硕士学位论文

社交网络数据获取技术与实现

SOCIAL NETWORK DATA ACQUISITION TECHNOLOGY AND IMPLEMENTATION

胡亚楠

哈尔滨工业大学

2011 年 6 月

国内图书分类号：TP391.2　　学校代码：10213
国际图书分类号：681.37　　密级：公开

工程硕士学位论文

社交网络数据获取技术与实现

硕士研究生：胡亚楠
导　　师：郑德权 副教授
申请学位：工程硕士
学　　科：计算机技术
所在单位：计算机科学与技术学院
答辩日期：2011 年 6 月
授予学位单位：哈尔滨工业大学

Classified Index: TP391.2
U.D.C: 681.37

Dissertation for the Master Degree in Engineering

SOCIAL NETWORK DATA ACQUISISTION TECHNOLOGY AND IMPLEMENTATION

Candidate： Hu Yanan

Supervisor： Associate Prof. Zheng Dequan

Academic Degree Applied for： Master of Engineering

Specialty： Computer Technology

Affiliation: School of Computer Science and Technology

Date of Defence： June, 2011

Degree-Conferring-Institution： Harbin Institute of Technology

摘　要

随着 Internet 和信息技术的快速发展，互联网上的信息空前丰富。Web 按其信息蕴藏的深度可分为 Surface Web 和 Deep Web，Deep Web 数据资源包括需要通过查询接口查询才能生成的页面和只有登录后才可查看的专有网络信息。搜索引擎的出现，一定程度上解决了查询信息的需求，但是传统搜索引擎无法索引到这些 Deep Web 页面。如今快速兴起的社交网站，吸引了大量的活跃网络用户，其 Web 信息资源更丰富并且具有很高的价值。本文分析了针对社交网络信息数据获取的框架，设计了抓取 Twitter、Facebook 和人人网的爬虫，并且给出了爬虫管理和数据展示的设计与实现。具体研究工作如下：

1. 研究了 Deep Web 爬虫的框架和模块设计。Deep Web 包括可搜索数据库和专有网络。针对可搜索数据库，爬虫首先要发现数据源，然后对查询接口归约，再把抓取结果聚合。针对专有网络，爬虫首先要获得网站授权，然后抓取页面，再对页面进行分析，最后把结果聚合。

2. 设计实现了 Twitter、Facebook 和人人网的爬虫。Twitter 爬虫的数据获取策略是首先通过 OAuth 认证获取 Access Token，然后调用 Twitter API 增量抓取用户 Twitter 数据。Facebook 爬虫，抓取策略是使用 HtmlUnit 登录，获得一个 Access Token，然后调用 Facebook Graph API 增量抓取用户的新鲜事，解析返回的 JSON 数据并且统一格式。人人网爬虫抓取策略是使用 HtmlUnit 构造浏览器 WebClient 登录，并保存 Cookie，然后使用 WebClient 增量抓取用户页面，解析状态和日志。经过功能测试和大规模性能测试，爬虫能够满足实际工作需要，具有稳定性和适应性。

3. 研究了爬虫管理系统的实现。设计了一个管理控制台和部署在每个抓取机器上的守护程序，他们通过互相通信来实现管理控制爬虫节点的任务分配与负载平衡。守护程序监视爬虫节点运行并解析普通爬虫抓取的数据入库。经过实验测试分析，管理系统能够准确完成设计的功能，并且在大规模通信和数据情况下，性能良好。

4. 研究了利用 Flash 的 ActionScript2.0 语言实现信息可视化效果。完成了抓取数据的可视化展示的动态饼状图。

关键词：Deep Web；社交网络；爬虫；HtmlUnit；Access Token

Abstract

With the rapid development of Internet and the Information Technology, the information on the Web is becoming unprecedented rich. The Web can be divided into Surface Web and Deep Web by the depth of their information. Deep Web data resources include the dynamic page, which need to be generated through the database query interface and the proprietary network information which should log in before you can view. The emergence of search engines, to some extent address the needs of information query, but the traditional search engines cannot index these Deep Web pages. Today, rapidly emerging SNS is attracting a large number of active cyber-citizen. Their information resources are more abundant and have high value. This thesis analyzes the framework of acquire data for the social network, and designs the crawler for Twitter, Facebook and Renren. It completes the design and implementation of crawlers management and data vision. Specific studies are as follows:

1. The design of Deep Web crawler framework and module is researched. The Deep Web includes searchable databases and proprietary networks. For the searchable databases, the crawler should find the data source interface first, then query the interface, and combine the results at last. For proprietary networks, spiders must get a site license at first, then crawl the page and analyzes it. Finally, combine the results as before.

2. Design and implement the crawlers of the Twitter, Facebook and Renren. The Twitter crawler's strategy is that, first to obtain an Access Token through the OAuth certification, and then acquires the data of Twitter by the Twitter API incrementally. The Facebook crawler's strategy is to use HtmlUnit to login and get an Access Token at first, and then call the Facebook Graph API to acquire user's news feed incrementally, and finally parse the returned JSON data and uniform their format. The Renren crawler's strategy is to use HtmlUnit's WebClient to login and save the Cookie firstly, and then use the WebClient crawl user's pages incrementally, the status and notes will be analyze. After functional tests and large-scale performance tests, these crawlers can meet the needs of practical work, with the excellent stability and adaptability.

3. The Crawler’s Management System is researched. A console is designed and daemon is deployed in each crawl machine, they communicate with each other to achieve the tasks allocation and loads balancing. The daemon monitors the node’s status and parses the data of common crawler. After the experiment and analysis, the Crawler’s Management System executes instructions complete accurately and has a good performance of communications and data in large-scale cases.

4. The Flash's ActionScript2.0 language to realize data visualization is studied. We complete a dynamic pie chart to show the acquired data.

Keywords: Deep Web, social network, crawler, HtmlUnit, Access Token

目　录

第1章 绪 论

1.1 课题背景

随着 Internet 和信息技术的快速发展，互联网上信息资源空前丰富，信息的组织方式也趋向于多样化。目前，越来越多的企业、个人等在互联网上建立自己的网站，发布相关的信息；许多互联网提供商通过门户网站、论坛、博客、社区等方式向用户提供新闻、社交和在线游戏服务；还有一些电子商务网站提供在线的电子商务业务。这些信息在互联网上以网页形式存储，部分网页之间通过超链接相关联，从而构成了一个巨大的信息资源网。由此可见，Web 中蕴含着各个领域海量的信息，如何从网络中获取用户感兴趣的信息，一直以来是一个热门的研究课题。搜索引擎的产生和发展，一定程度上解决了用户查询信息的需求。搜索引擎首先尽可能地抓取网页，然后用户提交关键词执行查询，返回最相关的网页。但是，仍然有相当一部分网页无法被搜索引擎检索到，有的只能通过网站的查询接口提交来动态生成，有的需要注册和登录之后才能查看。这些网页隐藏在互联网数据库中，没有超链接的互相关联，其信息量却占整个互联网的绝大部分。传统的 Web 处理相关技术研究和应用无法获取这些信息，亟需新的技术来实现。

1.1.1 课题来源

互联网中网页结构存在很大的差异，网页内容存在多样性和动态性等特点，互联网的规模也在急剧膨胀，自动从网络中抓取有价值的信息已成为当前一项具有挑战性的课题。

互联网上的信息资源按照显示方式主要可分为静态网页和动态网页两种。静态网页的每个网页都有一个固定的 URL，网页内容一旦发布，无论是否有用户访问，网页内容都保存在服务器上，即是说静态网页是保存在服务器上的独立文件。静态网页使用 HTML 语言，内容稳定，网页中没有程序代码，不需要在服务器和客户端执行，容易被搜索引擎检索到并把数据抓取下来[1]。

动态网页是相对静态网页而言的，它是服务器端动态生成的网页。一般情况下，动态网页以数据库技术为基础，实现了视图与控制相分离。动态网页并不是直接存在的网页文件，而是当用户请求时，服务器执行程序代码动态生成

的网页。因此，动态网页可以实现更丰富的功能，包括注册、登录、管理、更新、查询等。但是，一般的网络爬虫和 Web 处理技术无法完全获取这些信息。

随着 Web 的发展，越来越多的信息存储由静态网页转向动态网页，人们开始关注如何有效地获取这些隐藏在后台数据库中和需要登录之后才能查看的数据[2]。按照信息的深度可以将互联网划分为表层网络（Surface Web）和深层网络（Deep Web）。Surface Web 主要是指静态网页等资源，而 Deep Web 则主要指动态网页资源。

一般的 Deep Web 信息资源内容隐藏在网站后台数据库中，只有通过网站的查询接口生成动态网页来访问数据库内容。另外，还有相当一部分信息在存在于专有网络中，比如当前热门的社交网站，拥有大量活跃的网络用户，这些社交网站往往能够以链式反应吸引用户注册，并且用户能够保持长期访问，每天停留时间也长。用户在其中进行互动和交流，这样就集中了时下网络的热点话题与动态，其网页数据一般是用户的观点言论和互动信息，具有很高的分析价值。专有网络往往需要注册登录，访问时有权限要求，而且需要过滤大量的噪声。传统的爬虫无法如何获取这些数据，需要新的技术来解决这些问题，这是我们所着重关注的[3]。

因为这些网站具有的复杂性和异构性，我们需要针对不同的网站设计专门的爬虫，每个爬虫有对应的配置文件，各爬虫也部署在不同的机器上。为了保证各爬虫稳定高效运行，需要设计一个管理系统来管理这些爬虫。管理系统包括一个管理控制台和每个抓取机器上的守护程序，构成一个分布式的爬虫系统。同时，需要设计合理的展示方法，来展现我们所获取的数据。

现今互联网纷杂繁多，各种各样的信息鱼龙混杂，如何对互联网进行管理和监测，也是一个非常重要的问题。普通的 Surface Web 仅仅能够显示少量的信息，我们需要关注的角度应该重点放在互动性强的社交网络上，以追踪当前热门话题，分析舆情动向。本课题正是基于这些具有挑战性的工作而设立的，不仅需要开拓性的研究工作，同时需要大量工程性质的开发来保证系统的功能和效率，符合实际工作的要求。

1.1.2 课题研究的目的和意义

因特网信息资源的迅速增长为人们提供了一片广阔的天地，但是，很显然人们还不能方便地获取自己感兴趣的信息。虽然搜索引擎在一定程度上满足了

用户查询信息的需求，但是，众所周知，搜索引擎的检索结果不尽如人意，而且据调查，作为当前最好的搜索引擎 Google 索引到的信息量也仅占信息总量的 2%，这些信息大多是互相链接的静态网页，即 Surface Web 信息。

与 Surface Web 相比，Deep Web 蕴涵了更加丰富的信息。据 2000 年 7 月美国 Bright Planet 公司的调查，Deep Web 中包含的数据大概是 Surface Web 的 400～500 倍[4]。2004 年 4 月，伊利诺伊大学香槟分校相关研究人员也做了一次大致的估算，他们推测整个互联网上大概有 307000 个包含数据库的网站，比 Bright Planet 公司在 2000 年估计的数目增长了 6 倍。更为重要的是，Deep Web 中包含的信息资源质量高而且分类明确，具有更高的获取和分析价值[5]。

Deep Web 中的数据主要可分为孤立网页和动态网页两类。其中，动态网页已经在前面叙述过，它主要是由可搜索数据库(searchable database，SDB)支持，由服务器来生成。而孤立网页主要是指没有超链接指向的网页，通常情况下是这类网页通过设置权限等策略，使得直接通过超链接访问无效，必须登录之后才能查看这些信息，以防止爬虫等应用程序直接获取数据。这类网页通常存在于专有网络以保证用户的隐私。

如何高效快捷地从 Deep Web 中获取信息，传统的网络爬虫和信息检索是无能为力的，只有通过 Deep Web 爬虫才可以有针对性地搜集 Deep Web 信息资源，然后通过某一方式统一展示给用户查看。调查发现，对 Deep Web 信息的检索是搜索引擎研究的重点，也是发展方向之一，这不仅有利于提高搜索引擎的查找精确率和效率，而且能够很好地提高用户体验。

如今的大公司企业都在收集用户对自己产品的评论信息，以能够及时地对产品做出相应调整，保持市场活力。目前已经有很多优秀的舆情分析工具，可以分析网络上各种信息，通过数据挖掘技术分析出用户的喜好和潜在的购买行为等。那么如何获取这些信息，也显得至关重要。而且，网络用户最为活跃的网站，就是网络言论最集中的地方。因此，针对社交网站上的数据获取与分析，对于各公司的发展也有重要的意义。

本课题针对社交网站进行数据获取研究，这是专有网络的一种，获取难度比较大，需要解决登录问题、session 的保持、JavaScript 的解析等问题。如果能够获取这些数据，对于分析互联网舆论热点和人际关系图和社会舆论热点等，有极大的价值。

1.2 国内外相关研究现状

无论是普通应用程序，还是搜索引擎，为了自动地从互联网上获取信息，都需要设计一个高效的网络爬虫（Spider 或 Crawler）来完成工作。普通爬虫的基本工作原理如下：首先从一个工作种子集出发，将网页 URL 放到待抓取队列里，从队列中取出一个 URL 抓取下来对应的网页，然后提取出网页相应的 URL 放到队列里，如此按照广度优先或者深度优先策略不断地抓取网页，直到达到设定的停止标准为止。爬虫的工作原理简单，但是完成一个高性能高质量的爬虫却是一项比较有技术挑战的工作。

高性能的爬虫主要有三个特点，一是可伸缩，即增加硬件资源就能线性提高性能；二是分布式架构，需要处理协调各爬虫节点的负载和功能；三是个性化，需要能够完成不同的任务，包括本文所研究的 Deep Web 任务[6]。

Google 公司使用的爬虫系统是由斯坦福大学设计的[7]，每个爬虫运行在一台机器上，有 300 个连接并行抓取，由 URL 服务器来分配抓取队列。后期 Google 公司开发了独有的 GFS 分布式文件系统[8]，使用 Map 和 Reduce[9]来处理。Allan Heydon 和 Marc Najork 设计了名为 Mercator[10]的爬虫，它的独到之处是采用了很多优化策略比如 DNS 缓冲、延迟存储等来提升爬虫性能。

另外，还有一些其他的著名的爬虫系统。IRLBOT[11]是由 TAMU 开发的大规模爬虫，它的性能很强大并且较为智能。UbiCrawler[12]同样是一个性能优越的爬虫，它的爬虫具有分布式特点，而且容错率较高。Internet Archive[13]是专门针对 64 个网站进行抓取，每个爬虫负责一个网站。

但是，目前来看，网络爬虫基本上只能根据超链接的关联抓取 Surface Web 的资源，并没有专门针对 Deep Web 数据进行处理。尽管搜索引擎公司在不同的层次提出过垂直检索、个性化检索、框计算等技术，但是仍然只有少数的 Deep Web 资源能够被搜索引擎检索到。如果在检索结果中返回能够加入 Deep Web 数据资源，将会使搜索引擎更加智能化而且极大提升用户体验。

1.3 Deep Web 发展及现状

1.3.1 Deep Web 定义

我们在前面详细阐述了静态网页和动态网页的区别，以及 Surface Web 和

Deep Web 的基本知识。粗略地讲，互联网上的信息资源可以按照存储的深度分为表层网络(Surface Web)和深层网络(Deep Web)。我们把存储在互联网上的静态网页、文件等可以通过超链接关联抓取到的资源称之为表层网络(Surface Web)。

Deep Web 是与 Surface Web 相对应的，这个概念最早是 Dr.Jill Ellsworth 于 1994 年提出的“不可见网络” (Invisible Web)[4]。1998 年，Cho 等人提出了开发“不可见网络”中信息资源的想法，得到了一致的认可[14]。2001 年，Christ Sherman 和 Gary Price 对这个概念进行了扩展，指通过浏览器可以获取，但是普通搜索引擎无法索引到的高质量信息资源。2001 年，Bright Planet 公司公布了《Deep Web 白皮书》，文中把深层网络的概念用“Deep Web”代替了“Invisible Web”，并把 Deep Web 定义为：“保存在可检索的数据库(SDB)中，只能通过查询接口直接查询数据库才能获取的资源”[15]。另一种定义认为：Deep Web 是指可以通过浏览器查看，但由于技术条件限制或其他原因，搜索引擎不能或者不愿对些信息建立索引的网页、文件或者其他信息资源[16]。

综上所述，给出 Deep Web 的定义就是指那些存储在后台数据库中，只有通过查询接口查询才能由服务器动态生成并返回或者获得权限后才能查看的 Web 信息资源，没有超链接指向这些页面，不能被传统搜索引擎索引到[17]。

1.3.2 Deep Web 特点

根据给出的 Deep Web 定义，通常意义上来讲，Deep Web 信息资源主要包括以下四种：

(1) 通过提供的查询接口，填写表单后提交，服务器查询数据库后返回的动态网页信息；

(2) 只针对注册用户开放的网络信息，只有登录后才可查看的专有网络信息；

(3) 由于缺乏超链接指向或者不愿意被检索到的网络信息；

(4) 可以通过浏览器访问的非网页文件。

Deep Web 数据相对于 Surface Web 蕴涵了更加丰富的信息，信息的质量和数量都远高于 Surface Web。根据 Bright Planet 公司和 UIUC 近些年来的调查研究[5]，Deep Web 有以下的几个基本特点[18]：

(1) 信息量大。据调查，Deep Web 信息资源量大约是 Surface Web 的 500

倍，2004 年大约有 307000 个站点，450000 个后台数据库和 1258000 个查询接口。而且它仍在迅速增长，增长速度是平均每年 3 到 7 倍。

(2) 分布领域广。Deep Web 信息资源分布在各种各样的领域，且发展方向更加多元化。电子商务、社交网站等占主要一部分。

(3) 主题分明。Deep Web 网站大多是针对某个特定领域的，可以根据不同的网站设计不同的数据抓取策略。

(4) 信息质量高。相对于 Surface Web 非结构化的网页，Deep Web 信息资源存储大多是结构化的，内容更精深。

(5) 数据获取难度大。Deep Web 信息一般为动态网页资源，需要通过查询接口或者登录来获取[19]。

1.3.3 社交网络定义

社交网络属于 Deep Web 的专有网络。社交网络就是指人和人在互联网上通过朋友、爱好、交易等关系建立起来的社会网络结构。在社交网络中，人和人之间通过互联网服务商提供的功能进行讨论、点评、日志等功能。

