**Nutch网页爬取总结**

目录

[前言 2](#_Toc335579294)

[Nutch搭建过程[1] 2](#_Toc335579295)

[1 准备工作 2](#_Toc335579296)

[2 开始部署 2](#_Toc335579297)

[2.1 安装linux系统 2](#_Toc335579298)

[2.2 安装jdk 2](#_Toc335579299)

[2.3.配置nutch创建索引 3](#_Toc335579300)

[2.4安装tomcat 6](#_Toc335579301)

[2.5配置nutch查询索引 6](#_Toc335579302)

[参考文献 9](#_Toc335579303)

[Nutch爬取内容解析[1] 10](#_Toc335579304)

[1 Nutch的输出文件 10](#_Toc335579305)

[1.1 Crawldb 10](#_Toc335579306)

[1.2 Linkdb 10](#_Toc335579307)

[1.3 Segments 10](#_Toc335579308)

[1.4 Indexes 12](#_Toc335579309)

[1.5 Index 12](#_Toc335579310)

[2 Nutch爬取内容解析 13](#_Toc335579311)

[2.1 nutch的所有命令[2] 13](#_Toc335579312)

[2.2 nutch的页面转存命令 19](#_Toc335579313)

[2.3 nutch页面转存程序 21](#_Toc335579314)

[参考文献 21](#_Toc335579315)

[Nutch的爬取设置和配置文件 22](#_Toc335579316)

[1 nutch的爬取深度 22](#_Toc335579317)

[2 nutch的配置文件[2] 23](#_Toc335579318)

[2.1 配置文件的优先级 23](#_Toc335579319)

[2.2 配置文件详解 25](#_Toc335579320)

[2.3 nutch抓取动态网页配置[3] 33](#_Toc335579321)

[2.4 关于配置文件的一些细节问题[4][5] 34](#_Toc335579322)

[参考文献 36](#_Toc335579323)

[后序内容 37](#_Toc335579324)

# 前言

学习使用nutch的时间加起来能有一个月了，在网上也搜集了不少的资料，但是比较零散，所以想把这一个月使用它的全部过程、原理的讲解和中间的注意事项总结一下。主要汇总写网上已有的资料，方便查阅嘛。

# Nutch搭建过程[1]

## 1 准备工作

我是在virtualbox下用虚拟机进行的nutch搭建，所需要的工具如下：

（1）Virtualbox最新版

（2）ubuntu-desktop最新版（以后会改为ubuntu-server）

（3）jdk-6u25-linux-i586.bin

（4）apache-nutch-1.2-bin.tar.gz

## 2 开始部署

### 2.1 安装linux系统

首先安装VirtualBox，然后安装ubuntu，就是传统的安装方法。上传所用的包到linux系统~/tmp目录下,~表示用户目录。

### 2.2 安装jdk

终端命令**[收藏代码](javascript:void())**

1. #cd  ~/tmp
2. #chmod a+x  jdk-6u23-linux-i586.bin
3. #sh jdk-6u23-linux-i586.bin
4. #mkdir /usr/java
5. #mv jdk1.6.0\_23  /usr/java/

**配置用户环境变量[2]**

这个比较重要，最开始nutch多次爬取失败，就是因为环境变量配置的不对。

最开始是在~/.bash\_profile下配置，后来又在/etc/profile下配置，但是由于nutch有些命令需要管理员权限，当进入管理员权限时，上述配置的环境变量就不生效了，所以最后在/etc/environment下配置环境变量才好使。

在/etc/profile下配置环境变量（这个最好也配置上吧）。

终端命令**[收藏代码](javascript:void())**

1. #sudu gedit /etc/profile
2. （在该文件中添加如下内容）
3. export JAVA\_HOME=/usr/java/jdk1.6.0\_25
4. export JRE\_HOME=/usr/java/jdk1.6.0.25/jre
5. export CLASSPATH=.:$JAVA\_HOME/lib:$JRE\_HOME/lib:$CLASSPATH
6. export PATH=$JAVA\_HOME/bin:$JRE\_HOME/bin:$PATH

在/etc/environment下配置环境变量。

终端命令**[收藏代码](javascript:void())**

1. sudu gedit /etc/environment
2. （在该文件中添加如下内容）
3. JAVA\_HOME=/usr/java/jdk1.6.0\_25
4. JRE\_HOME=/usr/java/jdk1.6.0.25/jre
5. CLASSPATH=.:$JAVA\_HOME/lib:$JRE\_HOME/lib:$CLASSPATH
6. PATH直接在该文件本来有的PATH后面添加即可

注意1：每段路径中间用冒号（:）分开，别忘记了那个点（.）

注意2：在尝试的过程中，environment文件内容的错误添加可能导致电脑无法进入，反复的停留在登陆界面。（我在最开始修改这个文件的时候遇到过），只需按crl+alt+F2，在命令行界面把错误的修改删除即可。

**测试**

终端命令**[收藏代码](javascript:void())**

1. #java –version

**输出**

1. java version "1.6.0\_25"
2. Java(TM) SE Runtime Environment (build 1.6.0\_25-b05)
3. Java HotSpot(TM) Client VM (build 19.0-b09, mixed mode, sharing)

Jdk安装成功

另外，在终端下输入env，能够看到所有已经配置的环境变量，看看是否有这几个变量。输入sudo su，然后输入env，看在管理员权限下，是否有这几个变量。

### 2.3.配置nutch创建索引

终端命令**[收藏代码](javascript:void())**

1. #unzip apache-nutch-1.2-bin.zip
2. #mv nutch-1.2  /opt/
3. #cd /opt/nutch-12

终端命令**[收藏代码](javascript:void())**

1. #vim conf/crawl-urlfilter.txt

注1：这里用vim，vi，gedit都可以，如果是vim的话，可能还需要下载vim，使用apt-get install命令安装。

注2：crawl-urlfilter.txt是用来配置所爬取网站的范围，域名和它的子网页的正则表达式，类似于爬取规则。这里是以爬取新浪网第一页为例子的。

找到该文件中如下内容的部分

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

更改为

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

接下来修改nutch-site.xml文件，这类似于对我要爬取的网站进行一下声明，不声明的话，会导致爬取失败

终端命令**[收藏代码](javascript:void())**

1. #vim conf/nutch-site.xml

在该文件中添加

1. **<configuration>**
2. **<property>**
3. **<name>**http.agent.name**</name>**
4. **<value>**HD nutch agent**</value>**
5. **</property>**
6. **<property>**
7. **<name>**http.agent.version**</name>**
8. **<value>**1.0**</value>**
9. **</property>**
10. **</configuration>**

这里面的值（value）应该是可以更改的（并未进行过尝试）。

然后建立种子文件，说明爬取的网站域名列表，现在只有新浪网一个。

终端命令**[收藏代码](javascript:void())**

1. #cd bin
2. #mkdir urls
3. #vim urls/nutch.txt

这个文件的名称是自定义的，如nutch.txt，seed.txt，url.txt，随意。在该文件中添加需要爬取的网站的最高层域名。这里添加内容如下

1. http://www.sina.com.cn/

此后创建日志，用来基础爬取时的输出情况，不指定输出文件的话，默认是输出到终端。

终端命令**[收藏代码](javascript:void())**

1. #mkdir logs
2. #vim logs/log1.log
3. :wq

创建索引，用来存储爬取下来的所有结果。

终端命令**[收藏代码](javascript:void())**

1. #mkdir /opt/index

此后便可以进行爬取工作，输入nuch创建索引shell指令,这也是nuch不能在window下安装原因,在windows需要装虚拟机或用cygwin(在windows下运行unix shell command需要)。而且不能在win7下用cygwin模拟，试了一天才知道这一点。爬取命令如下，得加sudo，因为读写了home的上层目录。

