

北京邮电大学

本科毕业设计(论文)



题目：面向微博网页的爬虫设计与实现

姓 名 田 园

学 院 计算机学院

专 业 计算机科学与技术

班 级 2008211303

学 号 08211227

班内序号 29

指导教师 闫丹凤

2012 年 6 月

面向微博网页的爬虫设计与实现

摘 要

微博是现代信息社会的新生儿，是当今最热门的信息发布平台。它广泛分布在桌面、浏览器、移动终端等多个平台上，人们可以随时随地使用微博分享财经世界的大情小事、热门话题、财经动态，更有股市专家为实时解盘。可以随时随地与百万网友分享理财心得，与财经名人零距离接触，与股市达人探讨学习，与亲朋好友保持联络。为我们所需的金融信息提供了很大的信息源。而我们基于微博的金融咨询检索平台项目正式看中了微博的种种有利特点而开发，基于微博的金融检索平台是金融资讯电子杂志的上游项目。实现金融名人及机构的微博数据的抓取、分析、索引及用于挑选杂志内容的检索、展现、筛选及杂志内容入库及管理。我负责的便是项目中抓取子系统模块里的网络爬虫部分。

面向微博网页的爬虫，只对微博中各关注用户的信息进行爬取。故该爬虫采用定向爬虫模式，具有高度的垂直性，和主题相关性。只关注最有价值的信息。充分提高了用户检索信息的效率。采用深度搜索的网页抓取策略，更精确的获得相关信息。实时更新功能更能满足微博用户活跃度较高的特征，同时也保证了用户在检索信息时总是获取最新的信息。同时采用多线程并行抓取策略，大幅度提高爬虫的抓取速度和效率。本文包括了对该爬虫的设计与实现全面详细的讲述，并在最后给出了将其测试结果及通用化设置方法等内容。

关键词 定向爬虫 蜘蛛 微博 网络

The Design and Implementation of The Crawler to Micro-blog Web

ABSTRACT

The Micro-blog is the newborns of our modern information society, and it is the most popular information platform. It is widely distributed in desktop , browser, mobile terminal, and other platform, people can use a share of finance and economics at big feeling small, hot topic, finance and economics dynamic, more information of stock market at any time and any place of the world, experts for real-time solution plate and so on. And it also can share financial assistance with each other, and do financial celebrity zero distance contact, decide the stock market study, keep in touch with your friends and family at any time anywhere. It gives us very much useful financial information with great information source. Our Micro-blog based financial advisory retrieval platform is built for the reasons above. It is the upstream of the projects in the financial information electronic magazine. It have the functions like data grab, analysis, and index. I am responsible for the module which is used to grab data from web parts in the project, called the part of crawler.

The crawler to Micro-blog web is only focused on the user's information which we take up. So the crawler using the mode of directional crawler, and it has a high degree of vertical sex, and theme correlation. It fully improves the efficiency of the user when access to information. It uses the depth search page strategy, which is more accurate in the access to relevant information. Real-time update function could match the micro-blog users who are more active, also ensure the users in the information retrieval could always access the latest information. At the same time the threads parallel grab strategy, also greatly improved the crawler to grab with the speed and efficiency. This paper includes the details about design and implementation of crawler to micro-blog web, and in its last, it gives the test results.

KEY WORDS directional crawler spider micro-blog network

目 录

第一章 绪论.....	1
1.1 课题研究背景	1
1.2 工作内容	1
1.3 论文结构	2
第二章 背景知识.....	3
2.1 网络爬虫.....	3
2.1.1 网络爬虫概念.....	3
2.1.2 网络爬虫的常用策略.....	5
2.1.3 网络爬虫体系结构.....	7
2.1.4 部分开源爬虫介绍.....	9
2.2 Java 多线程编程技术.....	11
2.2.1 创建多线程的方法.....	11
2.2.2 Java 语言对线程同步的支持	11
2.2.3 Java 语言对线程阻塞的支持	12
2.3 Html 解析.....	12
2.3.1 Html 语言简介	12
2.3.2 正则表达式.....	13
第三章 面向微博网页爬虫的设计.....	15
3.1 需求分析	15
3.1.1 项目中爬虫的需求分析.....	15
3.1.2 与通用爬虫的区别.....	15
3.2 面向微博网页爬虫概述	16
3.2.1 爬虫功能介绍.....	16
3.2.2 爬虫设计策略.....	18
3.2.3 爬虫总体结构.....	20
3.3 系统各模块介绍	21
3.3.1 从服务器获取源码模块.....	21
3.3.2 解析获取的源码模块.....	22
3.3.3 源码本地存储模块.....	23
3.3.4 更新模块.....	26
3.3 本章小结.....	27
第四章 面向微博网页爬虫的实现.....	28
4.1 爬虫工作总体流程	28
4.2 爬虫的具体实现过程	29
4.2.1 获取源码部分.....	29
4.2.2 解析源码部分.....	31
4.2.3 存储源码部分.....	33
4.2.4 更新部分.....	35
4.3 爬虫的适应性参数设置	36

4.3.1 URL 种子库的设置	36
4.3.2 本地存储文档的设置	38
4.3.3 Html 源文件解析的设置	39
4.3.4 线程数量的设置	39
4.4 本章小结	40
第五章 面向微博网页爬虫的测试	41
5.1 测试环境介绍	41
5.1.1 硬件配置	41
5.1.2 软件环境	41
5.2 爬虫功能测试	41
5.2.1 爬取初始数据测试	42
5.2.2 增量更新数据测试	44
5.3 并行爬虫效率测试	47
5.4 本章小结	48
第六章 结束语	49
6.1 工作总结	49
6.2 仍需改进的地方	50
参考文献	51
致谢	52

第一章 绪论

1.1 课题研究背景

微博被定义为是一种通过关注机制分享简短实时信息的广播式的社交网络平台。微博客草根性更强，且广泛分布在桌面、浏览器、移动终端等多个平台上，有多种商业模式并存，或形成多个垂直细分领域的可能，但无论哪种商业模式，都离不开用户体验的特性和基本功能。人们更可以随时随地使用微博分享财经世界的大情小事、热门话题、财经动态，更有股市专家为实时解盘。可以随时随地与百万网友分享理财心得，与财经名人零距离接触，与股市达人探讨学习，与亲朋好友保持联络。为我们所需的金融信息提供了很大的信息源^[1]

而我们基于微博的金融咨询检索平台项目正是看中了微博的种种有利特点而开发，基于微博的金融检索平台是金融资讯电子杂志的上游项目。实现金融名人及机构的微博数据的抓取、分析、索引及用于挑选杂志内容的检索、展现、筛选及杂志内容入库及管理。基于微博的金融咨询检索平台分为五个子系统：抓取子系统、分析子系统、索引子系统、检索子系统和用户交互子系统。数据存储包括：原始的 html 页面数据、文本数据（抽取后的内容）、金融词典（用于中文切词）、过滤词典（用于黄反词过滤）、微博状态文件（微博消息标记删除）、索引数据、索引数据、名人机构库（包含名称及 URL）及入库数据（用作电子杂志内容的数据）。抓取子系统中又包括了两种方式，网络爬虫：对初始 URL 库过滤及选取，抓取指定 URL 的网页；使用微博 API：使用新浪、网易、腾讯及搜狐微博提供的微博 API，直接获取指定名人、机构的微博文本数据。由于微博要发展壮大，最终都要开放。通过新浪、搜狐、腾讯及网易微博等大微博网站的深入调研，其微博 API 都是开放的。API 获取数据子系统利用微博网站提供的 API，获取指定名人、机构的微博内容。其方便、快捷，省去了搜索引擎抓取网页、分析网页、噪声去重等步骤。不仅能保证数据获取的实时性，还能降低开发代价。但还有少数未开放 API 的微博，如和讯财经微博。故需要自己编写网络爬虫。即抓取子系统中网络爬虫的方式。本文则以和讯财经微博为例，介绍面向微博网页爬虫的设计与实现。下面将具体介绍我的研究内容。

1.2 工作内容

正如背景介绍中所说，由于大多数微博网站，如新浪微博，腾讯微博等都有其提供开放的微博 API，可直接获取指定名人、机构的微博文本数据。其方便、快捷，省去了搜索引擎抓取网页、分析网页、噪声去重等步骤。不仅能保证数据

获取的实时性，还能降低开发代价。但还有少数未开放 API 的微博，如和讯财经微博。故需要自己编写网络爬虫，对初始 URL 库过滤及选取，抓取指定 URL 对应的 html 网页。要对微博内容自动采集，抓取内容可随着被采集网页内容的更新而更新。在抓取的过程中，可实现对非相关数据能进行初步的过滤，并对数据进行本地存储。

根据以上内容本课题的主要任务及目标有：

- ✓ 对指定的 URL 库进行过滤及筛选；
- ✓ 在网络中抓取相关 URL 对应的 html 页面；
- ✓ 采用增量更新的方式，可以根据网页属性判断网页内容是否更新并进行更新；
- ✓ 原始 Web 页面的存储。

基于上述研究目标，本文作者在本科毕业设计工作期间开展了如下几个方面的工作：

- ✓ 学习并熟练应用 Java 语言及其有关技术，如多线程技术；
- ✓ 学习并了解爬虫的基本概念并清楚其工作原理；
- ✓ 名人机构微博 URL(初始 URL 整理)；
- ✓ 爬取规则的编写(数据结构、java 多线程) java 基本类，封装的 jar 包, 增量更新的实现；
- ✓ 原始 Web 页面的存储。

功能完成后要与同项目相关部分进行联调，例如文本抽取模块。

1.3 论文结构

本论文按以下章节组织：

第一章，绪论。本章简要介绍了项目的相关背景，工作内容以及论文结构。

第二章，背景知识。本章较为详细的介绍了网络爬虫，Java 技术，Html 文件解析等背景知识为后续的设计及实现过程做铺垫。

第三章，面向微博网页爬虫的设计。本章主要介绍了面向微博网页爬虫的设计思路。具体包括系统概述和分模块介绍。

第四章，面向微博网页爬虫的实现。本章详细介绍了爬虫工作的总体流程及具体实现过程，并展示了各部分的数据结构，各函数间联系等。

第五章，面向微博网页爬虫的测试。本章首先将目前爬虫工作的结果进行了展示，之后介绍了对其进行功能测试及效率测试部分的结果。

第六章，面向微博爬虫的通用化。首先介绍了通用化的目的及作用，之后进行了具体通用化设置各部分的详细介绍。

第七章，结束语。包括了工作总结，及仍需改进的地方的阐述。

本文的最后是参考文献和致谢。

第二章 背景知识

2.1 网络爬虫

2.1.1 网络爬虫概念

网络爬虫（又被称为网页蜘蛛，网络机器人，在 FOAF 社区中间，更经常的称为网页追逐者），是一种按照一定的规则，自动的抓取万维网信息的程序或者脚本。另外一些不常使用的名字还有蚂蚁，自动索引，模拟程序或者蠕虫[2]。随着网络的迅速发展，万维网成为大量信息的载体，如何有效地提取并利用这些信息成为一个巨大的挑战。搜索引擎(Search Engine)，例如传统的通用搜索引擎 AltaVista, Yahoo! 和 Google 等，作为一个辅助人们检索信息的工具成为用户访问万维网的入口和指南。但是，这些通用性搜索引擎也存在着一定的局限性，如：（1）不同领域、不同背景的用户往往具有不同的检索目的和需求，通用搜索引擎所返回的结果包含大量用户不关心的网页。（2）通用搜索引擎的目标是尽可能大的网络覆盖率，有限的搜索引擎服务器资源与无限的网络数据资源之间的矛盾将进一步加深。（3）万维网数据形式的丰富和网络技术的不断发展，图片、数据库、音频/视频多媒体等不同数据大量出现，通用搜索引擎往往对这些信息含量密集且具有一定结构的数据无能为力，不能很好地发现和获取。（4）通用搜索引擎大多提供基于关键字的检索，难以支持根据语义信息提出的查询。为了解决上述问题，定向抓取相关网页资源的聚焦爬虫应运而生。

聚焦爬虫是一个自动下载网页的程序，它根据既定的抓取目标，有选择的访问万维网上的网页与相关的链接，获取所需要的信息。与通用爬虫不同，聚焦爬虫并不追求大的覆盖，而将目标定为抓取与某一特定主题内容相关的网页，为面向主题的用户查询准备数据资源。聚焦爬虫工作原理以及关键技术概述：网络爬虫是一个自动提取网页的程序，它为搜索引擎从万维网上下载网页，是搜索引擎的重要组成。传统爬虫从一个或若干初始网页的 URL 开始，获得初始网页上的 URL，在抓取网页的过程中，不断从当前页面上抽取新的 URL 放入队列，直到满足系统的一定停止条件。下图 2-1 为聚焦爬虫在整个检索过程中的角色位置的图示。

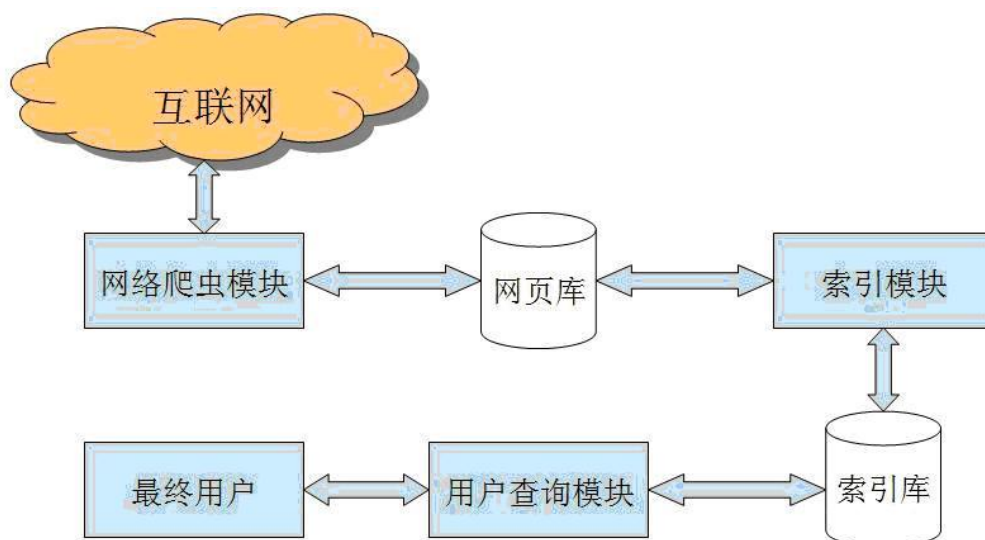


图 2-1 聚焦爬虫的角色位置图

聚焦爬虫的工作流程较为复杂，需要根据一定的网页分析算法过滤与主题无关的链接，保留有用的链接并将其放入等待抓取的 URL 队列。然后，它将根据一定的搜索策略从队列中选择下一步要抓取的网页 URL，并重复上述过程，直到达到系统的某一条件时停止。另外，所有被爬虫抓取的网页将会被系统存贮，进行一定的分析、过滤，并建立索引，以便之后的查询和检索；对于聚焦爬虫来说，这一过程所得到的分析结果还可能对以后的抓取过程给出反馈和指导。

相对于通用网络爬虫，聚焦爬虫还需要解决三个主要问题：

- (1) 对抓取目标的描述或定义；
- (2) 对网页或数据的分析与过滤；
- (3) 对 URL 的搜索策略。

抓取目标的描述和定义是决定网页分析算法与 URL 搜索策略如何制订的基础。而网页分析算法和候选 URL 排序算法是决定搜索引擎所提供的服务形式和爬虫网页抓取行为的关键所在。这两个部分的算法又是紧密相关的。

现有聚焦爬虫对抓取目标的描述可分为基于目标网页特征、基于目标数据模式和基于领域概念 3 种。基于目标网页特征的爬虫所抓取、存储并索引的对象一般为网站或网页。根据种子样本获取方式可分为：

- (1) 预先给定的初始抓取种子样本；
- (2) 预先给定的网页分类目录和与分类目录对应的种子样本，如 Yahoo! 分类结构等；
- (3) 通过用户行为确定的抓取目标样例，分为：
 - a) 用户浏览过程中显示标注的抓取样本；
 - b) 通过用户日志挖掘得到访问模式及相关样本。

