





# 流程解析：

## 1 Inject

* Mr1: 把输入的文本文件转换成DB格式
  + Map(InjectMapper，input：urls（在url后面可以自己指定一些参数）)
    - **URLNormalizers** 、**filter** 和**scoreFilter** 三个过滤
    - **setFetchTime, setScore，setFetchInterval，setStatus**
    - **output：**<url, CrawlDatum(*STATUS\_INJECTED*)>
  + reduce(默认)：
    - Output: inject-temp-/<url, CrawlDatum(*STATUS\_INJECTED*)>;
* Mr2:合并到现有的DB
  + Map(CrawlDbFilter，input：第一步的输出和已存在的current/DB文件):过滤url；
    - Output：<url, CrawlDatum(*STATUS\_INJECTED*)>；
  + Reduce（InjectReducer）：根据key（url）合并CrawlDatum；
    - Output: newCrawlDb/<url, CrawlDatum(*STATUS\_DB\_UNFETCHED*)>;
* *CrawlDb.install：原来目录变更为* old，newCrawlDb*文件夹改名成*current，解除目录锁定？
* 删除inject-temp- 目录；

## 2.1 generate

* 锁定db目录；
* Mr1:
  + Map(Selector,input:current/<url, CrawlDatum>)
* 过滤url：*URLFilter*，*check fetch schedule（allows it to be included in the further selection process）*，meta数据中的*GENERATE\_TIME*是否超时
* 对url进行打分
* 检查：限制状态、分值、抓取间隔
* *MetaData中*写入生成时间*(GENERATE\_TIME\_KEY)*
* Output: <score, SelectorEntry>
  + Selector：Hashcode（Byhost）分区
  + Reduce（Selector，input<score, <SelectorEntry>>）
* 检查topN,URLNormalizers （urlString）
* 检查：byhost主机条目
* 写入segnum
  + - Output: generate-temp-/ fetchlist-\*/<score, SelectorEntry>
* 检查有无生成抓取子目录fetchlist-\*
* Mr2: SelectorInverse
  + Map(SelectorInverseMapper，input：generate-temp-/ fetchlist-\*/<score, SelectorEntry>)
    - Output: <url, SelectorEntry>
  + URLPartitioner：Hashcode（Byhost）分区
  + Reduce（PartitionReducer< url, SelectorEntry>）
    - Output：crawl\_generate*/*< url(不同的url可能hashcode一样), CrawlDatum >
* Mr3: 更新db
  + Map（CrawlDbUpdater， input：crawl\_generate*/*<url, CrawlDatum>，current）：
    - Output: <url,CrawlDatum>
  + Reduce（CrawlDbUpdater< url, <CrawlDatum>>）
    - Meta数据更新*GENERATE\_TIME*
    - Output：generate-temp-2/< url, CrawlDatum >
* *CrawlDb.install：原来目录变更为* old，generate-temp-2*文件夹改名成*current，解锁db目录；
* 删除generate-temp-2 目录，解锁db目录；删除generate-temp- 目录；

## 2.2 Fetch

* 检查http.robots.agents，设置限制时间点用于失败任务的重跑；
* Mr
* Map（自定义MapRunnerClass，处理单条记录，map类默认）：Input:*GENERATE\_DIR\_NAME*/<url, CrawlDatum>
  + Feed线程：读URL记录到队列中；
  + FetcherThread：优先检查meta中的reprUrl，robot协议检查、delay处理，按协议获取输出，  
    判断返回状态：*SUCCESS\_REDIRECT（加入队列，初始化分值），TEMP\_MOVED（加入队列，初始化分值）*handleRedirect：urlFilters、scfilters过滤，是否忽略外链（ignoreExternalLinks）  
    **Output**: 把url的score保存下来,构造<url, NutchWritable>
  + fetch主线程若检测到超时则退出fetch。
* Reduce（reduce类默认，< url, <NutchWritable >>）
  + **Output**：segment/crawl\_fetch*/*<url, CrawlDatum>*,* segment/content/<url, Content>

## 2.3 parse

* Mr1
* Map（）：Input:*GENERATE\_DIR\_NAME*/Content/<url, Content >
  + 检查成功抓取状态，截断；
  + 解析，给页面计算指纹，计算分值写入meta。
  + **Output**:<url, ParseImpl>（ParseData，里面是放除了正文以外的其他元数据，从页面上解析出的link也是放到其中）
* Reduce（< url, < ParseImpl >>）
  + **Output**：segment/ parse\_text*/*<url, ParseText>*,* segment/ parse\_data/<url, ParseData>, segment/ crawl\_parse/<url, CrawlDatum >

