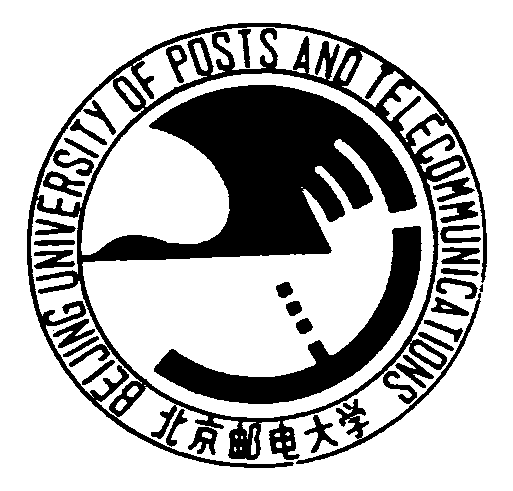
**密级： 保密期限：**



硕士研究生学位论文



**题目： 支持AJAX的定址网络爬虫**

**系统的研究与实现**

**学 号： 106900**

**姓 名： 刘凡凡**

**专 业： 计算机科学与技术**

**导 师： 吴国仕**

**学 院： 软件学院**

**2012年 12 月**

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名： 日期：

关于论文使用授权的说明

本人完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。

本学位论文不属于保密范围，适用本授权书。

本人签名： 日期：

导师签名： 日期：

支持AJAX的定址网络爬虫

系统的研究和实现

摘 要

在Web2.0的概念出现后，一种被称为RIA的具有高度互动性和丰富用户体验的网络应用程序出现了，如博客、微博等，而AJAX技术由于符合Web2.0时代的需求，被越来越多的运用在Web开发中。AJAX技术采用客户端JavaScript动态修改DOM结构，实现了网页的无缝化重构，提高了网页的互动性、速度以及可用性。但与此同时，它改变了传统的Web应用模型，打破了传统爬虫依赖于分析页面中超链接的爬行模式，使传统爬虫不能采集AJAX网页中的动态内容，这意味着大量有意义的数据无法通过搜索引擎检索。

本文设计并实现了一种支持AJAX的定址网络爬虫系统，旨在解决AJAX网站的动态数据采集问题。本文首先通过对传统网络爬虫的研究，分析得出AJAX爬虫的技术难点，并从一个实际的AJAX网站出发，说明了传统爬虫在爬行使用AJAX技术实现的网站时存在的关键问题以及本文设计的系统的应用场景；然后介绍了本研究的相关概念和问题模型，并设计了系统运行流程与系统架构；最后通过对AJAX爬虫中关键问题的分析和设计，实现了支持AJAX的定址网络爬虫系统。

本系统将传统网络爬虫工作过程中的URL提取和下载网页两个功能分离开，成为两个独立的功能模块。通过URL抽取模块实现网站URL抽取，形成URL资源库。采用Webkit渲染引擎实现的浏览器来加载HTML网页并解析JavaScript代码，并结合脚本生成器生成的JavaScript翻页脚本，能够从页面DOM表示中识别用于页面导航的页面元素，自动触发页面元素上的事件，生成并提取分页内容。爬虫系统只采集URL资源库中链接地址导向的网页信息，也就是说爬虫的爬行范围完全由URL资源库限定，是受控的，即为“定址”的网络爬虫。

最后，为测试系统查全率、准确度与性能，将系统应用于三类共六个真实网站，实验结果表明：本系统的查全率达到了100%；在不翻页采集的情况下，平均抓取速率达到50.55kb/s，系统展现出很好的效能。

本系统能够准确抓取AJAX网站动态内容，并对相似网页结构的目标网页进行分页数据的采集，系统具有较高的灵活性和适用性，可用于建设垂直搜索，或应用于开源情报采集领域等等。

**关键词：**AJAX JavaScript 网络爬虫 数据采集 定址

WEB CRAWLER FOR URL-SPECIFIED CRAWLING OF

AJAX-BASED WEB APPLICATIONS

ABSTRACT

Along with the emergence of Web2.0, a new kind of web application called Rich Internet Applications(RIAs) emerged, providing highly interactivity and rich user experience, such as blog and twitter. A technology that has gained a prominent position lately, under the umbrella of Web2.0, is AJAX. It has becoming a widespread technology for the development of web applications.In AJAX, the combination of JavaScript and dynamic DOM manipulation, along with asynchronous server communication is used to achieve a high level of user interactivity, speed and usability.

At the same time, it changes the traditional model of web applications, shatters the metaphor of web ‘pages’ with unique URLs, on which traditional web crawlers are based.This new change seriously impaires the ability of existing crawlers to truly crawl these applications. Current search engines either ignore AJAX applications or produce false negatives.

This paper designs and implements a novel technique for crawling AJAX-based applcation through an URL-specified web crawler, aiming at addressing the problem of crawling AJAX content induced by client-side JavaScript. First, we discuss traditional web crawlers, and analyze the key challenges involved in crawling AJAX-based applications. For illustration, an real-world public AJAX application is utilized as an example to explain the difficulties of crawling AJAX and scenarios. Second, we present the terminology used in this paper, model AJAX web sites, pages and events, describe the overall workflow and architecture of our system. Finally, we present a detailed discussion of our crawling algorithm and implements an URL-specified web crawler, using the concepts and algorithms discussed in this paper.

We separate traditional crawling into two independent modules: hyperlink extractor and web crawler. The hyperlink extractor is used to extract hyperlinks from web sites, and store them into the URL repository. Our approach is based on a webkit browser, in which we open the AJAX application, exercise client-side JavaScript code, identify clickable elements and fire events on those elements. The web crawler only downloads pages whose URL in the URL repository, which means the scope of crawling is totally identified by the URL repository, that is why we call it “URL-specified” crawling.

We have performed a number of empirical studies to systematically analyze the overall performance of our approach. We evaluate the precision(percentage of dynamic content recovered), accuracy(percentage of correct states) and performance. The results show that the precision is 100%. In the case of non-flip crawling, the average crawling rate is 50.55kb/s. The result reveals that our system produces effective results.

This crawling technique can crawl AJAX applications accurately. The system has high flexibility and scalablity, we believe that the crawling technique can be used, for instance, in building of vertical search, the field of open source intelligence gathering, and so on.