终端命令**[收藏代码](javascript:void())**

1. # sudo sh
2. # bin/nutch crawl urls –dir /opt/index –depth 10 –threads 2 –topN 100000 | tee logs/log1.log

crawl：通知nutch.jar，执行crawl的main方法。

urls：存放需要爬行的url.txt文件的目录

dir：爬行后文件保存的位置

depth：爬行次数，或者成为深度，不过还是觉得次数更贴切，建议测试时改为1

threads 指定并发的进程这是设定为4

topN ：一个网站保存的最大页面数。

终端命令**[收藏代码](javascript:void())**

1. #ls /opt/index

下面有索引内容表示成功,没的话看logs/log1.log日记的异常。

到此算是爬取的过程。

**可能出现的错误如下[3][4]**

|  |
| --- |
| Exception in thread "main" java.io.IOException: Job failed!     at org.apache.hadoop.mapred.JobClient.runJob(JobClient.java:1252)     at org.apache.nutch.crawl.Injector.inject(Injector.java:226)     at org.apache.nutch.crawl.Crawl.main(Crawl.java:124) |

这是由于JAVA\_HOME没有设置好造成的，重新在/etc/environment中设置，并重启就好了。

|  |
| --- |
| Exception in thread "main" java.io.IOException: Job failed!  at org.apache.hadoop.mapred.JobClient.runJob(JobClient.java:1252)  at org.apache.nutch.crawl.Injector.inject(Injector.java:217)  at org.apache.nutch.crawl.Crawl.main(Crawl.java:124) |

一说是语言设置的不对，因为函数返回值的类型无法匹配造成，需要在环境变量（/etc/environment）中配置“LANG="zh\_CN.GBK"，我尝试了，但是没有解决。后来查看源代码，是因为有一个文件没有找到造成的，需要把根目录中的nutch-1.2.jar拷贝到lib子目录中去。，因为在/bin/nutch的文件中需要读取这个包，但是这个包的路径不正确。”这个问题很严重，尤其在最初搭建的时候，很多的网址几乎都没有写这个解决方案。导致很久爬取不成功。

应该没有其他的问题了，至少我没遇到。

### 2.4安装tomcat

终端命令**[收藏代码](javascript:void())**

1. #cd  ~tmp
2. # tar  xzvf  apache-tomcat-6.0.29.tar.gz
3. #mv apache-tomcat-6.0.29 tomcat
4. #mv tomcat  /usr/local/

启动tomcat

终端命令**[收藏代码](javascript:void())**

1. #/usr/local/tomcat/bin/startup.sh

会有如下的输出

Using CATALINA\_BASE:   /usr/local/tomcat

Using CATALINA\_HOME:   /usr/local/tomcat

Using CATALINA\_TMPDIR: /usr/local/tomcat/temp

Using JRE\_HOME:        /usr/java/jdk1.6.0\_23

Using CLASSPATH:       /usr/local/tomcat/bin/bootstrap.jar

终端命令**[收藏代码](javascript:void())**

1. #netstat –tln

查看8080端口是否开启

测试访问[http://localhost:8080](http://localhost:8080/)

正常出现tomcat首页,测试成功后关闭tomcat

终端命令**[收藏代码](javascript:void())**

1. #/usr/local/tomcat/bin/shutdown.sh

### 2.5配置nutch查询索引

把nutch的war包到tomcat的webapps目录下

终端命令**[收藏代码](javascript:void())**

1. #cd /opt/nutch-1.2

终端命令**[收藏代码](javascript:void())**

1. #mv nutch-1.2.war /usr/local/tomcat/webapps

启动tomcat

终端命令**[收藏代码](javascript:void())**

1. #/usr/local/tomcat/bin/startup.sh

在webapps下会把nutch-1.2.war 解压出nutch-1.2文件包

测试 <http://localhost:8080/nutch-1.2>

进入nutch 主页,输入”新浪”,你发现没有结果而且出现乱码,下面解决这些问题先关闭tomcat。

终端命令**[收藏代码](javascript:void())**

1. #/usr/local/tomcat/bin/shutdown.sh

配置索引文件位置，因为并没有把刚才爬取下的网页和服务器关联。

终端命令**[收藏代码](javascript:void())**

1. #cd /usr/local/tomcat/webapps/nutch-1.2/WEB-INF/classes/

终端命令**[收藏代码](javascript:void())**

1. #vim nutch-site.xml

Xml代码**[收藏代码](javascript:void())**

1. **<configuration>**
2. **<property>**
3. **<name>**searcher.dir**</name>**
4. **<value>**/opt/index**</value>**
5. **</property>**
6. **</configuration>**

配置tomcat 编码，在server.xml文件的对应部分更改即可，同时删去原来的配置。

终端命令**[收藏代码](javascript:void())**

1. #cd /usr/local/tomcat/

终端命令**[收藏代码](javascript:void())**

1. #vim conf/server.xml

Xml代码**[收藏代码](javascript:void())**

2. **<Connector** port="8080"
3. maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
4. enableLookups="false" redirectPort="8443" acceptCount="100"
5. connectionTimeout="20000" disableUploadTimeout="true"
6. URIEncoding="UTF-8" useBodyEncodingForURI="true" **/>**

网页快照乱码问题

终端命令**[收藏代码](javascript:void())**

1. # cd /usr/local/tomcat/nutch-1.2
2. #vim cached.jsp

修改其中的第63行。

原来的代码是：content = new String(bean.getContent(details);

修改后的代码是：content = new String(bean.getContent(details),"gb2312");

保存:wq

重启tomcat

终端命令**[收藏代码](javascript:void())**

1. #/usr/local/tomcat/bin/startup.sh

地址栏输入<http://localhost:8080/nutch-1.2/zh/>输入新浪点查询。



图1 tomcat服务器下nutch网页的原始结果

结果类似于图1，我是爬取了两层深度的网页，所以结果比较多。

到此，爬取部分的搭建过程结束。

## 参考文献

[1] [linux下部署nutch-1.2](http://sunwendong.iteye.com/blog/1033733) <http://sunwendong.iteye.com/blog/1033733>

[2] [Ubuntu手动安装JDK](http://www.iteye.com/topic/340916) <http://www.iteye.com/topic/340916>

[3] Nutch ubuntu安装解惑

<http://vera8733.blog.163.com/blog/static/173698346201131234742609/>

[4] [ubuntu下nutch-1.0的安装和配置错误排除](http://deepfuture.iteye.com/blog/573988) <http://deepfuture.iteye.com/blog/573988>

# Nutch爬取内容解析[1]

## 1 Nutch的输出文件

nutch爬取下来的网页信息，保存在如图1的目录结构中

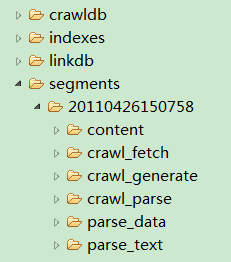


图 1

Nutch的输出文件（不包括临时文件）主要可分为crawldb、index、indexs、linkdb和segments。

### 1.1 Crawldb

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）

web link目录，存放 url 及 url 的互联关系，作为爬行与重新爬行的依据，页面默认30 天过期（可以在 nutch-site.xml 中配置）。

### 1.2 Linkdb

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。读取出来的格式如下（图2）：



图2

### 1.3 Segments

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“20061014163012”，打开此文件夹可以看到，它下面还有6个子文件夹，分别是：

crawl\_generate ：names a set of urls to be fetched

crawl\_fetch ： contains the status of fetching each url

crawl\_parse ： contains the outlink urls, used to update the crawldb

content ： contains the content of each url

parse\_text ： contains the parsed text of each url

parse\_data ： contains outlinks and metadata parsed from each url

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。格式如下：

////////////////////crawl\_generate、crawl\_fetch、crawl\_parse、crawldb的格式//////////////

http://www.163.com/ Version: 4

Status: 5 (fetch\_success)