其中，网页特征可以是网页的内容特征，也可以是网页的链接结构特征，等等^[2]。

2.1.2 网络爬虫的常用策略

（1）网页的抓取策略

网页的抓取策略可以分为深度优先、广度优先和最佳优先三种。深度优先在很多情况下会导致爬虫的陷入(trapped)问题，目前常见的是广度优先和最佳优先方法^[2]。

深度优先搜索策略：深度优先搜索属于图算法的一种，英文缩写为 DFS 即 Depth First Search. 其过程简要来说是对每一个可能的分支路径深入到不能再深入为止，而且每个节点只能访问一次。深度优先搜索是一种在开发爬虫早期使用较多的方法。它的目的是要达到被搜索结构的叶结点(即那些不包含任何超链的 HTML 文件)。在一个 HTML 文件中，当一个超链被选择后，被链接的 HTML 文件将执行深度优先搜索，即在搜索其余的超链结果之前必须先完整地搜索单独的一条链。深度优先搜索沿着 HTML 文件上的超链走到不能再深入为止，然后返回到某一个 HTML 文件，再继续选择该 HTML 文件中的其他超链。当不再有其他超链可选择时，说明搜索已经结束。优点是能遍历一个 Web 站点或深层嵌套的文档集合；缺点是因为 Web 结构相当深，有可能造成一旦进去，再也出不来的情况发生^[3]。

广度优先搜索策略：广度优先搜索策略是指在抓取过程中，在完成当前层次的搜索后，才进行下一层次的搜索。该算法的设计和实现相对简单。在目前为覆盖尽可能多的网页，一般使用广度优先搜索方法。也有很多研究将广度优先搜索策略应用于聚焦爬虫中。其基本思想是认为与初始 URL 在一定链接距离内的网页具有主题相关性的概率很大。另外一种方法是广度优先搜索与网页过滤技术结合使用，先用广度优先策略抓取网页，再将其无关的网页过滤掉。这些方法的缺点在于，随着抓取网页的增多，大量的无关网页将被下载并过滤，算法的效率将变低。

最佳优先搜索策略：策略按照一定的网页分析算法，预测候选 URL 与目标网页的相似度，或与主题的相关性，并选取评价最好的一个或几个 URL 进行抓取。它只访问经过网页分析算法预测为“有用”的网页。存在的一个问题是，在爬虫抓取路径上的很多相关网页可能被忽略，因为最佳优先策略是一种局部最优搜索算法。因此需要将最佳优先结合具体的应用进行改进，以跳出局部最优点。将在第 4 节中结合网页分析算法作具体的讨论。研究表明，这样的闭环调整可以将无关网页数量降低 30%~90%^[2]。

（2）重新访问策略

网络具有动态性很强的特性。抓取网络上的一小部分内容可能会花费

真的很长的时间，通常用周或者月来衡量。当爬虫完成它的抓取的任务以后，很多操作是可能会发生的，这些操作包括新建，更新和删除。从搜索引擎的角度来看，不检测这些事件是有成本的，成本就是我们仅仅拥有一份过时的资源。最常使用的成本函数，是新鲜度和过时性（2000 年，Cho 和 Garcia-Molina）^[4]

新鲜度：这是一个衡量抓取内容是不是准确的二元值。在时间 t 内，仓库中页面 p 的新鲜度定义如下（式 2-1）：

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases} \quad \text{式 (2-1)}$$

过时性：这是一个衡量本地已抓取的内容过时期度的指标。在时间 t 时，仓库中页面 p 的时效性的定义如下（式 2-2）：

$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ is not modified at time } t \\ t - \text{modification time of } p & \text{otherwise} \end{cases} \quad \text{式 (2-2)}$$

爬虫的目标是尽可能高的提高页面的新鲜度，同时降低页面的过时性。这一目标并不是完全一样的，第一种情况，爬虫关心的是有多少页面时过时的；在第二种情况，爬虫关心的页面过时了多少。

两种最简单的重新访问策略是由 Cho 和 Garcia-Molina 研究的(Cho 和 Garcia-Molina, 2003)^[4]：统一策略：使用相同的频率，重新访问收藏中的所有的链接，而不考虑他们更新频率。正比策略：对变化越多的网页，重新访问的频率也越高。网页访问的频率和网页变化的频率直接相关。（两种情况下，爬虫的重新抓取都可以采用随机方式，或者固定的顺序）^[2]

（3）平衡礼貌策略

爬虫相比于人，可以有更快的检索速度和更深的层次，所以，他们可能使一个站点瘫痪。不需要说一个单独的爬虫一秒钟要执行多条请求，下载大的文件。一个服务器也会很难响应多线程爬虫的请求。

就像 Koster (Koster, 1995) 所注意的那样，爬虫的使用对很多工作都是很有用的，但是对一般的社区，也需要付出代价。使用爬虫的代价包括：网络资源：在很长一段时间，爬虫使用相当的带宽高度并行地工作。服务器超载：尤其是对给定服务器的访问过高时。质量糟糕的爬虫，可能是服务器或者路由器瘫痪，或者会尝试下载自己无法处理的页面。个人爬虫，如果过多的人使用，可能是网络或者服务器阻塞。

对这些问题的一部分解决方法是漫游器排除协议 (Robots exclusion protocol)，也被称为 robots.txt 议定书 (Koster, 1996)^[6]，这份协议对

于管理员指明网络服务器的那一部分不能到达是一个标准。这个标准没有包括重新访问一台服务器的间隔的建议，虽然访问间隔是避免服务器超载的最有效的办法。最近的商业搜索软件，如 Ask Jeeves, MSN 和 Yahoo 可以在 robots.txt 中使用一个额外的 “Crawl-delay” 参数来指明请求之间的延迟。对连接间隔时间的第一个建议由 Koster 1993 年给出，时间是 60 秒。按照这个速度，如果一个站点有超过 10 万的页面，即使我们拥有零延迟和无穷带宽的完美连接，它也会需要两个月的时间来下载整个站点，并且，这个服务器中的资源，只有一小部分可以使用。这似乎是不可以接受的。Cho (Cho 和 Garcia-Molina, 2003) 使用 10 秒作为访问的间隔时间，WIRE 爬虫 (Baeza-Yates and Castillo, 2002) 使用 15 秒作为默认间隔。MercatorWeb (Heydon 和 Najork, 1999) 爬虫使用了一种自适应的平衡策略：如果从某一服务器下载一个文档需要 t 秒钟，爬虫就等待 $10t$ 秒的时间，然后开始下一个页面。Dill 等人 (Dill et al., 2002) 使用 1 秒。对于那些使用爬虫用于研究目的的，一个更详细的成本-效益分析是必要的，当决定去哪一个站点抓取，使用多快的速度抓取的时候，伦理的因素也需要考虑进来。

访问记录显示已知爬虫的访问间隔从 20 秒钟到 3-4 分钟不等。需要注意的是即使很礼貌，采取了所有的安全措施来避免服务器超载，还是会引来一些网络服务器管理员的抱怨的。Brin 和 Page 注意到：运行一个针对超过 50 万服务器的爬虫，会产生很多的邮件和电话。这是因为有无数的人在网上，而这些人不知道爬虫是什么，因为这是他们第一次见到。(Brin 和 Page, 1998) ^[2]

(4) 并行策略

一个并行爬虫是并行运行多个进程的爬虫。它的目标是最大化下载的速度，同时尽量减少并行的开销和下载重复的页面。为了避免下载一个页面两次，爬虫系统需要策略来处理爬虫运行时新发现的 URL，因为同一个 URL 地址，可能被不同的爬虫进程抓到 ^[2]。

2.1.3 网络爬虫体系结构

一个爬虫不能像上面所说的，仅仅只有一个好的抓取策略，还需要有一个高度优化的结构。Shkapenyuk 和 Suel (Shkapenyuk 和 Suel, 2002) 指出：设计一个短时间内，一秒下载几个页面的颇慢的爬虫是一件很容易的事情，而要设计一个使用几周可以下载百万级页面的高性能的爬虫，将会在系统设计，I/O 和网络效率，健壮性和易用性方面遇到众多挑战 ^[2]。

在网络爬虫的系统框架中，主过程由控制器，解析器，资源库三部分组成。控制器的主要工作是负责给多线程中的各个爬虫线程分配工作任务。解析器的主要工作是下载网页，进行页面的处理，主要是将一些 JS 脚本标签、CSS 代码内容、空格字符、HTML 标签等内容处理掉，爬虫的基本工作是由解析器完成。资源库是用来存放下载到的网页资源，一般都采用大型的数据库存储，如 Oracle 数据库，并对其建立索引。控制器：控制器是网络爬虫的中央控制器，它主要是负责根据系统传过来的 URL 链接，分配一线程，然后启动线程调用爬虫爬取网页的过程。解析器：解析器是负责网络爬虫的主要部分，其负责的工作主要有：下载网页的功能，对网页的文本进行处理，如过滤功能，抽取特殊 HTML 标签的功能，分析数据功能。资源库：主要是用来存储网页中下载下来的数据记录的容器，并提供生成索引的目标源。中大型的数据库产品有：Oracle、Sql Server 等。

一个设计良好的爬虫架构必须满足如下需求。

- （1）分布式：爬虫应该能够在多台机器上分布执行。
- （2）可伸缩性：爬虫结构应该能够通过增加额外的机器和带宽来提高抓取速度。
- （3）性能和有效性：爬虫系统必须有效地使用各种系统资源，例如，处理器、存储空间和网络带宽。
- （4）质量：鉴于互联网的发展速度，大部分网页都不可能即使出现在用户查询中，所以爬虫应该首先抓取有用的网页。
- （5）新鲜性：在许多应用中，爬虫应该持续运行而不是只遍历一次。
- （6）更新：因为网页会经常更新，例如论坛网站会经常有回帖。爬虫应该取得已经获取的页面的新的拷贝。例如一个搜索引擎爬虫要能够保证全文索引中包含每个索引页面的比较新的状态。对于搜索引擎爬虫这样连续的抓取，爬虫访问一个页面的频率应该和这个网页的更新频率一致。
- （7）可扩展性：为了能够支持新的抓取协议，爬虫架构应该设计成模块化形式^[7]。

实际的爬虫逻辑架构如下图 2-2 所示：

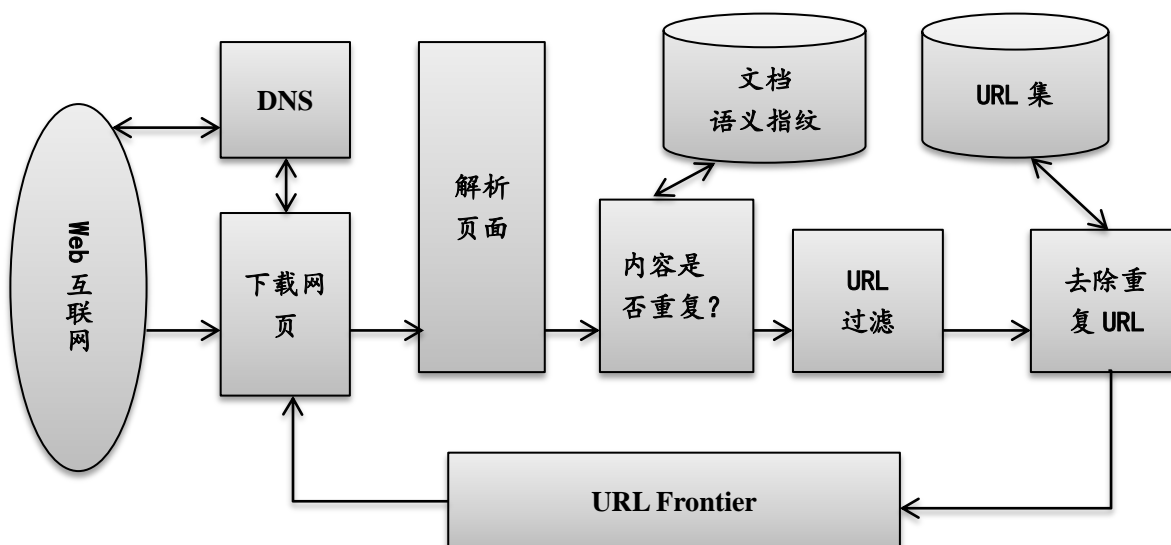


图 2-2 单线程爬虫结构

其中：

1. URL Frontier 包含爬虫当前带抓取的 URL（对于持续更新抓取的爬虫，以前已经抓取的 URL 可能会回到 Frontier 重抓）。
2. DNS 解析模块根据给定的 URL 决定从哪个 Web 服务器获取网页。
3. 获取模块使用 HTTP 协议获取 URL 代表的页面^[7]。

爬虫身份识别：网络爬虫通过使用 http 请求的用户代理字段来向网络服务器表明他们的身份。网络管理员则通过检查网络服务器的日志，使用用户代理字段来辨认哪一个爬虫曾经访问过以及它访问的频率。用户代理字段可能会包含一个可以让管理员获取爬虫更多信息的 URL。邮件抓取器和其他怀有恶意的网络爬虫通常不会留任何的用户代理字段内容，或者他们也会将他们的身份伪装成浏览器或者其他的知名爬虫。对于网路爬虫，留下用户标志信息是十分重要的；这样，网络管理员在需要的时候就可以联系爬虫的主人。有时，爬虫可能会陷入爬虫陷阱或者是一个服务器超负荷，这时，爬虫主人需要使爬虫停止。对那些有兴趣了解特定爬虫访问时间网络管理员来讲，用户标识信息是十分重要的^[2]。

4. 解析模块提取文本和网页的链接集合。
5. 重复消除模块决定一个解析出来的链接是否已经在 URL Frontier 或者最近下载过^[7]。

2.1.4 部分开源爬虫介绍

DataparkSearch 是一个在 GNU GPL 许可下发布的爬虫搜索引擎。

GNU Wget 是一个在 GPL 许可下，使用 C 语言编写的命令行式的爬虫。

它主要用于网络服务器和 FTP 服务器的镜像。

Heritrix 是一个互联网档案馆级的爬虫，设计的目标为对大型网络的大部分内容的定期存档快照，是使用 java 编写的。

HTTrack 用网络爬虫创建网络站点镜像，以便离线观看。它使用 C 语言编写，在 GPL 许可下发行。

ICDL Crawler 是一个用 C++编写，跨平台的网络爬虫。它仅仅使用空闲的 CPU 资源，在 ICDL 标准上抓取整个站点。

JSpider 是一个在 GPL 许可下发行的，高度可配置的，可定制的网络爬虫引擎。

LLarbin 由 Sebastien Ailleret 开发；

Webtools4larbin 由 Andreas Beder 开发；

Methabot 是一个使用 C 语言编写的高速优化的，使用命令行方式运行的，在 2-clause BSD 许可下发布的网页检索器。它的主要的特性是高可配置性，模块化；它检索的目标可以是本地文件系统，HTTP 或者 FTP。

Nutch 是一个使用 java 编写，在 Apache 许可下发行的爬虫。它可以用来连接 Lucene 的全文检索套件；

Pavuk 是一个在 GPL 许可下发行的，使用命令行的 WEB 站点镜像工具，可以选择使用 X11 的图形界面。与 wget 和 httptrack 相比，他有一系列先进的特性，如以正则表达式为基础的文件过滤规则和文件创建规则。

WebVac 是斯坦福 WebBase 项目使用的一个爬虫。

WebSPHINX(Miller and Bharat, 1998)是一个由 java 类库构成的，基于文本的搜索引擎。它使用多线程进行网页检索，html 解析，拥有一个图形用户界面用来设置开始的种子 URL 和抽取下载的数据；

WIRE-网络信息检索环境(Baeza-Yates 和 Castillo, 2002)是一个使用 C++编写，在 GPL 许可下发行的爬虫，内置了几种页面下载安排的策略，还有一个生成报告和统计资料的模块，所以，它主要用于网络特征的描述；

LWP: RobotUA(Langheinrich, 2004)是一个在 Perl5 许可下发行的，可以优异的完成并行任务的 Perl 类库构成的机器人。

Web Crawler 是一个为 .net 准备的开放源代码的网络检索器(C#编写)。

Sherlock Holmes 收集和检索本地和网络上的文本类数据（文本文件，网页），该项目由捷克门户网站中枢（Czech web portal Centrum）赞助并且主用商用于这里；它同时也使用在。

YaCy 是一个基于 P2P 网络的免费的分布式搜索引擎（在 GPL 许可下发行）；

Ruya 是一个在广度优先方面表现优秀，基于等级抓取的开放源代码的网络爬虫。在英语和日语页面的抓取表现良好，它在 GPL 许可下发行，并且完全使用 Python 编写。按照 robots.txt 有一个延时的单网域延时爬虫。