KEY WORDS：AJAX, JavaScript, Web crawler, Data collection, URL-specified

目 录

[第一章 绪论 1](#_Toc343979902)

[1.1 研究背景及意义 1](#_Toc343979903)

[1.2 研究现状 3](#_Toc343979904)

[1.3 论文研究工作 4](#_Toc343979905)

[1.4 论文架构 5](#_Toc343979906)

[第二章 相关背景知识 6](#_Toc343979907)

[2.1 Web2.0 6](#_Toc343979908)

[2.2 JavaScript 7](#_Toc343979909)

[2.3 AJAX 8](#_Toc343979910)

[2.3.1 AJAX简介 8](#_Toc343979911)

[2.3.2 AJAX的运作方式 9](#_Toc343979912)

[2.3.3 AJAX与传统网页的比较 10](#_Toc343979913)

[2.4 HTML DOM 12](#_Toc343979914)

[2.4.1 DOM的树状结构 12](#_Toc343979915)

[2.4.2 DOM常用的方法 13](#_Toc343979916)

[2.5 XPath 14](#_Toc343979917)

[2.6 本章小结 14](#_Toc343979918)

[第三章 网络爬虫系统的研究 15](#_Toc343979919)

[3.1 网络爬虫 15](#_Toc343979920)

[3.1.1基本原理 15](#_Toc343979921)

[3.1.2爬行策略 16](#_Toc343979922)

[3.2 AJAX爬虫的技术难点 17](#_Toc343979923)

[3.2.1JavaScript的解析 17](#_Toc343979924)

[3.2.2识别页面状态 17](#_Toc343979925)

[3.2.3 DOM操作 18](#_Toc343979926)

[3.2.4 触发事件 18](#_Toc343979927)

[3.3 本章小结 19](#_Toc343979928)

[第四章 网络爬虫系统的设计 20](#_Toc343979929)

[4.1 AJAX应用实例 20](#_Toc343979930)

[4.2 相关概念 22](#_Toc343979931)

[4.2.1 页面状态与状态切换 22](#_Toc343979932)

[4.2.2 clickable元素 23](#_Toc343979933)

[4.3 AJAX建模 23](#_Toc343979934)

[4.3.1 事件模型 23](#_Toc343979935)

[4.3.2 AJAX页面模型 23](#_Toc343979936)

[4.3.3 AJAX网站模型 24](#_Toc343979937)

[4.4 系统架构 25](#_Toc343979938)

[4.5 系统流程 27](#_Toc343979939)

[4.6 本章小结 28](#_Toc343979940)

[第五章 爬虫系统的实现 30](#_Toc343979941)

[5.1 WebKit渲染引擎 30](#_Toc343979942)

[5.1.1渲染引擎分析 30](#_Toc343979943)

[5.1.2 WebKit框架结构 33](#_Toc343979944)

[5.1.3 WebKit解析流程 33](#_Toc343979945)

[5.1.4 Qt WebKit 34](#_Toc343979946)

[5.2 链接抽取模块 36](#_Toc343979947)

[5.2.1过滤条件 36](#_Toc343979948)

[5.2.2 Robots协议 37](#_Toc343979949)

[5.2.3 URL消重策略 38](#_Toc343979950)

[5.2.4抽取流程 39](#_Toc343979951)

[5.3 脚本生成模块 40](#_Toc343979952)

[5.3.1样本页面 40](#_Toc343979953)

[5.3.2关键参数设置 40](#_Toc343979954)

[5.3.3定位clickable元素 41](#_Toc343979955)

[5.3.4脚本执行逻辑 42](#_Toc343979956)

[5.4 Cookie管理模块 44](#_Toc343979957)

[5.5 模块交互与关键算法 45](#_Toc343979958)

[5.6 系统实现 47](#_Toc343979959)

[5.7 本章小结 48](#_Toc343979960)

[第六章 爬虫系统实验与结果分析 49](#_Toc343979961)

[6.1实验环境 49](#_Toc343979962)

[6.2 评估方法 49](#_Toc343979963)

[6.3 实验设计及结果分析 50](#_Toc343979964)

[6.3.1查全率 50](#_Toc343979965)

[6.3.2准确度 51](#_Toc343979966)

[6.3.3抓取速率 52](#_Toc343979967)

[6.4本章小结 54](#_Toc343979968)

[第七章 结束语 55](#_Toc343979969)

[7.1 结论 55](#_Toc343979970)

[7.2 未来方向 56](#_Toc343979971)

[参考文献 57](#_Toc343979972)

[致谢 59](#_Toc343979973)

[攻读学位期间发表的学术论文 60](#_Toc343979974)

第一章 绪论

1. 研究背景及意义

随着网络的蓬勃发展，数以万计的网站林立，这些网站除了有公司企业、门户网站，还有相当数量的个人网站以及各种社交网站，它们拥有着众多但却杂乱无章的信息，如果我们把互联网看成一个非常庞大的资料库，那么这些网站则成为了互联网主要的资料来源。

人们使用搜索引擎在如此浩瀚、复杂的巨大资料库中快速、准确地查找需要的资料，而网络爬虫作为搜索引擎的重要组成部分，其抓取页面的数量和质量对搜索引擎的查询结果影响很大。

过去的，所有网站的信息及内容，皆由各网站的管理者或所属公司产生并放置上去，提供给所有的网络用户观看及使用；但在近几年，网络用户不再只是使用者，“创造与分享”的使用者行为如异军突起，彻底颠覆网络用户原本只能被动接收信息及内容的故有生态。O’Reilly媒体公司创办人暨执行长Tim O’Reilly赋予了它一个具代表性的名称——Web2.0。网络用户不论身在哪一国、相距有多远，都能透过具Web2.0功能的网站，进行相互分享信息的网络行为。

Web2.0与Web1.0最大的差异，是在于使用者在享用网际网络系统服务里，其所扮演的角色为何。其差异性是在于使用者对于网际网络系统服务的参与及信息交付制作分量加重，甚至把人和网际网络系统服务的角色互换。而系统可以依据用户的消费和体验的过程以及需求服务的请求，让服务系统学习到这些体验服务的经验累积，进而促进系统改善的依据，以使系统整体的服务效益放大。图1-1的内容即可简单的表示其两者的差异性。