Fetch time: Thu Oct 26 19:20:33 CST 2006

Modified time: Thu Jan 01 08:00:00 CST 1970

Retries since fetch: 0

Retry interval: 30.0 days

Score: 1.0

Signature: 3029c621e26c43797241a8d34980b2dc

Metadata: null

parse\_data的格式如下：（主要是每个URL解析出的外部链接和元数据）



outlink: toUrl: http://cha.so.163.com/so.php?key=001001006&q=??? anchor: 手机

Content Metadata: nutch.crawl.score=1.0 nutch.content.digest=3029c621e26c43797241a8d34980b2dc Content-Type=text/html;

charset=GB2312 date=Thu, 26 Oct 2006 11:20:33 GMT Cache-Control=max-age=146

Content-Encoding=gzip Content-Length=19628 Connection=close Expires=Thu, 26

Oct 2006 11:23:00 GMT nutch.segment.name=20061026192026 Accept-Ranges=bytes

Server=Apache/2.0.55 (Unix) X-Cache=HIT from www.163.com Vary=Accept-Encoding

Parse Metadata: OriginalCharEncoding=GB2312 CharEncodingForConversion=GB2312

parse\_text是网页的文本信息。

### 1.4 Indexes

索引目录，我运行时生成了一个“part-00000”的文件夹。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/\*。

### 1.5 Index

lucene的索引目录（使用luke工具查看），是indexs里所有index合并后的完整索引，注意索引文件只对页面内容进行索引，没有进行存储，因此查询时要去访问 segments目录才能获得页面内容。在非分布式搜索时优先根据查找index文件夹，如果不存在则查找indexs。Indexes：存放每次下载的独立索引目录。（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。）

## 2 Nutch爬取内容解析

### 2.1 nutch的所有命令[2]

nutch的所有命令如图3所示

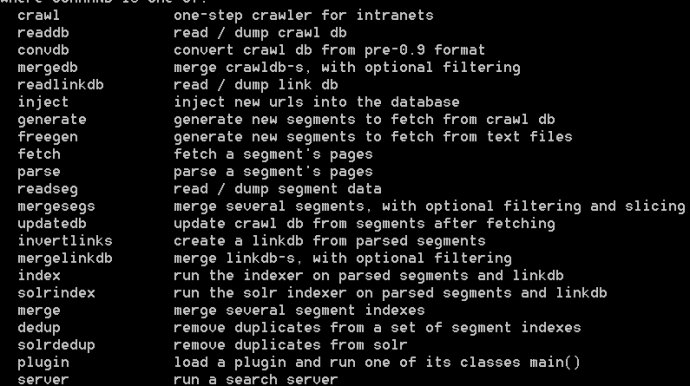


图3

**（1）bin/nutch crawl <urlDir> [-dir d] [-threads n] [-depth i] [-topN]**

Crawl是“org.apache.nutch.crawl.Crawl”的别称，它是一个完整的爬取和索引过程命令。

参数说明：   
    <urlDir>：包括URL列表的文本文件，它是一个已存在的文件夹。   
    [-dir <d>]：Nutch保存爬取记录的工作目录，默认情况下值为：./crawl-[date]，其中[date]为当前目期。   
    [-threads <n>]：Fetcher线程数，覆盖默认配置文件中的fetcher.threads.fetch值（默认为10）。   
    [-depth <i>]：Nutch爬虫迭代的深度，默认值为5。   
    [-topN <num>]：限制每一次迭代中的前N条记录，默认值为 Integer.MAX\_VALUE。

配置文件：   
hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml、crawl-tool.xml

   其他文件：   
   crawl-urlfilter.txt

**（2）bin/nutch readdb <crawldb> (-stats | -dump <out\_dir> | -url <url>)**

Readdb命令是“org.apache.nutch.crawl.CrawlDbReader”的别称，返回或者导出Crawl数据库（crawldb）中的信息。

参数说明：   
<crawldb>：crawldb目录.   
[-stats]:：在控制台打印所有的统计信息   
[-dump <out\_dir>]：导出crawldb信息到指定文件夹中的文件   
[-url <url>]：打印指定URL的统计信息   
实例：   
Shell代码

$ bin/nutch readdb fullindex/crawldb -stats

CrawlDb statistics start: fullindex/crawldb

Statistics for CrawlDb: fullindex/crawldb

TOTAL urls:     468030

retry 0:        467361

retry 1:        622

retry 2:        32

retry 3:        15

min score:      0.0

avg score:      0.0034686408

max score:      61.401

status 1 (db\_unfetched):        312748

status 2 (db\_fetched): 80671

status 3 (db\_gone):     69927

status 4 (db\_redir\_temp):       1497

status 5 (db\_redir\_perm):       3187

CrawlDb statistics: done

配置文件：   
hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

说明：

-stats命令是一个快速查看爬取信息的很有用的工作，其输出信息表示了：

DB\_unfetched：链接到已爬取页面但还没有被爬取的页面数（原因是它们没有通过url过滤器的过滤，或者包括在了TopN之外被Nutch丢弃）

DB\_gone：表示发生了404错误或者其他一些臆测的错误，这种状态阻止了对其以后的爬取工作。

DB\_fetched表示已爬取和索引的页面，如果其值为0，那肯定出错了。

**（3）bin/nutch readlinkdb <linkdb> (-dump <out\_dir> | -url <url>)**

readlinkdb是"org.apache.nutch.crawl.LinkDbReader"的别称，导出链接库中信息或者返回其中一个URL信息。

参数说明：   
<linkdb>：linkdb工作目录   
[-dump <out\_dir>]：导出信息到文件夹下   
[-url <url>]：打印某个URL的统计信息

配置文件：   
hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

**（4）bin/nutch injector <crawldb> <urldir>**

injector是"org.apache.nutch.crawl.Injector"的别称，注入新URL到crawldb中。

参数说明：   
<crawldb>：crawldb文件夹   
<urldir>：保存有URL的文件的文件夹目录

配置文件：

hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

以下配置文件参数影响到了注入方式：

db.default.fetch.interval——按天设置爬取间隔，默认值30.0f

db.score.injected——设置URL的默认打分，默认值1.0f

urlnormalizer.class——规范化URL的类，默认值为 org.apache.nutch.net. BasicUrlNormalizer

**（5）bin/nutch generate <crawldb> <segments\_dir> [-topN <num>] [-numFetchers <fetchers>] [-adddays <days>]**

generate是“org.apache.nutch.crawl.Generator”，从Crawldb中抓取新的Segment。

参数说明：

<crawldb>：crawldb目录

<segments\_dir>：新建的爬取Segment目录

[-topN <num>]：选取前多少个链接，默认值为Long.MAX\_VALUE

[-numFetchers <fetchers>]：抓取分区数量。 Default: Configuration key -> mapred.map.tasks -> 1

[-adddays <days>]: 添加 <days>到当前时间，配置crawling urls ，以将很快被爬取db.default.fetch.interval默认值为0。爬取结束时间在当前时间以前的。