Universal Information Crawler 快速发展的网络爬虫，用于检索存储和分析数据；

Agent Kernel，当一个爬虫抓取时，用来进行安排，并发和存储的 java 框架。

Dine 是一个多线程的 java 的 http 客户端。它可以在 LGPL 许可下进行二次开发。

2.2 Java 多线程编程技术

上面介绍爬虫架构时曾经提到过，为了提升爬虫性能，需要采用多线程的爬虫技术。采用多线程并行抓取能同时获取同一个网站的多个服务器中的网页，这样能极大地减少抓取这类网站的时间。

2.2.1 创建多线程的方法

多线程是一种机制，它允许在程序中并发执行多个指令流，每个指令流都成为一个线程，彼此间互相独立。线程又称为轻量级进程，他和进程一样拥有独立的执行控制，由操作系统负责调度，区别在于线程没有独立的存储空间，而是和所属进程中的其他线程共享存储空间，这时的线程间的通信较进程简单。

多个线程的执行是并发的，即在逻辑上是“同时”的。如果系统只有一个 CPU，那么真正的“同时”是不可能的，但是由于 CPU 切换的速度非常快，用户感觉不到其中的区别，因此用户感觉到线程是同时执行的。

为了创建一个新的线程，需要做的工作有，必须指明这个线程苏姚执行的代码，在 Java 语言中，通过 JDK 提供的 `java.lang.Thread` 类或者 `java.lang.Runnable` 接口，能够轻松地添加线程代码。

2.2.2 Java 语言对线程同步的支持

首先讲解线程的状态，一个线程具有如下四种状态。

- (1) **新状态**：线程已被创建但尚未执行（`start()` 方法尚未被调用）。
- (2) **可执行状态**：线程可以执行，但不一定正在执行。CPU 时间随时可能被分配给该线程，从而使得它执行。
- (3) **死亡状态**：正常情况下，`run()` 方法返回的是线程死亡。调用 `java.lang.Thread` 类中的 `stop()` 或 `destroy()` 方法亦有同样效果，但是不推荐使用这两种方法，前者会产生异常，后者是强制终止，不会释放锁。
- (4) **阻塞状态**：线程不会被分配 CPU 时间，无法执行。

编写多线程程序通常会遇到线程的同步问题，由于同一进程的多个线程共

享存储空间，在带来方便的同时，也会带来访问冲突这个严重的问题。Java 语言提供了专门机制以解决这种冲突，有效地避免了同一个数据对象被多个线程同时访问的问题。Java 语言中解决线程同步问题是依靠 `synchronized` 关键字来实现的，它包括两种用法：`synchronized` 方法和 `synchronized` 块。`synchronized` 方法：通过在方法生命中加入 `synchronized` 关键字来声明该方法是同步方法，即多线程执行的时候各个线程之间必须顺序执行，不能同时访问该方法。`synchronized` 块：它是这样一种代码块，块的代码必须获得 `syncObject` 对象（可以是类实例或类）的锁才能执行。由于 `synchronized` 块可以是任意代码块，且任意指定上锁的对象，因此灵活性较高。

2.2.3 Java 语言对线程阻塞的支持

阻塞指的是暂停一个线程的执行以等待某个条件发生（如等待资源就绪），Java 提供了大量方法来支持阻塞。如下：

（1）**`sleep()` 方法**：`sleep()` 方法允许指定以毫秒为单位的一段时间作为参数，它使得线程在指定的时间内进入阻塞状态，不能得到 CPU 时间片，指定时间一过，线程重新进入可执行状态。

（2）**`suspend()` 方法和 `resume()` 方法**：两个方法配套使用，`suspend()` 使线程进入阻塞状态，并且不会自动恢复，必须对其应用 `resume()` 方法，才能使得线程重新进入可执行状态。

（3）**`yield()` 方法**：`yield()` 方法使得线程放弃当前分得的 CPU 时间片，但不使线程阻塞，即线程仍处于可执行状态，随时可能再次分配 CPU 时间。

（4）**`wait()` 和 `notify()` 方法**：两个方法配套使用，`wait()` 可以使线程进入阻塞状态，它有两种形式，一种允许指定以毫秒为单位的一段时间作为参数，另一种没有参数。前者当对应的 `notify()` 被调用或者超出指定时间时，线程重新进入可执行状态；后者则必须在对应的 `notify()` 被调用时，线程才重新进入可执行状态。

2.3 Html 解析

2.3.1 Html 语言简介

HTML(HyperTextMark-upLanguage)即超文本标记语言，是 WWW 的描述语言。html 是在 sgml 定义下的一个描述性语言，或可说 html 是 sgml 的一个应用程序，html 不是程式语言，它只是标示语言^[2]。下面这段内容说明了一个最基本的网页文件的组成结构。

```
<html>

<head>
<title>显示在浏览器左上方的标题</title>
</head>

<body bgcolor="red" text="blue">
<p>红色背景、蓝色文本</p>
</body>

</html>
```

这些尖括号对（< >）与其中的字符序列就是 HTML 标签，一个 HTML 标签必须是由“<”开头，由“>”结尾 HTML 语言中的标签通常是成对使用的，它使用一个开始标签和一个结束标签来标识文本，结束标签是在标签名称前加一个“/”，也就是以“<标签名>”表示标签的开始，以“</标签名>”表示标签的结束。一对标签中还可以嵌套其他的标签，所以，成对标签又称之为容器。许多 HTML 标签都可以设置一个或多个属性来控制标签的显示效果，例如，<marquee>标签中的 behavior、标签中的 size 和 color 就是 HTML 标签属性。属性设置的一般格式为：属性名=属性值，属性值部分可以用英文的双引号（" "）或单引号（' '）引起来，也可以不使用任何引号。有些属性是公共的，这些属性的名称和作用在每个 HTML 标签中都完全相同，有些属性是某个 HTML 标签专用的。HTML 标签、属性名与属性值都是大小写不敏感的，即、、以及的效果是一样的。

2.3.2 正则表达式

正则表达式（regular expression）就是用一个“字符串”来描述一个特征，然后去验证另一个“字符串”是否符合这个特征。比如表达式“ab+”描述的特征是“一个‘a’和任意个‘b’”，那么‘ab’，‘abb’，‘abbbbbbbbbbb’都符合这个特征。

正则表达式可以用来：（1）验证字符串是否符合指定特征，比如验证是否是合法的邮件地址。（2）用来查找字符串，从一个长的文本中查找符合指定特征的字符串，比查找固定字符串更加灵活方便。（3）用来替换，比普通的替换更强大^[10]。正则表达式语法示例：“^s*\$”匹配空行。

“\d{2}-\d{5}”验证由两位数字、一个连字符再加 5 位数字组成的 ID 号。

“<\s*(\S+)(\s[^>]*)?>[\s\S]*<\s*/\1\s*>”匹配 HTML 标记。

Java JDK 1.40 版本中，Java 自带了支持正则表达式的包。在 regex 包中，包括了两个类，Pattern(模式类)和 Matcher(匹配器类)。Pattern 类是用来表达和陈述所要搜索模式的对象，Matcher 类是真正影响搜索的对象。另加一个新的例外类，PatternSyntaxException，当遇到不合法的搜索模式时，会抛出例外。用下面一段代码来具体说明：

```
//查找以 Java 开头,任意结尾的字符串

Pattern pattern = Pattern.compile("^Java.*");

Matcher matcher = pattern.matcher("Java 是一门高级编程语言");

boolean b= matcher.matches();

//当条件满足时,将返回 true, 否则返回 false

System.out.println(b);
```

其中 pattern.compile() 中填写的即为想要匹配的正则表达式。

第三章 面向微博网页爬虫的设计

3.1 需求分析

3.1.1 项目中爬虫的需求分析

微博平台是当今信息分享平台中的大热点，知名度非常高。它广泛分布在桌面、浏览器、移动终端等多个平台上，人们更可以随时随地使用微博分享财经世界的大情小事、热门话题、财经动态，更有股市专家为实时解盘。可以随时随地与百万网友分享理财心得，与财经名人零距离接触，与股市达人探讨学习，与亲朋好友保持联络。为我们所需的金融信息提供了很大的信息源^[1]。

我们基于微博的金融咨询检索平台项目通过检索非常有价值的微博内容，来给用户提供金融资讯的帮助。因为我们的项目是基于微博的，所以所有的数据来源就是个大微博网站，而我们获取它们数据的方式有两种，一是网络爬虫：对初始 URL 库过滤及选取，抓取指定 URL 的网页；二是使用微博 API：使用新浪、网易、腾讯及搜狐微博提供的微博 API，直接获取指定名人、机构的微博文本数据。且通过新浪、搜狐、腾讯及网易微博等大微博网站的深入调研，其微博 API 都是开放的。API 获取数据子系统利用微博网站提供的 API，获取指定名人、机构的微博内容。其方便、快捷，省去了搜索引擎抓取网页、分析网页、噪声去重等步骤。不仅能保证数据获取的实时性，还能降低开发代价。

但因为微博热度的蒸蒸日上，各主题网站或小众网站也开始有了自己的微博，为了使我们的数据更加全面，将会有越来越多的小流量微博网站数据产生。这其中不乏大多都是未开放 API 的网站，如和讯财经微博就是其中之一。它有着我们所关注的大量财经信息，但却没有提供给外界开发者的 API 平台。可是这样与财经金融信息高度相关性的数据是非常宝贵的，所以需要我们自己编写网络爬虫来获取我们所需要的信息。即上段中关于项目获取数据的第一种方式。而这样的爬虫因为只针对微博网站，获取的信息模式和规则大多是相同的，它属于定向爬虫。通过对其的设计与实现，我们不仅解决了项目中关于小流量微博网站数据获取的问题，同时也可以作为一个爬取微博信息并实时更新的专用爬虫。

3.1.2 与通用爬虫的区别

随着网络的迅速发展，万维网成为大量信息的载体，如何有效地提取并利用这些信息成为一个巨大的挑战。搜索引擎成为我们最常用的检索工具，而搜索引擎的数据来源大多是通过爬虫去完成的。他们通过让爬虫进行大量数据的爬取，来进行检索。而通用爬虫大多是服务于这样的需求。首先，因为它们需要尽可能

多的获得网络上的资源，即追求覆盖面的最大化。因此大多采用广度优先搜索策略。其次，因为数据量大的因素，它们的并行爬取度非常高，并且从一个大的种子库进行抓取数据，互相之间有着非常强的互斥关系，以防抓取动作的重复。最后，因为它们的覆盖面很广，每页数据的相似度较低，不可能制定统一的格式化模板对信息进行格式化存储。也没有专门针对性的实时更新功能。数据量过大，也不适合将每次爬取的数据都进行存储在本地。所以，我们专门面向微博网页的爬虫跟通用爬虫还是有着很多鲜明差别的。且一个通用爬虫是无法简单替代一个面向微博网页的爬虫的。

我们面向微博网页的爬虫从形式上来说属于聚焦爬虫，因为它是定向地抓取相关网页资源的爬虫。是一个自动下载网页的程序，它根据既定的抓取目标，有选择的访问万维网上的网页与相关的链接，获取所需要的信息。首先与通用爬虫最大的不同就是，它并不追求大的覆盖，而是将目标定为抓取与某一特定主题内容相关的网页，为面向主题的用户查询准备数据资源，即为我们的基于微博的金融咨询检索平台提供数据来源。除了聚焦爬虫的相关特点以外，因为它只针对微博网站进行爬取，所以抓取策略是固定不变的，采用深度优先搜索策略，按照既定规则进行抓取直到最深一层，抓取工作便可以结束，是一个可估计的有限爬取。有着较为固定的模式。再之，整个抓取结果的每一页有着高度的相似性，我们可以制定好统一的模板进行信息的初步抽取。最后，为了保持信息的新鲜程度，我们有着根据抓取结果模板制定的更新策略。这些都是通用爬虫所没有的，也正是我们要进行研究的的目的。通过对其的设计与实现来解决通用爬虫所不能处理的问题。

3.2 面向微博网页爬虫概述

通过上一节的关于面向微博网页爬虫的需求分析及其与通用爬虫区别的介绍，我们可以了解到该爬虫的大致作用和实现其的目的。下面我们通过对其功能、采用的设计策略、总体结构的分别介绍来更加深入的了解它。

3.2.2 爬虫功能介绍

首先，我们的项目基于微博的金融检索平台是金融资讯电子杂志的上游项目。实现金融名人及机构的微博数据的抓取、分析、索引及用于挑选杂志内容的检索、展现、筛选及杂志内容入库及管理。以下为它的功能示意图 3-1：

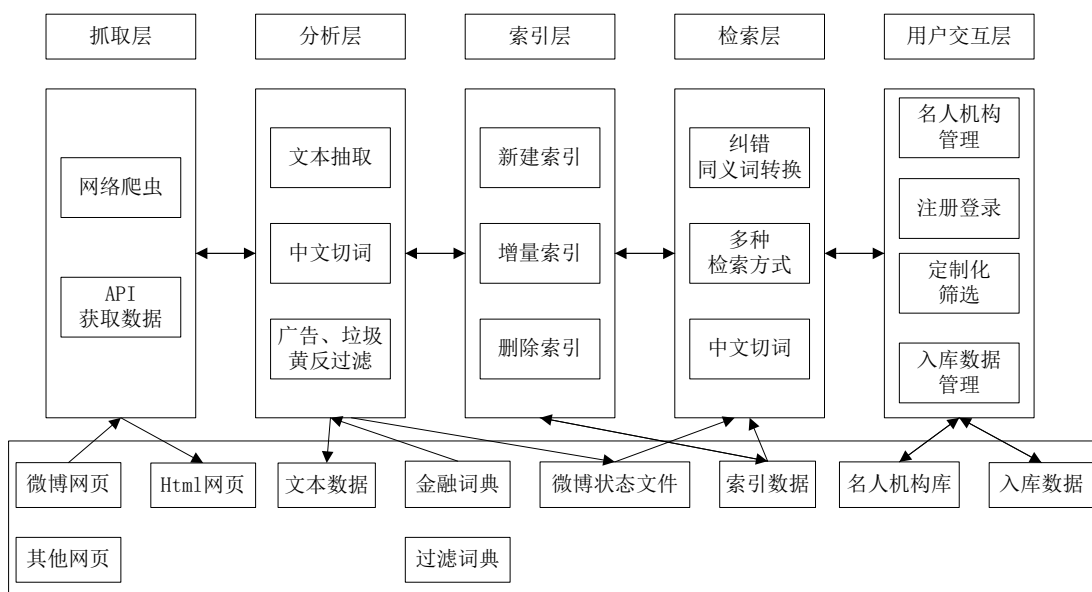


图 3-1 基于微博的金融检索平台功能示意图

本课题要完成的爬虫即上图抓取层中的网络爬虫部分。由于大多数微博网站，如新浪微博，腾讯微博等都有其提供开放的微博 API，可直接获取指定名人、机构的微博文本数据。其方便、快捷，省去了搜索引擎抓取网页、分析网页、噪声去重等步骤。不仅能保证数据获取的实时性，还能降低开发代价。即上图 3 中抓取层的 API 获取数据部分。但还有少数未开放 API 的微博，如和讯财经微博。故需要自己编写网络爬虫。本文则以和讯财经微博为例，介绍面向微博网页爬虫的设计与实现。通过爬虫的工作，要得到像 API 一样提供给用户的源数据。故需要爬虫具备以下功能：

(1) URL 库过滤及选取。因为和讯财经微博中的信息量较大，如果我们爬取所有用户的微博信息，数据量过大，且有很多非相关性信息，不能突出重点，且有可能影响数据的准确性和可用性。所以要对初始 URL 库过滤及选取，只抓取指定 URL 对应的 html 网页。

(2) 抓取原始数据。访问站点服务器，抓取该 URL 的源码数据。

(3) 更新数据。要对微博内容自动采集，抓取内容可随着被采集网页内容的更新而更新。

(4) 抓取内容的信息抽取。在抓取的过程中，可实现对相关信息的抽取工作，例如该数据的用户 ID 及其当前最新微博 ID。

(5) 并对数据进行本地存储。因为数据量过大，所以在存储时要按设计的格式存储，且分布存储在不同的文档中。

(6) 多线程并行抓取。因为数据两过大，为提高效率一定要多线程并行抓取原始数据及更新数据。

3.2.2 爬虫设计策略

因为我们的平台只关注金融资讯，我们是面向微博网页的爬虫，所以该爬虫只关注微博中的金融信息。和讯金融微博刚好满足了这两个条件，以其为例即只爬取该站点中的特定信息。属于定向爬虫，定向爬虫是网络爬虫的一种。定向爬虫可以精准的获取目标站点信息。定向爬虫获取信息，配上手工或者自动的模版进行信息匹配，将信息进行格式化分析存储。它的优势：基于模版的信息提取技术，能提供更加精准的信息。劣势：目标网站难以大面积覆盖，因为基于模版匹配的信息提取技术，需要人工的参与配置模版，欲要大面积覆盖各个目标网站，需要大量的人力成本，同样维护模板也需要很大的人力成本。