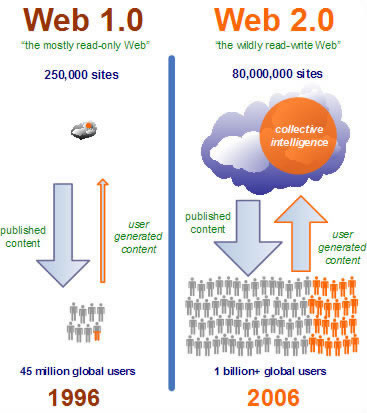


图1-1 Web1.0与Web2.0的差异性[1]

Web2.0的最大特征是Web作为平台，其特色为“互动”与“分享”，强调的是“双向互动”而非“单向传播”，“用户分享”而非“独断”，“集体智慧”而非“单一智慧”，进而产生创新的服务模式与价值链。

在Web2.0的概念出现后，对于网络内容服务化的趋势，配合创新的网络服务，并且强调“分享互动”与“使用者体验”的意识，使得各企业都致力于更为互动的技术，如AJAX[2]及RSS等，提供更为深度的使用者体验及服务。

AJAX（Asynchronous JavaScript and XML）的特点是：用户操作与服务器响应的异步化。用户访问以这种技术建设的网站时，通过异步模式向服务器发送并取回需要的数据，减少了服务器和浏览器之间交换的数据量，也减少了Web服务器的处理时间；同时由于该技术采用JavaScript处理来自服务器的响应，异步交互网络引擎运行在客户端，可以使很多的处理工作在发出请求的客户机完成，提高了应用响应的速度，降低了服务器的负载[3]。

由于该技术采用异步交互网络应用方式，符合Web2.0时代的需求，代表了互联网领域新技术的发展，在Web开发中运用AJAX技术的网站越来越多，Google Mail[4]，Yahoo! Mail[5]，Google Docs[6]都是非常著名的AJAX网络应用，我们称它们为“RIAs”（Rich Internet Applications，丰富互联网应用）。由于AJAX技术能够动态地改变客户端的页面内容，它给这些网站带来了更好的用户体验和新颖的UI设计。传统的网络应用如Amazon[7]和YouTube[8]也开始在自己的页面中加入动态的AJAX内容。

然而，AJAX应用程序是一把双刃剑，一方面给用户带来了高可交互性的应用程序，另一方面却造成搜索引擎没办法抓取页面的内容。主要表现在传统爬虫——单纯抓取静态页面的网络爬虫——不能爬取AJAX动态内容。一方面是大量基于AJAX的网站正在不断涌现，另一方面是这些网站的内容被搜索引擎忽略。这意味着越来越多有意义的数据将无法通过搜索引擎检索。最近几年，如何设计一种支持AJAX的网络爬虫，如何爬取这些网站，成为一个研究课题。

本论文所涉及的课题——“支持AJAX的定址网络爬虫的设计与实现”，正是在综合考虑上述发展趋势、需求因素的基础上提出的，设计并实现了支持AJAX的网络爬虫系统。此系统能够在AJAX网页的爬行过程中提取运行时的页面内容。从页面DOM树中识别用于页面导航的页面元素，自动触发页面元素上的事件，生成并提取新页面的内容，循环工作直到获取AJAX站点中全部页面内容并写入本地静态HTML文件中，从而形成AJAX站点的本地镜像。

1. 研究现状

人们对于网络爬虫的研究起始于Web的出现[Pinkerton 1994; Heydon and Najork 1999; Cho et al. 1998; Brin and Page 1998; Burner 1997]。近些年来，随着互联网技术的发展，一些Web数据被隐藏了起来，这些数据被称为Deep Web[9]（或称Hidden Web[10]、Invisible Web[11]）。根据文献[12]提出的观点，可以将隐藏数据分为两种：基于服务端的数据和基于客户端的数据，第一种隐藏数据是针对隐藏在form表单背后的数据，例如隐藏在Web数据库提供的查询接口后面，只有通过向查询接口提交查询才能获得，对于该类数据的获取已有广泛的研究[Raghavan and Garcia-Molina 2001; de Carvalho and Silva 2004; Lage et al. 2004; Ntoulas et al. 2005; Barbosa and Freire 2007; Dasgupta et al. 2007; Madhavan et al. 2008]。第二类数据主要是针对由于客户端脚本动态操作Web页面引起的，对于如何获取该类数据，还处在研究阶段。

国内在AJAX网络爬虫技术方面的研究还处于起步阶段[16]。罗兵[13]扩展了传统网络爬虫的功能。使其包含JavaScript代码解析、支持DOM操作的功能。通过解释和执行JavaScript代码，获取从服务器返回的内容，通过DOM支持模块修改页面内容，生成AJAX页面的内容，并从中提取新的超链接URL地址以获取更多页面。王映[14]、金晓鸥[15]在基于脚本语言分析的网络爬虫技术方面从事了一些研究，分别利用开源JavaScript引擎SpiderMonkey、Rhino解析JavaScript动态页面，从中提取并执行JavaScript代码，获取页面包含的所有URL地址，从而完成爬行任务。

国外在AJAX网络爬虫技术方面还处在研究阶段。Shah[17]描述了一种事件驱动的AJAX爬行方法，该方法基于三个核心功能，即实现AJAX功能的JavaScript代码解析、DOM事件处理、分发和动态DOM上下文抽取。该方法能够执行JavaScript代码并解析动态DOM内容，可以实现基本的AJAX爬行功能，但它并没有考虑AJAX应用的状态识别和状态空间遍历问题，且需要在人工指导下设定爬行路径。

G. Frey基于开源Java工具包Cobra和Rhino实现了一个对简单AJAX应用进行爬行、索引及重要性评分的实验系统[18]，该系统前端的AJAX爬虫首先对AJAX应用建模，然后通过执行所有的JavaScript事件逐步生成代表整个AJAX应用的所有状态及其关系的转换图。该系统的不足在于它仅处理了因执行客户端JavaScript而导致的状态变化，并没有考虑因接收服务器端异步响应而导致的状态变化，而后者在AJAX应用中非常普遍；此外，该系统通过直接比较DOM的hash值来判断是否产生了新状态，因而无法避免计数器、数字时钟等无关因素所产生的影响，可能造成状态爆炸。Matter[19]在Frey工作的基础上进行了改进，提出了一种启发式爬行算法，该算法将封装与服务器交互的XmHttpRequest（XHR）对象的JavaScript函数标记为热点。由于对同一个函数的每次调用都将导致同样的服务器端响应，算法通过缓存响应可以减少与服务器端通信带来的性能损耗。该算法的不足同样在于会面临状态爆炸问题。