配置文件：

hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

说明：

generate.max.per.host – 设置单个主机最大的URL数量，默认情况下：unlimited。

**（6）bin/nutch fetch <segment> [-threads <n>] [-noParsing]**

fetch是“org.apache.nutch.fetcher.Fetcher”的代称，它负责一个segment的爬取。

参数说明：

<segment>：segment目录

[-threads <n>]：运行的fetcher线程数默认值为 Configuration Key -> fetcher.threads.fetch -> 10

[-noParsing]：禁用自动解析segment数据

配置文件：

hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

说明：

Fetcher依赖于多个插件以爬取不同的协议，目前已有的协议及支撑插件如下：

http: protocol-http、protocol-httpclient

https: protocol-httpclient

ftp: protocol-ftp

file: protocol-file

当爬取网上文档的时候，不应该使用protocol-file，因为它是用于爬取本地文件的。如果你想爬取http、https，应当使用protocol-httpclient。

**（7）bin/nutch parse <segment>**

parse是“org.apache.nutch.parse.ParseSegment”的代称，它对一个segment运行ParseSegment。

参数说明：

<segment>：Segment文件夹

配置文件：

hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

说明：

它依赖于多个插件来解析不同格式的内容，支持的格式及插件有：

|  |  |  |
| --- | --- | --- |
| 内容格式 | 插件 | 备注 |
| text/html | parse-html | 使用NekoHTML 或者TagSoup解析HTML |
| application/x-javascript | parse-js | 解析JavaScript 文档(.js) |
| audio/mpeg | parse-mp3 | 解析MP3 Audio文档(.mp3). |
| application/vnd.ms-excel | parse-msexcel | 解析MSExcel文档 (.xls). |
| application/vnd.ms-powerpoint | parse-mspowerpoint | MSPower!Point 文档 |
| application/msword | parse-msword | 解析MSWord文档 |
| application/rss+xml | parse-rss | 解析RSS文档(.rss) |
| application/rtf | parse-rtf | 解析RTF文档(.rtf) |
| application/pdf | parse-pdf | 解析PDF文档 |
| application/x-shockwave-flash | parse-swf | 解析Flash 文档 (.swf) |
| text-plain | parse-text | 解析Text文档(.txt) |
| application/zip | parse-zip | 解析Zip文档(.zip) |
| other types | parse-ext | 通过基于content-type或者路径前缀的外部命令来解析文档 |

默认情况下只有txt、HTML、JS格式的插件可用，其他的需要在nutch-site.xml中配置使用。

**（8）bin/nutch segread <segment>**

segread是"org.apache.nutch.segment.SegmentReader"的代称，它读取并导出Segment数据。

参数说明：

<segment>：Segment文件夹

配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

说明：

在Nutch0.9后的版本中改为了readseg

**（9）bin/nutch updatedb <crawldb> <segment> [-noadditions]**

updatedb是“org.apache.nutch.crawl.CrawlDb”的代称，用fetch过程中获取的信息更新crawldb。

参数说明：

<crawldb>：crawldb目录

<segment>：已经爬取的segment目录

[-noadditions]：是否添加新的链接到crawldb中

配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

**（10）bin/nutch invertlinks <linkdb> (-dir segmentsDir | segment1 segment2 ...)**

invertlinks是“org.apache.nutch.crawl.LinkDb”的代称，它用从segment中获取到的信息更新linkdb。

参数说明：   
<linkdb>: linkdb目录   
<segment>: segment目录，可以指定至少一个的文件夹

配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

**（11）bin/nutch index <index> <crawldb> <linkdb> <segment> ...**

index是“org.apache.nutch.indexer.Indexer”的代称，创建一个segment的索引，利用crawldb和linkdb中的数据对索引中的页面打分。

参数说明：   
<index>: 索引创建后的保存目录   
<crawldb>: crawldb目录   
<linkdb>: linkdb目录   
<segment>: segment目录，可以指定多个

配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

**（12）bin/nutch merge [-workingdir <workingdir>] <outputIndex> <indexesDir> ...**

Merge是“org.apache.nutch.indexer.IndexMerger”的代称，它合并多个segment索引。

参数说明：   
[-workingdir <workingdir>]：提定工作目录   
<outputIndex>：合并后的索引存储目录   
<indexesDir>：包含待合并的索引目录，可以指定多个

配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

**（13）bin/nutch merge output\_crawldb crawldb1 [crawldb2 crawldb3 ...] [-filter]**

它是“org.apache.nutch.crawl.CrawlDbMerger”的代称，合并多个CrawlDb，URLFilter可选择性地过滤指定内容。可以合并多个DB到一个中。当你分别运行爬虫并希望最终合并DB时，它会相当有用。可选择地，可以运行当前URLFilter过滤数据库中的URL，以滤去不需要的URL。当只有一个DB时也很有用，它意味着你可以通过这个工作去滤掉那些DB中你不想要的URL。 只用这个工具来过滤也是可能的，在这种情况下，只指定一个crawldb。 如果同一个URL包括在多个CrawlDb中，只有最近版本的才会被保留，即由org.apache.nutch.crawl.CrawlDatum.getFetchTime()值决定的。然而，所有版本的元数据被聚合起来，新的值代替先前的值。

参数说明：   
output\_crawldb：CrawlDb输出文件夹   
crawldb1 [crawldb2 crawldb3 ...]：一个或者多个CrawlDb(s).   
-filter：采用的URLFilters

配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

**（14）bin/nutch mergelinkdb output\_linkdb linkdb1 [linkdb2 linkdb3 ...] [-filter]**

它是“org.apache.nutch.crawl.LinkDbMerger”的代称，用于合并多个linkdb，可以选择性的使用URLFilter来过滤指定内容。 当分别从多个segment群中分布式建立LinkDb而又需要合并为一个时很有用。或者，也可以指定单个LinkDb，只是用它来过滤URL。只用这个工具来过滤也是可能的，在这种情况下，只指定一个LinkDb。如果一个URL包含在多个LinkDb中，所有的内部链接被聚合，但是最多db.max.inlinks 指定的内链数会添加进来。 如果被激活，URLFilter可以应用到所有的目标URL及其内链中。如果目标链接被禁止，所有的该目标链接的内链将和目标链接一起被移去。如果某些内链被禁止，那么只有他们会被移去，在校验上面提到的最大限制数时他们不会被计算在内。

参数说明：

output\_linkdb：输出linkdb

linkdb1 [linkdb2 linkdb3 ...]: 多于一个的输入LinkDb(s)

-filter: Actual URLFilters to be applied on urls and links in LinkDb(s).   
配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

**（15）mergesegs bin/nutch mergesegs output\_dir (-dir segments | seg1 seg2 ...) [-filter] [-slice NNNN]**

它是“org.apache.nutch.segment.SegmentMerger”的代称，用于合并多个segment，可以选择性地输出到一个或者多个固定大小的segment中。

参数说明：   
output\_dir：结果segment的名称或者segment片的父目录   
-dir segments：父目录，包括多个segment   
seg1 seg2 ...：segment目录列表   
-filter：通过URLFilters过滤   
-slice NNNN: 创建多个输出segment，每一个中包括了NNNN个URL。

