crawler4j源码简单分析

唐皆哲

crawler4j是一个开源的、轻量级的、单机多线程的开源爬虫框架，其在github上的仓库地址为<https://github.com/yasserg/crawler4j>。为方便将其改造为分布式爬虫，本文简单分析了其源码。

对于其最基本的使用，用户需要实现一个WebCrawler的子类，主要需要重写visit和shouldVisit两个方法，visit方法里写用户希望对爬下的网页进行的操作，shouldVisit方法用于判断是否应当爬取一个网页。然后创建一个CrawlerController的对象，进行配置，并加载自己的WebCrawler子类，最后启动即可。详细的使用方法可参考作者github上的介绍。

crawler4j主要有以下几个包：

crawler

frontier

fetcher

parser

robotstxt

url

util

对于改造其为分布式爬虫，关键是要修改crawler包和frontier包中的代码，其他代码基本可保持不变，故文章主要分析这两个包。

**crawler包**

crawler包中有一下几个类：

1. Configurable
2. CrawlConfig
3. Page
4. WebCrawler
5. CrawlController

1）Configurable类很简单，它中包含一个CrawlConfig对象，无其他功能。

2）CrawlConfig主要用于设置这个爬虫的各种参数。比较重要的参数有resumableCrawling，它决定爬虫是否是可恢复的，若设置为true，则爬虫会用数据库记录当前的工作进度，当关闭后再次启动，爬虫会从上次关闭时的进度继续开始。crawlStorageFolder用于决定爬虫关于工作进度的数据库文件存放在哪里。此外，这个类中还有一个validate方法用于检查一些必要的参数是否被正确地设置了。

2）Page类用于保存一个网页内容，方便之后的处理，里面的具体属性和方法较多请参考源码。

3）WebCrawler类实现了Runnable接口，是单个线程的爬虫工作类，在爬虫工作时会启动多个WebCrawler对象同时工作。

4）WebCrawler里的关键的域有：myId，它是这个爬虫线程的id；myController，它是控制这个爬虫的CrawlController对象的引用；myThread，它是装载这个爬虫的线程的引用；isWaitingForURLs，它为true时表示该爬虫在等待Controller分配任务。

作为实现了Runnable的类，它最重要的方法是run方法。这个方法里是一个while循环，循环里首先尝试从frontier(任务队列)中获取50个URL(当获取URL时会将isWaitingForURLs设置为false，获得后再置为true)，如果没获得，则判断frontier是否已被关闭，若关闭则结束，否则等待3s。如果获得了URL，则调用processPage方法对每个URL进行处理，然后调用frontier的setProcessed方法将这些URL标记为已经处理了。如此反复循环。

再来看processPage方法，它首先调用pageFetcher的fetchPage方法，它会返回一个FetchResult对象，它包含一个状态码，若返回的状态码表示重定向，则将需要重定向后的URL加入到frontier中结束，若是其他异常状态码，则调用onUnexpectedStatusCode方法处理。若是能正确访问的状态码，则调用Parser对象的parse方法处理这个页面，并将结果存于Page对象中。再取出这个页面中的指向其他地方的URL，根据shouldVisit方法判断其是否应当访问，若是，则加入到frontier中。然后调用visit方法处理这个页面。

5） CrawlerController用于管理多个爬虫线程，用户也是控制它来启动和关闭整个爬虫。它里面有个monitorThread，它会时刻检查是否有线程挂掉以及是否任务已经完成。它还有个addSeed方法用于添加爬虫初始的URL地址。

**frontier包**

这个包的功能主要是用于管理爬虫的任务队列，为了使爬虫在关闭后能记住任务进度，这个包里使用了Berkeley DB将任务进度的相关信息保存到数据库中。Berkeley DB是一个Key-Value型的简易数据库，具体使用请参考网上的资料。这个包里有一下几个类：

1. Counters；
2. DocIDServer；
3. WorkQueues；
4. InProcessPagesDB；
5. Frontier；
6. WebURLTupleBinding；

1）Counters用来记录已经被安排的URL(Scheduled\_Pages)数量和已经处理过的URL(Processed\_Pages)数量，若爬虫是resumable的，则保存到数据库中，若不是，则保存到一个Map中。

2）DocIDServer管理一个叫做”DocIDs“的数据库，这个数据库存放URL->id的键值对。每一个新的URL会赋给它一个新的id，id是从1开始递增的。爬虫也是用这个数据库判读一个URL是否已经出现过了。

3）WorkQueues用于管理一个数据库，里面存着的是带有优先级的URL队列。具体可看源码中的注释：The key that is used for storing URLs determines the order they are crawled. Lower key values results in earlier crawling. Here our keys are 6 bytes. The first byte comes from the URL priority. The second byte comes from depth of crawl at which this URL is first found. The rest of the 4 bytes come from the docid of the URL. As a result, URLs with lower priority numbers will be crawled earlier. If priority numbers are the same, those found at lower depths will be crawled earlier. If depth is also equal, those found earlier (therefore, smaller docid) will be crawled earlier.

4）InProcessPagesDB，它是WorkQueues的子类，多了一个remove方法，其他类似。

5）Frontier是整个的大管家，也是对面的接口。它包含了Counters，DocIDServer，WorkQueues，InProcessPagesDB的对象。其工作的主要思路就是一个新的URL会被放到WorkQueues中，当它被分配给工作爬虫后就从WorkQueues中移动到InProcessPagesDB，当这个URL被处理完后就会从InProcessPagesDB中被移除。还有一个要注意的是，当爬虫从一个空的WorkQueues中获取URL时会进入阻塞，当WorkQueues加入新的URL后才会解除阻塞。