(1) 网页抓取策略：在之前背景知识介绍中，有三种网页的抓取策略，基于以上原因，且因为抓取每个用户的微博网页都是有页数限制的。所以刚好避免了深度优先搜索策略的缺点，不会因为 Web 结构相当深，造成一旦进去，再也出不来的情况发生，所以深度优先搜索策略是最符合该爬虫的抓取策略。深度优先搜索策略即在一个 HTML 文件中，当一个超链被选择后，被链接的 HTML 文件将执行深度优先搜索，即在搜索其余的超链结果之前必须先完整地搜索单独的一条链。深度优先搜索沿着 HTML 文件上的超链走到不能再深入为止，然后返回到某一个 HTML 文件，再继续选择该 HTML 文件中的其他超链。当不再有其他超链可选择时，说明搜索已经结束。

根据和讯财经微博的网页结构，它的网页最下端显示该用户发表微博的页数，如下图 3-2 所示：

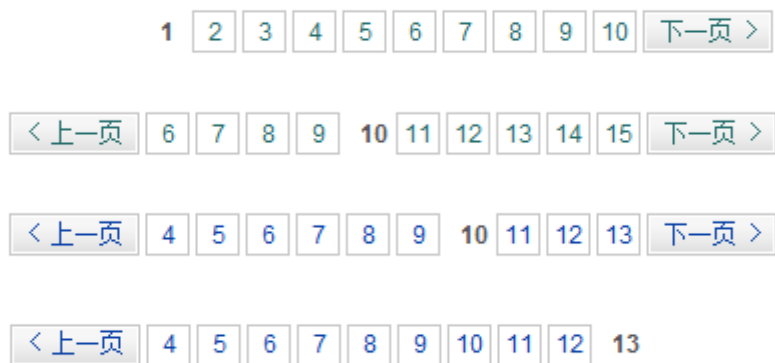


图 3-2 和讯财经微博页面页码

如上图 4 所示，第一行为当前页面为第 1 页是，页码栏的显示情况；第二行为当前用户的微博数量为 15 页以上，当前页面为第 10 页的显示情况；第三行为该用户微博数量为 13 页，当前页面为第 10 页的显示情况；最后一行为该用户微博共 13 页，当前页面为最后一页第 13 页的显示情况。通过以上页码栏的显示规律，则可以总结出深度爬寻的具体抓取策略。因

为不管该用户共有多少页的微博数量，只要当前页面不是最后一页，就会存在“下一页”的出发按钮，通过查看页面源码发现下一页中的 URL 即为该用户微博下一页的 URL 访问连接的后半部分。如下图 3-3 所示。

```
<li>
<li class="nextbtn2 a555">
<a href="/dszmdr/p2/default.html">下一页</a>
</li>
```

图 3-3 下一页 URL 源码

故可直接即对“下一页”按钮的 URL 进行判断，如果存在则说明当前页面还不是最后一页，并从源码中获取 URL 继续深度爬寻直到没有“下一页”按钮为止。

（2）数据更新策略：网络具有动态性很强的特性。抓取网络上的一小部分内容可能会花费真的很长的时间，通常用周或者月来衡量。更不要说像微博这样实时更新的信息，内容的更新周期较短。所以，当爬虫完成它的抓取的任务以后，很多新的微博会根据时间的变化不断的增加进来。所以我们要设计一个完善的更新策略来专门负责处理这部分内容的增加。

检测时间最常使用的成本函数，是新鲜度和过时性（2000 年，Cho 和 Garcia-Molina）^[4]。根据我们面向微博网页爬虫的特性，其更新周期较短，所以采用新鲜度这样的成本函数比较合适，我们只需比较其在相同时间当前最新数据与本地数据是否相同来判断是否有更新即可。

新鲜度：这是一个衡量抓取内容是不是准确的二元值。在时间 t 内，仓库中页面 p 的新鲜度定义如下（式 3-1）：

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases} \quad \text{式 (3-1)}$$

并且采用统一策略：使用相同的频率，重新访问 URL 库中的所有的链接，而不考虑他们更新频率。因为微博用户的数量是非常多的，如果一一进行更新频率的测评是一个非常大的工作量，且各用户的更新频率的不稳定性也影响了更新频率的测评结果的准确性。所以采用统一策略，即定期进行刷新，即设定一个频率来进行更新每一个用户的微博数据是比较适合的一个策略。

本文中针对和讯财经微博，做了相关调研工作最后确定其更新频率为一小时较为合理。根据调研，1 小时内和讯所有用户可发布大概 1800 条微博（这个是上午 10-11 点左右，每个小时有差别，此时段为较高峰，为了计算出 1 小时可能发表微博的最大值，所以选择高峰时数据）。因为“随便看看”栏目里的微博 ID 是连续的，说明 ID 是按时间顺序进行排序的。之后的更新策略中根据微博 ID 大小来判断微博的新旧程度也是至关重要的部分。本文针对和讯财经微博中所调查的名人+机构共 1634+109=1743 个。因为微博活跃用户数或总用户数是一个很难调研到的数量，所以根据《和

讯财经微博：不同于大众微博《投资人交流天堂》^[10] 一文中 2010 年和讯公测时的数据，有 30000 用户注册，因为是作为分母进行计算，所以仍然是按最大值计算。最后的得出 1 小时我们针对和讯财经微博的可能发布微博信息量 I 的计算公式如下：

$$I = 1743 / 30000 * 1800 \leq 104 \text{ (条微博)} \quad \text{式 (3-2)}$$

说明即使按最大值计算 一小时也最多 104 条微博，按照开启 6 个线程的环境 1 小时 420M（相当于 2100 条微博）的抓取速度，大约 3 分钟可完成一次更新，远小于 1 小时。所以以一小时为刷新频率较为合理。

（三）并行策略：为了提高整个爬虫的效率，所以引入了多线程技术，目标是最大化的下载的速度。为了防止线程之间重复抓取 URL 页，故该爬虫的多线程设计采用的具体策略是，每个线程负责一个 URL 子库，URL 子库在爬虫抓取工作开始前就已经分好。每个线程只抓取与自己相关的 URL 子库中的 URL 开始的源数据。这样的封装性大大提高了爬虫工作的准确性，且保证重复抓取这样的事情不会发生。并且可以根据抓取速度的要求对线程数量进行控制，增加或减少均可。只要改动相应的 URL 子库便可完成。

每个原始数据抓取线程的功能是相同的，除了 URL 子库和本地存储文档不同以外功能是相互克隆的。更新数据抓取线程也是如此。可替换性较强。且添加新的线程也较为简便。只需按原有线程模板上更改少量参数便可添加一个新的线程。

又因为是向服务器发送请求访问数据，所以很有可能服务器很长时间没有给反应，所以我们需要设定一个线程监控的策略来监控并行线程是否在正常运行。我们采用 `Thread.isalive()` 来判断线程的运行状态。当所有线程都启动之后，不断的判断各线程的运行情况，若有线程死掉，则重启该线程，重启的最大次数为 2 次。若 2 次重启后均失败，则抛出线程运行超时的异常。通过这样的监控策略可以更好的提高并行线程的性能。

3.2.3 爬虫总体结构

正如前面爬虫的并行策略中所说，为了防止线程之间重复抓取 URL 页，故该爬虫的多线程设计采用的具体策略是，每个线程负责一个 URL 子库，爬虫多线程并行结构图如下图 3-4 所示，单个线程的逻辑结构图如下图 3-5 所示：

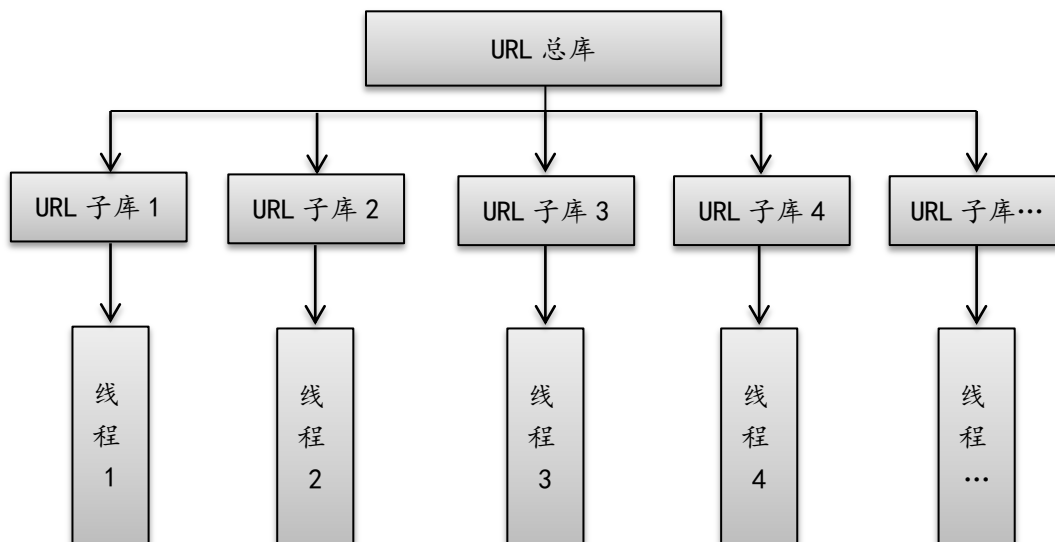


图 3-4 爬虫的多线程并行结构

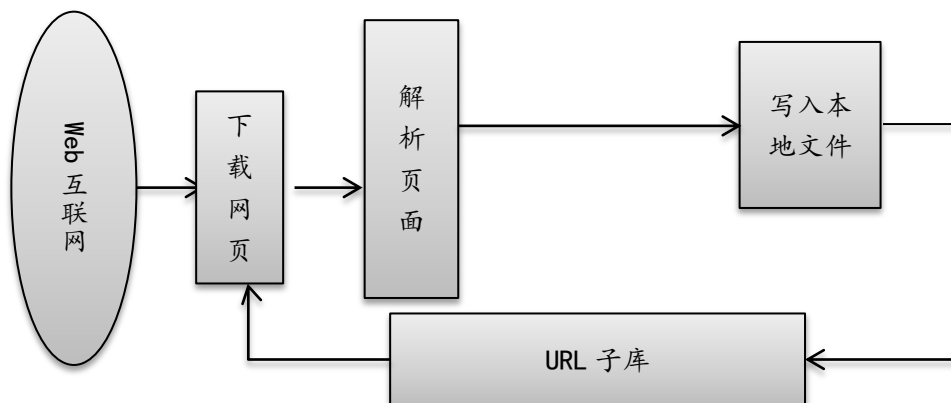


图 3-5 单线程爬虫逻辑结构

3.3 系统各模块介绍

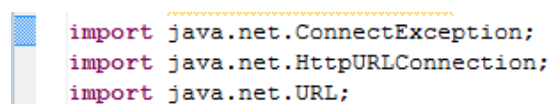
通过上一节对于面向微博网页爬虫的概述，我们大致的了解了爬虫的功能、设计策略以及它的总体结构。下面我们就将其分为从服务器获取源码、解析获取的源码、源码本地存储、更新这四个模块分别进行详细介绍。

3.3.1 从服务器获取源码模块

从上节3.2.3爬虫总体结构的介绍中，我们可以看到，从互联网web中下载网页时整个爬虫工作中的第一步。故它也是爬虫中非常重要的一个环节。下面我们就来介绍这个模块的详细内容。

该爬虫使用Java语言进行编写，Java JDK中提供了可以访问服务器并获取源

代码的类。这也是使用Java进行编码的好处之一。爬虫从服务器获取源码部分就使用了Java.net中的类（如下图3-6所示）来实现本功能。



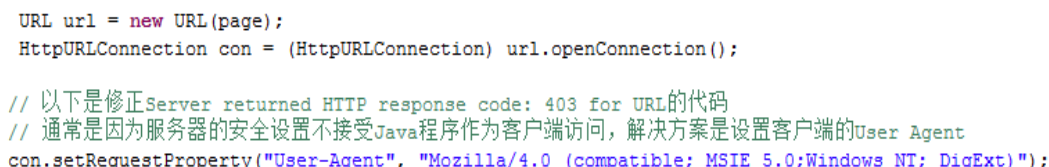
```
import java.net.ConnectException;
import java.net.HttpURLConnection;
import java.net.URL;
```

图 3-6 爬虫中引用的 Java.net 中的类

从服务器获取源码模块，在 GetPage 类中，通过 getPage 方法进行实现。该模块中的主要内容有：

（1）打印 log 日志：因为使用了多线程的并行抓取策略，且各线程都会通过此模块来获取源码。为了能清晰的看到各线程在该模块中的工作情况，在获取源码之前和之后都需将其状态 log 进行输出，保证良好的监控性。

（2）设置 user-agent：由于和讯财经微博不像新浪、腾讯、搜狐等微博较为大众化，故其服务器对爬虫的限制也较高，刚开始实现抓取单个 web 时采用 httpclient 的 java 包来实现，抓取功能正常，但唯有抓取和讯财经微博网页时会出现返回错误码 403 的问题，起初一直找不到服务器返回 StatusCode=403 的原因，因为一旦返回的不是正常的 200，就无法进行接下去的抓取工作。而在抓取其他微博，如新浪，腾讯等时均没有出现过此类问题。猜测有可能因为和讯财经微博没有那么大量的数据访问，对服务器的安全设置权限可能较高，不接受 java 程序作为客户端访问。故终于找到了最终的原因，要改变 user-agent 使其判断为真正的浏览器形式而返回请求的数据。最后改用 URLconnection 类及其改 user agent 的函数，解决了此问题。如下图 3-7 所示：



```
URL url = new URL(page);
HttpURLConnection con = (HttpURLConnection) url.openConnection();

// 以下是修正Server returned HTTP response code: 403 for URL的代码
// 通常是因为服务器的安全设置不接受Java程序作为客户端访问，解决方案是设置客户端的User Agent
con.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible; MSIE 5.0; Windows NT; DigExt)");
```

图 3-7 为爬虫设置 user-agent

（3）通过 HttpURLConnection 类获取源码：通过 HttpURLConnection 类建立连接后，获取一个输入的字符流，但是在写入 txt 文件时，需将其当文本方式写入。故需将其按行读出并添加“\r\n”即回车符之后拼接成一个大的字符串。方便写入读出操作，及后续的抽取信息操作等。最后以字符串形式返回源码。

3.3.2 解析获取的源码模块

因为从服务器抓取过来的信息是庞大且文理性较差的，为了方便存储及之后的更新策略的实现。我们需要通过一些初步的解析工作来疏导这些源码，并让他们以设定好的规范格式存储在本地文件中。

在前面背景知识章节中我们关于正则表达式做了一部分介绍。这也是本模块的核心内容。整个解析工作的内容就是通过正则表达式从获取的源码中进行信息抽取。在这里仍然要说的是采用 Java 作为该程序实现语言的很多优势。我们在这个模块则采用的就是 java 提供的正则表达式开源包 `java.util.regex.*`。它是 JDK 1.40 版本中自带的支持正则表达式的开源包。

因为获取源码部分都是按照一页一页的形式获取的，且在之后的源码存储过程中，需要直接传入用户 ID 及其最新微博 ID。故需在获取源码后直接从整页源码中解析出相关信息。这部分是在 `TestString` 类中完成。它们大多是通过分析源码格式，找出相关规律来编写正则表达式后进行信息抽取的。

(1) 下一页 URL 解析：抽取源码中“下一页的”URL，例如`<li class="nextbtn2 a555">下一页 >`中的“11561892/p2/default.html”部分，因为它不是下一页的完整 URL，故需为其加上头部 `t.hexun.com/`即可。

(2) 用户 ID 解析：抽取 URL 中的用户 ID，例如 `http://t.hexun.com/citicbankbj/default.html` 中的“citicbankbj”即为用户的 ID。

(3) 最新微博 ID 解析：抽取微博展现模板中的第一条微博的 id，例如源码中的`<div class="blogcon" id="listWeibo"><div class="nr_con" id="articleItem_16378983">`中“articleItem_16378983”即为用户最新一条微博的 id。