Mesbah等人[20]设计了一个AJAX应用爬行和镜像站点生成系统，通过分析并重构用户界面状态的变化，动态推导出一个状态流图以对AJAX应用中各种导航路径和状态建模，并生成原始AJAX应用的多页面静态版本。该系统实现了一个全自动的爬行算法，首先尝试点击所有的DOM节点，通过比较页面DOM是否变化来判断是否产生了新的状态，然后在新的状态中重复上述过程，最终以深度优先方式遍历整个状态空间。

该算法的缺陷在于AJAX应用的状态变化可以由多种事件触发，而它仅模拟了点击等鼠标操作，这将造成状态丢失；而且该算法不能实现状态切换，需要回退到之前的状态时只能由初始状态开始逐步转换，这严重制约了算法在处理大型AJAX应用时的性能。

1. 论文研究工作

尽管本文主要聚焦于相似网页结构的AJAX动态信息采集，但本研究所设计的爬虫系统不仅可用于AJAX动态页面的采集，也可应用于静态页面的信息采集；不仅能够采集相似网页结构的页面信息，还可针对整个Web、目标站点或用户个性化的需求进行信息采集；不仅可以通过触发click事件采集分页数据（如用户评论内容），还可采集网页中任何需要触发事件（doubleClick, mouseOver等）才能显示的内容。

论文的主要工作包括以下几个方面：

1.分析了AJAX爬虫的技术难点，并针对AJAX问题领域进行建模。

2.设计并实现了支持AJAX的定址网络爬虫系统，详细介绍了系统中各个模块功能的具体实现与主要算法。

3.为测试系统准确度与性能设计了详尽的实验，包括针对三大类共六个真实网站做测试。

1. 论文架构

本论文总共分为七个章节，每一章节的内容如下：

第一章是绪论，内容包含研究背景、研究意义、国内外的相关研究现状和论文架构。

第二章是背景知识介绍，论述本研究所应用的相关技术概念和原理，包括Web2.0、JavaScript、AJAX、DOM、XPath等。

第三章是系统研究，首先介绍了传统网络爬虫的基本原理、工作流程以及爬行策略；然后详细讨论了支持AJAX的网络爬虫的设计难点，主要包括客户端JavaScript代码的执行、识别页面状态、DOM操作和复杂多变的事件处理等问题。

第四章是系统设计，从一个AJAX网站出发，分别介绍本研究的相关概念、问题模型。然后介绍了支持AJAX的网络爬虫系统的流程及架构设计及系统相关模块。最后详述了系统架构中各个包和类的功能与情况等。

第五章是系统实现，详述系统架构中各个模块功能的具体实现及主要算法。

第六章是实验与结果分析，介绍了支持AJAX的网络爬虫系统的实验设计，并对实验结果进行分析及讨论。

第七章是结论与未来研究方向，根据现有的知识与了解，提出本系统在未来有待深入探讨的地方与后续可能的改进方向。

第二章 相关背景知识

本章是针对本研究相关的概念与技术，进行整理与说明。此章有5小节，包括Web2.0、JavaScript、AJAX、HTML DOM、XPath相关技术探讨，依序说明如下。

## Web2.0

关于Web2.0的概念，是从O’Reilly与MediaLive讨论国际研讨会议题开始的。它并不是一项新的技术或是新的标准规范，连维基百科中的说明内容，也证实它只是一个新生名词而已。但这个名词的概念及其核心涵义到底有那些呢？我们将它的内容整理如下：

* 网络服务是一个平台，不再只是一个信息提供者
* 集合群体用户的智能与使用的体验
* 服务内容是下一个“Intel Inside”（例如Google Inside、Yahoo Inside或是Wiki Inside等等）
* 软件服务的升级或改版的模式终将结束，没有版本的界限，而是永远的BETA版本
* 轻巧的用户界面设计模式、开发模式及商业模式
* 服务的内容将超越单一装置或软件所提供的服务
* 丰富使用者的操作感受与体验
* 网站不能是封闭的——它必须可以很方便地被其他系统获取或写入数据
* 使用者应该在网站上拥有他们自己的资料内容
* 网站除了保留一些功能与机制的变动权限外，内容与呈现将由使用者自行提供和定义
* 完全基于Web Base——多数成功或较具名气的Web 2.0概念服务公司通常是透过浏览器来使用他们的服务内容
* 提供服务，而不是只是软件包，并能以符合成本效益的方式扩充
* 控制独特的、难以再制的资料或服务来源，随着越多人使用而累积越丰富的资料及新版本的升级
* 信任使用者为共同的开发者
* 使用者的自助服务及个人化服务的效能产生

Web 2.0的应用技术相对于Client/Server或是单机应用系统的技术，是比较复杂的，而且它还在继续发展中，但可以肯定的是包括服务器端的软件、内容、组织结构、协议、以及基于标准的浏览器和各种不同的客户端应用程序。这些提供了Web2.0信息存储、创建和分发的服务能力，这些内容远远超出了人们先前对网站的期望。换句话说，如果一个网站服务内容使用了以下各项技术或应用概念作为其网站服务的内容或核心表征，那我们就可以说它是利用了Web2.0技术的服务网站，这些技术包含下列各项：

* 内隐式的丰富应用技术（例如AJAX、XFORM）
* CSS，语义化有效的XHTML标记，和Microformats(微格式)
* 信息订阅与发布，RSS/ATOM与SMTP指定信息发布
* RSS/ATOM数据的Mashup(聚合)
* 规则且有意义的URL(Web Service)
* Blog服务及应用及社群性服务内容
* REST或者是XML Web服务API
* Mashup混搭应用
* 提供Widgets、gadgets小工具服务或Mashup这些小工具的服务应用