社交网络来源于网络交友，随着互联网的迅速发展，社交网络也逐渐演变，为人们的生活提供更便捷的信息交流。社交网络一直朝着节约交流时间和成本，获取高速、高质量信息的方向发展。社交网络通过互联网这一平台，把各式各样的人联系起来，形成具有鲜明特点的社会化集合。

研究表明，社交网络覆盖了社会的各个层次，从达官显贵到平民百姓，从国家外交至家庭关系，都逐渐出现在社交网络上。并且社交网络对于矛盾的化解、企业的运营管理以及个人的成功都有关键的作用。

1.3.4 研究现状

Deep Web 信息资源获取至今仍是一个方兴未艾的热门研究领域，国内外已经开展了大量的相关工作。但是，仍然有很多方面的问题需要解决。

在国外，从 Deep Web 概念的提出到现在，已经有十多年的研究历史，发表了大量的学术专著和论文，研究比较深入。Dr.Jill Ellsworth 于 1994 年第一次提到了不可见网络(Invisible Web)的概念[4]。1998 年，美国的 Lawrence 和 Giles 提出互联网信息利用率问题，他们指出普通搜索引擎的索引率过低，认

为网络资源没有被充分地开发利用，因此，他们提出了获取 Deep Web 信息资源的想法，这是真正的 Deep Web 研究开始。一些公司比如 Bright Planet 和 Complete Planet 也把目光转向 Deep Web 领域，并推出了相关的在线产品。目前，国外 Deep Web 的研究已经成为信息检索和数据挖掘领域的热门研究方向，而且出现了一些 Deep Web 相关的网站，比如 completeplanet.com 和 invisible-web.net，它们按领域对一些 Deep Web 数据库都做了分类。同时，更深层次的 Deep Web 研究，比如数据库查询接口发现、查询接口抽取、聚类、表单自动填充及数据源的自动选择等，也有相关理论和技术。但是，完整的 Deep Web 信息资源获取与集成系统还在研究阶段，而且比较分散，没有真正的产品出现。

我国的互联网起步较晚，发展时间短，而 Deep Web 正是伴随着互联网发展而发展的。因此，与国外对 Deep Web 研究的形成对照的是，国内的学者对 Deep Web 的研究并不太深入，对 Deep Web 相关论述较少。绝大多数网络用户对 Deep Web 的还很陌生，Deep Web 没有得到研究者足够的重视。伴随着 Deep Web 信息资源的不断扩展，部分学者也开展了 Deep Web 探索和研究，比如中国人民大学的数据库小组的孟小峰教授，他们对 Deep Web 信息资源集成的研究有了一定的积累[17]。2008 年，《软件学报》期刊开设了 Deep Web 专刊，收录了 9 篇关于的 Deep Web 信息资源集成的论文。同时一些大学对 Deep Web 的研究是建立在国外相关技术和理论的基础上，开展了 Deep Web 的初步探索，提出了一些相关理论。

总体来说，国内外研究人员目前在 Deep Web 领域已经开展了大量的工作，但各方向发展不均衡。Deep Web 主要的研究技术包括三种：一是接口集成，这种方式是将多个 Deep Web 站点的数据库查询接口抽取出来，构建一种统一的查询界面，主要研究内容包括接口抽取、模式匹配、查询分发、结果合并等部分。对于接口抽取，文献[20]根据对接口可视化分析结果，建立抽取语法规则，抽取语义相关的标签和元素，文献[21]提出一种三层模式模型来捕获 Web 接口。模式匹配是接口集成的核心问题，He 提出了 DCM 方法[22]，利用关联挖掘方法进行复杂的模式匹配，Wu 等利用模式间的桥接效应提出基于聚类的匹配方法[23]。接口集成的典型系统有 MetaQuerier 和 Wise-integrator 等。二是分类目录，基于分类目录的 Deep Web 数据集成是把 Deep Web 查询接口按其主题进行分类，用户可以根据所属类别找到需要的资源。Bright Planet 公司、Invisible Web 公司等均有相关产品。三是 Deep Web 搜索技术，通过一定的策略自动发现互联网中存在的 Deep Web 信息资源，对其进行数据抓取并建

立索引。首先需要设计一个 Deep Web 爬虫，自动识别页面中的检索接口，并填写提交表单。根据爬虫的设计不同，可以分为基准 Deep Web 爬虫和面向主题的 Deep Web 爬虫。基准 Deep Web 爬虫是通过对网页抓取和分析，有效定位可搜索数据库入口并抓取数据[24][25]。面向主题的 Deep Web 爬虫是在发现可搜索数据库的过程中，增加了对主题的分类，使得爬虫更有主题针对性地抓取 Deep Web 的数据源[26]。本体作为最强大的网络信息语义的描述工具，在面向特定领域的检索中被广泛使用，文献[27]中提出了一个结合本体的 Deep Web 语义搜索引擎设计。

对专有网络的数据获取有一定的困难，目前尚未有学者进行过相关的研究。一般的方法是首先通过注册用户的用户名和密码登录，然后获得一个登录凭据，每次都附加上凭据来抓取网页数据。同时，要有一定得抓取策略来循环抓取，保证实时更新。网页如果包含 JavaScript 代码，要能够准确解析执行。

1.4 本文主要研究内容

随着互联网上 Deep Web 信息成倍速度递增，越来越多的信息资源需要查看和管理分析，如何从中准确地找到有价值的信息，将是我们需要解决的首要问题。

Deep Web 数据获取主要是为了有效地抓取那些隐藏在后台数据库中和专有网络的信息资源，普通的网络爬虫无法抓取到这些资源，需要设计有效的 Deep Web 爬虫来实现。本文提到的 Deep Web 的四种类型信息资源，我们将着重研究第二种类型的 Deep Web 信息，即那些需要注册登录的社交网络的数据获取技术以及数据的管理与展示。

本课题的用户的需求主要是完成对互联网的监测与分析，以把握舆论动向和社会热点。因此，在满足了新闻网站的实时抓取与分析之后，抓取网络用户最晚活跃的社交网站的数据，成为我们的研究重点。前文也提到，社交网站往往是舆论中心，大量的用户在其中通过发表日志、微博、分享等表达自己的看法，同时与其他人交流。

根据课题的要求，本文主要研究以下三个方面。

(1) Deep Web 中的数据获取。

包括 Twitter、Facebook、人人网的数据抓取。由于 Twitter 数据的开放性，Twitter 的数据资源是公共的，我们针对 Twitter 采用的抓取策略是直接调用 Twitter 的用户 API，根据返回的结果进行数据解析，以获取该用户的数据。

针对 Facebook，我们需要首先模拟浏览器填写并提交表单来登录，然后获取一个认证序列，并由认证序列来保持身份凭据，然后抓取 Facebook 网站上一些特定用户信息。针对人人网，我们同样需要模拟登录，由模拟的浏览器来保存用户凭据，再抓取人人网的数据。

(2) 数据管理与控制。

设计完成一个管理系统来管理多个节点的爬虫。管理控制台通过 Socket 方式与每台抓取机器的守护程序通信，管理各爬虫节点。守护程序通过配置文件管理爬虫节点，同时检测文件变化，以解析数据存入数据库。

(3) 数据展示。

把抓取的数据以 Flash 形式动态地展示出来。

1.5 论文主要组织结构

本文共分为四章，文章的主要内容与结构如下：

第一章，介绍了课题的背景、来源和研究的目的和意义，介绍了爬虫和 Deep Web 研究的发展和现状。

第二章，介绍了 Deep Web 相关技术，包括自动登录、保持凭据、解析 JavaScript 等技术，给出了 Deep Web 爬虫的工作原理与系统结构。介绍了 Flash 技术和 HtmlUnit 框架。

第三章，介绍了 Deep Web 网站数据抓取策略，分别针对 Twitter、Facebook 和人人网设计了相应的爬虫。

第四章，介绍了爬虫数据管理与展示的基本结构与设计框架。

第2章 相关研究技术

本章首先介绍了 Deep Web 数据获取的关键技术，然后介绍了 Deep Web 爬虫的工作原理和系统框架结构，最后介绍了课题用到的 HtmlUnit 框架和 Flash 技术。

2.1 Deep Web 数据获取设计

Deep Web 蕴涵着丰富的信息，如何从互联网中获得这些信息资源，传统的爬虫无能无力，需要设计 Deep Web 爬虫来完成任务。目前，Deep Web 研究主要有三个主要方向：接口集成、分类目录和 Deep Web 爬虫。本文中只关注 Deep Web 的数据获取技术，首先就是设计 Deep Web 爬虫。

2.1.1 Deep Web 爬虫工作原理

根据定义，Deep Web 数据主要包含两个方面，一是需要通过查询接口查询才能由服务器动态生成并返回 Web 信息资源。二是只针对注册用户开放的网络信息，只有登录后才可查看的专有网络信息[28]。针对这两种信息资源，需要设计不同的数据抓取策略来进行。如下图 2-1 显示了 Deep Web 爬虫的基本工作方式。

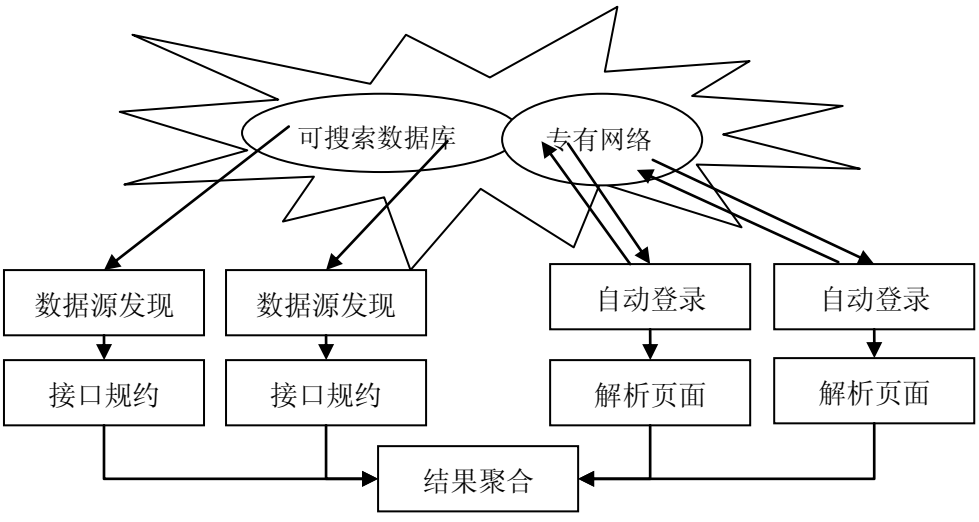

图 2-1 Deep Web 爬虫工作原理

针对第一种类型的 Deep Web 数据资源的获取，其的关键技术包括数据源的发现和选择、完整的查询关键词、结果的聚合。

其中数据源的识别，主要是解决如何使爬虫能够在互联网中快速找到可搜索数据库，并识别数据库的查询接口[29]。W. Meng 提出过一种自动发现查询接口的方法，通过抓取网页并且匹配文字来识别页面，然后提取出页面上的查询表单的参数[30]。Australian 提出过一种基于决策树来识别最优查询接口的方法，通过表单的 Form 标记和文本输入域 input-text 以及 search 字样来识别，准确度不太高[31]。也有研究者提出了动态探测、图模型的方法来识别查询接口。

完整的查询关键词，需要解决如何针对各种不同的查询接口能够给出较为完整的查询关键词，以尽可能返回多的检索结果。J. Wang 提出了基于实例探测的方法，通过对执行查询返回的结果进行分析，得出最适合的查询模式。Sriram Raghavan 提出过名为一个 HIWE 的系统，它能够结合领域知识生成合理的查询关键词提交给查询接口，获得查询结果。

结果的聚合，需要解决如何把信息量大、主题多样的 Deep Web 数据资源抽取成统一的格式并且去除掉不相关和重复信息。C. Yu 提出了从抓取的结果中按数据的格式、内容相似度、表现形式等实现抽取数据的对齐[32]。另外还有一些常用的模式匹配[33]、数据挖掘[34]等方法来完成数据集成工作[35][36]。

针对第二种类型的 Deep Web 信息资源的获取，其关键技术包括自动登录、Session 的保持、页面解析、实时更新。这种类型的 Deep Web 资源一般是专有网络的信息，这类网络需要注册和登录之后才能够访问其中的数据，典型的专有网络有社交网络 Twitter、Facebook、人人网等。由于各专有网站注册时一般需要填写验证码以防止不良用户恶意注册，而且很多验证码一般都是复杂的图片，无法通过图像识别来获取，这就给我们的实现自动注册带来了极大的困难。因此，我们暂时不研究自动注册技术，手工注册账号之后获得用户名和密码在 Deep Web 爬虫里面实现登录。

为了实现自动登录，首先需要找到登录表单，填写登录用户名和密码并与服务器建立连接，然后通过 HTTP 协议的 POST 请求，把用户凭据传到服务器，以实现登录的目的。登录的同时，要对给用户的凭据建立 Cookie，每次访问时在 HTTP 请求头中附加上 Cookie，以保持登录。由于 Cookie 有生命周期的限制，因此需要为 Cookie 设置最大保存时间，过期之后再访问就需要重新登录。

页面的解析技术包括 JavaScript 解析、Ajax 解析等方面。专有网站为了实现良好的用户体验，往往嵌入了大量的脚本代码，在服务器端返回的网页中包

含了一些 JavaScript 语言，只有在 Deep Web 爬虫中实现自动解析 JavaScript 语言才能完全获取到 Deep Web 信息资源。Ajax 技术是基于 JavaScript 的，需要完成向服务器异步请求数据才能完成页面的解析。还有一些服务器返回 JSON 格式的数据，这是一种轻量级的数据交换格式，它是基于 JavaScript 标准的一个子集，也易于解析和生成。

实时更新技术包括如何增量式地抓取 Deep Web 信息资源，如何实时获取数据两个方面。由于 Deep Web 信息源随时都在更新，保持数据的实时性非常重要。有研究曾经提出通过页面的更新频度和重要性来决定抓取频率[37]。也可以手动设置重新抓取时间来实现。

2.1.2 Deep Web 数据获取系统框架

无论 Deep Web 数据获取系统针对何种信息资源，都包括以下几个部分：资源分析模块、数据抓取模块、数据处理模块、用户接口模块。其框图如下图 2-2 所示。

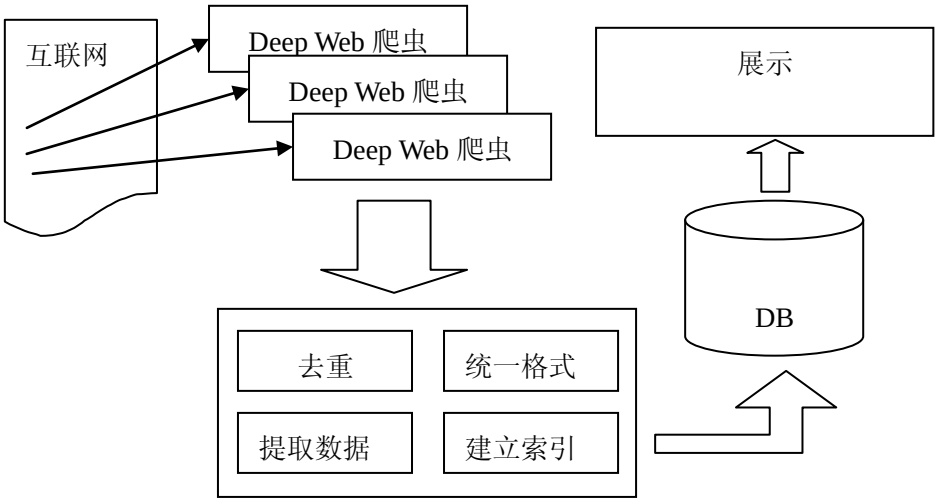

图 2-2 Deep Web 数据获取框架

其中，资源分析模块主要用来完成从互联网中提取 Deep Web 查询接口或者登录到专有网站的功能，数据抓取模块由爬虫来完成网页的抓取，数据处理模块完成对抓取下来的数据解析处理入库，用户接口模块完成对数据的展示。

资源分析、数据抓取和处理模块属于 Deep Web 爬虫部分，这在 2.1.1 节已

经详细阐述。而数据用户接口模块，需要完成把抓取下来的数据展示给用户。Deep Web 信息资源的类别各不相同，需要根据主题设计不同的展现方式。我们可以给用户提供信息检索接口，或者实时刷新网页形式展现，或者采用 Flash 动态效果来展现。

为了实现一个高效可靠的 Deep Web 爬虫，还需要考虑性能以及网页抓取覆盖率、更新策略等问题。爬虫要长期持续运行，需要考虑稳定性以及对系统资源的消耗。爬虫能够在有限硬件和网络的条件下支持海量的数据的抓取，首先就需要设计良好的算法和数据结构，以降低系统资源消耗，其次要有负载的平衡和调度来管理这些爬虫，还要保证一定的抓取速率。为了提高爬虫的抓取数据的质量，还需要对抓取的网页进行分析，包括重要性评价、去除重复、避免下载垃圾信息等。另外，Deep Web 爬虫还需要设计良好的更新策略，一种办法是周期性更新，每隔一段时间重新抓取新的数据替换掉陈旧数据，这种方法性能较低；另外一种办法是增量式更新，根据网页的变化频率抓取最新的信息，这种策略效率更高，但是确定更新间隔时间较难。综上，实际设计中，Deep Web 爬虫还需要增加 DNS 和数据缓存模块、爬虫分布式管理模块、更新控制模块等。