(4) 解析本地数据：与之前的情况不同观点是，本地数据不像新获取的数据一样是一整页一个字符串的形式，而是每个 URL 子库的数据存放在一个 txt 文件中。又因为一个 txt 文件时相当庞大的，故不可能将整个数据读取为一个字符串再进行解析。所以采用读取一行进行判断解析的方法来获取相关信息。这部分内容在更新微博类 `Update` 中进行实现。在下一节我们介绍的源码本地存储模块中将会说明存储的格式。所以，这部分的正则表达式也是根据存储格式中的分隔符来进行编写实现的。

3.3.3 源码本地存储模块

因为该爬虫爬取的是某站点关注用户的所有微博信息，因为每个用户从注册到使用较长时间后，其微博原始数据是一个非常庞大的数据量。所以其原始数据的存储分类及格式也是相当重要的一部分。下面我们来主要介绍源码本地存储模块的详细内容。

分块存储策略：根据上章面向微博网页爬虫的设计中爬虫的并行策略中，我们知道每个线程会负责一部分用户的整个抓取过程。所以我们的存储也是按照线程的负责块来分块进行存储，即每个线程按照其 URL 子库进行爬取后则存储在相应的 txt 文档中，每个 txt 都有一个与它对应的线程和一个 URL 子库。这样也方

便信息的查取，因为若要查取某用户的本地源码信息，通过了解他属于哪一个 URL 子库便可找到它存储的文档是哪个。此方法较为方便且不易出错。

该模块在 GetPage 类中的 WriteTXT 方法中实现，接下来我们介绍该模块中的主要内容：

（1）设置存储文档的路径：因为是将数据存储在本地，故必须将其存放位置的详细路径事先设定好。在方法开始前进行初始化。如下图 3-8 所示：

```
public void WriteTXT(String data,boolean first,String ID,int no)
{
    String filePath_c1,filePath_c2,filePath_c3,filePath_c4,filePath_c5,filePath_m,filePath_UPc1,filePath_UPc2,filePath_UPc3,filePath_UPc4,filePath_UPc5,filePath_UPm;
    TestString U=new TestString();

    id=U.id(data);
    filePath_c1=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a1.txt"); //名人源码存储的对应线程的文件
    filePath_c2=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a2.txt");
    filePath_c3=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a3.txt");
    filePath_c4=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a4.txt");
    filePath_c5=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a5.txt");
    filePath_m=("E:\\MyEclipse\\Workspaces\\2012\\spider\\b.txt"); //机构的源码存储的文件（只有一个线程负责）

    filePath_UPc1=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c1.txt"); //更新源码存储的对应线程的文件
    filePath_UPc2=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c2.txt");
    filePath_UPc3=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c3.txt");
    filePath_UPc4=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c4.txt");
    filePath_UPc5=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c5.txt");
    filePath_UPm=("E:\\MyEclipse\\Workspaces\\2012\\spider\\b1.txt"); //更新机构的源码存储的文件（只有一个线程负责）
}
```

图 3-8 存储文档路径的初始化设置

（2）文档初始化及接续写入：因为数据的写入过程是接续的，即不是一次性写入，而是每抓取一次网页就写入一次数据，而并非一次将文档的所有数据写入。所以采用 FileWriter 类来实现这一功能。它可以通过设置第二个参数为 false 或 true 来判断是覆盖当前数据重新写入数据(false)，还是不覆盖当前数据接上文继续写入(true)。

但如此一来，如果是第一次在文档中写入数据，若原有旧数据没有清空则会有冗余数据、错误数据产生。故需要在各文档第一次写入数据时，将文档初始化，即清空文档先前存在的数据。这一工作通过线程在工作时设定一个第一次抓取数据的参数传入此模块，且该模块通过自己的标记来防止各线程间冲突的方法来进行判断。如下图 3-10 所示：

```

if(first==true&&flag==0)
{
    out = new FileWriter(fc1,false);
    out = new FileWriter(fc2,false);
    out = new FileWriter(fc3,false);
    out = new FileWriter(fc4,false);
    out = new FileWriter(fc5,false);

    out = new FileWriter(fm,false);
    out = new FileWriter(u_fm,false);

    out = new FileWriter(u_fc1,false);
    out = new FileWriter(u_fc2,false);

    flag=1;//因为有了多线程，所以必须要控制first标记，不然会出现混乱
}

```

图 3-10 第一次写入文档的初始化设置

(3) 根据线程序列号判断存储文件：根据之前介绍的存储策略，每个线程对应一个 URL 子库，同时对应一个分块存储的 txt 文档。所以当线程使用该模块进行信息存储时，需要将线程序列号作为一个参数传入，该模块通过线程序列号来判断其存储文档名，根据文档名对应的路径，将其抓取的数据写入即可。如下图 3-11 所示：

```

switch(no)
{
    //原始数据抓取线程文档存储位置
    case 1:out = new FileWriter(fc1,true);break;
    case 2:out = new FileWriter(fc2,true);break;
    case 3:out = new FileWriter(fc3,true);break;
    case 4:out = new FileWriter(fc4,true);break;
    case 5:out = new FileWriter(fc5,true);break;
    case 6:out = new FileWriter(fm,true);break;
    //更新数据住区线程文档存储位置

    case 11:out = new FileWriter(u_fc1,true);break;
    case 12:out = new FileWriter(u_fc2,true);break;
    case 13:out = new FileWriter(u_fc3,true);break;
    case 14:out = new FileWriter(u_fc4,true);break;
    case 15:out = new FileWriter(u_fc5,true);break;
    case 16:out = new FileWriter(u_fm,true);break;
}

```

图 3-11 根据线程序列号判断存储文件路径

(4) 数据写入格式：因为各用户的微博数据庞大，所以为了方便之后的信息检索，我们需要在数据进行写入时，就保持良好的数据写入格式和规范。所以我们设定了这样的写入格式。首先，爬虫抓取工作是一页一页抓取的，所以当每一页被获取后，需先抽取该微博用户的 ID，和其当前最新微博的微博 ID。因为更新策略中，判断微博是否有更新完全靠当前用户最新微博 ID 和本地存储的之前的最新 ID 之间的比对来完成，所以此信息对之后的更新策略是至关重要的。并且除了在存储时获取该信息以外，为了方便之后更新策略的对比，需要将其一较易查找的位置呈现在存储文档中。且每页微博数据之间也需要相应的分隔符来进行

区分。故设定了如下分割符：

```
String start="\r\n@@@@@@@@@@@@@@@@@@@@@"+ID+"@@@@@@@@@@@@@@@@@@@@@r\n";  
String Item="\r\n#####"+id+"#####r\n";  
String date="\r\n-----"+current_d+"-----r\n";  
String end="\r\n!!!!!!!!!!!!!!!!!!!!!!TIANYUAN!!!!!!!!!!!!!!!!!!!!!!r\n";
```

其中字符串 start 即微博用户的用户 ID（ID 通过解析获取的源码模块来获取并作为参数传入该模块），用来分割每个用户的数据。当第一次写入该用户微博数据时，首先写入字符串 start，再写入其当前最新微博 id（ID 通过解析获取的源码模块来获取并作为参数传入该模块）的信息，即字符串 Item。又因为项目中，做数据解析部分的同学要求提供抓取数据时系统的当前时间，以便进行之后的数据抽取解析工作，所以设计了字符串 date 的写入，它是获取的当前系统时间。之后再写入其微博数据，每页数据写完之后都要写入字符串 end，来区分每一页的数据。这样的存储格式可以方便信息检索及之后的更新策略在比对微博是否有更新时，可快速定位到用户及其本地存储数据的最新微博 ID。

3.3.4 更新模块

微博作为一个实时更新频率非常高的信息发布平台。对其数据的更新更是非常重要的环节。根据前面介绍的该爬虫的更新策略可以了解，更新模块的主要工作就是根据相应频率来刷新数据，每次刷新都通过比对各微博用户的最新微博是否与本地存储的该用户微博 ID 相同来判断是否需要更新数据。下面介绍该模块详细内容：

（1）**多线程并行更新**：即使是要更新的数据量不是很大，但因为我们是定期扫描所有关注的用户。故采用多线程进行更新判断，是一个提高爬虫效率的有效措施。这部分的策略跟抓取原数据是一样的，每个线程负责相应的 URL 子库。线程定期开启后，通过判断每个 URL 的用户是否有更新后，再进行后续的抓取工作。

（2）**新鲜度判断**：首先要说明的是，微博 ID 数字越大表明此条微博越新。除了判断第一页的最新微博 ID 是否与本地不同以外，为了防止该用户的突发数据量，如一小时内更新的微博数量大于 30 条，则有可能第二页中也有新的微博数据。故更新后必须继续进行第二页的判断，若第二页的最新微博 ID 仍大于本地存储中的最新微博 ID 说明第二页中仍有新的微博数据，需要抓取。

（3）**更新数据存储格式**：因为抓取的更新微博是按页抓取的，且只抓取有更新的用户的最新一页微博信息。故其存放顺序与 URL 子库中不完全相同。所以需要每一页源码前加入该用户的 ID 和微博最新 ID。并且，抓取的更新数据是与原数据进行分开存储的，即存储在另一个文档中。因为我们在下一次更新时，只与

上一次更新数据进行对比，而不再与原始的数据进行对比。这样做也可以减少每次对比时的信息读取负担。加快读取速度及分析速度。

3.3 本章小结

本章按照整个爬虫的设计流程顺序详细的介绍了面向微博网页的爬虫的设计过程。先是阐述了该爬虫的需求分析及其与现在已有的通用爬虫的区别，来表明我们此次研究的目的，并且说明了它的设计背景。之后通过对其功能，及其所选用的设计策略，总体结构的设计的介绍让我们对它有了框架性的了解。最后的各模块分别的详细介绍即更加深入的了解整个爬虫的工作原理和细节部分。这样以时间轴为主线由浅入深的介绍，希望能使读者很快得了解爬虫的各个部分，在后续的实现章节中理解其所完成的具体工作内容。

4.1 爬虫工作总体流程

```
graph TD
    URL库([URL 库]) --> 线程1[线程 1]
    URL库 --> 线程2[线程 2]
    URL库 --> 线程3[线程 3]
    URL库 --> 线程4[线程 4]
    URL库 --> 线程更多[线程...]
    线程3 --> 从服务器获取[从服务器获取网页数据]
    从服务器获取 --> 用正则表达式[用正则表达式筛选出下一页]
    用正则表达式 --> URL库
    用正则表达式 --> 按照规定的存储格式[按照规定的存储格式存入本地 txt]
    按照规定的存储格式 --> 用正则表达式
    用正则表达式 --> 循环深度爬寻[循环深度爬寻该用户微博主页]
    循环深度爬寻 --> 用正则表达式
```

```
graph TD; A[更新] --> B[先从服务器请求得到当前最新微博]; B --> C[筛选出微博 ID 与本地存储的用户最新微博 ID 相比较]; C --> D{ }; D -- "若无更新，则继续检查其他用" --> E[URL 库]; E --> A; D -- "若有更新，则将新数据入库" --> F[若有更新，则将新数据入库];
```

The flowchart illustrates the logic of the Weibo crawler. It begins with a box labeled "更新" (Update). An arrow points down to a box labeled "先从服务器请求得到当前最新微博" (First request the current latest Weibo from the server). Another arrow points down to a box labeled "筛选出微博 ID 与本地存储的用户最新微博 ID 相比较" (Select Weibo ID and compare it with the latest Weibo ID stored locally for the user). From this comparison box, an arrow points down to a decision point. If there is no update ("若无更新，则继续检查其他用"), an arrow points left to a box labeled "URL 库" (URL Library), which then points back to the "更新" box via a large curved arrow. If there is an update ("若有更新，则将新数据入库"), an arrow points right to a box labeled "若有更新，则将新数据入库" (If there is an update, store the new data in the database).

28

4.2 爬虫的具体实现过程

根据上面的流程图，我们可以大致地了解到整个爬虫的工作原理及工作流程。同时也根据其工作流程将其所完成的功能进行了分模块的实现，每个模块都有自己相应的所完成的功能。上章也对各模块进行了详细介绍。所以，下面就针对之前各模块的设计来对应的描述其具体实现过程。

4.2.1 获取源码部分

在上一章的各模块详细介绍中，我们已经详细介绍了从服务器获取源码模块的详细介绍。可以了解到从服它也是整个爬虫工作的首个环节，因此它的实现是整个爬虫功能实现至关重要的一部分。从务器获取源码部分的主要流程实际上就是通过 `HttpURLConnection` 类跟服务器建立连接后得到数据流，因为要写入 `txt` 文档中，所以需在每行结尾处加入 “`\r\n`” 来分行存储，最后以一个大的字符串作为返回值写入 `txt` 文档中。

因为在程序运行的过程中，我们为了监控爬虫的执行结果及效率加入了很多打印出的日志信息。通过在获取数据前和完整数据后打印输出的 `log` 信息可以发现，整个爬虫工作中，这部分代码是耗时最大的一部分。因为服务器的反馈速度可能受忙时闲时，服务器反应速度及安全性设置等多个因素的影响，所以为了提高这部分的执行效率，保证其实现过程尽量简洁快速。所以没有设定太多其他的功能在这个模块。下面是该模块的流程图和主要数据结构的展示。

（一）流程图

下图 4-3 为获取源码部分的流程图：

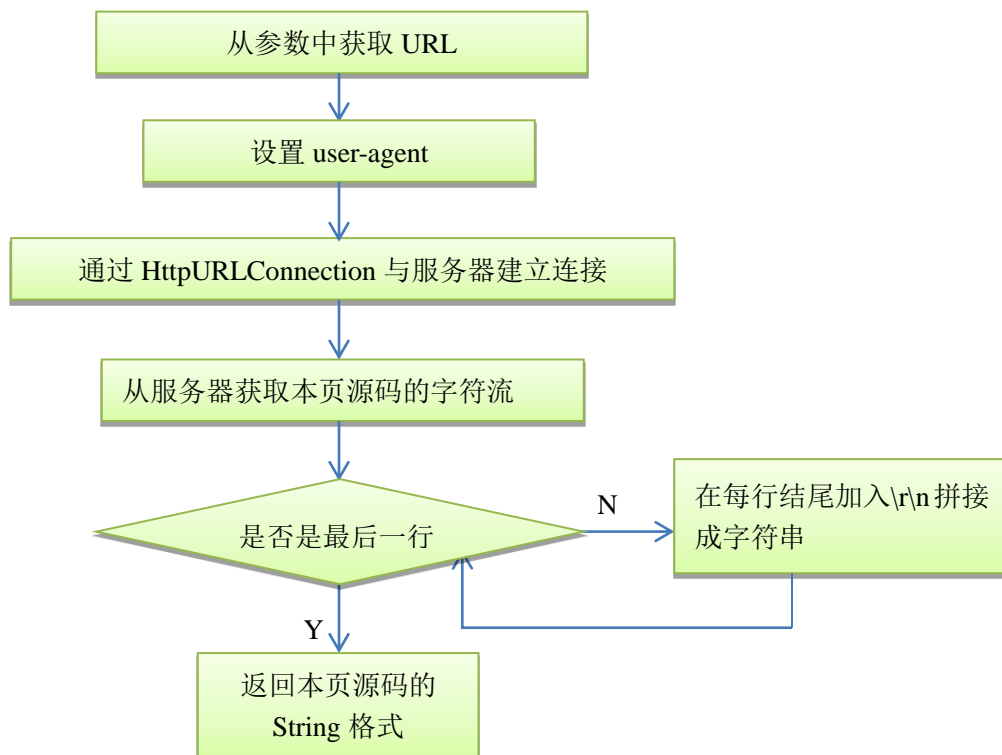


图 4-3 获取源码部分流程图

（二）数据结构

1. 使用 java.net 开源包中的相关类介绍，如下表 4-1 所示：

类名	类的作用
URL	代表一个统一资源定位符，它是指向互联网“资源”的指针；
URLConnection	它是抽象类，是所有类的超类，它代表应用程序和 URL 之间的搜索路径加载类和资源；
HttpURLConnection	支持 HTTP 特定功能的 URLConnection；
ConnectException	试图将套接字连接到远程地址和端口时发生错误的情况下，抛出此异常。

表 4-1 java.net 相关类介绍表

2. 实现该部分的类 GetPage 数据结构

◆ 全局变量

GetPage 类中共有两个全局变量用来临时存储当前用户 ID，和最新微博 ID，如下表 4-2 所示：

变量名称	变量类型	变量作用
Temp_ID	String	存储当前的用户 ID；

Temp_id	String	存储当前的用户的最新微博 id。
---------	--------	------------------

表 4-2 GetPage 全局变量表

◆ 主要方法

GetPage 类中的主要方法 getPage 的相关说明，如下表 4-3 所示：

方法名称	getPage (string page, int no)
方法作用	获取指定 URL 的源代码，并以字符串型是返回结果；
输入参数	page 即为要获取源代码的页面的 URL； no 即为正在使用此方法进行数据抓取的线程序列号；
返回值	类型 string，即为拼接好的源代码。