## JavaScript

JavaScript是一种轻型的、解释型的脚本语言，具有面向对象能力。在句法构成上，JavaScript的核心与C、C++和Java相似，都具有诸如if语句、while循环和服运算符这样的结构。但是，JavaScript与这些语言的相似之处也仅限于句法上的类同。解释语言的弱点是安全性较差，而且在JavaScript中，如果一条执行不了，那么下面的语言也无法执行。而且由于每次重新载入都会重新编译，载入后，有些代码会延迟至运行时才解释，甚至会多次解释，所以速度较慢。

把JavaScript解释器嵌入Web浏览器就形成了客户端JavaScript，这是迄今为止最普通的JavaScript变体。当人们提到JavaScript时，通常所指的是客户端JavaScript。

客户端JavaScript将JavaScript解释器的脚本化能力与Web浏览器定义的文档对象模型（DOM）结合在一起。这两种技术是以一种相互作用的方式结合在一起的，产生的效果也大于两部分能力之和，即客户端JavaScript使得可执行的内容散布在网络中的各个地方，它是新一代动态DHTML（Dynamic HTML）文档的核心。这样一来，网页就不再是静态的HTML了，而是包含与用户进行交互、控制浏览器以及动态创建HTML内容的程序。

虽然更常用于网页，但JavaScript也可以用于其他场合，比如服务器端编程，而微软为了取得技术优势，推出了Jscript来迎战JavaScript的脚本语言。基于互用性，ECMA[21]建立了ECMA-262标准（ECMAScript），现在两者都属于ECMAScript的实作，尽管JavaScript作为给非程序人员的脚本语言，而非作为给程序人员的程序语言来推广和宣传，但是JavaScript是一门具有非常丰富特性的语言，它有着和其它程序语言一样的复杂性，或更甚复杂。实际上，技术人员必须对JavaScript有扎实的理解才能用它来撰写比较复杂的程序。

## AJAX

AJAX是**A**synchronous **J**avaScript **A**nd **X**ML的简写，它的原理非常类似Dynamic HTML（或者DHTML）,主要的目的在于提高网页的互动性（interactivity），速度（speed），以及可用性（usability）。我们在此对它的技术背景和运作方式作详细的说明。

### 2.3.1 AJAX简介

AJAX是一种客户端技术，它不需要回传就能进行服务端处理，因而使客户端（浏览器）具有丰富的服务端能力。例如像地图这种需要传输大量资料的应用，浏览器与服务器之间需要传输大量的图形资料，如果利用传统的同步方式，在全部的图形下载完成之前，我们没办法对网页进行操作，而使用者点选往左、往右的按钮后，又得等待另一次的图形传送。反之，使用AJAX技术的网页可以在使用者操作的时候，非同步的下载其他图形，这样速度会变快，也不必一再的重复刷新网页。

其实，这个概念并不是很新，只是这样的技巧大量的应用于Google的网页之后，如Gmail、Google Maps和Google Suggest等，才被重视，Google Maps给用户呈现出可以任意放大、缩小和移动图片的体验；Google Suggest能够根据你的输入立刻显示建议条目，而不需要更新页面。以往若要在使用者输入资料后立即检查，就必须借助客户端电脑的JavaScript程序；若要由服务器端检查，就必须重新更新整个网页；而透过AJAX，则可以在不需重新读取网页的情况下，由服务器端来检查。当AJAX引擎接收使用者全部或部分的表单数据之后，AJAX引擎会跟服务器连接并由服务器处理完成，并将完成的资料回传给AJAX引擎，然后再由AJAX引擎将成果动态的呈现在网页上。这些过程，使用者完全不会感受到，而且由于是非同步的，网页不会停止接收使用者的操作。

AJAX也不是一项单独的技术，它是由一堆现有的技术所组成，它的组成技术有[2]：

（1）使用XHTML和CSS来展现网页内容。

（2）使用JavaScript操作DOM进行动态显示和交换信息。

（3）使用XML和XSLT进行数据交换及相关操作。

（4）利用XMLHttpRequest对象与Web服务器进行异步的通信。

（5）使用JavaScript将所有的东西绑定在一起。

### 2.3.2 AJAX的运作方式

首先，在跟远端服务器要求数据之前，我们需要建立XMLHttpRequest对象。AJAX是利用JavaScript的XMLHttpRequest对象与服务器建立HTTP连接，并取得回应。表2-1是建立XMLHttpRequest对象的方法。

表2－1 建立XMLHttpRequest对象的方法

|  |
| --- |
| var xmlHttp;  function createXMLHttpRequest() {  //支持ActiveX的浏览器  if (window.ActiveXObject) {  xmlHttp = new ActiveXObject(“Microsoft.XMLHTTP”);  }  //不支持ActiveX的浏览器  else if(window.XMLHttpRequest) {  xmlHttp = new XMLHttpRequest();  }  } |

建立了XMLHttpRequest对象之后，我们就可以利用它的属性和方法来向服务器发出请求。表2-2为XMLHttpRequest对象的属性，而表2-3为它可以使用的方法。

表2－2 XMLHttpRequest对象的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| onreadystatechange | 每个状态都会触发这个事件处理器，通常会调用一个JavaScript函数 |
| readyState | 请求的状态，有5个可取值：  0：未初始化  1：正在载入  2：已载入  3：交互中  4：完成 |
| responseText | 服务器返回的文本 |
| responseXML | 服务器返回的XML对象，可以表示为一个DOM对象 |
| status | 服务器返回的HTTP状态码（200对应OK，400对应Not Found,等等） |
| statusText | HTTP状态码的相应文字 |

表2－3 XMLHttpRequest对象的方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| abort() | 停止当前请求 |
| getAllResponseHeaders() | 把HTTP请求的所有回应标头作为键值返回 |
| getResponseHeader(“header”) | 返回指定标头的字串值 |
| open(“method”,“URL”) | 建立对服务器的呼叫，并设置请求目标URL、方法和其他参数 |
| send(content) | 向服务器发送请求 |
| setRequestHeader(“header”,“value”) | 把指定标头设定为所提供的值 |

如果配合HTML的innerHTML属性，和responseText结合，就可以再HTML的网页上显示出XML文件的文字。

### 2.3.3 AJAX与传统网页的比较

用户在浏览传统的网页时，背后的流程模型如图2-1所示：



图2－1 传统网页的流程模式

用户在网页的操作时，经由鼠标或键盘动作，触发一个HTTP的请求服务，传给后端的网页服务器，服务器接受到这个请求后，根据数据与程序的设定完成逻辑处理后，例如与其它服务器运算或交换数据等，最后将结果以HTML网页的格式回传到客户端。这种处理流程对于大量使用HTML撰写的静态网页相当有用，但对于动态网页却相当不方便，因为当后端的服务器在处理数据时，用户处于等待的状态，直到服务器将结果回传客户端，使用者才知道结果。