配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

**（16）bin/nutch dedup <indexes> ...**

dedup是“org.apache.nutch.indexer.DeleteDuplicates”的别名，它segment indexes中去掉重复的页面。

参数说明：   
<indexes>：indexes索引文件

配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

**（17）bin/nutch plugin <pluginId> <className> [args ...]**

plugin是“org.apache.nutch.plugin.PluginRepository”的代称，用于从插件库中加载一个插件并执行其主方法。

参数说明：   
<pluginId>：期望执行的插件ID   
<className>：包含主方法的类名   
[args]：传入插件的参数

配置文件：hadoop-default.xml、hadoop-site.xml、nutch-default.xml、nutch-site.xml

### 2.2 nutch的页面转存命令

其中关于把爬取下来的页面信息转存为可读格式的命令有如下几个：

（1）查看crawldb数据库

查看url地址总数和它的状态及评分：bin/nutch readdb url/crawldb/ -stats

查看每个url地址的详细内容，导出数据：bin/nutch readdb url/crawldb/ -dump crawldb(导出的地址)

查看具体的url，以163为例：bin/nutch readdb url/crawldb/ -url http://www.163.com/

（2）查看linkdb数据库

查看链接情况：bin/nutch readlinkdb url/linkdb/ -url http://www.163.com/

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

（3）查看segments

bin/nutch readseg -list -dir url/segments/ 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。

bin/nutch readseg -dump url/segments/20090309103156 segdb 导出segments,其中url/segments/20090309103156为一个segments文件夹，segdb为存放转换后的内容的文件夹。

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。[3]

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

在实际中，进入nutch的安装目录，运行命令：

 终端命令**[收藏代码](javascript:void())**

1. # bin/nutch readseg -dump db/segments/20090525140014 segdb -nocontent -nofetch -nogenerate -noparse -noparsedata

db/segments/20090525140014为一个segments文件夹，它是爬取时根绝时间命名的（-no\*\*可以根据需要增减）。segdb为存放转换后的内容的文件夹。此后会在当前目录生成segdb文件夹，里面有一个名为dump的文件，它可以用vi、记事本等直接打开，内容为网页的url和文本信息（已去掉所有标签）。如图4。

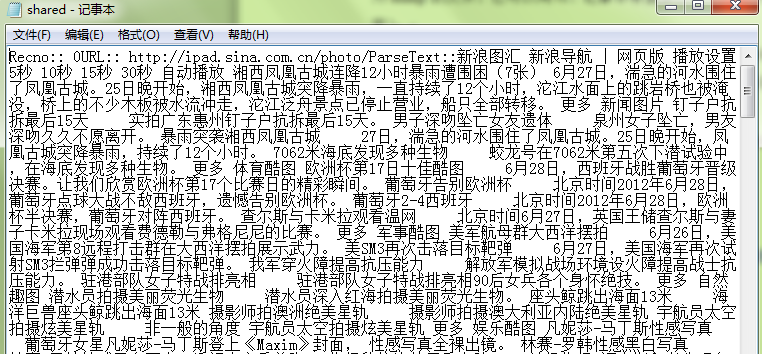


图4

### 2.3 nutch页面转存程序

另外这种转存可以通过java程序实现，需要加载nutch-1.2.jar包，主要利用nutch自带的解析器，因为通过命令可以实现，就并没有尝试这种方法，可以参照如下三个网页。

1. nutch索引里面,读取文章内容

<http://hi.baidu.com/sunky/blog/item/40702ff5c42a9b25bd310949.html>

1. Nutch:读取nutch抓取内容

<http://www.haogongju.net/art/141084>

1. [Nutch中的html页面的解析问题](http://www.blogjava.net/kxx129/archive/2010/04/23/319209.html)

<http://www.blogjava.net/kxx129/archive/2010/04/23/319209.html>

## 参考文献

[1] Nutch读取网页字段

<http://blog.163.com/bit_runner/blog/static/53242218201141393943980/>

[2] nutch命令详解<http://blog.csdn.net/an342647823/article/details/7303033>

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

# Nutch的爬取设置和配置文件

## 1 nutch的爬取深度

在网页抓取时我们使用的命令如下：

终端命令**[收藏代码](javascript:void())**

1. # sudo sh bin/nutch crawl urls –dir /opt/index –depth 1 –threads 2 –topN 10 >&logs/log1.log

关于这个命令我遇到的问题是。可以在linux终端下首先输入“#sudo su”获得管理员权限，因为nutch需要这个权限。“-topN”选项可以不用填写，否则爬取不到所有的网页。“-depth”是爬取深度，大多数人对这个深度是有疑问的，因为在传统的网站建设中，网站的每一层目录是一个深度，但是在nutch中，是当前页面的所有超链接算作下一层深度。在设置时要考虑需要爬取多少页面，设定合适的深度。

有人做过一个实验，来验证nutch对深度的理解。[1]

先在Tomcat里面建一个示例网站Test，网站根目录下建四个网页index.html, a.thml, b.html, c.html.建一个文件夹1，内有1.html，在文件夹1中建文件夹2，建2.html。它们的链接关系：index.html中有a.html和1.html的链接，a.html中有b和c的链接，1.html中有2.html的链接。这个示例网站是按我原来的意思建的：第一层有四个文件，且包含了蜘蛛能访问到的链接，第二层有一个文件1.html,第三层为2.html。这是一个以文件夹为深度的例子。

启动nutch的爬虫爬行网站。

终端命令[收藏代码](javascript:void())

1. nutch crawl  urls -dir c:\Test -threads 5 -depth 1

先爬行一层，如果是以文件夹为深度，那么第一层的4个文件都将爬到。让我看下结果：

显示结果[收藏代码](javascript:void())

1. F:\code\nutch\bin>nutch readdb  c:\Test\db -dumppageurl
2. 061127 150112 parsing jar:file:/F:/code/nutch/nutch.jar!/nutch-**default**.xml
3. 061127 150112 parsing jar:file:/F:/code/nutch/nutch.jar!/nutch-site.xml
4. 061127 150112 No FS indicated, using **default**:local
5. org.apache.nutch.db.WebDBReader@13f5d07
7. Page 1: Version: 4
8. URL: http://localhost:8080/Test/
9. ID: 59ba405710e0b0a631562a040e30865c
10. Next fetch: Wed Dec 27 14:55:52 CST 2006
11. Retries since fetch: 0
12. Retry interval: 30 days
13. Num outlinks: 2
14. Score: 1.0
15. NextScore: 1.0

18. Page 2: Version: 4
19. URL: http://localhost:8080/Test/1/1.html
20. ID: 2c6dadf23768abaca03c8f7135b99436
21. Next fetch: Mon Nov 27 14:55:52 CST 2006
22. Retries since fetch: 0
23. Retry interval: 30 days
24. Num outlinks: 0
25. Score: 1.0
26. NextScore: 1.0

29. Page 3: Version: 4
30. URL: http://localhost:8080/Test/a.html
31. ID: a588737b7cd966929644a731445dffeb
32. Next fetch: Mon Nov 27 14:55:52 CST 2006
33. Retries since fetch: 0
34. Retry interval: 30 days
35. Num outlinks: 0
36. Score: 1.0
37. NextScore: 1.0

实是只有三个网页被抓到了：index.html，1.html，a.html。b和c都没有抓进来。可以说nutch爬行深度是依据网页上的链接的，也就是说nutch把爬行的第一个页面上的链接当作第一层，依次类推第一层页面上的链接是第二层。还可以看三个数据的Num outlinks。index的有两个，1和a明明设了链接的值却为零。可见nutch爬虫在扫描页面链接的时候就把设定层外的链接给丢弃了。