表 4-3 getPage 方法介绍表

4.2.2 解析源码部分

解析源码部分是整个实现过程中较为复杂的一部分，因为之后的更新模块的全部策略都是基于这部分的信息获取结果上的。如果该部分有任何的差池，都会导致更新结果的不正确。所以除了逻辑性流程上的正确以外，更要保证的是用正则表达式抽取到的信息的正确性。首先要对网页源码进行深度研究和调查，找出能够抽取所需信息（用户 ID，最新微博 ID，页面分隔符等）的正确内容的标识，再根据规则编写正确的正则表达式进行抽取。而这部分的详细内容已在前面的模块介绍中做了说明。我们除了了解到正则表达式在此部分中的关键作用以外，还要了解 java.util.regex.*包提供的正则表达式的实现功能。而下面的流程图和数据结构就是针对整个实现流程的介绍。

（一）流程图

在上一章节的各模块详细介绍中就已经说明，解析源码部分分成两种情况，一种是直接解析某一页的源码，一种是解析某一行的源码，所以其流程框图如下图 4-4 所示：

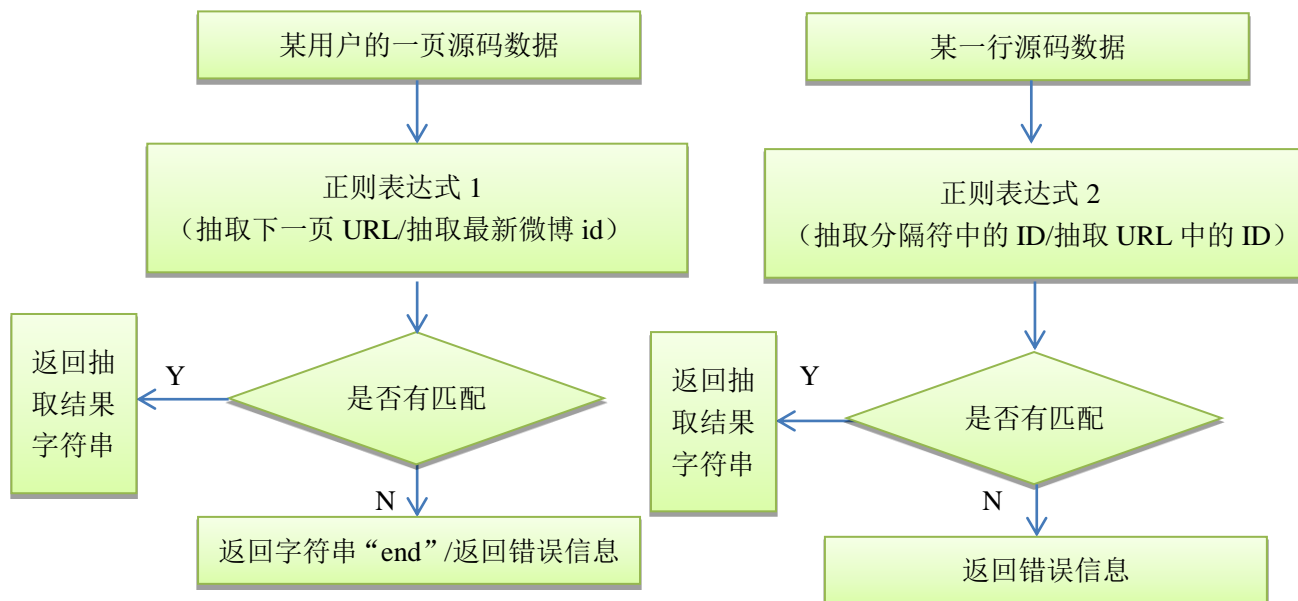


图 4-4 解析源码部分流程图

两个部分的解析流程是相似的，只是为了逻辑判断，抽取下一页的 URL 部分，若没有匹配项说明是没有下一页的按钮的，即已抓取到最深一层，所以在此需设定返回项字符串“end”方便深度爬寻的逻辑判断。其他若没有匹配项则返回错误信息，若有则直接返回抽取结果即可。

（二）数据结构

实现该部分的类 TestString 的介绍：

◆ 全局变量

TestString 中有如下表 4-4 所示的全局变量：

变量名称	变量类型	变量作用
patternString1	String	存储抽取下一页 URL 的正则表达式 1；
pattern1	pattern 类	用来匹配的存有正则表达式 1 的模式 1；
patternString2	String	存储抽取微博用户 ID 的正则表达式 2；
pattern2	pattern 类	用来匹配的存有正则表达式 2 的模式 2；
patternString3	String	存储抽取最新微博 id 的正则表达式 3；
pattern3	pattern 类	用来匹配的存有正则表达式 3 的模式 3。

表 4-4 TestString 类的全局变量表

◆ 主要方法

下面表 4-5, 表 4-6, 表 4-7 分别介绍了 TestString 类中主要的三个方法。

1. public String ID(String URL)

方法名称	ID(String URL)
方法作用	从传入的 URL 中抽取用户的 ID；

输入参数	要抓取源码的网页的 URL;
返回值	类型 string, 为正确的抽取结果或错误提示信息。

表 4-5 ID 方法介绍表

2. public String nextPage(String data)

方法名称	nextPage(String data)
方法作用	从传入的一页源码中抽取下一页要爬取的 URL;
输入参数	某一页源码;
返回值	类型 string, 为正确的抽取结果或“end”。

表 4-6 nextPage 方法介绍表

3. public String id(String data)

方法名称	id(String data)
方法作用	从传入的一页源码中抽取该用户当前最新微博 id;
输入参数	某一页源码;
返回值	类型 string, 为正确的抽取结果或错误提示信息。

表 4-7 id 方法介绍表

4.2.3 存储源码部分

存储源码部分相对来说是实现过程中较为简单的一个环节, 只要保证抓取到的数据按照规定的格式写入相应文件即可。规定的格式即模块详细介绍中各用户, 各页之间的分隔符, 相应文件即根据线程号分开存储的对应文件。下面是该实现部分的流程图和数据结构。

(一) 流程图

下图 4-5 为存储源码部分的流程图:

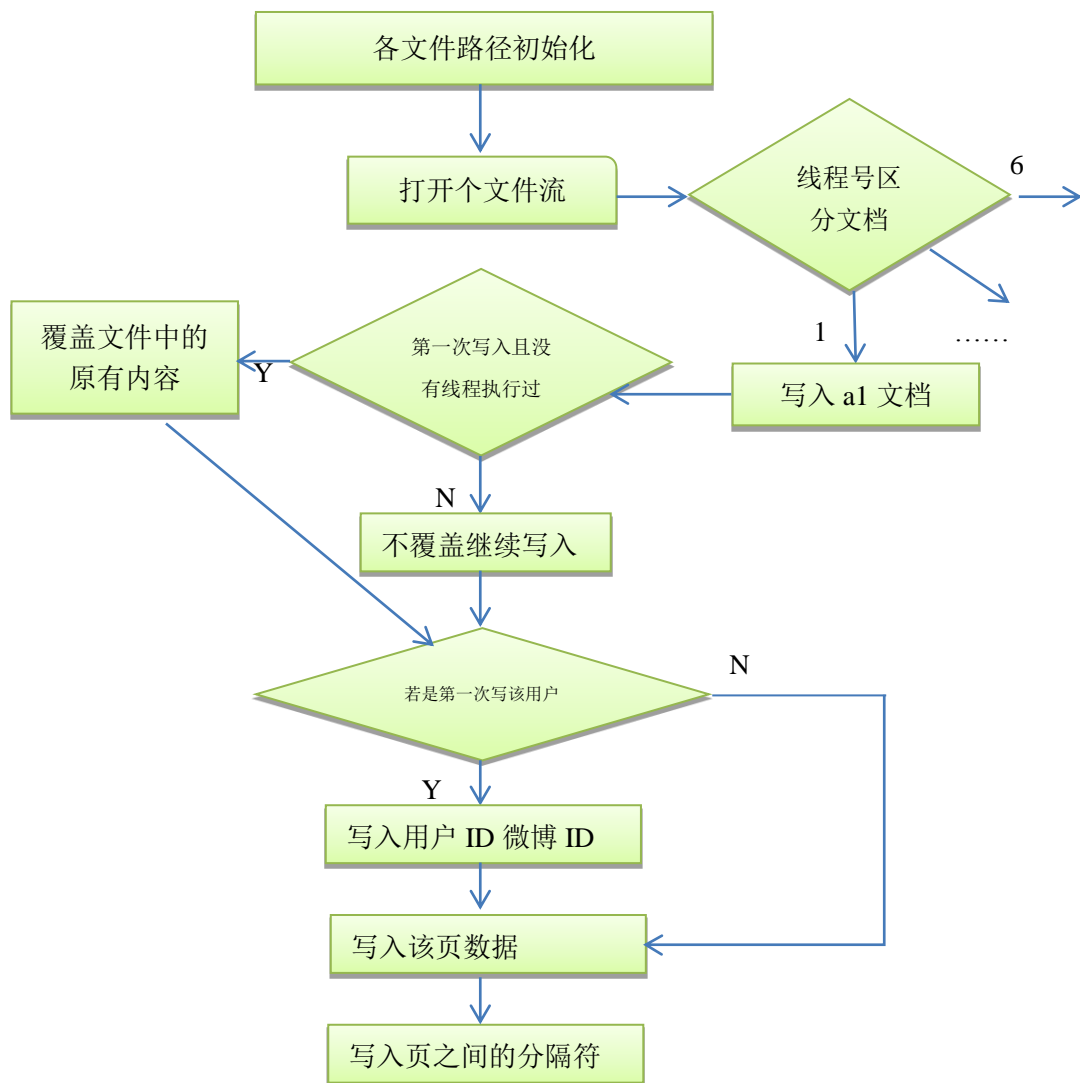


图 4-5 存储源码部分流程图

（二）数据结构

实现该部分的类 WriteTXT 的介绍：

◆ 主要方法

`public void WriteTXT(String data, boolean first, String ID, int no),`

如下表 4-8 所示：

方法名称	WriteTXT(String data, boolean first, String ID, int no);
方法作用	获取一页源码 data，根据参数 first 判断是否覆盖原文件，根据线程程序序号 no 判断写入哪个文件，写入数据前根据是否是第一次写入该用户数据结果在 data 前写入用户 ID 标识符；
输入参数	data 即为该页源码； first 即为判断是否是第一次进行写入数据操作，若为 true 则为第一次写入，false 为不是第一次写入；

	no 即为正在使用此方法进行数据写入的线程序列号；
返回值	无

表 4-8 WriteTXT 方法介绍表

在每次写入数据前，需要将各文档输入流打开，再根据线程序列号来判断应该写入的文且该路径，然后依次写入。判断是否为第一次写入，是为了在每次爬虫重新进行初始数据爬取时，覆盖原先的旧数据，否则就会接续写入，造成混乱。影响后续的分析工作。

4.2.4 更新部分

更新部分是该爬虫后续工作中的重点，因为当原始数据爬取完毕后，更新部分需要保证能长期不间断运行。它的功能主要是进行数据的对比，然后根据返回的对比结果，根据对比结果决定是否需要写入本次抓取的数据。再根据各线程所管理的 URL 子库的不同来确定将更新数据写入哪一个文件夹，以方便下次的更新工作进行。其中需要注意的问题就是每次判断当前抓取页如果有更新内容的话，需再次深度抓取下一页判断是否也有更新内容。因为有用用户也许突发性的发了很多条微博，而我们要将更新的内容全部抓取，不能有遗漏。下面则是该部分的流程图和数据结构的介绍。

（一）流程图

更新部分也采用的是多线程并行更新策略，下图 4-6 的流程图只是单线程的流程图：

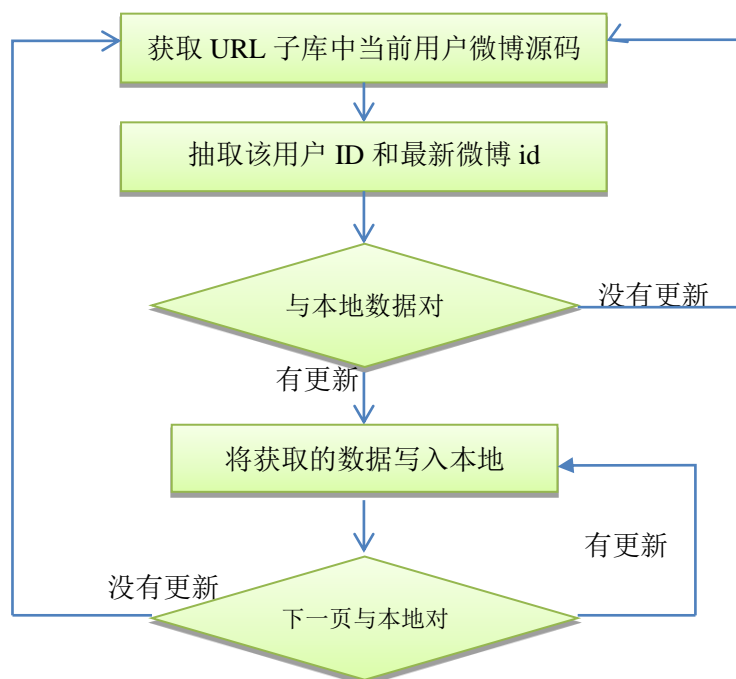


图 4-6 单线程更新部分的流程图

（二）数据结构

实现该部分的 Update 类的介绍：

◆ 主要方法

public boolean judge_new(FileReader f1,String URL_id,String article_id,int type,int no) 如下表 4-9 所示：

方法名称	judge_new(FileReader f1,String URL_id,String article_id,int type,int no);
方法作用	根据当前用户最新的微博 id 在其相应的本地存储文件中寻找上次更新的最新微博 id 进行对比，判断是否有更新；
输入参数	f1，即为根据线程号确定的要查询的本地文件； URL_id 为该用户的 ID； Article_id 为该用户当前的最新微博 ID； type 用来区分是查找该用户本地文件的第一页还是第二页，type=0 第一页，type=1 查第二页，依次类推； no 线程序号用来判断是要在哪个文档中进行比对。
返回值	为 true 表明有更新，为 false 说明无更新。

表 4-9 judge_new 方法介绍表

4.3 爬虫的适应性参数设置

因为本文是以和讯财经微博网站为例进行实现的，但面向微博网页的爬虫不可能仅仅适用于一个特定的网站，故在实现过程中，我们发现了很多因爬取网站不同，需要更改的部分爬虫参数。为了方便该爬虫适应更多的微博网站，我们在代码编写时，也进行了相应的代码优化，方便对这些影响因素参数的修改。这样的做法大大提高了我们面向微博网页爬虫的是适应性性，独立性，可利用价值也更大。所以，我们在本章的最后对其应对不同微博网站时应该更改的参数设置进行了详细的介绍。

4.3.1 URL 种子库的设置

根据前面第三章中各模块的介绍，了解到，我们种子库的大致结构。我们是从 URL 的总库中获取整个 URL 的来源，再进行分解给各个子库。所以若更换了定向抓取的站点，则整个 URL 总库需要全部更新。则直接在其存储文档中进行替换即可。如下图 6-1 所示为 URL 总库：

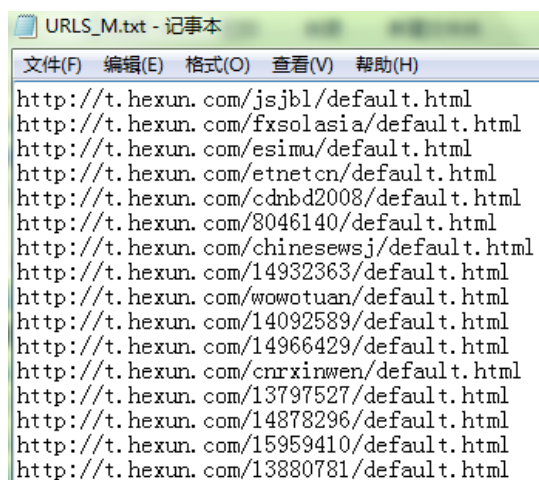


图 6-1 URL 总库表单

该项工作需在爬虫工作开始之前进行，接下来就是种子库的初始化，代码段如下图 6-2 所示：

```
public class Mypublic {
    //*****url数量初始化*****
    static final int C=1586;//所抓微博的名人总数
    static final int M=107;//所抓微博的机构总数

    //*****线程数量初始化*****
    static int Thread_C_num=5;//现在正运行的抓取名人微博的总线程数
    static int Thread_M_num=1;//现在正在运行的抓取机构微博的总线程数
    static int Thread_num=Thread_C_num+Thread_M_num;//现在正在运行的总线程数

    //*****url种子库分解*****
    static int C_num=C/Thread_C_num+1;//分解后各种子库url数量
    static int M_num=M/Thread_M_num+1;

    static String seeds_c1[]=new String[C_num],seeds_c2[]=new String[C_num],seeds_c3[]=
    static String seeds_m[]=new String[M_num];
}
```