这个问题在于使用者只是为了验证一笔网页中的数据，或是单纯地想知道更一进的内容，但是，他必须等到服务器处理完毕后，才知道资料正确与否或是不是他所想要知道的内容。如果每个请求都需要刷新页面或换页，使用者会因为冗长的等待，而失去耐性，进而不再造访该网站。

这种差异，在处理流程图中的时间轴看出，使用者在传统网页中所触发的每个动作，必须等待服务器处理完成后回传，才能操作下一步的浏览动作，也就是浏览器与服务器循序地交谈，造成两者的数据处于同步状态。

但在AJAX网页中，使用者发出HTTP请求，都经过AJAX引擎与服务器交谈，此时，使用者可以继续操作页面中其它动作，因此页面的呈现与处理过程，借由AJAX层的代理，而能不同步运作。AJAX引擎作为服务器与浏览器间的中介层，主要以JavaScript撰写而成。

当一个会话开始时，浏览器会先加载AJAX引擎，负责处理用户接口，以及与后端的服务器交换数据。从图2-2中可以看出，AJAX引擎与服务器交谈时，都是透过XML格式，所以处理完成的数据，不会直接回传HTML格式的网页给浏览器，而是先回传XML格式到AJAX引擎，再经由它解析成HTML格式呈现在浏览器中。因此，使用者除了不需要等待处理结果回传才能决定下一个动作外，也不需要刷新页面才能浏览更多的内容，AJAX引擎代理这些动作，并让所有过程都在同一页面中完成，所以称为“异步”。使用者浏览网页，与浏览器和服务器间交换数据间的动作是分开处理的，最明显的是，用户不会看到鼠标指针呈现沙漏的等待状态。



图2－2 AJAX的运作方式

## HTML DOM

DOM[22]是文档对象模型(Document Object Model)的缩写，它的相关标准是由W3C(World Wide Web Consortium)制订出来的，它是一种跨平台的应用程序接口(API)，允许程序设计者利用脚本语言进行动态存取及更新HTML及XML文档的内容和结构。

DOM有两种，HTML DOM和XML DOM。HTML DOM是一种特殊的DOM，它仅支持使用getElementById和getElementsByTagName两个方法来进行查询，而XML DOM则可以与XPath API相结合，基于强大的正则表达式来进行查询。AJAX开发中所用到的主要是HTML DOM。

HTML文档是以文字格式来储存的档案，当一份HTML载入浏览器之后，经过DOM的处理，就会自动地将该份HTML文件内的元素标签、属性、属性值及注解进行分解，然后将分解后的结果编成一个树状结构，存入内存中。利用DOM的方法，可以控制HTML中的节点，来进行新增、删除或更新节点的应用，程序的开发者就能够根据这样的方式处理该份HTML文档。

### 2.4.1 DOM的树状结构

一份HTML文件经由DOM处理后，会呈现规则的树状结构，在此我们透过表2-4的范例来说明：

表2－4 HTML文件范例

|  |
| --- |
| <html>  <head>  <title>文档标题</title>  </head>  <body>  <a href = ”next.html”>我的链接</a>  <p>我的内容 </p>  </body>  </html> |

这个文件经过DOM处理之后会在浏览器内存中形成图2-3的树状结构图。



图2－3 DOM树状结构图

### 2.4.2 DOM常用的方法

根据DOM，HTML文件中的每个成分都是一个节点。可以通过JavaScript来访问这棵DOM树，遍历树上的节点、动态添加、删除树上的节点、设置或修改某个节点的样式、设置或修改某个节点中保存的数值等等。通过JavaScript对于这棵DOM树所做的任何修改都会立即生效，JavaScript不能控制浏览器重新呈现DOM树的时间，因为浏览器重新呈现DOM树是一个非常耗时的操作，所以应该尽量减少做这种操作的次数。表2-5列出了DOM元素的一些属性，表2-6列出了一些有用的方法。

表2－5 处理HTML文件的DOM元素属性

|  |  |
| --- | --- |
| 属性名 | 描述 |
| parentNode | 返回元素的父节点 |
| firstChild | 返回目前元素的第一个子元素 |
| lastChild | 返回目前元素的最后一个子元素 |
| nodeName | 返回节点的名称 |
| nodeValue | 返回节点值 |
| nodeType | 返回节点的类型 |

表2－6：处理HTML文件的DOM元素方法

|  |  |
| --- | --- |
| 方法名 | 描述 |
| getElementById(id) | 取得有指定唯一id属性值文件中的元素 |
| getElementsByTagName(name) | 返回目前元素中有指定标签名的子元素的列表 |

## XPath

XPath是XML路径语言（XML Path Language）的缩写，它用于定位XML文档中的节点。有关XPath的详细标准，请参加[23]。

我们利用XPath表达式在DOM树中定位节点，比如执行该表达式“/html/body/a”以后返回值为“我的链接”。（图2-3）

## 本章小结

本章主要是探讨本研究所应用的相关理论及技术，包括Web2.0、JavaScript、DOM、AJAX、XPath等。这些内容都是本文对支持AJAX的网络爬虫系统展开研究与实现的理论基础。

第三章 网络爬虫系统的研究

作为搜索引擎关键技术之一的网络爬虫（网络蜘蛛、机器人），能够沿Web页面中的超链接在万维网中自动漫游，并读取漫游的Web页面的程序。网络爬虫的漫游是指根据HTML文档中的超链接，利用HTTP协议连续地从不同的服务器中获取Web页面的过程[13]。

## 网络爬虫

### 3.1.1基本原理

网络爬虫是通过网页的链接地址来寻找网页，从网站某一个页面（通常是首页）开始，读取网页的内容，找到在网页中的其他链接地址，将这些链接全部放到一个有序的待采集队列中，而网络爬虫从这个队列里按序取出URL，访问URL所指向的页面，然后从这些页面中提取出新的URL，并将它们继续放入待采集队列中，这样一直循环下去，直到网络爬虫根据自己的策略停止采集。对于大多数的网络爬虫来说，到此就算完结；而对于有些网络爬虫而言，它还要将采集到的页面数据和相关处理结果存储、索引并在此基础上对内容进行语义分析[43]。网络爬虫采集的网页包括多种格式，如HTML、图片、doc、pdf、多媒体、动态网页等。