针对专有网络，我们设计的 Deep Web 数据获取基本框架如下图 2-3 所示。

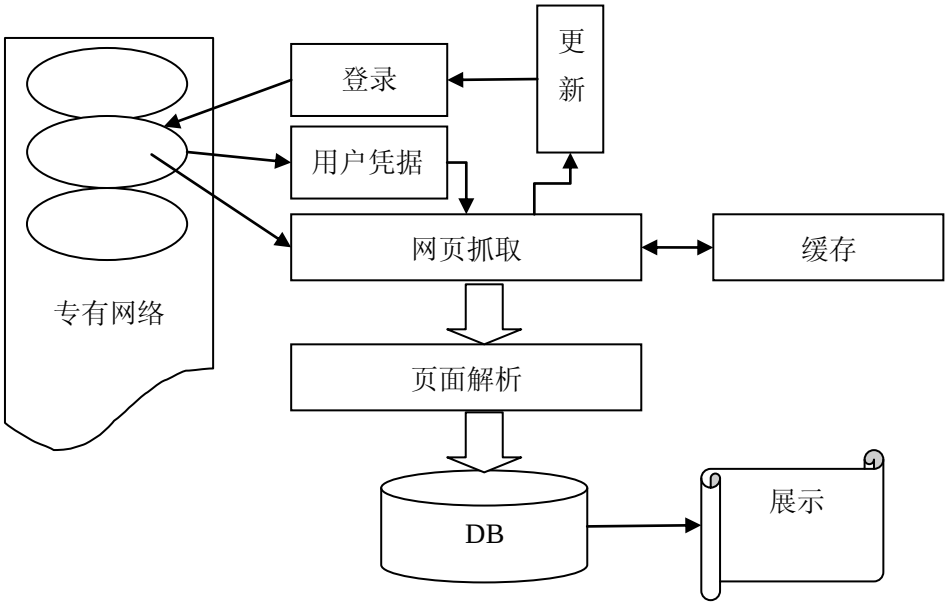

图 2-3 专有网络 Deep Web 爬虫框架

专有网络 Deep Web 数据获取主要功能包括登录模块、网页抓取模块、页面解析模块和数据展示模块。登录模块实现 Deep Web 爬虫的自动登录，并且保存凭据以供抓取模块使用，如果 Session 过期，需要重新登录。网页抓取模块主要是向服务器请求数据，抓取下来相应的网页。页面解析模块需要解析返回的网页文件，包括网页数据、XML 数据、JSON 数据等，并且解析执行页面中的 JavaScript 代码。展示模块实现 Deep Web 数据的展示，包括信息检索、实时刷新的网页、Flash 动态效果来展现等。同时，为了实现实时抓取最新数据，设计针对不同的页面使用不同间隔抓取的增量式抓取策略，以保证信息的实时性和动态性。

我们将在第三章和第四章着重介绍专有网络 Deep Web 数据获取的具体实现，并且以图 2-3 框架，设计实现了基于社交网络 Facebook、Twitter、人人网的数据获取与展示。

2.2 JAVA 无界面浏览器 HtmlUnit

HtmlUnit 是 JUnit 的开源扩展测试框架之一，实现了对 HTML 文件的建模，可以直接对 HTML 文件进行处理。HtmlUnit 是一个纯 Java 的无界面浏览器，它将 HTML 文档抽象成对象，提供了大量的 API 供调用，开发者可以很轻易的使用 HtmlUnit 模拟浏览器的功能，比如点击链接、填写并提交表单、保存 Cookies 等。HtmlUnit 一大特点就是对 JavaScript 有着非常好的支持，HtmlUnit 甚至支持完全基于Ajax的页面应用，所以 HtmlUnit 往往用来测试动态网站的页面。HtmlUnit 还可以根据配置模拟 Firefox 或者是 Internet Explorer 两种浏览器的特性[38]。

总的来说，HtmlUnit 具有以下几个基本的特点：

(1) 无界面，运行速度快；

(2) 使用 Java 开发，提供丰富的 API，扩展性好；

(3) 可移植性好，不仅跨平台而且跨浏览器，只需简单的参数配置；

(4) 由于扩展 JUnit，可以转化为性能测试，共享同一脚本，简单方便。

(5) HtmlUnit 支持 HTTP，HTTPS，COOKIE，表单的 POST 和 GET 方法；

(6) 对 JavaScript 支持较全面，支持主流的 jQuery 类库，可以轻松解析一般网页的 JavaScript 代码；

HtmlUnit 的基本原理就是将从服务器请求的返回数据以 HTML 格式建

模，用类的方式对 HTML 控件进行操作。HtmlUnit 使用 table、form 等 HTML 标识符，将测试文档作为 HTML 来处理，它也继承自 TestCase，同样需要遵循 JUnit 测试框架结构的 Java 测试程序。

HtmlUnit 封装了 Java 的 HttpClient，定义了很多对象来表示每个页面组件，比如 HtmlPage 对象对应一个页面、HtmlForm 对应 Form 表单、HtmlImg 对应页面中的 img 标签、HtmlInput 对应页面中的 input 输入域、HtmlAnchor 对应超链接 href 标签等。

工作时，首先创建一个 WebClient 对象来模拟浏览器，然后根据一个 url 创建的 URL 对象，使用 getPage()方法获取对应的页面，返回一个 HtmlPage 对象。HtmlPage 包含了很多方法来解析网页，比如 getTitleText()方法返回页面的标题、getForms()方法返回所有的表单、getAnchors()方法返回所有的超链接等。HtmlPage 对象就是包装了一个 HTML 页面，通过方法 getWebResponse().getContentAsString()就可以直接返回页面的 HTML 代码。在我们的浏览器里，一个网页中表单的提交，超链接的跳转，都是通过 click()方法即通过鼠标点击事件触发的。使用 HtmlUnit 来模拟点击动作，它们都可以从服务器返回相应的 HtmlPage 对象，然后通过 HtmlPage 对象的相应方法来解析页面。

HtmlUnit 的 JavaScript 引擎采用的是Rhinojs 引擎，能够解析并执行页面上的 JavaScript 代码，模拟 JavaScript 运行。一般说来，HtmlUnit 的这一功能对爬虫项目非常有用，可以有效的分析出页面标签，并且能够的运行页面上的 JavaScript 代码以便得到一些需要在客户端执行 JS 脚本才能得到的数据。

因此，我们在课题研究中，为了实现 Deep Web 信息资源的获取，使用了 HtmlUnit 框架模拟浏览器来实现表单的填写与提交、自动登录以及 Cookie 的保持，同时使用了 HtmlUnit 的 JavaScript 引擎，解析页面中的 JavaScript 代码，解析 JSON 数据等。

2.3 脚本语言 ActionScript

为了实现数据动态展示的效果，可以使用 Flash、Flex、Silver Light 等技术来可视化。Flash 和 Flex 都是用 ActionScript 脚本语言作为其核心编程语言，并且将代码编译成 swf 文件在 Flash Payer 中运行[39]。相对于 Flex 注重于用户应用，Flash 更注重特效的处理和动画设计，而且 Flash 编程模型是基于时间轴的，可以做出持续的交互动画效果。我们选择使用了 Flash 展示的方式，

向用户展示最新的 Deep Web 信息、社会热点信息以及热点趋势。

随着 FLASHMX2004 的推出，Macromedia 公司（后来被 Adobe 公司收购）推出了一种新的脚本语言称为 ActionScript 2.0，相应的 Flash8 是专业的 Flash 创作工具，可以制作出极富感染力的 Flash 效果和 Web 内容。随后，伴随着 Flash 9 的问世，又推出新的 ActionScript 3.0 脚本语言。ActionScript 是用来向 Flash 应用程序添加交互性的动作，这种应用程序可以是普通的 SWF 动画文件，也可以是更复杂的功能丰富的网络应用程序。如果我们需要提供用户的交互、按钮和影片剪辑等除内置于 Flash 中的对象之外的其它对象或者能提升用户体验的 Flash 特效，则需要使用 ActionScript 脚本语言。

Flash 中使用 ActionScript 脚本语言给动画增加与用户的交互性。在简单 Flash 动画里，会按照时间轴的顺序来播放 Flash 中的每一帧的场景，而在交互 Flash 动画中，用户可以使用键盘或鼠标与动画交互。例如，可以单击某一按钮，弹出一个网页或者跳转到动画的其他部分；可以移动动画中的对象元素；可以在动画的表单中填写信息等等。总之，使用 ActionScript 脚本语言可以控制 Flash 动画中的对象，添加与用户交互动作，实现 Flash 动画特效，扩展 Flash 网络应用的用户体验。

ActionScript 语言类似于 JavaScript 语言，是针对 Flash Player 运行时环境的编程语言，它在 Flash 文件中和一些应用程序中实现了与用户交互性、数据处理以及其它许多特效和动作。ActionScript 脚本语言是由 Flash Player 中的 ActionScript 虚拟机 (AVM，类似于 Java 虚拟机) 来执行的，ActionScript 代码通常被编译器编译成字节码格式，字节码嵌入到 swf 文件中，swf 文件由 Flash Player 执行。ActionScript 2.0 对应的虚拟机为 AVM1，基于 ECMA-262 V3（相当于 JavaScript 1.5）。ActionScript 3.0 对应的虚拟机为 AVM2，基于 ECMA-262 V4（相当于 JavaScript 2.0）。

ActionScript 是一种面向对象的编程语言，它的核心是动作对象（Action），同时提供了完成其他功能的能力，比如创建类和对象、触发事件（Events）等，非常便于开发人员学习和使用。ActionScript 也支持解析 XML 文件，可以使用 XML 配置文件来改变 Flash 动画的显示效果。

2.4 本章小结

本章我们介绍了 Deep Web 爬虫的基本工作原理，以及 Deep Web 数据获取的框架和各部分功能。Deep Web 信息资源主要包含两个方面，一是需要通

过查询接口查询才能由服务器动态生成并返回的 Web 信息资源。二是没有只针对注册用户开放的网络信息，只有登录后才可查看的专有网络信息。针对这两种信息资源，设计了不同的数据抓取策略来进行。

我们给出了 Deep Web 数据获取系统的框架以及每一部分功能的介绍，提出了设计 Deep Web 数据系统的各个模块，包括资源分析模块、数据抓取模块、数据处理模块、用户接口模块等。另外，着重介绍了针对专有网络的 Deep Web 数据获取设计方案以及设计框架。

同时，介绍了本课题所涉及到的 HtmlUnit 开源框架和 ActionScript 语言。HtmlUnit 框架通过对 Html 网页进行建模，以模拟浏览器功能。ActionScript 是一种面向对象的脚本语言，用来实现 Flash 效果的动画、交互、网络应用功能。

我们将在后续章节介绍 Deep Web 数据获取与展示的具体实现。

第3章 社交网络数据获取技术

本章我们介绍 Deep Web 数据获取的技术以及设计方案，同时提出我们的针对社交网络的爬虫系统，完成系统的框架与三个网络爬虫的详细设计与开发。最后给出系统的功能测试和性能测试，并对实验结果进行分析。

3.1 相关研究

如今，随着互联网的发展，人们在网络上的人际交往越来越频繁，互联网企业推出了越来越多的社交工具，从最初的电子邮件、聊天室到现在的即时通讯工具，从之前简单的沟通到现在的实时在线聊天，从早期的论坛、BBS 发展到现在的博客、微博、社交网站等。越来越多的网络应用占据了人们的生活，社交网站经过多年的积累已经具有相当大的规模并且还在急剧膨胀，已经成为继搜索引擎和电子商务之后，互联网新的一个发展方向。最新调查显示，Facebook 在全世界范围内拥有超过 7 亿的注册用户，每天有上亿的网民活跃在其中，发表日志，分享新鲜事等，而另一大热门网站 Twitter 的注册用户数也超过了 2 亿。

社交网站是一种专有网络，他们的信息资源在 Deep Web 中占了很大的一部分。社交网站的信息往往具有规模效应和群体倾向性，当一个热门话题开始出现时候可能在社交网站上被广泛讨论和迅速传播，如果能获取社交网站上这些数据，对于分析舆论热点和热点趋势有很大的帮助。社交网站上集中了最活跃的网络用户，所以也是网络言论最集中的地方，挖掘其信息资源，将能获取到舆情分析所需要的极有价值的原始数据，以便相关的使用者查看和分析[40]。

如今的大公司企业都在收集用户对自己产品的评论信息，以能够及时地对产品做出相应调整，保持市场活力。目前已经有很多优秀的舆情分析工具，可以分析网络上各种信息，通过数据挖掘的技术分析出客户的喜好和潜在的购买力等。那么如何获取这些信息，就显得至关重要。因此，针对社交网站上的数据获取与分析，对于各公司的发展也有重要的意义。2010 年，《哈佛商业评论》针对 2000 多家企业进行了的调研，主要包括企业对社交网络媒体的使用以及评估。从中我们看出，许多企业在社交网络媒体使用上依然处在的起步阶段，而中国的企业目前更是未现端倪。调研发现有以下结果：

(1) 75%的企业不知道最有价值的用户在社交网络上讨论他们；

(2) 31%的企业对社交网络媒体使用效果未作评估；

(3) 不足 25%的企业使用社交网络分析工具；

(4) 有 12%的企业可以游刃有余地使用社交网络媒体；

(5) 只有 7%的企业将社交网络与自己的营销手段相结合，从战略和运营角度应用社交网络；

(6) 平均每个企业使用三种社交网络平台；

(7) 25%表示不打算使用社交网络媒体。

由此可见，许多企业已经认识到社交网络对企业的冲击，并开始着手做一些尝试，希望能够尽快进入社交网络媒体，并且从中获益。如果我们能够向企业提供有关该企业的 Deep Web 数据，企业就能够分析出所投放广告的效益、用户对企业产品的评价、企业的影响力等，这对企业的发展将会有很大的帮助。

社交网站最重要的两点是以人为本和关系网络化。我们的 Deep Web 爬虫从社交网站上抓取下来的数据可以用来分析社会关系网络，根据分析出的用户的好友关系以及相同的兴趣爱好来构建社会实体之间的关联，还可以根据一些热点信息的关注度来分析舆论领军人物，根据热点人物的言论分析其情感倾向等[41]。

为了抓取这些 Deep Web 数据，我们需要使用用户名和口令登录，深入到网站的内部，通过关注大量的用户来抓取尽可能多的数据。当然，由于某些用户隐私设置，我们无法抓取到所有的用户数据，但是对于那些属性为 public 的用户，比如公共主页、公众微博等，我们都能够在登录后直接访问。往往这些人也是名人或者代表企事业单位，针对这些用户的数据获取基本能够满足我们分析舆论热点的需求。有些社交网站为了与其他网站相连结，提供了详尽的 API 接口供调用，这给我们的 Deep Web 爬虫抓取数据提供了很大的便利。当然，为了保证安全性，网站基于 OAuth 认证来验证用户，同样不能避免登录问题[42]。有关 OAuth 认证协议，我们将在第 3.3 节详细叙述。

3.2 设计方案

3.2.1 OAuth 认证

为了获取这些专有网络的数据，首先就需要登录到网站，并且通过保持凭据的来完成对信息资源的抓取。而需要抓取的 Facebook、Twitter 和人人网都

采用 OAuth 认证来对第三方应用程序进行权限控制，因此我们有必要介绍一下 OAuth 认证的相关知识。

OAuth 是由 Blaine Cook、Chris Messina、Larry Halff 及 David Recordon 共同发起的，OAuth 协议目的在于为第三方应用程序的授权提供了一个简单、安全、开放的标准。OAuth 不会让第三方的应用接触到用户的账号信息，也就是说第三方的应用不需使用用户名与密码就可以获得网站对该用户资源的授权。同时，任何一个服务商都可以实现自己的 OAuth 认证，任何第三方也都可以使用这个 OAuth 服务。很多公司都提供 OAuth 认证多语言开发包，不管是用户还是开发者使用都非常方便。

OAuth 认证的过程如下图 3-1 所示。

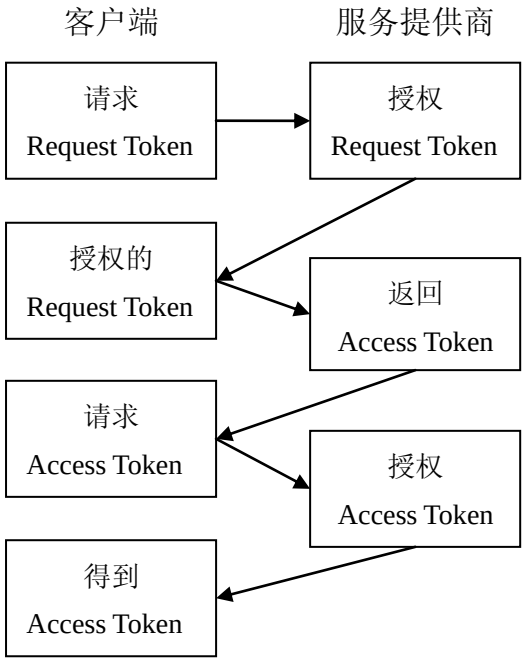

图 3-1 OAuth 认证过程

这个过程的详细步骤描述包括：

(1) 第三方向 OAuth 服务商请求未授权的 Request Token，请求头要附加参数；

(2) OAuth 服务商同意第三方的请求，授权 Request Token 并返回未授权的 OAuth Token 和密钥；

(3) 第三方向 OAuth 服务商请求授权的 Request Token，请求头附加未授权

的 OAuth Token 和密钥；

(4) OAuth 服务商返回未授权的 Access Token；

(5) 第三方向 OAuth 服务商请求未授权的 Access Token，并附加 Request Token；

(6) OAuth 服务商授权 Access Token，并返回密钥；

(7) 第三方就可以使用这个 Access Token 获得该用户的资源。

由于 OAuth 认证的简单安全，越来越多的互联网服务商提供了 API 访问接口，同时使用 OAuth 认证来向用户授权。开放 API 是 SaaS（Software as a Service，软件即服务）模式下的一种应用，互联网服务商将自己的网站封装成 API 接口供第三方调用。网站提供开放 API，可以吸引一些开发人员在这个平台上开发应用。网站可以借此获得更多的流量与关注度，第三方开发者则用很小的成本完成有价值的应用，是一个双赢的局面。Twitter 使用了开放 API 来实现自己网站的 API 开放性，初期使用 HTTP Basic Authentication 验证机制。当使用 HTTP Basic Authentication 时，只需要使用在 Twitter 注册的“用户名”作为 Session 或 Cookie 的“用户名”部分的内容即可。但是，在 2010 年 6 月 30 日，Twitter 宣布终止对 HTTP Basic Authentication 认证的支持，转而全面启用 OAuth 认证方式，这说明 OAuth 认证的强大魅力。Twitter 的 OAuth 认证流程与基本的认证流程一致，只需要得到一个 Access Token，在向 Twitter API 需要验证身份的每个 HTTP 请求中，附加上 Access Token 即可。Facebook 的 Graph API 使用 OAuth2.0 作为权限控制，它是一个简化版的 OAuth，使用 SSL 协议作为 API 的通信，而不需要复杂的 Access Token 交换。获取 Facebook 的 Access Token 很简单，只需要在使用用户名密码登录后抓取链接 http://developers.facebook.com/docs/api 对应的页面就能通过对页面的分析获得当前的 Access Token。人人网同样使用 OAuth2.0 来进行验证，但是人人 API 使用独有的 Session Key 机制来认证应用和用户，所以调用人人 API 前，需要通过 OAuth Token 获取 Session Key。