把index中1的链接改成2的链接更可以验证nutch的深度是依据链接的。这样设计的爬虫容易控制。

## 2 nutch的配置文件[2]

### 2.1 配置文件的优先级

Nutch的配置文件主要有三类：

1.Hadoop的配置文件，Hadoop-default.xml和Hadoop-site.xml。

2.Nutch的配置文件，Nutch-default.xml和Nutch-site.xml。

3.Nutch的插件的配置文件，这些插件的配置文件在加载插件的时候由插件自行加载，如filter的配置文件。

配置文件的加载顺序决定了配置文件的优先级，先加载的配置文件优先级低，后加载的配置文件优先级高，优先级低的配置会被优先级高的配置覆盖。因此，了解Nutch配置文件加载的顺序对学习使用Nutch是非常必要的。下面我们通过对Nutch源代码的分析来看看Nutch加载配置文件的过程。

Nutch主要命令--crawl，下面我们就从crawl的main类（org.apache.nutch.crawl.Crawl)的main方法开始分析：

Crawl类main方法中加载配置文件的源码如下：

Configuration conf = NutchConfiguration.create();

conf.addResource("crawl-tool.xml");

JobConf job = new NutchJob(conf);

上面代码中，生成了一个NutchConfiguration类的对象，NutchConfiguration是Nutch管理自己配置文件的类，Configuration是Hadoop管理自己配置文件的类。下面我们进入NutchConfiguration类的create()方法。

/\*\* Create a {@link Configuration} for Nutch. \*/

public static Configuration create() {

Configration conf = new Configuration();

addNutchResources(conf);

return conf;

}

create()方法中，先生成了一个Configuration类的对象。请看Configuration类中的源码：

/\*\* A new configuration. \*/

public Configuration() {

this(true);

}

/\*\* A new configuration where the behavior of reading from the default

\* resources can be turned off.

\*

\* If the parameter {@code loadDefaults} is false, the new instance

\* will not load resources from the default files.

\* @param loadDefaults specifies whether to load from the default files

\*/

public Configuration(boolean loadDefaults) {

if (LOG.isDebugEnabled()) {

LOG.debug(StringUtils.stringifyException(new IOException("config()")));

}

if (loadDefaults) {

resources.add("hadoop-default.xml");

resources.add("hadoop-site.xml");

}

}

由上面代码可以看出，在创建Configuration对象的时候，会依次加载hadoop-default.xml和hadoop-site.xml这两个配置文件。所以Hadoop-site.xml中的配置会覆盖hadoop-default.xml中的配置。了解完Hadoop配置文件的加载，我们回到刚才的create()方法里面。创建了Configuration对象后，将调用addNutchResources(conf)方法。

/\*\* Add the standard Nutch resources to {@link Configuration}. \*/

public static Configuration addNutchResources(Configuration conf) {

conf.addResource("nutch-default.xml");

conf.addResource("nutch-site.xml");

return conf;

}

我们看到，先加载了nutch-default.xml文件，后加载了nutch-site.xml文件。所以nutch-site.xml中的配置会覆盖nutch-default.xml中的配置。下面我们回到crawl类的main方法，继续往下看。调用了conf.addResource("crawl-tool.xml");这表明crawl-tool.xml配置文件是最后加载。

通过上面简单的源码分析，我们不难看出Nutch配置文件的优先级。

Nutch自己的配置文件：crawl-tool.xml --> nutch-site.xml --> nutch-default.xml

hadoop的配置文件：hadoop-site.xml --> hadoop-default.xml

当然，因为nutch的配置文件后于Hadoop的配置文件的加载，nutch的配置也会覆盖Hadoop配置文件中的配置。需要明白的是覆盖的不是整个配置文件，而是单独的Property。

在实际应用时，可以不更改nutch-default.xml和hadoop-default.xml的内容，把需要更改的属性写到nutch-site.xml和hadoop-site.xml文件中，这样既方便阅读了解更改了哪些内容，又不会丢失默认的配置。

### 2.2 配置文件详解

Nutch 的配置文件几乎覆盖了Nutch 所有的功能。以前在网上看到过一个关于Nutch-default.XML 配置项解释，地址我没收藏，和本文配合起来看，会更好一些。

<property>

<name>http.max.delays</name>

<value>20</value>

<description>The number of times a thread will delay when trying to

fetch a page. Each time it finds that a host is busy, it will wait

fetcher.server.delay. After http.max.delays attepts, it will give

up on the page for now.</description>

</property>

爬虫的网络延时线程等待时间，以秒计时，默认的配时间是3秒，视网络状况而定。如果在爬虫运行的时候发现服务器返回了主机忙消息，则等待时间由fetcher.server.delay 决定，所以在网络状况不太好的情况下fetcher.server.delay 也设置稍大一点的值较好，此外还有一个http.timeout 也和网络状况有关系。

<property>

<name>http.content.limit</name>

<value>65536</value>

<description>The length limit for downloaded content, in bytes.

If this value is nonnegative (>=0), content longer than it will be truncated;

otherwise, no truncation at all.

</description>

</property>

描述爬虫抓取的文档内容长度的配置项。原来的值是 65536 ， 也就是说抓取到的一个文档截取 65KB左右，超过部分将被忽略，对于抓取特定内容的搜索引擎需要修改此项，比如XML文档。

<property>

<name>db.default.fetch.interval</name>

<value>30</value>

<description>The default number of days between re-fetches of a page.

</description>

</property>

这个功能对定期自动爬取需求的开发有用，设置多少天重新爬一个页面。

<property>

<name>fetcher.server.delay</name>

<value>5.0</value>

<description>The number of seconds the fetcher will delay between

successive requests to the same server.</description>

</property>

<property>

<name>fetcher.threads.fetch</name>

<value>10</value>

<description>The number of FetcherThreads the fetcher should use.

This is also determines the maximum number of requests that are

made at once (each FetcherThread handles one connection).</description>

</property>

<property>

<name>fetcher.threads.per.host</name>

<value>1</value>

<description>This number is the maximum number of threads that

should be allowed to access a host at one time.</description>

</property>

<property>

<name>fetcher.verbose</name>

<value>false</value>

<description>If true, fetcher will log more verbosely.</description>

</property>

这几个是爬虫线程的配置项，看名称就容易理解。

<property>

<name>parser.threads.parse</name>

<value>10</value>

<description>Number of ParserThreads ParseSegment should use.</description>

</property>

解析爬取到的文档线程数， 和爬虫线程对应，因为爬虫主要的处理类是有很多地方使用到了同步，所以此配置项和爬虫线程保持一直对处理有好处。

<property>

<name>fs.default.name</name>

<value>local</value>

<description>The name of the default file system. Either the

literal string "local" or a host:port for NDFS.</description>

</property>

分布式文件系统 使用的配置项，默认的是local 表示 使用本地文件系统，如果使用 host:port 的形式表示使用分布式文件系统NDFS，此处的文件系统地址是 nameserver ，也就是通过 bin/nutch namenode xxxx启动的 主机地址和端口号。

<property>

<name>ndfs.name.dir</name>

<value>/tmp/nutch/ndfs/name</value>

<description>Determines where on the local filesystem the NDFS name node

should store the name table.</description>

</property>

分布式文件系统namenode 使用的存放数据的目录，Namenode 会使用此项，另外在启动namenode或datanode的时候也可以加上 路径作为参数也可以生效。

<property>

<name>ndfs.data.dir</name>

<value>/tmp/nutch/ndfs/data</value>

<description>Determines where on the local filesystem an NDFS data node

should store its blocks.</description>

</property>

分布式文件系统ndatanode使用的存放数据的目录，datanode会使用此项，另外在启动 datanode的时候也可以加上 路径作为参数也可以生效。

<property>

<name>indexer.max.tokens</name>

<value>10000</value>

<description>

The maximum number of tokens that will be indexed for a single field

in a document. This limits the amount of memory required for

indexing, so that collections with very large files will not crash

the indexing process by running out of memory.