其工作流程如图3-1所示。



图3－1 传统爬虫的工作流程

### 3.1.2爬行策略

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。



图3－2 爬行策略

（1）广度优先搜索

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[eTesting L.，2000]。这种方法的特点是可以让网络爬虫并行处理，提高其抓取速度。对于图3-2，广度优先的抓取顺序是A-B、C、D、E、F-H、G-I。

（2）深度优先搜索

深度优先是指网络爬虫会从起始页开始，一个链接一个链接抓取下去，处理完这条线路之后再转入下一个起始页，继续抓取链接。这个方法的优点是网络爬虫在设计时比较容易，但是会导致爬虫的陷入问题。对于图3-2，深度优先的抓取顺序是A-F-G、E-H-I、B、C、D。

（3）最佳优先搜索

最佳优先策略[24]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。

由于不可能抓取所有的网页，有些网络爬虫对一些不太重要的网站，设置了访问的层数。例如，在图3-2中，A为起始网页，属于0层，B、C、D、E、F属于第1层，G、H属于第2层，I属于第3层。如果网络爬虫设置的访问层数为2的话，网页I是不会被访问到的。这也让有些网站上一部分网页能够在搜索引擎上搜索到，另外一部分不能被搜索到。对于网站设计者来说，扁平化的网站结构设计有助于搜索引擎抓取其更多的网页。

## AJAX爬虫的技术难点

传统网络爬虫获取页面源码并提取URL的工作方式不适用于爬行AJAX页面，主要存在以下困难：

### 3.2.1JavaScript的解析

由于AJAX网络应用使用AJAX引擎作为服务器与浏览器间的中介层，并由其负责处理用户接口，以及与后端的服务器交换数据。AJAX是使用JavaScript发挥作用，而通用的搜索引擎（如Google、Bing）不支持JavaScript，因此搜索引擎机器人看不到AJAX发送的内容。比如，如果一个网站导航采用AJAX技术制作，那么搜索引擎无法通过导航系统深入抓取该网站首页以外的其它链接页面，同样，如果网页内容由AJAX生成，也无法为搜索引擎识别。

关于JavaScript解析，即动态内容的提取，主要存在如下两种解决方案[25]：一是使用完整的开源浏览器(例如早期的Netscape Navigator，以及Mozilla)渲染整个网页，从浏览器的输出结果中提取页面中的动态内容；二是自行构建脚本解析环境，利用开源浏览器项目中的脚本解释引擎，实现相关脚本片段的解析，从而获得动态页面内容，参见文献[14]。

本研究使用第一种解决方案，利用WebKit渲染引擎实现的浏览器加载、解析并渲染网页，执行客户端的JavaScript代码，从而获取页面中的动态内容。

### 3.2.2识别页面状态

在传统的Web应用程序中，用户的每一次请求都对应一个页面的响应，也就是说每一个URL标志着一个静态页面，每一个响应的页面实际上记录着每一次的状态信息。但是，使用AJAX技术的网站通常若干个页面共用一个URL，每一个状态没有对应一个明确的URL。考虑一种极端的情况，一个AJAX站点只存在着一个具有独立URL的网页，在此页面上的所有操作都将动态的修改此网页的内容，而保持URL不变。因此，这种状态变化并不能通过页面的URL体现出来，而是通过DOM结构的动态变化来展现。

另一方面，静态页面内部包含的超链接以URL的方式直接嵌入页面源文件的HTML标签中。传统爬虫可以使用HTML标记识别的方法[3]，实现页面所含超链接网络地址的提取功能。然而，在AJAX动态页面中，除了包含少量静态URL外，大量的超链接被包含在AJAX动态内容中，必须通过浏览器执行脚本才能得到，并且状态之间的转变依赖于事件的触发，仅仅依赖通过网页中的内嵌超链接地址来寻找其他网页并获取资源的方式无法适用于AJAX网站。AJAX爬虫需要模拟用户操作以触发事件，并利用DOM结构来标志页面状态。

### 3.2.3 DOM操作

AJAX利用JavaScript绑定一切，包括各个元素的事件触发，使用 XMLHttpRequest对象与服务器进行异步通信，同时利用服务器返回的信息动态地对DOM进行更新。这种动态的内容更新机制导致了一些资源对于传统爬虫来讲是透明的。对使用了AJAX技术的网页源码进行分析，发现嵌入网页源码的JavaScript中往往包含修改DOM树的语句，而AJAX正是利用它们动态改变页面内容的呈现。因此爬虫不仅要能够解析客户端的JavaScript代码，还要加入支持DOM操作的部分，利用DOM特有的方法修改页面元素内容，如通过ID属性或Xpath路径获取页面元素并对其进行管理，从而得到与页面呈现基本一致的页面内容。

### 3.2.4 触发事件

分页是指目标网站上的某块数据分为好几页显示，一个HTML页面所能显示的数据是有限的，大量有意义的数据隐藏在分页中，而由于AJAX技术的广泛采用，传统的翻页链接不复存在，取而代之的是JavaScript/Js动态生成的超链接，也许根本就不是超链接，而是用户点击触发了一段JavaScript/Js代码，在代码中使用XMLHttpRequest等通信函数动态的下载分页内容，完全依靠从现有HTML页面中匹配出超链接的其他网络爬虫和网站信息抓取软件无法应对这种情形，爬虫必须能够模拟用户点击DOM节点触发上述与网站服务器的通信，动态地获取分页内容。

表3-1展示了在AJAX网页中打开下一页的不同方式，事件处理函数（如onClick，onMouseOver）被绑定到DOM元素上，例如在表3-1中，一个DIV元素（第三行）由于被onClick事件绑定从而变成了一个clickable元素，因此当用户点击该元素时，onClick事件被触发，从而修改DOM树的内容。