3.2.2 基本设计方案

针对专有网络的 Deep Web 数据获取其流程大致相同，都包括有登录模块、网页抓取模块、页面解析模块和数据展示模块四大部分，不同之处就在于一些抓取细节比如登录方法、缓存、更新周期等需要针对不同的网站设计不同的策略。

我们的 Deep Web 爬虫基本方案是，登录模块使用 HtmlUnit 创建一个 WebClient 对象，用来模拟浏览器。然后通过该网站的入口 URL，得到一个 HtmlPage 页面对象。提取出页面中的所有的表单，然后分析出登录表单。通过表单的 HtmlTextInput 对象，填写用户名，通过表单的 HtmlPasswordInput 对象填写密码，通过表单的 HtmlSubmitInput 提取出登录按钮，最后执行 click()方法，模拟登录。根据登录响应判断是否登录成功，如果不成功则过一段时间之后重新登录。

登录成功之后，我们就可以通过 WebClient 来抓取网页。由于 WebClient 具有浏览器的功能，因此我们可以把 Cookie 附加到 WebClient 里，这样每次向服务器发送 HTTP 请求时都在协议头里附加用户凭据，以通过服务器验证。如果用户凭据过期，还需要重新登录。根据 3.2.1 的分析，在调用 Facebook 和 Twitter API 时，我们使用 OAuth 协议来验证身份，每次请求时通过一个 Access Token 来以获得用户资源。通过这些控制，就实现了用户 Session 的保持。

抓取页面时，可以通过向服务器发送 HTTP 请求来实现。首先根据爬虫待抓列表里的 url 建立一个 URL 对象，然后使用 openConnection()方法打开一个与服务器的 HTTPUrlConnection，然后通过这个连接建立一个输入流，从服务器中读取返回的数据，最后关闭连接。也可以直接使用 WebClient 的来实现，直接对 URL 获得一个 HtmlPage 对象，但是效率较低。

页面解析模块包括对返回的页面提取数据，解析 JavaScript 代码等。如果返回的页面是普通网页文件，我们首先要得到提取出文本块，然后分析文本块的属性，统一成数据格式存储。如果网页文件中某一块包含 JavaScript，我们需要调用 JavaScript 解析器来执行这段代码，然后再提取数据。Java 提供了完整的解析 JavaScript 的包，使用时只需要创建一个 JavaScript 引擎，然后直接执行脚本公式，就能得到数据结果。如果页面返回的是 XML 文件，我们使用 JDOM 来解析 XML，提取其中的数据信息。如果返回的页面是 JSON 数据，我们需要使用 JSON 解析包，创建 JSON 对象来实现 JSON 数据解析。把数据形成统一的格式，然后存入数据库。

入库方面，实现了 Web Service 接口，客户端是 Deep Web 爬虫，服务器端与数据库在同一台机器上。爬虫首先把数据通过 Web Service 接口传输到服务器上，然后服务器通过 Hibernate 持久化解析这样就避免了 Deep Web 爬虫直接对数据库进行操作，防止不安全发生。

基本的设计方案如下图 3-2 所示。

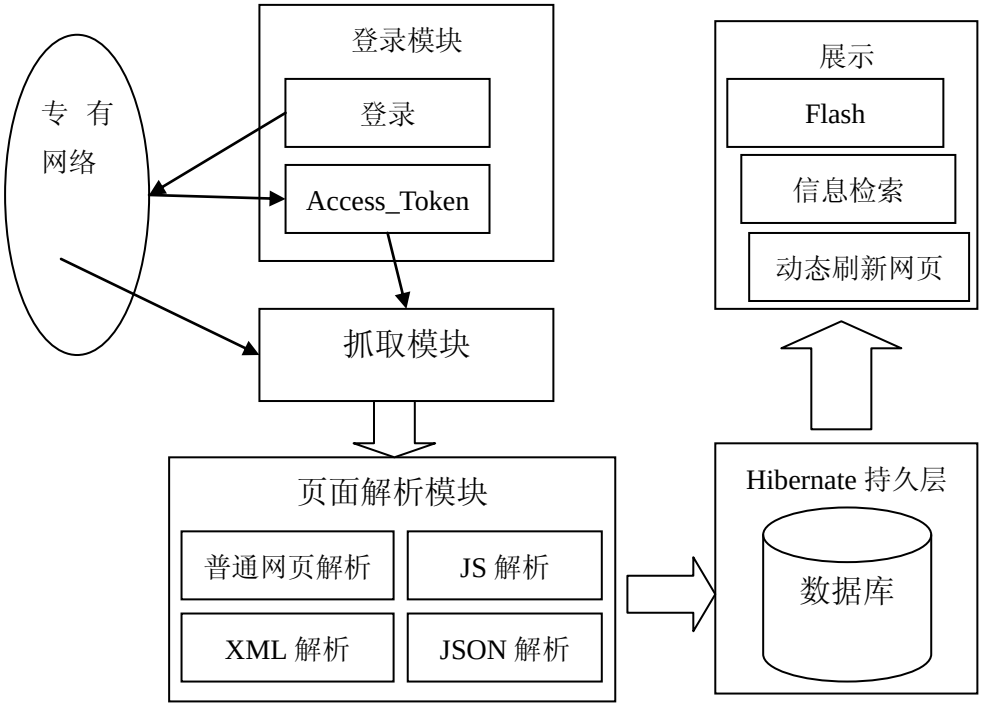

图 3-2 Deep Web 数据获取基本设计方案

Web Service 是基于 SOAP 协议（简单对象访问协议）实现的，可以实现跨平台跨域的数据交换。Web Service 的优点就是实现在线的服务，用户可以在线地使用这种服务，而且可以避免服务器暴露过多数据逻辑。SOAP 协议提供了标准的 RPC 方法来调用 Web service，客户端通过服务器提供的 WSDL 语言（Web Service 描述语言）来得到用于描述其 Web service 函数、参数和返回值。在我们的实现中，服务器实现了 Web Service 接口，提供带参数的数据传输的方法，客户端通过这个方法把数据传输到服务器端，由服务器端入库。在 Eclipse 中，已经提供了生成 Axis2 的 Web Service 服务端和客户端的方法，可以直接生成。

Hibernate 是一个数据库持久层框架，它把数据库的表封装成对象，通过对象操作来实现数据库的持久化存取。Hibernate 有五个核心接口：Session、SessionFactory、Transaction、Query 和 Configuration。通过这些接口，不仅可以对持久化对象进行存取，还能够进行事务控制。Session 接口负责数据库的增加、删除、查询和更新操作；SessionFactory 是一个 Session 工厂，负责初始化 Hibernate，并创建 Session 连接；Configuration 接口负责配置并启动

Hibernate，创建 SessionFactory 对象；Transaction 接口负责事务处理的相关操作；Query 和 Criteria 接口负责执行各种 HQL（Hibernate 查询语言）和 SQL 查询。 一般情况下，存储数据的流程是：首先从 Hibernate 的 SeeionFactory 中获得一个 Session，把数据保存到表所对应的对象中，然后使用 Session 接口的 save 方法把该对象存储到服务器，最后关闭 Session。

3.3 爬虫详细设计

本课题实现了三个 Deep Web 爬虫，分别是针对 Twitter、Facebook 和人人网，采用了相同的架构设计，但是在详细设计方面做了不同的处理。以下我们对这三个爬虫作详细的阐述。

3.3.1 Twitter 信息抓取

为了抓取 Twitter 信息资源，我们需要抓取每个感兴趣的用户主页上的每条 Twitter 信息。但是，Twitter 界面设计比较复杂，直接访问 Twitter 来抓取的话，只能得到 Twitter 消息的用户名和消息内容，无法得到 Twitter ID、发布时间、评论和转发人数等其他数据，机器自动处理这些数据很难保证信息的准确性[43]。而且，有些信息资源需要登录到 Twitter 才能查看，这对于爬虫来说又是一个难题。幸好，Twitter 提供了 openAPI 供调用，我们只需通过 OAuth 认证获取 Access Token，方便地使用 API 来获取相关数据。

Twitter 爬虫系统抓取策略是，首先从 Twitter 里获得认证授权的 Access Token，然后对跟踪列表里的用户调用 Twitter 的 API 分别进行抓取。如果该用户为公开用户，则可以直接获取到其 Twitter 数据；如果该用户为私密用户，则在 API 链接中附加 Access Token 来获得授权，然后抓取其 Twitter 数据。抓取时页面请求返回的是 XML 页面，需要解析该页面，首先判断当前一条 Twitter 消息的是否已抓，方法就是解析出当前的 Twitter ID，与已抓 ID 进行比对，如果已抓则放弃，否则解析该条 Twitter 的其他信息。解析数据完成之后，把数据保存在本地，同时连接服务器的 Web Service 接口，使用 Hibernate 入库。爬虫为每个跟踪的用户设置了一个定时器，按照时间段循环抓取最新的数据。

Twitter 抓取的流程图如下图 3-3 所示。

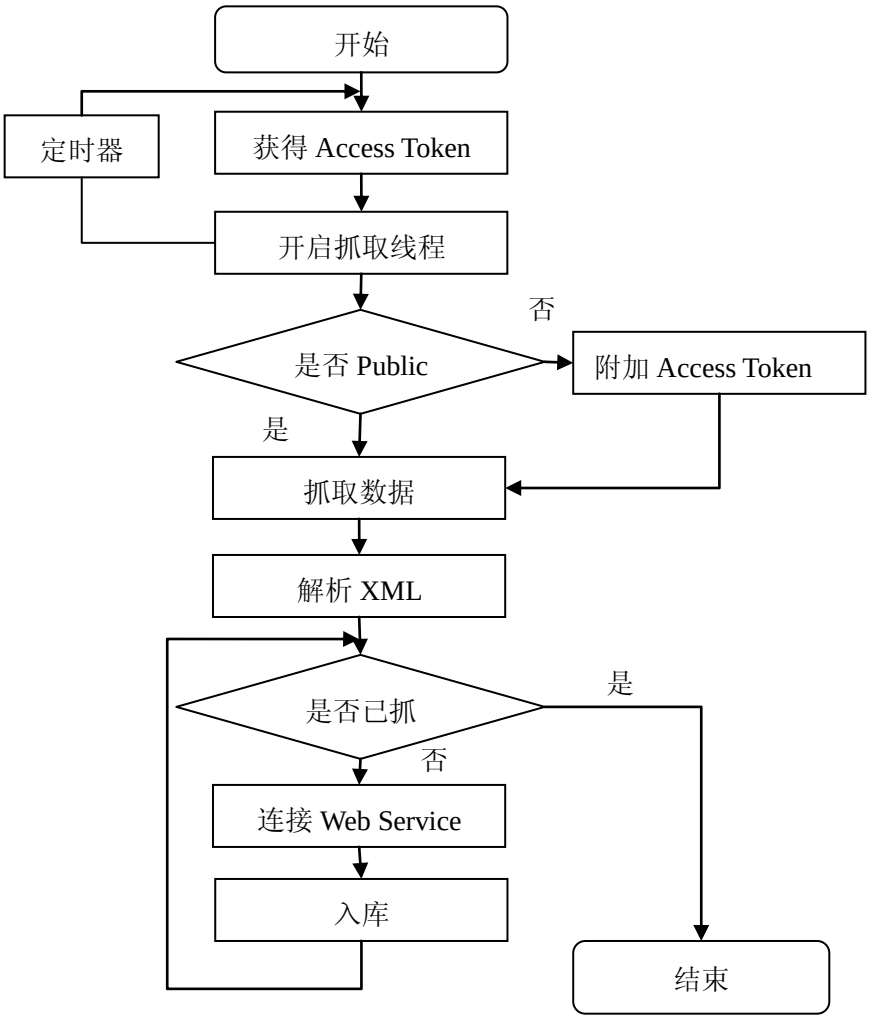

图 3-3 Twitter 爬虫流程图

为了实现 Twitter 的 openAPI 调用，我们首先要通过 Twitter 的 OAuth 认证，获取一个授权信息。获取 OAuth 认证授权信息的过程是，首先根据自己得到的 Consumer key 和 Consumer secret，放入 HTTP 请求中发送到 Twitter API 的链接地址 https://api.twitter.com/oauth/request_token，就能得到一个 Request Token，然后再发送 HTTP 请求附加上刚刚获得的 Request Token 到链接 https://api.twitter.com/oauth/authorize?oauth_token=<Request Token>，就会被重定向到一个附加 Verifier 参数和新的 Token 的 URL。得到 Verifier 和新的 Token

后，再向 Twitter 的 API 发送附加了新的 Token 的 HTTP 请求，链接是 https://api.twitter.com/oauth/access_token，就能获得一个 Access Token。然后我们在每次需要的时候，把这个 Access Token 附加到每一个 Twitter API 的 HTTP 请求中，就能够抓取需要的 Twitter 数据。每次在定时器开启抓取线程的时候，只要获得一个 Access Token 就行，不需要重复去 Twitter 获取授权。有的认证用户设置属性为 public，能够随意抓取其数据，不需要附加 Access Token；有的用户设置为私密，需要附加 Access Token 才能抓取其数据。所以抓取时候要判断页面返回的是否为 401 Not Authorized，然后再决定附加 Access Token 到 API 里。

抓取数据时，首先创建一个 HttpURLConnection 对象，然后向 Twitter 的 API：http://api.twitter.com/statuses/user_timeline.xml?<screen_name>&<count>打开一个连接，再从连接中打开一个输入流，从输入流里把数据读取出来。其中两个参数 screen_name 代表需要抓取用户的 ID，count 代表需要抓取的条数。连接返回一个 XML 页面。这个返回的结果是该用户前 count 数的最新 Twitter 消息，并且以时间顺序排列。

解析 XML 文档时，首先从本地读取已抓 Twitter 的 ID 放入一个 HashMap 中，其键值对的 key 代表已抓的 ID，value 代表是否抓。然后按顺序解析 XML 文档的每一条 Twitter ID，如果已经存在于 HashMap，则停止解析，否则，继续解析该条 Twitter 的其他数据，包括用户账号、用户名、发表时间、微博正文和转发数。转发数主要用来判断该条 Twitter 的热度，以决定这条信息的重要程度。解析完毕的数据分别存储在本地磁盘和数据库里。

为了保证数据库的安全，避免爬虫直接操作数据库，我们实现了 Web Service 方式把数据入库。首先，应用服务器实现了 Web Service 接口，提供了 WSDL（Web Service 描述语言），客户端得到用于描述其 Web service 函数、参数和返回值，就可以把数据通过参数传输到应用服务器的 Web Service 接口。服务器端在接收到数据之后，使用 Hibernate 持久层，把数据存储到数据库里。

使用 Hibernate 持久化框架主要是通过对象的方式来封装数据库接口，保证数据库 Session 连接和数据库存取的持久化。我们首先要创建一个类，以对应数据库中的 Twitter 数据表，类中的每一个全局变量代表着表里的一个属性，通过 set 和 get 方法来设置变量的值。然后给每一个类创建一个配置文件，完成数据库中表和类的映射，需要指明表的主键。然后还需要定义一个 Session 工厂类，来提供 Session，封装与数据库的连接，完成向数据库的存储

和查询等操作。数据入库流程是：首先从 Hibernate 的 SeeionFactory 中获得一个 Session，把数据保存到表所对应的对象中，然后使用 Session 接口的 save 方法把该对象存储到服务器，最后关闭 Session。

我们为 Twitter 的爬虫设计了一个界面，界面中提供了增加删除用户、立即更新等功能，同时打印抓取的提示信息。在界面里点击增加用户时候，需要填写上用户名和抓取该用户的间隔时间，以实现增量式更新，同时为该用户添加一个定时器。当开启爬虫的时候，将使用定时器每隔设定的间隔时间抓取该用户的数据，以保持所抓取的数据的实时性。

为了保证抓取性能，使用了增量式抓取模式。每次首先从本地读出已抓的 Twitter 编号，然后在解析前先对 Twitter 编号进行比对再解析数据，避免了资源的浪费。经过大量的测试，Twitter 限制调用 API 为的频率为一小时 150 次，因此在抓取用户数量多的时候，增大每个用户抓取的时间间隔，以保证数据的及时被抓取下来。另外，Twitter API 返回的一定数量的该用户最新的 Twitter 消息，其参数 count 如果设置过大，将会增大爬虫负担，每次都需要抓取大量的数据然后再与已抓队列比对，导致性能下降；如果参数 count 设置过小，将有可能会漏过那些发言频率较高的用户的某些数据，即该用户有可能在这个时间间隔内发布了超过 count 值的 Twitter 消息，导致所抓取数据不能完全覆盖。因此，需要根据设置的时间间隔和用户的活跃程度适当调整参数，以兼顾性能和完备性的需求。