## 2.4 crawlDbTool.update：

* lock目录，
* Mr：（）
  + Map（CrawlDbFilter）：Input: segments[i]/crawl\_fetch/<url, CrawlDatum >， segments[i]/crawl\_parse/<url, CrawlDatum >，crawl/current
  + Reduce（CrawlDbReducer）：合并记录
    - Output：newCrawlDb
* *CrawlDb.install：原来目录变更为* old，newCrawlDb*文件夹改名成*current，解除目录锁定；

## 3 linkDb

* 过滤目录？锁定linkdb目录；
* Mr：
  + Map（LinkDb）：输入：segment/ parse\_data/<url, ParseData>;过滤url；
    - Output:< toUrl, inlinks< fromUrl, anchor>>
  + Combiner（LinkDbMerger）：合并
  + Reduce（LinkDbMerger）：合并
    - Output：linkdb- /< toUrl, inlinks< fromUrl, anchor>>
* Mr2合并原有的linkdb目录；
  + Map（LinkDbFilter）：输入：linkdb-，current；过滤url
    - Output:< url, inlinks< fromUrl, anchor>>
  + Reduce（LinkDbMerger）：合并
    - Output：linkdb-merge-/< toUrl, inlinks< fromUrl, anchor>>
* 删除 linkdb- 目录；
* LinkDb.install：*原来目录变更为* old，linkdb-merge-*文件夹改名成*current
* 解除锁定；

# 改造工作：

## Host提供黑白名单过滤功能；

* **数据结构：**
  1. hostdb<<host, channel>，hostdata<seed,topicnum,total, rate, metadata<key,value>>>
* **数据存储：**
  1. Hdfs文件：仿crawldb，有保障。
* **主要流程：**
  1. 读入黑白名单数据：
     1. 读hdfs上的文件，包装成distributedCache，分发到node节点;
     2. Node本地读取hostdb文件，载入内存（Map），用于主题判断；
  2. 数据使用：
     1. 返回url是否是主题相关的结果
     2. 返回url的主题类型（白、黑、待识别）；
  3. 数据更新
     1. 在一次nutch流程crawlDbTool.update后，开始hostdb的更新，使用mapreduce统计数据（**crawl/current**），计算每个url的主题值超过某个设定的临界值的host的个数。

## url提供主题分值功能；

* **使用场景：**
  1. 取代nutch中的打分插件，实现按主题相关度打分。
* **数据结构：**
  1. 直接使用crawldb中的score字段；
* **数据存储：crawldb。**
* **主要流程：**
  1. 数据使用：
     1. 添加处理主题的ScoringFilter，实现各个接口方法，nutch流程会在适当的地方调用；
  2. 数据更新
     1. 原有的crawldb的更新流程不变。

## 总体流程：



## Hostdb数据量估算：

* **依据：**

中国互联网络信息中心：截至2013年12月，中国网站数量为320万；

通过对 Alexa 报告中 250 家顶级网站的域名分析发现，其在 70% 的网站域名长度小于或等于 8，平均长度刚刚过 7。  
<http://thenextweb.com/socialmedia/2014/04/06/ideal-length-everything-internet/>

* **单条hostdb字节数（hostdb<<host,** channel**>，hostdata<seed,topicnum,total, rate>>）**

Host channel seed topicnum totoal rate /n 总计（byte）

4+8+4 +8 +1 +8 +8 +4 +1 46

* **数据总量：**

46\*3200000= 147,200,000 byte = 140.1 M

# 附件：hadoop mapreduce缓存技术

MapReduce编程中相对有效的设置全局共享数据的方法。

## 读写HDFS文件

磁盘、网络io；

## 配置Job属性

Jobconfig设置属性，数据量非常小。

## 使用DistributedCache

只会为每个作业每个节点复制一次，缺点是它是只读的。

# 附件：目录结构

 crawldb：存放需要抓取的的超链接地址；

{current, old}

 segments：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、crawl\_parse：用来更新crawl的外部链接库

4、content：每个下载页面的内容

5、parse\_text：包含每个解析过的url文本内容

6、parse\_data：每个url解析出的外部链接和元数据

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。

# 附件：数据结构：

Crawl DB

* CrawlDb 是一个包含如下结构数据的文件: <URL, CrawlDatum>
* CrawlDatum:<status, fetchTime, retries，fetchInterval, score, modifiedTime, signature，metaData<…>>
* Status: {db\_unfetched, db\_fetched, db\_gone,linked,fetch\_success, fetch\_retry, fetch\_gone，…}