Note that this effectively truncates large documents, excluding

from the index tokens that occur further in the document. If you

know your source documents are large, be sure to set this value

high enough to accomodate the expected size. If you set it to

Integer.MAX\_VALUE, then the only limit is your memory, but you

should anticipate an OutOfMemoryError.

</description>

</property>

这个配置项的功能是限制索引的时候每个文档的单个字段最大10000个Tokens，也就是说在采用默认的一元分词的情况下，最大的文档字数限制是10000，如果采用其他中文非一元分词系统，则索引的单个文档单个字段将会超过10000个，对内存有影响。

<property>

<name>indexer.mergeFactor</name>

<value>200</value>

<description>The factor that determines the frequency of Lucene segment

merges. This must not be less than 2, higher values increase indexing

speed but lead to increased RAM usage, and increase the number of

open file handles (which may lead to "Too many open files" errors).

NOTE: the "segments" here have nothing to do with Nutch segments, they

are a low-level data unit used by Lucene.

</description>

</property>

合并因子，在建立索引的时候用到，表示索引200个文档的时候回写到存储设备。

<property>

<name>indexer.minMergeDocs</name>

<value>50</value>

<description>This number determines the minimum number of Lucene

Documents buffered in memory between Lucene segment merges. Larger

values increase indexing speed and increase RAM usage.

</description>

</property>

这个配置项对内存影响巨大，功能是在建立索引的时候最小的合并文档数量，这个值设置太小一个会影响索引速度，另外一个在需要索引的文档数量很大的时候会出现 Too Many Open files 的错误，这个时候需要调整此配置项，有试验表明1000的时候会有比较快的索引速度，但我把此项值调整到10000，索引的时候最高内存占用到1.8G，索引创建速度是25page/sec，并且多次索引的时候有一个衰减。不过对查询的相应时间有很大提升，如果内存足够的话修改较大的值比较好。

<property>

<name>indexer.maxMergeDocs</name>

<value>50</value>

<description>This number determines the maximum number of Lucene

Documents to be merged into a new Lucene segment. Larger values

increase indexing speed and reduce the number of Lucene segments,

which reduces the number of open file handles; however, this also

increases RAM usage during indexing.

</description>

</property>

这个好像不需要设置，因为默认的值是 Integer.MAX\_VALUE ， 不会比这个更大了。

<property>

<name>searcher.summary.context</name>

<value>5</value>

<description>

The number of context terms to display preceding and following

matching terms in a hit summary.

</description>

</property>

这个比较有用，在前面的文章里有介绍。

<property>

<name>searcher.summary.length</name>

<value>20</value>

<description>

The total number of terms to display in a hit summary.

</description>

</property>

在前面的文章里也有介绍。

<property>

<name>plugin.folders</name>

<value>plugins</value>

<description>Directories where nutch plugins are located. Each

element may be a relative or absolute path. If absolute, it is used

as is. If relative, it is searched for on the classpath.</description>

</property>

<property>

<name>plugin.includes</name>

<value>protocol-httpclient|urlfilter-regex|parse-(text|html|rss)|index-more|

query-(basic|site|url)</value>

<description>Regular expression naming plugin directory names to

include. Any plugin not matching this expression is excluded. By

default Nutch includes crawling just HTML and plain text via HTTP,

and basic indexing and search plugins.

</description>

</property>

这两个是配置插件功能的配置项，plugin.folders制定插件加载路径，plugin.includes表示需要加载的插件列表，关于插件后面会专门做介绍。

<property>

<name>parser.character.encoding.default</name>

<value>windows-1252</value>

<description>The character encoding to fall back to when no other information

is available</description>

</property>

解析文档的时候使用的默认编码windows-1252好像比较少用到的一种编码，我不太熟悉。

<property>

<name>parser.html.impl</name>

<value>neko</value>

<description>HTML Parser implementation. Currently the following keywords

are recognized: "neko" uses NekoHTML, "tagsoup" uses TagSoup.

</description>

</property>

制定解析HTML文档的时候使用的解析器，NEKO功能比较强大，后面会有专门的文章介绍Neko 从HTML到 TEXT以及html片断的解析等功能做介绍。

<property>

<name>extension.clustering.hits-to-cluster</name>

<value>100</value>

<description>Number of snippets retrieved for the clustering extension

if clustering extension is available and user requested results

to be clustered.</description>

</property>

聚合功能，对搜索结果有聚合需求的应用可能会用到。

<property>

<name>extension.ontology.extension-name</name>

<value></value>

<description>Use the specified online ontology extension. If empty,

the first available extension will be used. The "name" here refers to an 'id'

attribute of the 'implementation' element in the plugin descriptor XML

file.</description>

</property>

人工智能，这个功能在我以后的开发过程中会逐步深入，等我有相关的经验以后在给大家介绍。◎\_◎

<property>

<name>query.url.boost</name>

<value>4.0</value>

<description> Used as a boost for url field in Lucene query.

</description>

</property>

<property>

<name>query.anchor.boost</name>

<value>2.0</value>

<description> Used as a boost for anchor field in Lucene query.

</description>

</property>

<property>

<name>query.title.boost</name>

<value>1.5</value>

<description> Used as a boost for title field in Lucene query.

</description>

</property>

<property>

<name>query.host.boost</name>

<value>2.0</value>

<description> Used as a boost for host field in Lucene query.

</description>

</property>

<property>

<name>query.phrase.boost</name>

<value>1.0</value>

<description> Used as a boost for phrase in Lucene query.

Multiplied by boost for field phrase is matched in.

</description>

</property>

以上的几个关于搜索结果排序的分值计算因子在以后的搜索结果排序会专门做介绍，这几个项对垂直搜索的用处不太大。

<property>

<name>lang.analyze.max.length</name>

<value>2048</value>

<description> The maximum bytes of data to uses to indentify

the language (0 means full content analysis).

The larger is this value, the better is the analysis, but the

slowest it is.

</description>

</property>

和语言有关系，分词的时候会用到，不过我没用过这个配置项。

还有几个重要的配置项在 nutch-site.xml里面配置。

<property>

<name>searcher.dir</name>

<value>C:\</value>

</property>

可以有两种方式，如果指向的目录下面有search-servers.txt文件，那么优先处理 search-servers.txt文件中的内容，并解析其中符合hostname port格式的内容（即分布式查询请求），解析到后就想该服务器发送查询请求，如果没有则查找 segements 目录 ，segments 是本地索引文件。如果两个都没有找到，她就要报错了。search-servers.txt内容很简单，例如：127.0.0.1 9999。不过需要注意的是，这个9999的端口启动的是查询服务器，是用 bin/nutch server 9999 的命令启动的，和namenode启动比较相似，我当初接触的时候就以为是 namenode的地址，郁闷的很久。

namenode 和searchserver 结合不太好，没有提供直接从namenode到searchserver的文件访问接口，需要自己开发。

### 2.3 nutch抓取动态网页配置[3]

注意在conf下面的2个文件：regex-urlfilter.txt，crawl-urlfilter.txt  
# skip URLs containing certain characters as probable queries, etc.  
-[?\*!@=] （-改+）

这段意思是跳过在连接中存在? \* ! @ = 的页面，因为默认是跳过所以，在动态页中存在？一般按照默认的是不能抓取到的。可以在上面2个文件中都修改成：

# skip URLs containing certain characters as probable queries, etc.

# -[?\*!@=]

另外增加允许的一行（在实际中可以不用增加允许条目，直接注释掉-[?\*!@=]条目就是可以的）

# accept URLs containing certain characters as probable queries, etc.

+[?=&]