Twitter 爬虫的关键技术包括获得 Access Token、时间间隔和抓取条数参数调整、Web Service 技术、Hibernate 入库等方面。

3.3.2 Facebook 信息抓取

为了获取 Facebook 的信息资源，我们需要访问感兴趣的用户的涂鸦墙（Wall），上面包含了该用户的状态、日志、评论、相册等全部信息。Facebook 的前端用户界面做得非常美观，使用了大量的新 UI 技术和 JavaScript 脚本技术，可以带来丰富用户体验和特效。用户登录到 Facebook 之后查看页面源代码就会发现 Facebook 界面的主要部分都需要使用脚本代码在浏览器端执行，包括涂鸦墙的信息。但是，这些前端展示技术的服务对象是人，而爬虫在处理这些非结构化数据和脚本代码有许多困难，随着 Facebook 页面布局和页面脚本技术的不断更新和变化，爬虫直接获取 Facebook 的数据手段很难保证信息的准确性。而且，需要登录到 Facebook 才能查看，这对于爬虫来说又

是一个难题[44]。

同样地，Facebook 也为第三方应用程序提供了丰富的 API 接口，即 Facebook Graph API。我们调用 Facebook Graph API 时候，网站将会返回一个 JSON(JavaScript Object Notation)格式的数据，包含了每条消息的 ID、发布时间、作者、摘要、内容等信息。JSON 是一种轻量级的数据交换格式，它是基于 JavaScript 标准的一个子集，适合服务器与客户端的交互，也易于解析和生成。但是，Facebook 中用户属性只有少部分设置为 public，可以随意访问和获取，大部分用户属性为 private，需要通过授权才能够查看其信息。前文提到过，Facebook 使用简单的 OAuth2.0 认证，只需要提供正确的 Access Token 就能通过授权认证，然后访问私人用户涂鸦墙抓取数据。

Facebook 的抓取策略是，首先登录 Facebook，从中获得一个认证授权的 Access Token，然后对待抓取列表中的用户使用 Facebook 的 Graph API 进行抓取其涂鸦墙的新鲜事，即在 API 链接中附加 Access Token 字段来向服务器发送 HTTP 请求，无论该用户如果该用户是公开用户还是为私密用户。抓取时页面请求返回的是 JSON 页面，需要解析 JSON 数据，首先判断当前一条新鲜事是否已抓，方法就是从 JSON 中解析出当前的 ID，与已抓 ID 进行比对，如果已抓则放弃，否则解析该条新鲜事的其他信息。因为 Facebook 新鲜事包括留言、日志、状态、分享链接、分享视频等各种类型，需要把数据整合成统一的格式。然后把数据保存在本地，同时连接服务器的 Web Service 接口，使用 Hibernate 入库。爬虫也为每个跟踪的用户设置了一个定时器，按照时间段循环抓取最新的数据。

Facebook 抓取的流程图如下图 3-4 所示。

为了实现 Facebook Graph API 的调用，需要获取 Access Token，然后在调用 API 链接的时候，附加上这个授权信息。获取 Facebook 的 Access Token 有两种方法，第一种是的获取授权许可的 OAuth 认证流程，主要面向在 Facebook 上发布应用的用户，第二种是通过登录在 Facebook 页面里获取 Access Token，这种方法比较简单。第一种方法的流程类似于我们在 3.2.1 节介绍的 OAuth 认证流程，这里不再赘述。

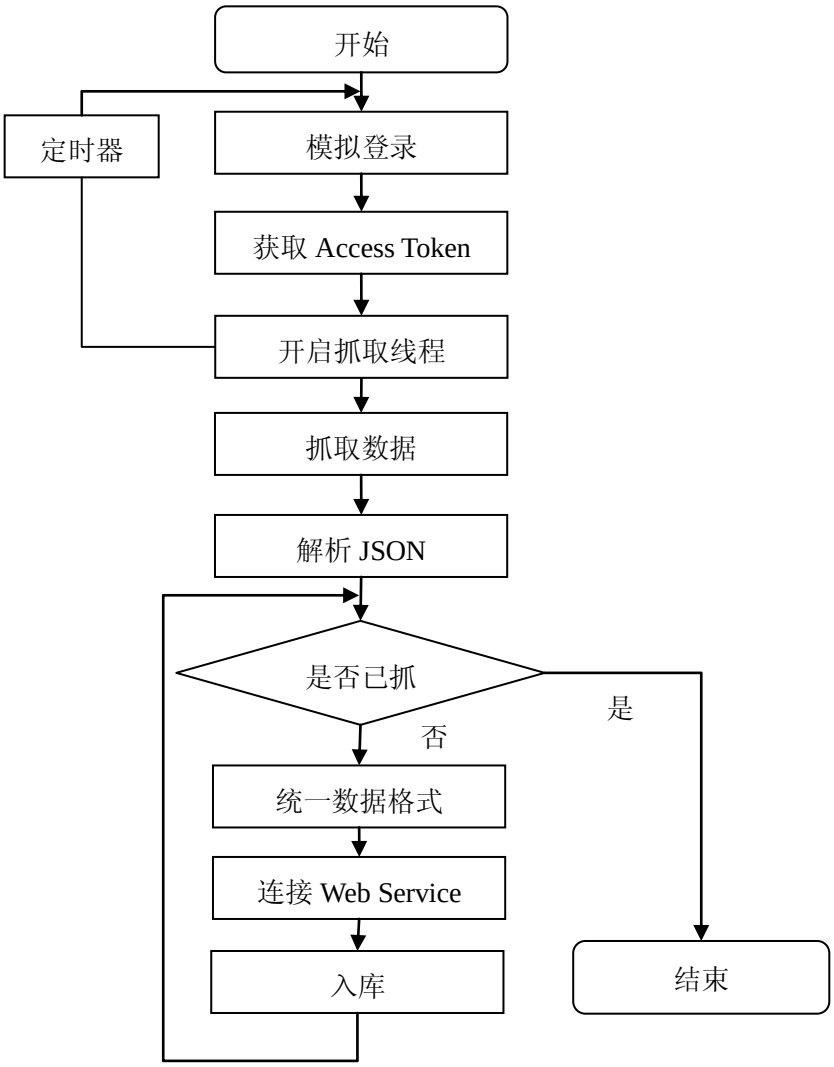

图 3-4 Facebook 爬虫流程图

我们采用直接的第二种方法来获取 Access Token。首先，使用 HtmlUnit 开源框架创建一个 WebClient 来模拟浏览器，然后使用 WebClient 的 getPage()方法，得到一个 HtmlPage，进入登录界面。使用 HtmlPage 的 getForms()方法提取界面中的表单元素，找到登录表单之后，填写上用户名域 HtmlTextInput 和密码域 HtmlPasswordInput，然后使用按钮 HtmlSubmitInput，执行其 click()方法，返回登录之后的页面。再用 WebClient 浏览器访问 http://developers.facebook.com/docs/reference/api/，从页面中解析出当前的

Access Token。拥有这个 Access Token 之后，我们就能随心所欲地抓取所有的用户涂鸦墙上的公开信息。Access Token 获得之后，继续使用 WebClient 不需要重新获取，但是 Access Token 有一个短暂的有效期，过期了就需要重新登录再获得一个新的 Access Token。

抓取数据时，使用 HtmlUnit 的 WebClient 浏览器向 Facebook 发送 HTTP 请求，请求链接是 https://graph.facebook.com/<userID>/feed?<access_token>，返回一个 JavaScript 页面，即是我们需要的该用户的 Facebook 信息资源，包含留言、分享等。这个 API 中包含两个参数，userID 即是待抓的用户 ID，access_token 就是刚刚获得的授权信息。其返回的页面是该用户的涂鸦墙上的新鲜事最新一页的数据信息，而且包含获得下一页信息的访问链接地址。返回 JSON 数据包含每一个新鲜事的详细描述，并且按照时间顺序排列。

解析 JSON 数据有两个工具，org.json 包和 json-lib 包，这两种方法都比较简单，使用方法也类似，但是 org.json 相对于 json-lib 更轻量一些，没有任何依赖任何其他组件，而 json-lib 需要依赖其它组件。我们采用了 org.json 包来处理 JSON 数据，首先就是创建一个 JSONObject 对象，然后根据每一个元素的关键字来判断是否有该数据，并且解析出该数据。同样地，首先从本地读取已抓 Facebook 新鲜事的 ID 放入一个 HashMap 中，其键值对的 key 代表已抓的 ID，value 代表是否抓。然后按顺序解析 JSON 页面的每一条新鲜事的 ID，如果已经存在于 HashMap，则说明后面的新鲜事已经抓取完毕，停止解析，否则，继续解析该条新鲜事的其他数据。由于 Facebook 每条新鲜事有多种类型，大致统计了一下，包括个人状态、好友留言、分享链接、分享视频、日志等，其格式各不相同。为了方便数据库存储和数据展示，我们把数据格式统一为 ID、用户账号、作者、发表时间、标题、描述和正文。然后把数据分别存储在本地磁盘和数据库里。

同样地，与 Twitter 数据入库方式类似，为了保证数据库的安全，避免爬虫直接操作数据库，我们也通过 Web Service 方式把数据传输到应用服务器上。服务器端在接收到数据之后，也使用了 Hibernate 持久层，把数据存储到数据库里。我们首先要创建一个类，以对应数据库中的 Facebook 数据表，类中的每一个变量代表着数据库 Facebook 的数据表里的一个属性，通过 set 和 get 方法来设置和读取变量的值。然后给这个类创建一个配置文件，完成数据库中表的属性和类的变量映射。Facebook 通过 Hibernate 入库方法类似于 Twitter 数据入库，这里不再详细阐述。

Facebook 爬虫添加用户时候，同样需要填写上用户名和抓取该用户的间隔

时间，以实现增量式更新，添加用户的同时为该用户添加一个定时器。当开启爬虫的时候，将使用定时器设定的间隔时间抓取该用户的数据，以保持所抓取的数据的实时性。为了保证抓取性能，使用了增量式抓取模式。每次首先从本地读出已抓的新鲜事编号，然后在解析前先对新鲜事编号进行比对再解析数据，避免了资源的浪费。Facebook 支持 FQL(Facebook Query Language)语言，类似于 SQL 语言，可以直接调用 API 执行数据库查询，能够提高 API 的执行效率和网络利用率。

Facebook 爬虫的关键技术包括登录、获得 Access Token、解析 JSON 数据、Web Service 技术、Hibernate 入库等方面。

3.3.3 人人网信息抓取

为了抓取人人网的数据资源，同样需要登录到而人人网，获得访问权限，抓取有价值的数据信息。人人网为第三方应用程序提供了丰富的 API 供调用，允许被人人网授予权限的第三方应用程序以用户的身份获取人人网的资源，比如好友基本资料、好友关系、日志相册等。2011 年 1 月下旬，人人网开放了 OAuth2.0 认证授权，允许桌面应用程序接入人人网。人人网获取授权有三种方式，包括用户页面授权方式、用户名密码授权方式和令牌刷新方式。用户页面授权方式需要登录到指定页面，然后选择同意授权就会获得授权服务器的 Authorization Code。每一个 Authorization Code 的有效期为一个小时，并且只能使用一次，过期或者再次使用需要重新申请。通过用户名和密码获取 Access Token 方式是为了方便缺乏浏览器支持的应用，OAuth2.0 认证增加了 Resource Owner Password Credentials Flow ，使得应用程序使用用户名和密码可以直接获取 Access Token，但是不方便之处就是必须首先向人人网提出申请并签署相关协议。令牌刷新方式是已申请到 Refresh Token 使用权限的第三方应用，在获取 Access Token 时，认证服务器将返回相应的 Refresh Token。

但是，人人网仅仅对基本 API 开放了访问的权限，高级 API 还需要申请开通。基本的 API 包括好友关系、通知、公共主页、状态等信息，高级 API 才包括我们需要的状态、日志、新鲜事等。这大大限制了我们的爬虫需要抓取的人人网信息资源，无法获得真正有价值的数据。人人网提供了一个高级 API 的测试方案，但是只能添加 5 个好友，不能满足需求。鉴于此，我们将不考虑使用人人网的 API，转而采用登录之后，分析待抓用户的个人主页数据进行抓取的策略。自动登录采用 HtmlUnit 模拟浏览器来完成，分析页面将对页面里的文

本、链接和 JavaScript 代码提取并且解析。人人网在 JavaScript 代码比较简单，易于解析，而我们仅仅关注状态、日志，只需要解析查看所有评论和查看全部新鲜事的脚本代码。我也需要抓取其他附加信息，比如全部学校学院列表等，但这些信息抓取下来之后可以一劳永逸。同样地，所抓取的信息需要整合成统一的格式。然后把数据保存在本地，同时连接服务器的 Web Service 接口，使用 Hibernate 入库。爬虫也为每个跟踪的用户设置了一个定时器，按照时间段循环抓取最新的数据。

人人网抓取爬虫的流程图如图 3-5 所示。

人人网抓取爬虫与 Twitter 和 Facebook 不同，它不采用 API 调用的方式来抓取数据，而是直接登录再解析页面。人人网爬虫登录模块首先调用 HtmlUnit 开源框架创建一个浏览器 WebClient，然后对人人网登录页面 URL 使用 WebClient 的 getPage()方法，得到一个登录页面的 HtmlPage 对象。然后调用 getForms()方法提取界面中的表单元素，找到登录表单之后，填写上用户名域 HtmlTextInput 和密码域 HtmlPasswordInput，然后使用按钮 HtmlSubmitInput，执行其 click()方法，返回登录后的首页。这样，就成功登录到人人网，并且把 Cookie 保存下来，每次向服务器发送 HTTP 请求的时候，都在请求头中附加上这个 Cookie，就能保证每次都有抓取权限。

抓取页面时，直接使用 WebClient 获取每一个待抓用户的主页 URL。首先向待抓用户的个人主页发送请求，判断该页面是否为公共主页，如果是个人主页，再判断其是否能够抓取。如果该个人主页是好友，可以直接抓取；如果不是好友的话，有的设置了好友才能查看，将无法获取其页面。然后返回的动态页面包含了该用户的最新新鲜事，要从中提取状态、日志以及其留言。通过标签 stats，可以提取出其 ID 和内容，然后判断是否为日志，如果为日志则提取日志链接，抓取该日志正文。在抓取时候，首先要从本地读取已抓保存到一个 HashMap 里面，然后每次都首先判断新鲜事的 ID 是否为已抓，如果以及抓取，则停止解析。我们把数据格式统一为新鲜事 ID、作者、类型、内容、正文。如果是状态，则正文与内容一致，都是状态的内容；如果是日志，则内容为日志标题，正文是日志的正文。然后把数据分别存储在本地磁盘和数据库里。

同样地，与 Twitter 和 Facebook 数据入库方式类似，为了保证数据库的安全，避免爬虫直接操作数据库，我们也通过 Web Service 方式把数据传输到应用服务器上。服务器端在接收到数据之后，也使用了 Hibernate 持久层，做了类与数据库表的映射，把数据存储到数据库里，这里不再详细阐述。

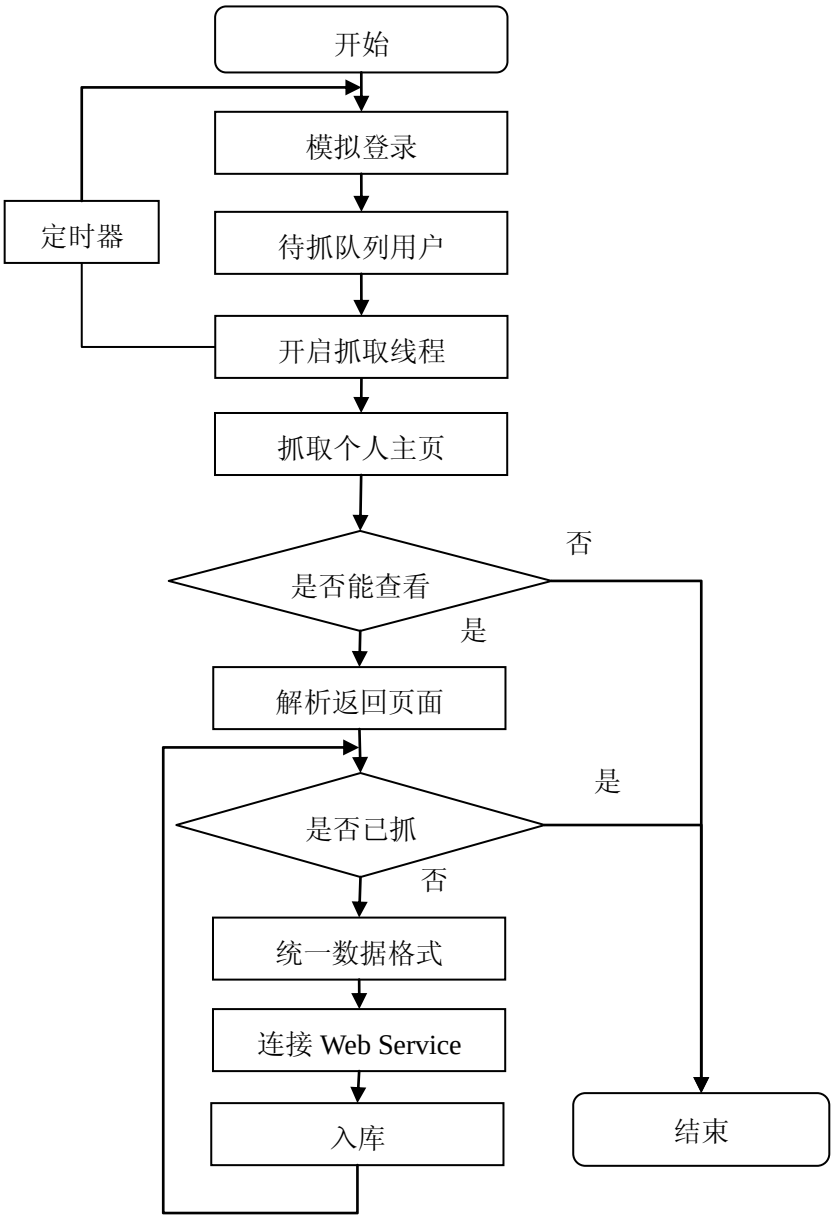

图 3-5 人人网爬虫流程图

人人网爬虫在添加用户时候，同样需要填写上用户名和抓取该用户的间隔时间，以实现增量式更新，添加用户的同时为该用户添加一个定时器。当开启

爬虫的时候，将使用定时器设定的间隔时间抓取该用户的数据，以保持所抓取的数据的实时性。为了保证抓取性能，使用了增量式抓取模式。每次首先从本地读出已抓的新鲜事编号，然后在解析前先对新鲜事编号进行比对再解析数据，避免了资源的浪费。