图 6-2 种子库初始化位置

首先需要在 Mypublic 全局变量类中修改 URL 的总数量 C 和 M。之后根据开启线程数量的确定，方可确定种子库分解后各线程爬取的 URL 数量之后才可从总 URL 库中读取到各子库中进行初始化操作。

在总爬虫类中，有 URL 子库的初始化函数 initCrawlerWithSeeds(), 如下图 6-3 所示，如 URL 总库在本地的存储地址或文件名有变化，则需在其中 URL 库文件地址中进行修改。

```

public static void initCrawlerWithSeeds()// 使用种子 url 初始化 URL 队列
{
    String add;
    int i=0;
    try{
        //url库文件地址
        FileReader fc=new FileReader("E:\\MyEclipse\\Workspaces\\2012\\spider\\URLS_C.txt");
        BufferedReader br=new BufferedReader(fc);
        add=br.readLine();
        if(add!=null)//从本地文件中读取名人URL列表存入种子组
        {
            for(i=0;i<=Mypublic.C_num-1&&add!=null;i++)
            {
                Mypublic.seeds_c1[i]=add;
                System.out.println(Mypublic.seeds_c1[i]);
                add=br.readLine();
            }
            for(i=0;i<=Mypublic.C_num-1&&add!=null;i++)
            {
                Mypublic.seeds_c2[i]=add;
                System.out.println(Mypublic.seeds_c2[i]);
                add=br.readLine();
            }
            for(i=0;i<=Mypublic.C_num-1&&add!=null;i++)
            {
                Mypublic.seeds_c3[i]=add;
                System.out.println(Mypublic.seeds_c3[i]);
                add=br.readLine();
            }
            for(i=0;i<=Mypublic.C_num-1&&add!=null;i++)
            {
                Mypublic.seeds_c4[i]=add;
                System.out.println(Mypublic.seeds_c4[i]);
                add=br.readLine();
            }
            for(i=0;i<=Mypublic.C_num-1&&add!=null;i++)
            {
                Mypublic.seeds_c5[i]=add;
                System.out.println(Mypublic.seeds_c5[i]);
                add=br.readLine();
            }
        }
    }
}

```

图 6-3 URL 种子库初始化函数

4.3.2 本地存储文档的设置

在 GetPage 类中的 WriteTXT 方法内修改存储原始数据和更新数据文档的存储路径即可。如下图 6-4 所示：

```

public void WriteTXT(String data,boolean first,String ID,int no)
{
    String filePath_c1,filePath_c2,filePath_c3,filePath_c4,filePath_c5,filePath_m,filePath_UPc1,filePath_UPc2,fi
    TestString U=new TestString();

    id=U.id(data);
    filePath_c1=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a1.txt"); //名人源码存储的对应线程的文件
    filePath_c2=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a2.txt");
    filePath_c3=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a3.txt");
    filePath_c4=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a4.txt");
    filePath_c5=("E:\\MyEclipse\\Workspaces\\2012\\spider\\a5.txt");
    filePath_m=("E:\\MyEclipse\\Workspaces\\2012\\spider\\b.txt"); //机构的源码存储的文件（只有一个线程负责）

    filePath_UPc1=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c1.txt");//更新源码存储的对应线程的文件
    filePath_UPc2=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c2.txt");
    filePath_UPc3=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c3.txt");
    filePath_UPc4=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c4.txt");
    filePath_UPc5=("E:\\MyEclipse\\Workspaces\\2012\\spider\\c5.txt");
    filePath_UPm=("E:\\MyEclipse\\Workspaces\\2012\\spider\\b1.txt");//更新机构的源码存储的文件（只有一个线程负责）
}

```

图 6-4 存储文档路径

4.3.3 Html 源文件解析的设置

这部分内容是通用化中变化最多的一部分。因为这部分是完全依照爬虫所要抓取站点的源码风格进行编写的。但只要找出微博用户 ID 及微博本身 ID 和深度爬取下一页微博的 URL 数据所在源码位置，用正则表达式解析出即可。而在更新时所进行的解析本地文件中的用户 ID 和分页识别符是不用改变的。这些都是有统一标识的，如“!!!!!!!!!!!!!!TIANYUAN!!!!!!!!!!!!!!”是分页符一样，它的解析规则不变。在 TestString 类中，将原有的正则表达式规则进行重新编写即可，如下图 6-5 所示：

```
public class TestString {
    public static final String patternString1="<li class=\"nextbtn2 a555\"><a href=\"(.+
    public static Pattern pattern1 =Pattern.compile(patternString1,Pattern.DOTALL);

    public static final String patternString2="http://t.hexun.com/(.+?)/default.html";//
    public static Pattern pattern2 =Pattern.compile(patternString2,Pattern.DOTALL);

    public static final String patternString3="<div class=\"blogcon\" id=\"listWeibo\">\
    public static Pattern pattern3 =Pattern.compile(patternString3,Pattern.DOTALL);
```

图 6-5 正则表达式的更改

4.3.4 线程数量的设置

首先，需根据实际抓取的站点及 URL 数量，用户微博数等信息进行评测来确定最佳的线程数量。数量确认后，即开始进行通用化设置。因为该爬虫的 URL 子库是根据线程数量进行划分的，所以首先也要在 Mypublic 全局变量类中进行线程数值的设置。并且线程中是将机构和名人分开的。如下图 6-6 所示：

```
//*****线程数量初始化*****
static int Thread_C_num=5;//现在正运行的抓取名人微博的总线程数
static int Thread_M_num=1;//现在正在运行的抓取机构微博的总线程数
static int Thread_num=Thread_C_num+Thread_M_num;//现在正在运行的总线程数
```

图 6-6 名人和机构分开的线程数设置

进行了线程数量的设置后，因为线程的序列号是写入文件，还有读取文件等的判断依据，故需与存储文件地址一一对应，如果线程数量增多，序列号也应按序增加。序列号在线程开始出进行了初始化设置。如下图 6-7 所示：

```
public class Thread_Crawl extends Thread{
    String name;
    final static int no=1; //Reminder:线程序列号
    public Thread_Crawl(String na)
    {
        name=na;
    }
}
```

图 6-7 线程序列号的设置

当前根据和讯财经微博抓取数量的情况，序号编写规则为，原始数据抓取线程的为起始号为 1，之后递加至名人数据的线程数结束，再往后继续递加为机构数，直至机构线程数结束。更新数据抓取线程的序号为以 11 开始跟原始数据抓取线程的排序规则相同一次递加编写。

4.4 本章小结

本章在前章设计的基础上，开始了真正的实现工作。也是按照是时间顺序，先画出整体的流程图，然后再分模块进行实现。分为跟模块划分对应一样的四个部分，获取源码，解析源码，存储源码，更新分别进行了详细介绍，并给出该部分的流程图和主要数据结构。并在实现完所有模块功能后，又对面向微博网页爬虫的适应性作用进行介绍。并在最后给出了该爬虫在面对不同微博网站时应该进行的适应性参数调整设置。希望通过本章的介绍，读者能够了解到整个爬虫各功能模块的具体实现过程。

第五章 面向微博网页爬虫的测试

通过对面向微博网页爬虫的设计与实现的详细介绍，我们的整个研究工作已经算是完成了一大半，但是光实现了功能并不能看做是整个工作的结束，最重要的测试环节才能展现出我们爬虫的实现效果和其真正的价值所在。因为爬虫不仅要完成抓取网页，更新网页的操作，其抓取速度，完成一次更新的时间，这些性能也是至关重要的。因为一旦不能达到一定的性能标准，整个爬虫就相当于一个没有实际功效的抓取工具。所以，在本章中我们通过对爬虫的功能和并行效率两方面的测试来向大家展示最后的结果。

5.1 测试环境介绍

5.1.1 硬件配置

- CPU 处理能力：Intel(R) Core(TM)2 Duo CPU T5870 @ 2.00GHz
- 内存：2.00GB
- 硬盘：250GB
- 系统：Microsoft Windows 7 旗舰版

5.1.2 软件环境

- 运行软件：MyEclipse 8.5
- User-agent：Mozilla/4.0
- JDK 版本号：1.6.0

5.2 爬虫功能测试

针对前面第三章设计章节中爬虫功能的介绍，我们了解到面向微博网页的爬虫具备以下功能：抓取原始数据、更新数据、抓取内容的信息抽取、并对数据进行本地存储、多线程并行抓取。下面我们就针对这些功能进行测试。又因为爬虫的整体功能是分为爬取初始数据和增量更新数据两部分的，所以按照这两个大功能分开进行测试。

5.2.1 爬取初始数据测试

运行 Crawl 程序，开始原始数据的爬取工作，即根据 URL 库中的 URL 深度爬寻抓取每一个关注用户的微博数据工作。当前开启的并行线程数是 6 个，其中线程 6 负责机构 URL 的抓取工作，其余线程 1-5 即为名人用户的抓取工作。

首先，程序启动后，各线程开始工作，并打印 log 日志。图 5-1 为各线程开启后工作台打印的 log 日志信息：

```
线程第 一个原始数据爬虫已启动!!!!!!!!!!!!!!
线程第 二个原始数据爬虫已启动!!!!!!!!!!!!!!
线程第 三个原始数据爬虫已启动!!!!!!!!!!!!!!
线程第 四个原始数据爬虫已启动!!!!!!!!!!!!!!
线程第 五个原始数据爬虫已启动!!!!!!!!!!!!!!
线程第 六个原始数据爬虫已启动!!!!!!!!!!!!!!
线程 1 正要获取http://t.hexun.com/gechang68/default.html的数据!!!!!!!!!!!!!!
线程 2 正要获取http://t.hexun.com/14410981/default.html的数据!!!!!!!!!!!!!!
线程 4 正要获取http://t.hexun.com/15844632/default.html的数据!!!!!!!!!!!!!!
线程 5 正要获取http://t.hexun.com/14509526/default.html的数据!!!!!!!!!!!!!!
线程 3 正要获取http://t.hexun.com/suishidiaolong/default.html的数据!!!!!!!!!!!!!!
线程 6 正要获取http://t.hexun.com/citicbankbj/default.html的数据!!!!!!!!!!!!!!
线程 5 已获取完http://t.hexun.com/14509526/default.html的数据!!!!!!!!!!!!!!
```

图 5-1 程序启动时的 log 日志

由上图可知，各抓取数据的线程均启动成功，并开始抓取 URL 的数据。观察本地数据的存储，可看到各本地存储文档也都创建成功，且开始写入数据，如下图所示：







 a1.txt	2012/6/1 14:07	文本文档	297 KB
 a2.txt	2012/6/1 14:07	文本文档	348 KB
 a3.txt	2012/6/1 14:07	文本文档	441 KB
 a4.txt	2012/6/1 14:07	文本文档	62 KB
 a5.txt	2012/6/1 14:07	文本文档	313 KB
 b.txt	2012/6/1 14:07	文本文档	336 KB

图 5-2 本地存储文档创建成功

可通过工作台输出的 log 信息查看当前各线程的运行情况，其中在线程打开文件，写入数据、从服务器获取数据、获取完数据以及正则分析时均有提示信息，通过观测这些数据可以详细的了解目前各线程的运行情况，如下图 5-3 所示：


```

线程4为了写liguoliang2010的数据打开文件了!!!!!!!!!!!!!!
线程4为了写liguoliang2010的数据文件已经打开了!!!!!!!!!!!!!!
线程4要写liguoliang2010的数据了!!!!!!!!!!!!!!
线程4写完liguoliang2010的数据!!!!!!!!!!!!!!
end
end
next是end
线程4正要获取http://t.hexun.com/14782603/default.html的数据!!!!!!!!!!!!!!
线程4已获取完http://t.hexun.com/15930716/default.html的数据!!!!!!!!!!!!!!
在做从URL筛选出用户ID为15930716的正则分析!!!!!!!!!!!!!!

true
在做从data中筛选出最新一条微博的ID为articelItem_5317452的正则分析!!!!!!!!!!!!!!

线程2为了写15930716的数据打开文件了!!!!!!!!!!!!!!
线程2为了写15930716的数据文件已经打开了!!!!!!!!!!!!!!
线程2要写15930716的数据了!!!!!!!!!!!!!!
线程2写完15930716的数据!!!!!!!!!!!!!!
在做从data中筛选出下一页URL是/15930716/p2/default.html的正则分析!!!!!!!!!!!!!!

/15930716/p2/default.html
/15930716/p2/default.html
next是http://t.hexun.com/15930716/p2/default.html
线程2正要获取http://t.hexun.com/15930716/p2/default.html的数据!!!!!!!!!!!!!!
线程3已获取完http://t.hexun.com/shjxq/default.html的数据!!!!!!!!!!!!!!

```

图 5-3 各线程运行情况展示

当陆续有线程抓取完 URL 子库中的数据时，就会打印线程结束的信息，并显示当前正在进行的线程还剩几个，如下图 5-4 所示，因为共有 6 个线程并行运行，线程 4 是第一个结束的线程。

```

end
end
next是end
目前仍在进行的线程有5个-----线程4结束*****
线程1已获取完http://t.hexun.com/gechang68/default.html的数据!!!!!!!!!!!!!!
在做从URL筛选出用户ID为gechang68的正则分析!!!!!!!!!!!!!!

```

图 5-4 线程结束时的 log 信息

当最后一个线程结束时，会如下图 5-5 所示，观察到这个信息时，说明所有并行爬取数据的线程都已完成了自己 URL 子库中数据的爬取。

```

true
在做从data中筛选出最新一条微博的ID为articelItem_5336251的正则分析!!!!!!!!!!!!!!

线程6为了写citicbankbj的数据打开文件了!!!!!!!!!!!!!!
线程6为了写citicbankbj的数据文件已经打开了!!!!!!!!!!!!!!
线程6要写citicbankbj的数据了!!!!!!!!!!!!!!
线程6写完citicbankbj的数据!!!!!!!!!!!!!!
end
目前仍在进行的线程有0个-----线程6结束*****

```

图 5-5 爬取初始数据结束 log 信息

当所有线程结束，我们可以在本地查看到抓取的所有原始数据的存储情况。

1. 抓取结果

图 5-6 为所有文档抓取完信息的展示，其中最大的文本文档已经达到了 1717M 以上的数据，说明数据写入的过程是正常的。

a1.txt	2012/5/30 3:49	文本文档	1,717,583...
a2.txt	2012/5/30 3:27	文本文档	875,086 KB
a3.txt	2012/5/30 3:27	文本文档	819,597 KB
a4.txt	2012/5/30 3:11	文本文档	337,009 KB
a5.txt	2012/5/30 3:05	文本文档	193,391 KB
b.txt	2012/5/30 3:11	文本文档	387,197 KB

图 5-6 爬取初始数据结束各 txt 文档信息

2. 本地文件的存储格式

下面我们来查看数据是否按照原有的存储格式进行本地存储，一下为抓取下来的数据结果存储格式展示。首先是某一用户写入第一页数据时的格式，首行为用户 ID，第二行为他的当前最新微博 ID，第三行为抓取本页信息的年月日信息（为信息抽取模块提供的数据依据）。如下图 5-7（1）所示：

```

#####14430420#####
#####articleItem_16506088#####
-----2012-06-01-----

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>博击盘股的微博-和讯财经微博</title>

<link href="/css/wb.css" rel="stylesheet" type="text/css" />
<link href="/css/vote.css" rel="stylesheet" type="text/css" />

```

图 5-7（1） 用户第一次写入的标识

因为微博数据为一页一页进行抓取的，故每页之间也需要进行分割，下图 5-7（2）为某一用户的两页微博数据间的分隔标识：

```

<!--弹出窗口：微博认证 start-->
<!--弹出窗口：微博认证 end-->

</body>
</html>

#####TIANYUAN#####
-----2012-06-01-----

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>博击盘股的微博-和讯财经微博</title>