意思是抓取时候允许抓取连接中带 ? = & 这三个符号的连接

# accept hosts in MY.DOMAIN.NAME

Nutch的爬取是按照网页的连接一层一层的进行的，如果只包含入口地址和产品详细信息的地址，很可能由于无法爬取中间的过渡页面而导致下载暂定。采集是有顺序的，不是随便写的。比如：你要采集产品页，你首先得把首页放进来，然后产品是放在分类页面的，你得把分类也得包括进来，然后再把具体产品规则的正则写进来，这样才能完成你所需要的任务。如：

+^<http://www.china-nengyuan.com/product/product_list.php>$ #分类页面

+^[http://www.china-nengyuan.com/product/product\_list.php\?gopage=([[0-9]\*[1-9][0-9]\*])&](http://www.china-nengyuan.com/product/product_list.php\?gopage=(%5b%5b0-9%5d*%5b1-9%5d%5b0-9%5d*%5d)&)$ #“下一页的链接表达式，用来跳转下一页”

+^[http://www.china-nengyuan.com/product/([0-9]{}5).html](http://www.china-nengyuan.com/product/(%5b0-9%5d%7b%7d5).html)$ #产品详细信息页面  
# skip everything else  
-.

这样就可以爬取该网站的所有产品列表和产品的详细信息了。

另外有时候我们需要对爬取的页面中剔除一些特定格式的页面，可以在# accept hosts in MY.DOMAIN.NAME 的上面添加如下内容：

# skip hosts in MY.DOMAIN.NAME

-+^<http://www.china-nengyuan.com/product/product_list.php\?gopage=3&>$

意思是不下载产品列表第三页的内容。

注意：两个文件都需要修改，因为NUTCH加载规则的顺序是crawl-urlfilter.txt-> regex-urlfilter.txt。（修改配置文件的urlfilter.regex.file项可以更改读取的规则文件）

通过如此修改再加上合理的过滤正则表达式就可以爬取动态网页了，例如按照下一页的方式垂直爬取页面列表，如果出现问题，一般是由于配置文件产生的，很少是因为正则表达式书写的错误。

### 2.4 关于配置文件的一些细节问题[4][5]

如果不注意这些细节问题，可能导致莫名其妙的就爬取失败了，很多人经常会被莫名其妙的失败困扰，而纠结好几天。所以这里把我遇到的和配置文件有关的细节问题列举出来。可以说大部分的爬取失败，都是因为配置文件配置的不合理造成的。

o 设置http.agent.name，如果http.agent.name为空，爬虫将无法正常启动。这里可以设置任意你喜欢的名字，比如设置为vik-robot。

o indexer.mergeFactor/indexer.minMergeDocs 这两个值都改成500。这两个参数值越大，性能越高，所需要花费的内存也需要相应的增加。如果设置过大，可能出现内存溢出。就实际使用情况，在当前参数下，内存的最大使用量为3xxM。

o http.timeout表示最大超时等待时间，如果访问的连接长时间没有反应将丢弃。所以对于较偏远的网站，该值可以考虑设置大一些。

o db.max.outlinks.per.page该参数表示单个页面最多支持多少个外连的连接，如果过多将不抓取。如果是自己的内部系统那就设置一个任意大的数吧，比如-1。

o如何绕过目标站点的robots.txt限制

多数站点都是只允许百度、google等搜索引擎抓取的，所以会在robots.txt里限制其他爬虫。 nutch自然是会遵循robots协议的，但是我们可以通过修改nutch源码来绕过限制。   
相关代码位于（nutch版本1.5.1，其他版本未测试）： org.apache.nutch.fetcher.Fetcher的run方法. 找到以下几行代码并注释掉就OK了。

**Java代码  [收藏代码](javascript:void())**

1. **if** (!rules.isAllowed(fit.u)) {
2. // unblock
3. fetchQueues.finishFetchItem(fit, **true**);
4. **if** (LOG.isDebugEnabled()) {
5. LOG.debug("Denied by robots.txt: " + fit.url);
6. }
7. output(fit.url, fit.datum, **null**, ProtocolStatus.STATUS\_ROBOTS\_DENIED, CrawlDatum.STATUS\_FETCH\_GONE);
8. reporter.incrCounter("FetcherStatus", "robots\_denied", 1);
9. **continue**;
10. }

o url掉转导致html parse不成功的问题

在抓取百度百科的景点数据时，发现部分页面不会html parse部分的逻辑，而我的plugin是基于HtmlParserFilter扩展点的，因而没有生效。后来发现请求部分页面的链接返回的http状态为301，跳转之后才会到真正页面，而nutch默认是不会抓取跳转后的页面的.这时需要修改nutch-site.xml，加入以下配置即可，nutch-default.xml里的默认值是0，我们这里改成一个大于0的值，nutch就会继续抓取跳转后的页面了。

**Xml代码  [收藏代码](javascript:void())**

1. **<property>**
2. **<name>**http.redirect.max**</name>**
3. **<value>**2**</value>**
4. **<description>**The maximum number of redirects the fetcher will follow when
5. trying to fetch a page. If set to negative or 0, fetcher won't immediately
6. follow redirected URLs, instead it will record them for later fetching.
7. **</description>**
8. **</property>**

o 抽取的过程中发现某些属性老是抽不到，而在不使用nutch抓取的情况下是能抽到的，进而怀疑nutch抓取的页面不全。于是去google了一下"nutch content limit"，发现nutch有这么一个配置项：

**Xml代码  [收藏代码](javascript:void())**

1. **<property>**
2. **<name>**http.content.limit**</name>**
3. **<value>**65536**</value>**
4. **<description>**The length limit for downloaded content using the http://
5. protocol, in bytes. If this value is nonnegative (**>**=0), content longer
6. than it will be truncated; otherwise, no truncation at all. Do not
7. confuse this setting with the file.content.limit setting.
8. **</description>**
9. **</property>**

用来限制抓取内容的大小，放大10倍后（或设置成-1），问题解决。

需要注意的是nutch还有一个很容易混淆的配置项：

**Xml代码  [收藏代码](javascript:void())**

1. **<property>**
2. **<name>**file.content.limit**</name>**
3. **<value>**65536**</value>**
4. **<description>**The length limit for downloaded content using the file://
5. protocol, in bytes. If this value is nonnegative (**>**=0), content longer
6. than it will be truncated; otherwise, no truncation at all. Do not
7. confuse this setting with the http.content.limit setting.
8. **</description>**
9. **</property>**

两个配置用于的协议不同，前者是http协议，后者是file协议。

## 参考文献

[1] [nutch爬行中的depth（层数，深度？）http://javaeyes.iteye.com/blog/36191](http://javaeyes.iteye.com/blog/36191)

[2] Nutch的配置文件<http://wenku.baidu.com/view/329470d176a20029bd642d1d.html>

[3] nutch抓取动态网页<http://www.blogjava.net/kxx129/archive/2010/04/24/319280.html>

[4] [nutch如何才能抓取到动态的url，配置文件解析](http://blog.csdn.net/jiutao_tang/article/details/6525825)

<http://blog.csdn.net/zfrong/article/details/5070683>

[5] nutch杂记<http://lc87624.iteye.com/blog/1625677>

# 后序内容

未来还有一些需要写的，用程序解析nutch爬取的页面、源代码的浅析、插件开发、和增量爬取分布式系统有关的内容，也不知道以后能不能接触到这些，现在只是进行了比较浅显的使用。