Facebook 爬虫的关键技术包括登录、获得 Access Token、解析 JSON 数据、Web Service 技术、Hibernate 入库等方面。

3.4 实验与分析

3.4.1 功能测试

三个爬虫分别进行功能测试，对于 Twitter 爬虫，我们在界面里手动添加了 100 个需要关注的测试用户，包括活跃和非活跃用户，然后分别对其设置了抓取间隔，开启自动抓取。对于 Facebook 爬虫，我们在界面里添加了 10 个公共主页，10 个个人主页进行抓取，包括了一些热门主页，并分别抓取间隔设置了抓取间隔。对于人人网爬虫，我们添加了 5 个公共主页，15 个个人主页，包括了一部分人气之星，并且分别对其设置了抓取间隔，开启自动抓取。抓取结果如下表 3-1 所示。

表 3-1 爬虫实验结果

爬虫	关注数	运行时间	抓取条数	抓取覆盖率
Twitter	100	3 天	482	100%
Facebook	20	3 天	161	100%
人人网	20	3 天	213	100%

三个爬虫分别运行了三天时间，实验统计了他们抓取的数据，其抓取覆盖率代表爬虫抓取的条数与网站上实际的条数的比值，本次测试三个爬虫都能够完全抓取到所关注的数据，抓取覆盖率达到了 100%。

Twitter 爬虫的调用 count 参数设置为 50，即每次抓取最新的 50 条信息。3 天一共抓取下来 482 条 Twitter 数据，下表 3-2 是统计结果。

我们仅仅统计这些数据，已经足够我们针对不同的用户抓取间隔进行调整。数据统计表明，一些名人的 Twitter 账号被转发和回复的比例远大于普通用户，而且名人原创微博更多，相应地，普通用户转发微博稍多。测试完毕之后，我们就可以对那些发 Twitter 频率较快的用户设置较短的时间间隔，对不

表 3-2 Twitter 数据统计

Twitter	数目	比例
被回复	105	37.23%
被转发	231	47.93%
原创	195	40.45%
转发	287	59.54%

活跃用户设置较长的时间间隔，以保证抓取的效率和实时性。统计还表明，没有用户能够在抓取间隔之内发送超过 50 条的数据，这样才保证了抓取覆盖率达到了 100%。

Facebook 爬虫 3 天一共抓取下来 161 条数据，表 3-3 是统计信息。

表 3-3 Facebook 爬虫数据统计

类型	数目	比例
留言	83	51.55%
状态	14	8.70%
日志	5	3.11%
分享-链接	44	27.33%
分析-视频	15	9.31%

由此可见，Facebook 新鲜事中，以其他用户留言和分享链接居多，日志最少。，公共主页的更新频率远高于个人主页，应该设置较短的抓取间隔。而且，调用 Facebook API 抓取，能够抓取所有的最新数据，所有保证了抓取覆盖率为 100%。

人人网爬虫 3 天一共抓取下来 153 条数据，表 3-4 是统计信息。

表 3-4 人人网爬虫数据统计

类型	数目	比例
公共主页日志	65	42.48%
公共主页状态	40	26.14%
个人主页日志	16	10.46%
个人主页状态	32	20.92%

根据表统计信息，我们只抓取了日志和状态两种数据，其中公共主页发表日志较多，个人发表状态较多。当然这也与个人的爱好有关，有人喜欢分享，有人喜欢发状态。热门公共主页经常会更新大量数据，需要设置很短的抓取间隔，而个人用户则一般比较平均，大约两三个小时更新一条新鲜事。

3.4.2 性能测试

性能测试是为了测试爬虫的性能能否满足稳定性、高效、大规模数据等要求，而且爬虫要求适应性强，容错性好。

我们分别对三个爬虫添加 100 个关注用户进行压力测试，设置 1 分钟的短暂抓取间隔，查看一个小时内爬虫运行情况，表 3-5 为实验结果。实验环境为奔腾 4 (2.6GHZ) CPU，2G 内存，Windows XP 操作系统。

表 3-5 压力测试结果

爬虫	内存	CPU	爬虫状态	入库状态
Twitter	持续增长至 84M，稳定	10%	正常	正常
Facebook	持续增长至 124M，稳定	13%	正常	正常
人人网	持续增长至 101M，稳定	12%	正常	正常

测试表明，在较大规模的情况下，爬虫都能够稳定地运行，内存和 CPU 表现在初期的小幅度增长之后，稳定在一个较低的水平，能够满足实际应用的需求。而且，入库的 Web Service 接口也能够满足入库的需求，没有异常或错误产生。

测试同时发现，Twitter 对第三方应用程序有抓取限制，具体为每小时不大于 150 次请求，否则返回 400 错误。而 Facebook 和人人网没有限制抓取频率。针对 Twitter 爬虫的解决办法是，我们可以开多个 Twitter 爬虫，每个爬虫只针对有限数量的用户来抓取，这样既满足实际抓取规模的需求，又能够突破 Twitter 的限制。

3.4.3 结果分析

经过功能测试和性能测试，根据我们的统计信息，爬虫能够满足我们的需求，抓取的数据能够覆盖所关注用户的全部信息资源，我们根据抓取数据的分析对抓取间隔进行了调整。抓取下来的数据具有统一的格式，能够完准确入库。并且在大规模情况下，爬虫能够满足稳定、高效的要求，并且占用系统资源较少，性能出色。

不足之处就是，三个爬虫都是专有爬虫，只能抓取 Twitter、Facebook 和人人网，不具有通用性。而且我们只针对感兴趣的用户进行抓取，并没有覆盖整个网站的全部 Deep Web 信息，当然这也是对实际需求与可行性综合考虑的结果。

3.5 本章小结

本章我们着重介绍了专有网络的 Deep Web 爬虫框架，包含认证模块、抓取模块、页面解析模块、入库模块和展示模块。然后分别就三个社交网站的爬虫 Twitter、Facebook 和人人网介绍了详细的设计流程，给出了各自的抓取策略，分析了抓取的关键技术，完成了爬虫的详细设计。

然后我们针对这三个爬虫进行了功能测试和性能测试，测试表明，我们的爬虫能够满足实际的需要，具有完整的功能和良好的性能。

第4章 爬虫管理与数据展示

本章我们介绍在多台机器上爬虫的分布式管理设计与实现，包括管理控制台的设计和每个抓取机器的守护程序的设计，介绍整个系统的功能然后给出性能测试与分析。另外，给出爬虫数据效果可视化的设计方案与实现。

4.1 引言

单机爬虫系统受限于 CPU 处理能力和磁盘的存取速度，不能处理大量的信息资源。为了提高爬虫系统的效率，尽可能多地利用的系统硬件资源和网络带宽，我们就需要把很多个爬虫分布到多台机器上来并行爬行，这就要求不同机器上的爬虫能够协同工作[6]。目前分布式网络爬虫有三种主要模式，主从模式、自治模式和混合模式。主从模式是指一台机器作为管理控制，其他爬虫节点与控制节点通信，从控制节点接收任务。这种模式实现简单，便于管理，因此我们的爬虫管理系统采用这种方式[45]。

为了实现爬虫的管理，多个爬虫协同工作，需要有一个专门的控制台来分配和协调爬虫抓取的任务，能够对爬虫的负载进行平衡。这就需要管理控制台与抓取机器进行通信，管理爬虫的开启、关闭以及增加删除任务等。因此，我们为每台机器的爬虫设置了一个守护程序，专门与管理控制台进行通信，接受指令，同时监护爬虫的稳定运行。

4.2 爬虫管理框架

在我们的互联网信息监测项目中，需要同时抓取几百个网站下的上千栏目，而且要保证信息的实时性与时效性，如果仅仅由一个爬虫完成显然不可能。因此，我们部署了多个普通的网络爬虫以及我们的 Deep Web 爬虫，每个爬虫负责抓取其中一部分网站或者网站的一部分抓取工作，多台机器上的爬虫实现并行协同工作。为了完成这个系统，我们实现一个管理控制台，同时为每台抓取机器部署一个守护程序，管理控制台通过一定的命令格式，与守护程序通信，管理爬虫节点的运行。管理控制台监控各抓取机器的负载状况，通过增加删除任务来平衡爬虫的任务数，并且能够从抓取机器上传下载配置文件和抓取数据以分析爬虫的运行状态。守护程序与管理控制台通信，同时监护本抓取

机器爬虫的运行，并把爬虫抓取下来的数据通过服务器 Web Service 接口存入数据库中[46]。下图 4-1 给出了爬虫管理系统的框架。

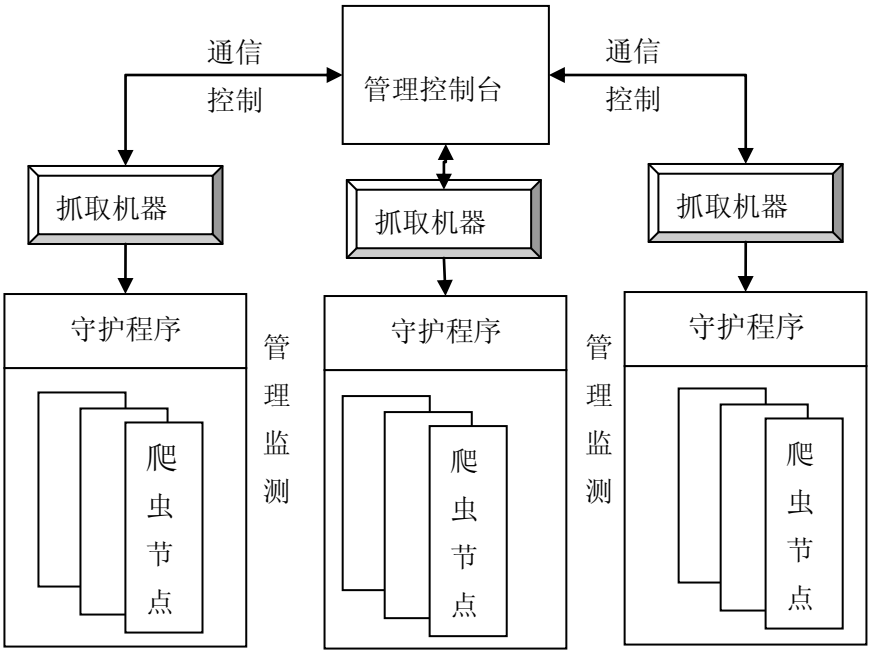

图 4-1 爬虫管理系统框架

爬虫管理系统共有两部分组成，分别是管理控制台和守护程序。管理控制台提供一个功能丰富的界面，以供管理员在控制台操作爬虫节点。提供的功能包括启动停止节点、增加删除电脑、增加删除节点、增加删除网站、增加删除栏目、迁移节点、迁移网站、查看状态、统计数据等功能。守护程序部署在抓取机器上，能够监测爬虫节点的运行，如果停止则自动重启。守护程序监视爬虫存储的文件夹变化，如果有文件抓取下来，解析文件并且把数据通过服务器的 Web Service 接口保存到数据库中。守护程序定时打包数据，并把数据传输到控制台，作为原始资料以供其他情况使用。

由于我们设计的 Deep Web 爬虫，抓取下来的数据直接传输到服务器的 Web Service 接口入库，因此不会被守护程序监控到。而普通爬虫由于最初设计问题和开发语言兼容性问题，只能够抓取数据以 XML 格式保存到本地。因此，我们的守护程序通过事件触发的方式监控存储文件夹变化，然后解析新增的文件，再把数据入库。

管理控制台与守护程序通信采用 TCP 的 Socket 编程，通过设定的命令格

式来向守护程序发送指令，守护程序收到指令之后向控制台返回确认信息，并且执行命令。如果需要传输数据，将开启数据传输线程通过 Socket 向控制台发送数据。

4.3 爬虫管理详细设计

4.3.1 管理控制台

管理控制台主要完成了界面的设计，提供方便的功能。其主界面如下图 4-2 所示。

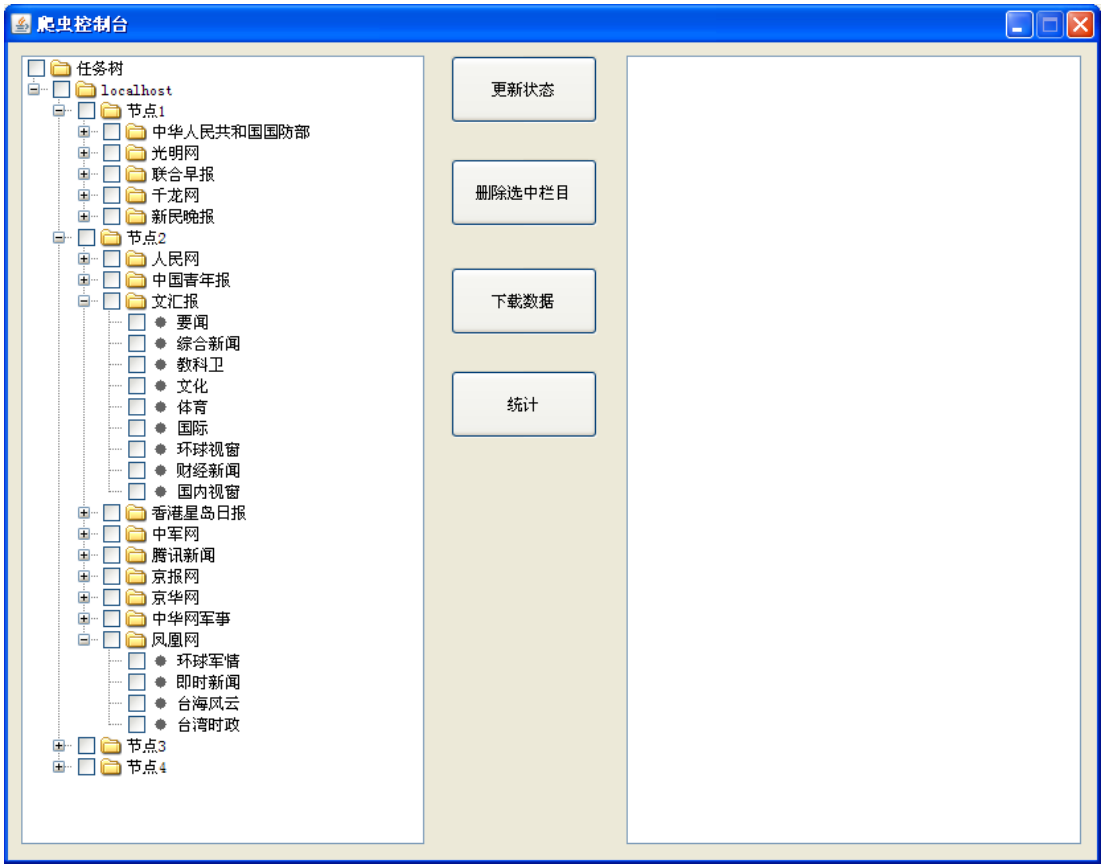

图 4-2 管理控制台界面

管理控制台左侧部分是整个任务树，展示了所有的抓取机器上的爬虫节点情况，可以根据各节点任务的规模进行调整。同时，提供了右键操作功能，可以用于对各抓取机器、节点、网站和栏目分别操作。中间功能区提供全局功

能，包括查看状态、删除选中栏目、下载数据和统计。其中，删除选中栏目需要在左侧任务树中首先勾选中要删除的栏目名。右侧部分为信息显示区，显示命令执行结果及提示结果等信息。

管理控制台与守护程序通信的命令格式如下表 4-1 所示：

表 4-1 通信命令格式设计

类型	命令	格式	功能
任务树	增加电脑	add computer	增加一个电脑
抓取机器	删除电脑	delete computer	删除当前电脑
	下载配置	load computer config	下载该机器所有配置文件
	下载日志	load computer log	下载该机器的日志文件
	增加节点	add node	在当前网站下增加栏目
节点	启动节点	start node	启动当前节点
	停止节点	stop node	停止当前节点
	删除节点	delete node	删除当前节点
	增加网站	add site	在当前接电线增加网站
	下载配置	load node config	下载当前节点的配置文件
	下载日志	load node log	下载当前节点的日子文件
	查看状态	view state	查看当前节点状态
网站	增加栏目	add item	在当前网站下增加栏目
	下载配置	load node config	下载当前网站的配置文件
	删除网站	delete site	删除当前网站
栏目	删除栏目	delete item	删除当前栏目
	下载配置	load item config	下载当前栏目的配置文件

管理控制台与守护程序根据这些命令格式互相通信，管理控制台通过守护程序控制着各个爬虫节点的运行，以完成分布式爬虫的负载均衡与协同工作。各个爬虫只负责一部分网站的数据获取，通过管理控制台的增加删除节点、网站、栏目来分配任务，分配任务时，需要上传或者下载相应的配置文件。

在通信的过程中，涉及到命令数据传输与文件互相传输问题，我们采用了 TCP 协议传输，使用了 Java 的 Socket 包建立连接，传输数据。每次通信时，传输方都要通知与接收方要进行传输，提醒接收方准备好，然后再开始传输数据。

传输命令的过程是，首先向目标地址的端口建立 Socket 连接，然后打开一个数据输出流 DataOutputStream 和一个数据输入流 DataInputStream，使用 writeUTF()方法向输出流中写命令字符串，然后从输入流里读响应字符串“got it”，如果没收到则继续尝试。文件传输的过程是，首先向目标地址的端口建立