```

图 5-7（2） 两个各用户数据的分隔符

通过对整个爬取初始数据工作的运行及测试结果展示，可得出其功能测试的结果，程序可正常运行，并且运行结果正确，输出正常。

5.2.2 增量更新数据测试

原始数据的爬取工作结束了，开始更新功能的测试。运行 Update 程序，当前开启的并行线程数是 6 个，其中线程 6 负责机构 URL 的更新工作，其余线程

1-5 即为名人用户的更新工作。更新工作并行进行的线程也为 6 个，与原始数据爬取时对应的 URL 子库相同。

首先，程序启动后，各线程开始工作，并打印 log 日志。图 5-8 为各线程开启后工作台打印的 log 日志信息：

```
<terminated> Update [Java Application] E:\Genuitec\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\java
线程第二个更新数据爬虫已启动!!!!!!!!!!!!!!
线程第三个更新数据爬虫已启动!!!!!!!!!!!!!!
线程第四个更新数据爬虫已启动!!!!!!!!!!!!!!
线程第五个更新数据爬虫已启动!!!!!!!!!!!!!!
线程14正要获取http://t.hexun.com/hechunying/default.html的数据!!!!!!!!!!!!!!
线程15正要获取http://t.hexun.com/hwn330/default.html的数据!!!!!!!!!!!!!!
线程第一个更新数据爬虫已启动!!!!!!!!!!!!!!
线程第六个更新数据爬虫已启动!!!!!!!!!!!!!!
线程11正要获取http://t.hexun.com/14430420/default.html的数据!!!!!!!!!!!!!!
线程13正要获取http://t.hexun.com/14811947/default.html的数据!!!!!!!!!!!!!!
线程16正要获取http://t.hexun.com/citicbankbj/default.html的数据!!!!!!!!!!!!!!
线程12正要获取http://t.hexun.com/9534015/default.html的数据!!!!!!!!!!!!!!
线程16已获取完http://t.hexun.com/citicbankbj/default.html的数据!!!!!!!!!!!!!!
在做从URL筛选出用户ID为citicbankbj的正则分析!!!!!!!!!!!!!!
```

图 5-8 程序启动时的 log 日志

跟抓取数据时一样，更新数据时也会显示各线程的运行状况，会打印目前正在比对的用户 ID 和最新微博 ID，以及该线程的判断结果（若 ifnew=false 则无更新，ifnew=true 有更新）等 log 信息。各线程的结束提示语也跟抓取数据时一样，在这里不再做展示。如下图 5-9 所示：

```
在做从data中筛选出最新一条微博的ID为articeItem_5336251的正则分析!!!!!!!!!!!!!!
当前用户ID url_id=citicbankbj!!!!!!!!!!
当前用户最新微博ID artice_id=articeItem_5336251!!!!!!!!!!
在本地匹配到的用户ID为citicbankbj
用户citicbankbj在本地最新微博ID为articeItem_5336251!!!!!!!!!!!!!!
判断前本地用户ID为citicbankbj!!!!!!!!!!!!!!
判断前本地用户最新微博为articeItem_5336251!!!!!!!!!!!!!!
判断前抓取的用户ID为citicbankbj!!!!!!!!!!!!!!
判断前抓取的用户最新微博为articeItem_5336251!!!!!!!!!!!!!!
线程16的用户citicbankbj的ifnew=false
目前仍在进行的线程有5更新线程16结束*****
```

图 5-9 程序运行时的 log 日志

下面我们针对一个具体情况进行测试：

某 ID 为 9534015 的用户在 53 分钟前进行了更新，因为我们是一小时爬取一次故本信息属于更新信息。如下图 5-10 所示：



图 5-10 ID 为 9534015 用户的更新微博

因为更新部分的数据是写在另外的 txt 文档中，并且是该 URL 子库中有用户产生了更新数据时才会创建此 txt 文档并将更新数据写入。ID 为 9534015 的用户是属于名人中第二个 URL 子库的用户，故其更新数据应写入 c2.txt 中。如下图 5-11 所示。

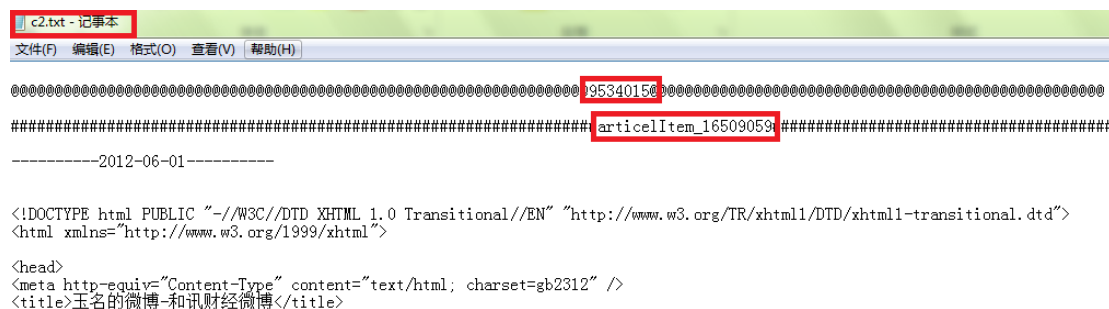


图 5-11 用户 9534015 在 c2.txt 中更新数据的存储

并且，在图 5-12 中，我们可以通过火狐浏览器的自带源码检索工具 firebug 看到该条更新微博的 ID，并与存储在 c2.txt 中的更新微博 ID 信息进行比对。



图 5-12 用户 9534015 最新微博 ID

通过以上 5-11，5-12 两张图中 articleItem_16509059 的对比，我们可以得

到更新结果为正确结果的结论。并且写入数据等操作也按预计进行，输出正常。所以结论为更新功能测试正常。

5.3 并行爬虫效率测试

在本章的开始，我们就已经说明，面向微博网页的爬虫除了要进行有关功能方面的测试以外，还要进行性能的测试。因为它不只是需要能将数据获取下来，根据微博的实时更新的特性，我们的爬虫在抓取数据的效率上也要满足一定的要求才可以称之为一个真正的面向微博网页的爬虫。并且，通过这方面的性能测试，也能让我们更加深入的了解该爬虫的效率受到哪些因素的影响，找出其中较重要的影响因素及最好情况，并将之后的定期抓取工作都放在最优条件下运行，提高其工作效率。

下面我们就通过在不同时段进行原始数据抓取工作的时间进行了统计。因为抓取数据时爬虫最基本的环节，且耗时较长，可以作为一个用来做统计的数据基础。如下图 5-13 所示，其横轴为一天中的不同时段，纵轴为在这些时段进行原始数据爬取所需的时间，单位为小时。

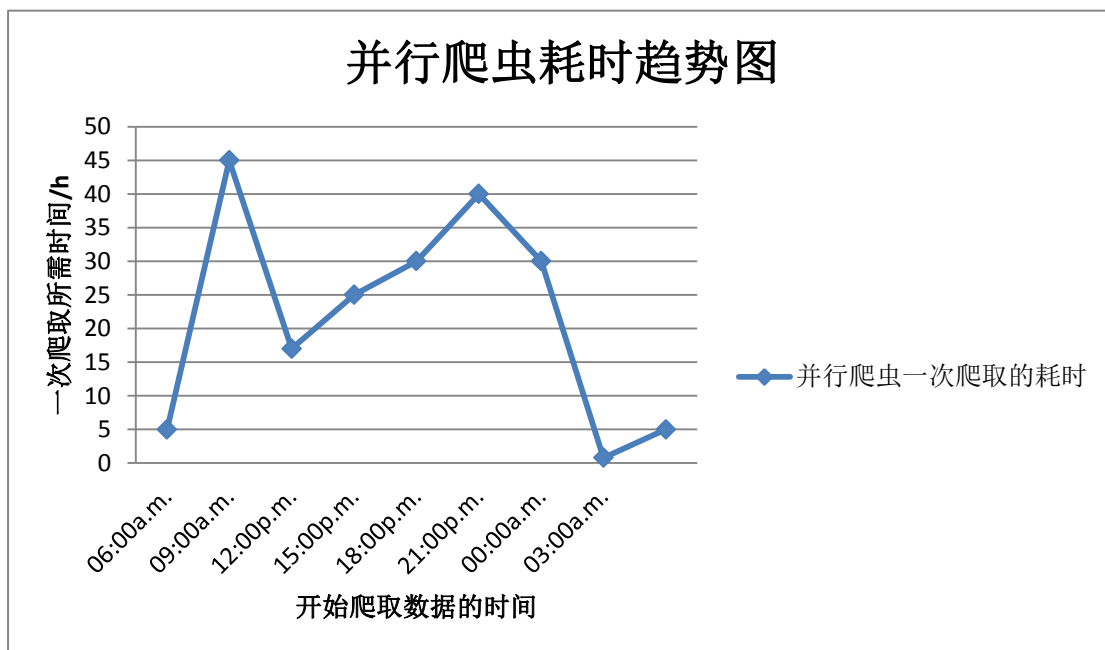


图 5-13 并行爬虫耗时趋势图

通过上图我们可以明显的看出一天中爬虫工作所耗时的长短变化，有两个明显的高峰，分别在上午 9-12 时，和晚间 9 时左右。而两个低谷分别在中午 12 时和凌晨以后。根据这样的趋势图，我们可以从图中直观的感受得到，这个耗时的趋势与网站访问量有着极度的相似。最低谷也是出现在晚上凌晨之后。根据具体的数据，晚上凌晨三点进行一次原始数据的爬取只需要 50 分钟，而当在最高峰时竟需要大概两天的时间才可能完成一次。

于是我们可以得出，爬虫的抓取效率与网站服务器的反馈效率是完全成正比的，当网络流量较大，访问用户较多时，服务器的反馈速度较低，我们爬虫抓取工作耗时也就越大。反之，当到了凌晨这样的用户极度减少的时间段，我们的爬虫工做耗时也变少，效率提高。因此，我们得出面向微博网页爬虫的工作效率是收网站访问效率影响的。为了避免其对爬虫工作的影响，我们把爬取工作设在凌晨，即最有条件下，便可提高整个爬虫的工作效率。

5.4 本章小结

本章主要介绍面向微博网页爬虫的测试工作。首先对测试环境进行了介绍。之后又根据爬虫的两大功能分别对其抓取功能和更新功能的工作进行了测试。各部分都对其工作过程进行了详细的描述，并得出了相应的正确的预计结果。最后，又对影响爬虫性能的因素进行了初步分析并得出爬虫工作在凌晨时效率最高的结果。希望读者通过本章的介绍，能对爬虫的整体功能和性能有一个全面的了解。

第六章 结束语

6.1 工作总结

毕设工作已经接近尾声，在这长达三个多月的时间里，我学到了很多新的知识，同时也是对我大学四年所学内容的总结和提高。我的毕设题目是《面向微博网页的爬虫设计与实现》。在初次选题时，这个题目就深深吸引了我。首先，微博是当今社会的热点项目，特别是我们年轻人都很爱使用它，这样紧跟潮流的题目是我所期望的。并且是做网络爬虫方面的设计与实现工作，这也是我的兴趣点。虽然此前对爬虫只是有所耳闻，知道它的大概工作原理和流程，但没有细致的研究过。因此，从刚开始，浓厚的兴趣便主导我来完成这次的毕设工作。

但是由于此前对爬虫方面的深度接触较少，并且采用从未用过的 Java 语言进行程序的实现。于是我在最开始的时间里通过细心研读《自己动手写网络爬虫》、《21 天学通 Java》等书籍迅速的补充这方面的知识作为整个工作开始的基础。通过学习，我了解到了很多关于网络爬虫的东西，例如爬虫所有的搜索策略，它们的体系结构和一些已经存在的知名开源爬虫的介绍。同时通过这些内容的了解，我再根据所研究课题的需求分析设计出自己的爬虫策略和架构。在之后的实现过程中，也因为之前有了 Java 语言的学习铺垫，能够快速上手。但也因为缺乏相关编码经验，遇到很多自己不能解决的问题，例如：刚开始访问和讯的网站总是不能正常获取到源码；对开发环境并不熟悉，不知道怎样加入外界的 jar 包；解析源码时所用到的正则表达式怎样使用等等。但都通过查阅资料，认真学习，询问学长或老师一一攻克。最后，当整个实现代码完成，看到自己努力辛苦得到的结果，心情是欣慰愉悦的。并且也更加明白，没有解决不了的问题，没有不可攀登的高峰。只要自己足够努力认真，就没有不能完成的任务。除了代码实现，测试环节也是不可缺少的关键环节。在最后的测试工作中，根据测试情况不断地更改优化自己的代码，并最终使爬虫工作正确高效的完成。实现了最初需求分析和设计中所希望达到的成果。

回顾整个毕设的过程，从开始的学习积累到之后的逐步实现再到最后的测试完善。整个过程中随处可以找到自己从发现问题到解决问题的小步骤。一个一个问题的解决促成了最后整个工作的完整收工。虽然毕设只是完成了一个课题任务，但是它更是对我大学四年的学习成果的有效检验。我们学到的也不光只有知识，更重要的是学习的方法和解决问题的能力。通过这样的过程，可以让我们更加领会到这一点。在之后的学习生活中，我们也要将它发扬光大。并且也学会了从失败中总结经验教训，来帮助自己走向成功。

6.2 仍需改进的地方

虽然毕设工作已经收尾，但是由于时间原因没有完善的问题还存在很多。下面我将一一列出并进行总结。

- ✧ 线程监控机制：虽然目前已经设立了检测线程是否正常运行，即检测 `Thread.isalive()` 来监控线程是否正常运行。但是发现当服务器很长时间没有给爬虫返回源码时，虽然 `Thread.isalive()` 的值仍然为真，但是线程其实已经停止运行了。若采用检测获取源码时长来进行判断的话，就需要在线程中再开启一个专门测量时长的线程，结构过于复杂，不易于实现和调试。所以这部分还可以再做优化。
- ✧ log 信息输出规范：因为在测试阶段为了方便调试，都是在工作台上直接输出 log 信息进行检测，且大多提示语都较为口语化。没有固定的输出模式。这部分在后面的工作中可以通过加入 log 包来实现 log 日志的输出。
- ✧ 动态页面的获取：由于微博中的评论是需要通过鼠标触发才能展开的，而我们获取数据都是直接下载整个页面没有动作进行，所以这部分数据目前是没有获取的。在之后的工作中可以继续研究，找到适当的方法来获取这部分数据。

参考文献

- [1] 和讯财经微博 <http://t.hexun.com/default.htm>
- [2] 百度百科 <http://baike.baidu.com/view/284853.htm>
- [3] DataMan'S Blog 《深度优先算法》 <http://hi.baidu.com/dingzhoufang/blog/item/f4867c160922164c20a4e9e0.html>.2008
- [4] Cho,Junghoo,Hector Garcia-Molina 著 《更改频率的估计》(Estimating frequency of change) .2003
- [5] Ipeirotis,P.,Ntoulas,A.Cho,J.,Gravano,L. 著 《文本数据库的建模和内容变化管理》(Modeling and managing content changes in text databases) 2005 年 4 月 页 606-617,
- [6] Koster,M.著 《机器人作家指南》(Guidelines for robots writers) .1993
- [7] 罗刚 王振东著 《自己动手写网络爬虫》.清华大学出版社. 2010
- [8] “北极星工作室” 软件. 《HTML 语言教程》.
- [9] Steve Mansour 著 《正则表达之道》.1999
- [10] 《和讯财经微博：不同于大众微博 投资人交流天堂》 <http://news.hexun.com/2010-06-07/123905451.html><http://news.hexun.com/2010-06-07/123905451.html>
- [11] 庞永庆 庞丽娟著 《21 天学通 Java》 电子工业出版社. 2009

致谢

在论文的结尾，我必须要感谢所有支持我给予过我帮助的人们。

首先是我的指导教师，闫丹凤老师，她活跃的思维，严谨的治学态度，精益求精的精神，都是我需要学习的地方。在毕设中遇到问题的时候，老师总能细心指导，为我指明方向。

此外，我还要特别感谢田瑞学长，他是我日常学习的指导组长，正是他的辛勤努力和耐心指导才使得我的毕设工作得以顺利完成。学长对问题的独到见解和快速解决问题的能力都是值得我学习的地方。

最后我还要感谢所有帮助过我的同学，不管是学习上还是生活上，你们都是促成我顺利完成毕业设计工作的一分子。在此，对你们表示衷心的感谢。也祝愿所有的人在以后的路上都能越走越好。