Socket 连接，然后打开一个数据输出流 DataOutputStream，在输出流里使用 writeUTF()方法写字符串“send start”，接收方使用 DataInputStream 的 readUTF()方法接收到命令之后，创建文件，准备接收数据。然后传输方开始传输数据，使用 writeLong()方法，每次传输 8192 字节的数据，直到文件全部被传输完毕。传输过程中，使用缓冲区来缓冲读取的文件，以提高文件传输效率。

4.3.2 守护程序

守护程序是部署在每一台抓取机器上的，负责与管理控制台通信，接受管理控制台的指令，并且对爬虫进行相关操作。守护程序根据配置文件设置数个定时器，定时打包数据并传输到管理控制台。守护程序还监视爬虫运行状况，如果有异常，重启爬虫。另外，由于普通爬虫设计原因，其抓取的数据只存储在本地磁盘，守护程序需要监视本地磁盘的文件变化，把新抓取下来的数据解析，存储到服务器的数据库中。

守护程序与控制台通信指令在第 4.3.1 节已经详细介绍，这里介绍守护程序执行指令的过程。守护程序收到控制台指令之后，根据规定指令格式，分析指令内容，然后通过添加删除相应的配置文件来执行添加删除功能，以平衡该抓取机器爬虫的负载。执行下载配置的相关指令时候，通过 Socket 向管理控制台发送配置文件。守护程序根据爬虫抓取的日志进行分析，如果在给定的时间间隔内无日志更新，则判定爬虫节点异常，通过该爬虫节点的进程 ID 停掉该爬虫，然后重新启动。

在本课题中，普通爬虫抓取的文件包括 XML 格式的数据以及相关的图片，全部保存在本地磁盘。为了把这些数据存入数据库，以展示给用户，我们监视文件变化，如果是 XML 文件则解析，并通过服务器的 Web Service 接口入库的。整个过程的流程图如下图 4-3 所示。

守护程序使用了 JNotify 包来监视文件变化，它首先使用 c 调用 Windows 的文件变化事件的 API，生成动态链接库，然后使用 JNI 包装成 jar 包。使用 JNotify 类似于 Windows 事件，能够对文件变化立即做出响应，实时性非常强。解析完的数据通过 Web Service 接口传输到服务器，再使用 Hibernate 入库的流程[47]与第三章的 Deep Web 爬虫入库是一致的。

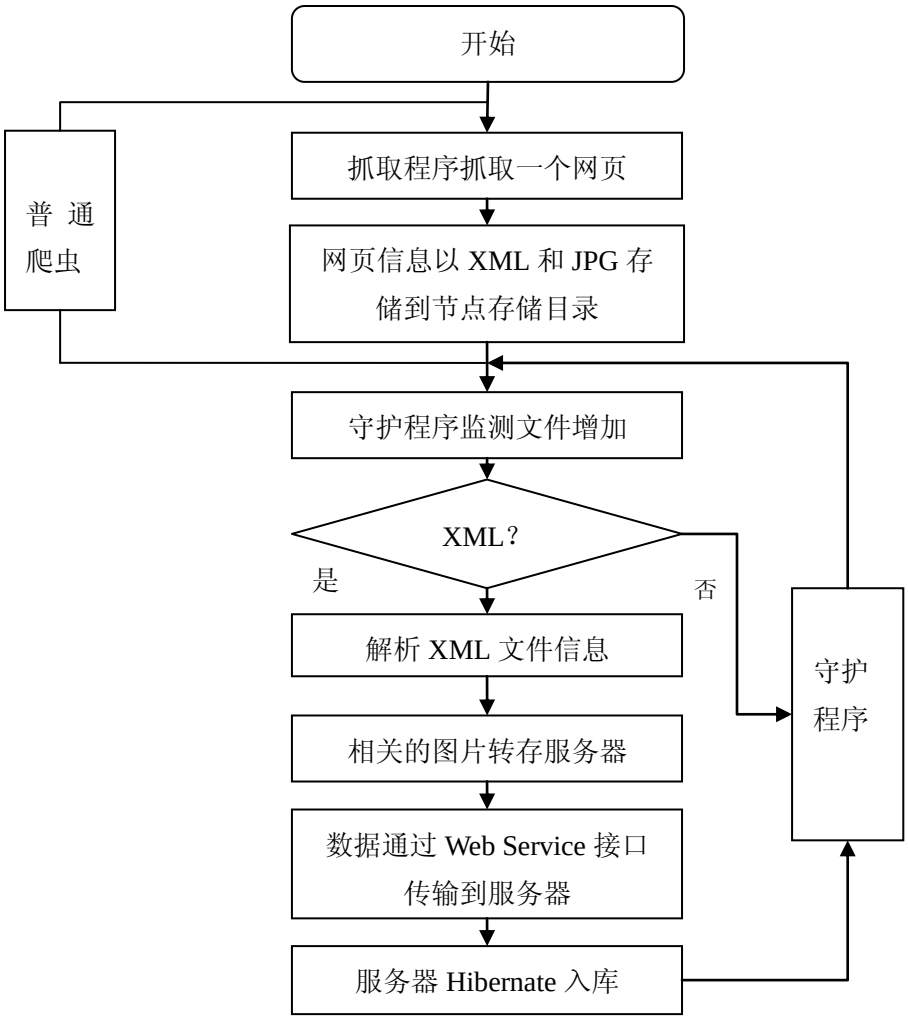

图 4-3 守护程序入库流程图

另外，我们用 JAVA 实现了 ZIP 格式文件压缩和解压的程序。由于 JAVA 自带的类 java.util.zip 压缩不支持中文文件名，因此我们重写了相关的类 ZipInputStream 和 ZipOutputStream。在读取文件名和写文件名的时候，使用 UTF-8 的编码格式来读写，以支持中文字符。

守护程序运行界面如下图 4-4 所示。用户可以更改守护程序端口号，以方便与管理控制台通信。界面中显示该抓取机器上守护程序所管理的爬虫节点，并且在信息显示区显示与管理控制台的通信的命令响应情况。

图 4-4 守护程序界面

图 4-5 显示了普通爬虫运行时的情况。界面展示了正在抓取的数据，这个爬虫由守护程序统一管理。

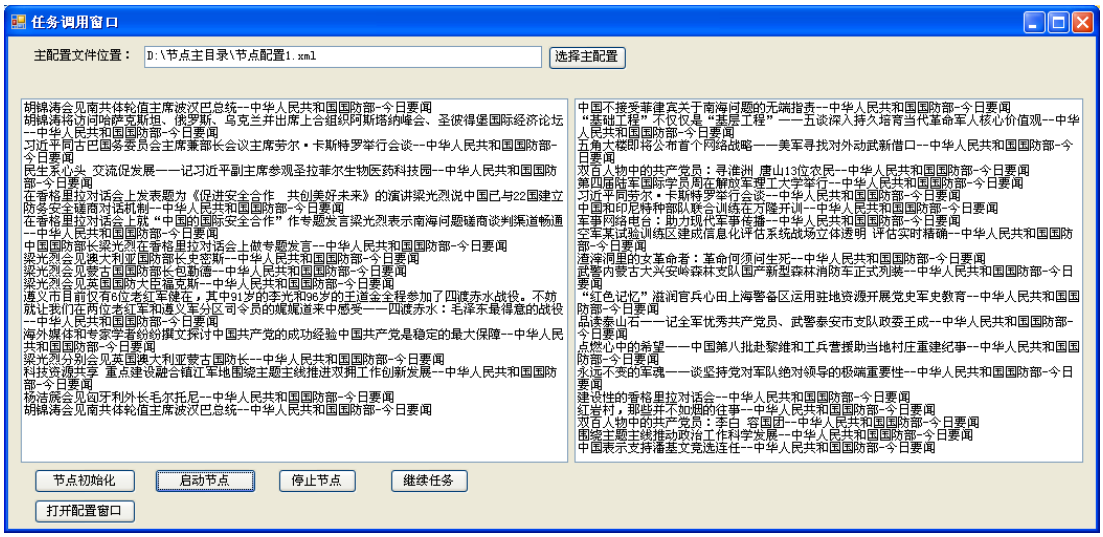

图 4-5 普通爬虫抓取界面

4.4 数据展示

4.4.1 设计思路

爬虫每天抓取了了大量的数据，如何把这些数据动态直观地展示给用户，并且根据分析出的热点话题，可视化地显示热点和趋势图，是我们需要研究的另外一个问题。最新最热门的专有网站的话题，也可以通过可视化的效果展示出来[48]。

每天对抓取下来的数据，包括 Deep Web 数据进行聚类分析[49]，得出热点话题和热点新闻，然后用动态效果图展示出来。实现可视化的方法有 Flash、Flex、Silver Light 等技术，我们选择了通过数据来直接生成 Flash 的方法来展现。Flash 和 Flex 都是用 ActionScript 脚本语言作为其核心编程语言，并且将代码编译成 swf 文件在 Flash Payer 中运行。相对于 Flex 注重于用户应用，Flash 更注重特效的处理和动画设计，而且 Flash 编程模型是基于时间轴的，可以做出持续的交互动画效果。

Action Script 动作脚本语言是面向对象的编程语言，它在 Flash 内容和应用程序中使用实现交互性、数据处理和其他功能，我们在第二章中专门介绍了这种技术。

4.4.2 整体设计

本次设计 Flash 动画，只使用时间轴的第一帧，然后在该帧添加 ActionScript2.0 动作代码，编程实现动态饼状效果。饼状图的每一个扇形代表一个热点主题，在扇形块中标注该块的名称和所占的比例。用户鼠标滑过饼状图的每一个扇形块，就会着重显示该块。点击该块，弹出对应的聚类主题的网页。

为了实现程序的交互，使用 XML 配置文件保存数据，ActionScript2.0 程序读取该配置文件来实现数据的展示。然后，其他程序就可以通过向配置文件中注入数据来生成动态效果图。

具体操作时，首先其他程序根据普通爬虫和 Deep Web 爬虫的数据进行聚类，得到各热点的主题以及相关新闻和话题的条数，然后保存到 XML 配置文件中。然后打开生成的 Flash 文件就可以展示可视化效果。

4.4.3 详细设计

首先要实现饼状图。我们在 ActionScript2.0 代码实现中，首先画个圆（扇形）作为上表面，然后分别画出两个侧面和一个曲面，最后画下表面的圆（扇形）。这样一个立体的小扇形就完成了，然后把所有的扇形拼在一起就构成了饼状图。为了确定每一个扇形的深度，让它们在视觉上看起来处于正常的前后关系位置，需要先计算出当前扇形的中间度数，然后判断这个度数是否在水平线以下。如果中间度数在水平线以下，则根据这个度数与垂直方向的角度判定，角度最小的在最前面，最大的在最后面。如果中间度数在水平线以上，那么则与中间度数在水平线以下的相反。需要注意的是，必须要从正上方作为 0 度开始计算，外侧面整个旋转了-90 度，比如，实际为 0 度的外侧面显示为 90 度，实际为 180 度的外侧面显示为 270 度。另外，扇形的每一块的通过随机生成的 RGB 值来着色，以使视觉上看起来更华丽。

然后再向图中元素添加鼠标动作。ActionScript 鼠标动作共有三种，包括鼠标点击（onRelease）、鼠标移入(onRollOver)和鼠标移出(onRollOut)。针对每个扇形块的鼠标点击（onRelease）添加了两个动作，一个是弹出相对应的热点主题聚类的网页，另外一个动作是扇形块向外移动或向内移动（使用 mx.transitions 包的 Tween 类），以着重展示该主题的信息。实现如下：

```
onRelease = function ()
    {
        getURL(url,"_blank");                    //弹出对应的 URL
                Tween = new Tween(MC 实例,MC 属性,运动模式,淡入或淡出,
目标值,持续时间/帧数,时间/帧数开关)
                保存当前扇形块位置，以决定淡入或淡出。
    }
```

针对每个扇形块的鼠标移入(onRollOver)和鼠标移出(onRollOut)添加一个向外移动的动作，使用方法同鼠标点击的向外移动动作。

另外，Flash 安全沙箱为保证安全性，必须设置本地回放安全性，选择只访问本地文件或者只访问网络文件。在 Flash 制作时，测试影片效果具有最高安全性，所以不需考虑配置文件从本地读取还是从服务器读取。但是，使用网络调用生成的 swf 文件时，将不能访问本地 XML 配置文件。因此，需要把配置文件也放在服务器上供调用。

XML 文件中对特殊字符“<”、“>”、“&”、“’”和“””分别会被转义成

“>”、“<”、“&”、“"”和“'”。因此，在保存配置时，特殊字符被转义，读取时需要替换成原先的字符。而 Action Script2.0 没有 replace 函数，需要手动实现。

实现的动态效果展示如下图 4-6 的示例所示。

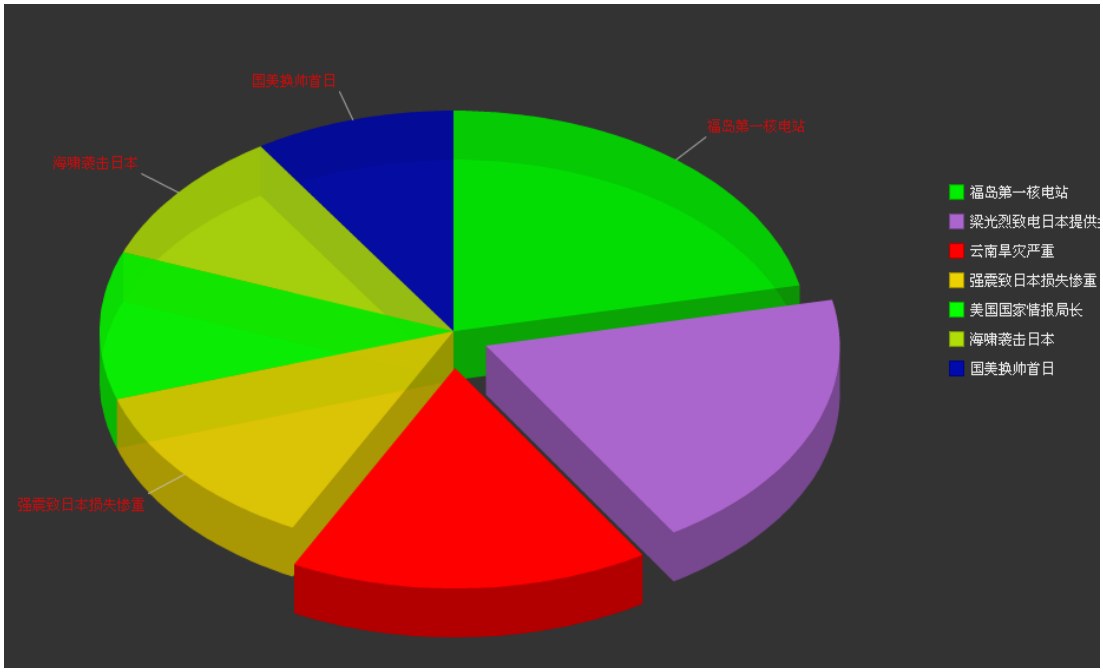

图 4-6 动态效果展示图

4.5 实验与分析

4.5.1 功能测试

针对管理控制台界面的功能进行测试，查看与守护程序通信的各个命令执行结果。我们针对每个命令在不同的环境下，在程序中添加测试代码，分别进行 10 次测试，下表 4-2 是详细测试结果。

通过测试结果可以看出，命令的平均响应时间都很短，能够及时做出响应，而且执行结果完整准确，没有出现错误，说明功能正常，能够满足需求。由于命令执行过程中可能需要压缩与解压缩文件、传输文件，所以有的命令平均执行时间较长，但是，这并不影响管理控制台的性能。

表 4-2 管理控制台功能测试结果

命令	平均响应时间/ms	平均执行时间/ms	执行情况
add computer	19.1	76.1	完整无误
delete computer	18.5	67.5	完整无误
load computer config	18.1	2387.4	完整无误
load computer log	18.8	6751.6	完整无误
add node	17.9	378.3	完整无误
start node	19.2	138.4	完整无误
stop node	18.7	127.1	完整无误
delete node	19.1	94.5	完整无误
add site	18.4	2643.5	完整无误
load node config	18.5	1765.1	完整无误
load node log	18.4	3655.2	完整无误
view state	19.1	197.3	完整无误
add item	18.2	2413.3	完整无误
load node config	19.6	1546.7	完整无误
delete site	18.7	105.7	完整无误
delete item	19.1	116.4	完整无误
load item config	19.0	1638.4	完整无误

4.5.2 性能测试

由于管理控制台同时管理很多个爬虫的时候，其通信功能可能会成为整个系统的瓶颈，导致系统性能下降[50]。因此我们在多台机器上部署大量的爬虫节点，然后使用管理控制台同时向这些守护程序发送指令，管理这众多爬虫节点的运行，测试管理控制台的平均响应时间。

硬件环境方面，控制台和抓取机器均为 Intel 酷睿 2(2.1GHZ)双核处理器、2G 内存。测试结果如下图 4-3 所示。

我们针对守护程序也完成了性能测试，使其能够满足大量爬虫同时抓取数据时，能够全部准确检测文件变化，并解析数据存入数据库中。并且能够与及时与管理控制台通信，准确接受指令，不影响整个系统的性能。我们测试的方法是，向本次磁盘存储的文件夹拷贝两天内已抓取的 65608 个 XML 文件数据资源，然后使用守护程序检测该文件夹，把数据解析入库，查看系统资源使用情况。同时使用管理控制台向该守护程序发送指令，计算平均命令响应时间。

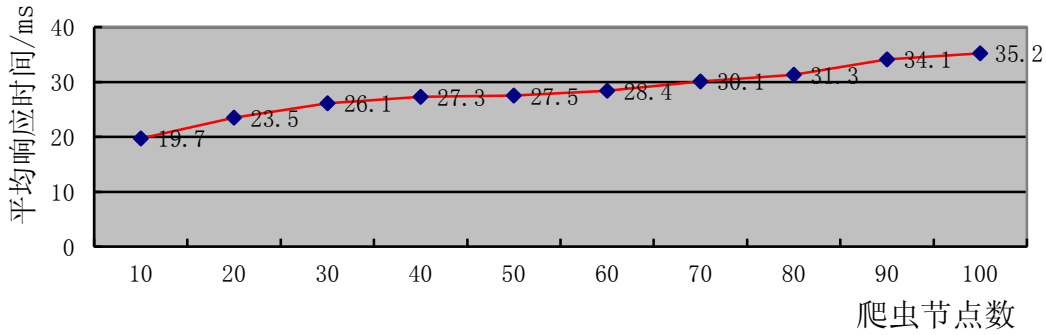

图 4-7 爬虫控制台性能测试结果

硬件环境方面，控制台和抓取机器均为 Intel 酷睿 2 2.1GHZ 双核处理器、2G 内存。拷贝时间总共为 538 秒，平均每秒钟拷贝 122 个文件。而守护程序入库总共耗时 2148 秒，入库条数为 65601，文件监测准确率达到了 99.99%。下表 4-3 是性能测试结果。

表 4-3 守护程序性能测试结果

测试项目	性能
内存	从初始的 21M 持续增长至 153M 稳定
CPU	CPU 使用率保持在 5%左右
管理控制台的命令平均响应时间	21.2ms

从上表中可以看出，内存使用一直在增长至一个较高的值才稳定，而 CPU 使用率保存稳定，说明在大量数据情况下，守护程序能够基本满足性能需求，不会对系统资源有过多的要求。而管理控制台与守护程序通信的命令响应时间也与正常情况下一致，说明守护程序的与控制台通信不受爬虫抓取数据的影响。但是，总共 65608 个 XML 文件，入库条数却为 65601，说明有一部分文件没有被监测到或者没有被解析。经过仔细检查日志文件发现，未入库的 7 条数据都被守护程序检测到，但是因为格式不完整而无法入库。

另外，我们针对内存持续增长问题进行了检测，对所有可能导致内存泄露或者垃圾的情况都检测了一遍，包括大量使用静态方法和变量、解析大量 XML 文件、大量使用 Map 和 List 容器、Log4j 记录日志、JVM 虚拟机强制回收垃圾、JNotify 的使用。最后发现，在我们的针对每个单独情况测试时，只有仅仅使用 JNotify 监视大量文件增加就会导致内存不断增长，因此判断内存增长的原因是 Jnotify 的未能及时释放内存导致的。但是，在不太影响系统性能

的情况下，使用 JNotify 是非常精确而且高效的。

4.5.3 结果分析

经过功能测试和性能测试，根据我们的实验结果和统计信息，爬虫管理系统在功能上能够满足我们的需求，管理控制台所有的命令都能够被有效准确地执行，而且响应和执行时间都较短。在大规模的通信和数据请求情况下，整个系统也能够稳定地运行，并且消耗较少的系统资源。

而系统的不足之处在于，使用了 JNotify 监视文件变化，涉及到 Windows 事件以及 JNI 封装，有可能存在内存泄露。从而可能导致在长时间运行时，守护程序所占内存不断增加，过于消耗系统资源。但是这也是在我们硬件资源可接受的范围内，JNotify 出色的功能也使得其不可或缺。

4.6 本章小结

本章我们介绍了爬虫管理系统的框架结构，这是一个类分布式系统，包括一个管理控制台和多台抓取机器上的守护程序。管理控制台与守护程序通信，控制各爬虫节点的运行。详细介绍了管理控制台的功能界面以及相应的命令格式，介绍了文件和命令传输的过程。详细介绍了守护程序的功能，除了与控制台通信，同时守护程序还监视爬虫运行、检测文件并入库、定时打包等。

另外，我们介绍了抓取下来的数据的可视化问题，提供了使用 ActionScript 编程实现动态饼状图的方法。

最后，我们进行了爬虫管理系统的功能测试和性能测试。功能测试主要针对管理控制台的功能执行情况，性能测试包括管理控制台与大量爬虫节点同时通信的性能以及大量数据抓取时守护程序性能两部分，结果表明，我们的爬虫管理系统能够准确完成所需的功能，并且在大规模数据和通信环境下，能稳定运行，消耗较少的资源。

结　论

Deep Web 信息包括可搜索数据库以及专有网络数据资源，本文研究了针对专有网络 Deep Web 爬虫的实现，实现了三种社交网站的爬虫框架与详细设计，并且给出了爬虫的管理与展示系统。主要研究内容如下几个方面：

(1) 提出了专有网络爬虫的设计方案，使用了 HtmlUnit 框架模拟浏览器实现了自动登录，然后抓取并解析页面信息。

(2) 设计实现了 Twitter 爬虫，数据获取策略是首先通过 OAuth 认证获取 Access Token，然后调用 Twitter AIP 增量抓取用户 Twitter 数据并入库。

(3) 设计实现了 Facebook 爬虫，抓取策略是使用 HtmlUnit 登录，获得一个 Access Token，然后调用 Facebook Graph API 增量抓取用户的新鲜事，解析返回的 JSON 数据并且统一了格式入库。

(4) 设计实现了人人网爬虫，抓取策略是使用 HtmlUnit 构造浏览器 WebClient 登录，并保存 Cookie，然后使用 WebClient 增量抓取用户页面，解析状态和日志并入库。

(5) 实现了一个爬虫管理系统，完成一个管理控制台和部署在每个抓取机器上的守护程序，通过通信来管理控制爬虫节点任务分配与负载平衡。守护程序监视爬虫节点运行并解析普通爬虫抓取的数据入库。

(6) 使用 Flash 的 ActionScript2.0 语言实现了抓取数据的可视化展示的动态饼状图效果。

虽然本文取得了大量阶段性成果，基本达到了课题预期目标，系统也已经部署到实际环境中使用。但是，本文有一些部分不是很成熟，还有待进一步的研究和创新。今后的工作重点主要有以下几点：

(1) 针对专有网络的 Deep Web 爬虫可以在一个统一的框架里实现通用性，能够适应抓取各种不同的专有网站，包括需要注册登录的论坛、社交网站等。本课题设计的爬虫有待改进，比如三个爬虫都没有抓取相应的留言，而且人人网只抓取了日志和状态。

(2) 管理控制台可以通过 SOAP 协议与守护程序通信，更方便高效。管理控制台还可以根据每台抓取机器的资源消耗实现自动负载平衡，更加智能。

(3) 可以根据一段时间内的数据展示出热点变化趋势的动态效果图。

参考文献

[1] 赵志宏, 黄蕾, 刘峰, 陈振宇. Deep Web 搜索技术进展综述[J]. 山东大学学报(工学版), 2009, 4.

[2] L. Barbosa, J. Freire. Siphoning hidden-web data through keyword-based interfaces. In SBBD, 2004.

[3] Tao Cheng, Kevin Chen-Chuan Chang. Entity Search Engine: Towards Agile Best-Effort Information Integration over the Web[C]. In Proceedings of the Third Conference on Innovative Data Systems Research, 2007, pp08-113.

[4] Bergman M K. The Deep Web: surfacing hidden value [J]. The Journal of Electronic Publishing, 2001, 7 (1): 891228914.

[5] He H. Automatic integration of Web search interfaces with WISE integrator [J]. VLDB Journal, 2004, 13 (3): 2562273.

[6] 周德懋, 李舟军. 高性能网络爬虫: 研究综述[J]. 计算机科学, 2009, 8(36).

[7] Brin S, Page L. The Anatomy of a Large—scale Hypertextual web Search Engine [J]. Computer Networks, 1998, 30: 107-117.

[8] Ghemawat S, Gobioff H, Leung Shun-Tak. The Google File System [A]. //Proceedings of the 19th ACM Symposium Oil-Operating Systems Principles[c]. 2003: 20-43.

[9] Dean J，Ghernawat S. MapReduce: Simplified Data Processing on Large Clusters [A]//Proceedings of the 8th Conference on Symposium on Operating Systems Design & Implementation[C]. San Francisco, CA, 2004: 16-10.

[10]Heydon A, Najork M. Mercator: A scalable, extensible Web crawler[J]. World Wide Web, 1999, 2(4): 219—229.

[11]Lee Hsin-Tsang, Leonard D. IRLbot：Scaling to Billion Pages and Beyond[A] //Proceedings of the 17th International World Wide Web Conference[C]. ACM

Press, 2008: 427—436.

[12]Boldi P, Codenotti B. Santini MUbiCrawler: A Scalable Fully Distributed Web Crawler [J]. Software: Practice & Experience. 2004, 34: 711-726.

[13]Burner M．Crawling towards Eternity: Building an Archive of the World Wide Web[J]．Web Techniques Magazine, 1997,2(5): 125-130.

[14]Cho J, Carcia-Molina H, Page L. Efficient crawling through URL ordering [J]. Computer Networks and ISDN Systems, 1998, 30 (7): 1612172.

[15]Pedley P. The Invisible Web: searching the hidden parts of the internet [M]. London: Europa Publications Ltd, 2001.

[16]Sherman C, Price G. The Invisible Web: uncovering sources search engines can’ t see [J]. Library Trends, 2003, 52 (2): 2822298.

[17]刘伟，孟小峰. Deep Web 数据集成问题研究[R]. WAMDM 技术报告, 2006.

[18]BrightPlanet.com. The deep web: Surfacing hidden value. Accessible at http://www.brightplanet.com/, July 2000.

[19]黄晓冬. Invisible Web 研究综述[J] . 情报科学, 2004, 22(9): 114421148.

[20]Zhang Z, He B, Chang K C. Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax[C/OL]. [2008205228]. http://eagle.cs.uiuc.edu/pubs/2004/parsing-sigmod04-zhc-mar04.pdf.

[21]He H, Meng W Y, Lu Y Y, et al. Towards Deeper Understanding of the Search Interfaces of the Deep Web[J] . World Wide Web, 2007, 10: 1332155.

[22]He B, Chang K C. Automatic Complex Schema Matching Across Web Query Interfaces: A Correlation Mining Approach[J] . ACM Transactions on Database Systems, 2006, 31 (1): 1245.

[23]Wu W, Doan A, Yu C T. Merging Interface Schemas on the Deep Web via Clustering Aggregation[C/OL]. [2008205228]. http: // ieeexplore.ieee.org/ stamp/ stamp. jsp ? tp = &arnumber = 1565786 &isnumber = 33217.

[24]Raghavan S, Molina H G. Crawling the Hidden Web [C/OL]. [2008205228]. http:// www.vldb.org/ conf/ 2001/ P129.pdf.

[25]Cope J, Craswell N, Hawking D. Automated Discovery of Search Interfaces on the Web [C/OL]. [2008205228]. http://crpit.com/confpapers/CRPITV17Cope.pdf.

[26]Barbosa L, Freire J. An Adaptive Crawler for Locating Hidden-Web Entry Points [C] / /Proceeding of the World Wide Web Conf. (WWW). New York: ACM Press, 2007: 4412450.

[27]谭春亮. 基于本体的 Deep Web 语义搜索引擎[D]. 广西:广西师范大学, 2008.
TAN C L. Ontology-based semantic search engine for Deep Web [D]. Guangxi: Guangxi Normal University, 2008.

[28]Barbosa L, Freire J. Searching for Hidden-Web Databases[C/OL]. [2008205228]. http://webdb2005.uhasselt.be/webdb05_eproceedings.pdf.

[29]王辉, 刘艳威, 左万利. 使用分类器自动发现特定领域的深度网入口[J]. 软件学报, 2008, 19(2): 2462256.

[30]Z WU, W. Meng, V. Raghavan, etal. Towards automatic incorporation of search engine into a large-scale meta-search engine[C]. IEEE/WICWI-2003 Conference, 2003: 658-661.

[31]Clement Yu, George Philip, and Weiyi Meng. Distributed top-n query processing with possibly uncooperative local systems[C]. In VLDB 2003: Proeeedings of the 29th international conference on Very large databases. VLDB Endowment, 2003, 117-128.

[32]Y. Lu, H. He, H. Zhao, W. Meng, C. Yu. Annotating Structured Data of the Deep Web[C]. IEEEICDE, 2007.

[33]Luciano B. Siphoning hidden-Web data through key word-based interfaces[C] //Proceedings of S BBD. Brasilia, Brazil, 2004: 3092321.

[34]Thanaa M.Ghanem, Walid G.Aief. Databases Deepen the web[J]. IEEE

Computer, 2004, 73(1), Pll6-117.

[35]He B, Patel M, Zhang Z, et al. Accessing the Deep Web: a survey[J]. Communication of the ACM, 2007, 50(5):P94-101.

[36]Wensheng Wu, AnHai Doan. Merging Interface Schemas on the Deep Web via Clustering Aggregation. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.31&rep=rep1&type=pdf.

[37]T.H.Haveliwala, Efficient Computation of PageRank[R]. Stanford Univ. Technical Report, 1999.

[38]M. Bowler. HTMLUnit. http://sourceforge.net/projects/htmlunit.

[39]O'Reilly Media. Essential ActionScript 2.0[M]. 2004: 35-165.

[40]Michael Trusov, Randolph E. Bucklin, Koen Pauwels. Effects of Word-of-Mouth Versus Traditional Marketing: Findings from an Internet Social Networking Site[J]. Journal of Marketing, 2009, 73(5): 90-102.

[41]Kikuo Yuta, Naoaki Ono, Yoshi Fujiwara. A Gap in the Community-Size Distribution of a Large-Scale Social Networking Site[J]. Yoshi Fujiwara, 2007.

[42]Joseph Bonneau, Sören Preibusch. The password thicket: technical and market failures in human authentication on the web[R]. The Ninth Workshop on the Economics of Information Security, 2010.

[43]Akshay Java, Xiaodan Song, Tim Finin, Bella Tseng. Why we twitter: understanding microblogging usage and communities[C]. WebKDD/SNA-KDD’07, 2007.

[44]Nicole B. Ellison, Charles Steinfield, Cliff Lampe. The Benefits of Facebook “Friends:” Social Capital and College Students’ Use of Online Social Network Sites[J]. Journal of Computer-Mediated Communication, 2007, 12(4): 1143-1168.

[45]刘爽等. 基于 GNP 算法的分布式爬虫调度策略[J]. 计算机应用研究，2010(2).

[46]Dumitru Roman, Uwe Keller. Web Service Modeling Ontology[J].Journal Ontology. 2005, 1(1).

[47]Christan Bauer, Gavin King. Hibernate In Action[M].2005: 30-251.

[48]石峰. 利用 Flash 和 XML 实现动态展示图[J]. 计算机网络. 2007(11).

[49]A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. In ICDE, 2005.

[50]P. G. Ipeirotis and L. Gravano. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. In VLDB, pages 394–405, 2002.

哈尔滨工业大学学位论文原创性声明及使用授权说明

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《社交网络数据获取技术与实现》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果。据本人所知，论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签名：胡亚楠　　日期：2011 年 6 月 25 日

学位论文使用授权说明

本人完全了解哈尔滨工业大学关于保存、使用学位论文的规定，即：

（1）已获学位的研究生必须按学校规定提交学位论文；（2）学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（3）为教学和科研目的，学校可以将学位论文作为资料在图书馆及校园网上提供目录检索与阅览服务；（4）根据相关要求，向国家图书馆报送学位论文。

保密论文在解密后遵守此规定。

本人保证遵守上述规定。

作者签名：胡亚楠　　日期：2011 年 6 月 25 日

导师签名：郭德成　　日期：2011 年 6 月 25 日

致　谢

光阴荏苒，岁月如梭，两年时间如白驹过隙般划过，留下了数不清的眷恋和道不尽的不舍。机器智能与翻译研究室一如天空一般广阔，如大海一样深邃，又如母亲一般温暖。您给了我们翱翔的翅膀，赐予我们搏浪的风帆，又教会了我们勇敢。两年前我们还青葱如玉风华初放，两年后我们已经意气风发即将远行，在跟您告别之前，请让我深情地向您们说一声感谢。

首先感谢我的导师郑德权副教授。郑老师您和蔼可亲平易近人，无论什么事情都能得到您慷慨大方的帮助和无微不至的关怀；郑老师您学识渊博经验丰富，在我的学习和工作中指引我迅速成长，激励我不断前进；郑老师您兢兢业业孜孜不倦，忘我的工作精神和踏实严谨的治学作风，都深深印在了我的心上。作为您的学生，我觉得非常的幸运，以后必将勇攀高峰，不负您的期望。

感谢实验室李生教授和赵铁军教授，您们德高望重大师风范令我无时无刻不景仰；您们高瞻远瞩高屋建瓴为实验室带来了今天明天的辉煌；您们求真务实的态度和忘我的工作精神令我们深深折服。

感谢实验室杨沐昀副教授，我一直都敬畏您严肃认真一丝不苟的精神，敬畏您严格管理严谨治学的态度，您的教诲将使我受用一生。

感谢李晗静老师，感谢徐冰老师，感谢朱聪慧老师，您们为实验室付出了大量的心血，也给予我非常多的指导。

感谢实验室的李世奇、刘水、刘宇鹏、陈宇、梁华参、刘乐茂、李风环、张春越众位师兄师姐，虽然很少聆听你们的教诲，但是微言大义常常令我醍醐灌顶。感谢小邹姐，你是实验室最勤劳善良的。感谢我的同学于墨、李震、李大任、朱晓宁、刘海波、王山雨、郑宏、王垚尧、郑博文、王超，我们永远兄弟般的情谊。感谢实验室的师弟师妹们，你们是如此的可爱，一定会有光明的未来。

感谢我的父母和姐姐弟弟，你们在背后默默地支持我，我每时每刻都觉得充满力量，不惧困难。

感谢我的女朋友牛宁，你的真诚和善良给我带来无限的温馨幸福，你的博爱和宽容陪我走过一段段的崎岖坎坷，你的勤劳与勇敢激励我闯过了无数的艰难险阻。

感谢所有关心支持我的人，谢谢你们。

社交网络数据获取技术与实现

作者：胡亚楠
学位授予单位：哈尔滨工业大学