第2行中A元素的链接地址为“#”，AJAX请求被绑定的onClick事件捕获。在URI规范中“#”叫做hash fragment，此hash fragment不作为URL请求的一部分。所以搜索引擎的爬虫无法对A元素的href值产生有效的URL地址，如果要产生有效的地址也就是当前页面的URL地址。又因为此URL已被抓取过，所以搜索引擎的爬虫放弃了这个URL，AJAX中的数据无法被搜索引擎收录。

AJAX技术实现异步操作的关键是XMLHttpRequest对象。该对象用于向服务器端异步发送请求，并接收服务器端的响应以进行局部刷新。除了直接使用 XMLHttpRequest对象的AJAX开发技术外，还有很多封装了XMLHttpRequest对象的第三方AJAX框架和库可以用来开发AJAX应用[26]。第三方AJAX框架除了以属性的形式将事件绑定到DOM元素节点外，还可能实现框架相关的事件附加模式。第4-13行使用jQuery[27]代码给class属性为“next”的DIV元素附加一个click事件处理函数。当这个元素被点击时，它会获取下一页的内容并且动态的注入到id属性为“content”的SPAN元素中。

表3－1 DOM元素绑定事件的多种方式

|  |
| --- |
| 1. <a href=” javascript:OpenNextPage();”> 2. <a href="#" onClick="OpenNextPage();"> 3. <div onClick="OpenNextPage();"> 4. <div class="next"/> 5. <span id=”content”> 6. <!-- jQuery function attaching events to elements having attribute 7. class="next"，The contents of next page are injected into the SPAN element 8. --> 9. <script> 10. $(".next").click(function() { 11. $("#content"). load("next.html"); 12. }); 13. </script> |

## 本章小结

本章首先介绍了网络爬虫的原理、工作流程以及现有爬虫使用的抓取策略。接着详细阐述了AJAX网络爬虫的技术难点，主要包括JavaScript代码的执行、识别页面状态和DOM事件处理等问题。

第四章 网络爬虫系统的设计

本章节将从一个AJAX网站出发，分别介绍本研究的相关概念、问题模型、系统设计流程与系统架构，并且通过本研究所建立的系统流程与架构来进行第五章的系统实现。

## AJAX应用实例

微博，即微博客（MicroBlog）的简称，是一个基于用户关系的信息分享、传播以及获取平台。用户可以借助Web、Wap、Mail、APP、IM、SMS以及各种客户端，通过文本、多媒体、图片、影音等多种形式更新、发布、传递消息。微博的本质是信息传播的一种方式，是基于Web2.0技术的社会化媒体传播。同时，它又和其他的Web2.0的信息传播方式不同，它的交互更具便捷性、自由性、时效性、互动性、扩散性和增值性[28]。

相对于其他社会性媒体，微博虽然很年轻，但是发展十分迅猛，近期接连发生的重大事件，例如华为接班人事件和[蒙牛诽谤门事件](http://www.gooseeker.com/cn/node/Fuller/2010102201)都是由微博引爆的。因此凡是忽视微博采集和跟踪的网络舆情监测系统都有全面性缺陷。

微博网站同早期的社会性媒体有很大的不同，更像一个综合应用软件，而不像传统的网站，也就是所谓的胖客户端或富媒体系统，上述特征有赖于 JavaScript/AJAX技术。但是，JavaScript/AJAX向来是网络数据采集和网络爬虫的克星，除非数据采集系统具有AJAX解析能力，普通的网络爬虫和全文搜索引擎难于获得AJAX管理的数据。本文将以采集新浪微博为例说明此爬虫系统如何克服AJAX数据采集困难。

新浪微博目前是国内排名第一的微博产品，截止2012年5月底，新浪微博注册用户已超过3亿。其主要特点在于力推名人圈，通过庞大的认证名人资源使得舆论领袖效应明显，目前基本已经覆盖大部分知名文体明星、企业高管和媒体人士。图4-1为个人微博主页（http://weibo.com/yaochen）的界面。



图4－1 新浪微博

此微博页面上含有许多重要的信息，如图4-1中红线框住的部分：个人信息、个人介绍、微博内容、发表时间、转发数及评论数。在浏览器中访问此微博，右键查看网页源代码，很容易发现源代码里的内容远少于页面呈现的内容，在源代码中搜索某条微博内容，比如“早上好”，将发现其在源代码中不存在。转发数与评论数也同样未在源代码中出现，如图4-2所示，由此可知此页面内容都是通过后加载方式利用DOM操作动态呈现出来的，用户察觉不出来。

除此之外，新浪微博还采用自动加载分页的机制，加载两页之后才会显示分页导航，这时用户才可手动翻页。也就是说，当我们拖动滚动条到页面底部时，网页中会短暂的显示“正在加载中”字样，如图4-2所示，随后更多的微博数据被自动加载进来，并且再次向下拖动滚动条时，这种情况还会出现一次，直到第三次将滚动条拖至网页底部时，才会看到分页导航，如图4-3所示。

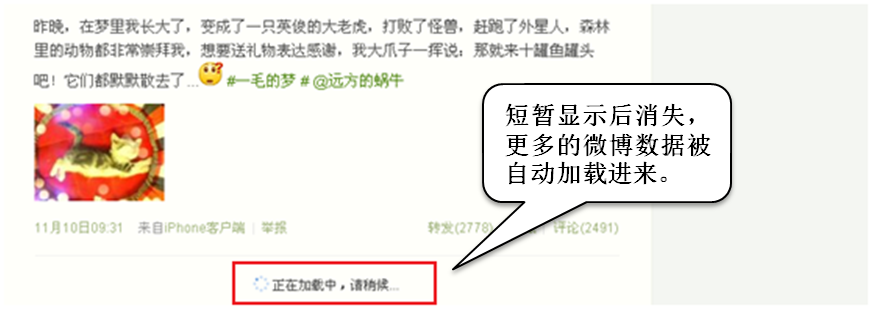


图4－2 新浪微博自动加载分页



图4－3 新浪微博分页导航

分页加载之前的源代码片段如下：

<div class="W\_loading" node-type="lazyload">

<span>正在加载中，请稍候...</span>

</div>

由以上的例子可知，微博是一种新媒体和通信工具，上面有大量[值得挖掘的信息](http://www.gooseeker.com/cn/node/Fuller/2010103001)，首先需要将微博的内容抓取下来。但是，从微博网站采用的编程技术来说，抓取微博又有很多障碍，最大的障碍是基于Javascript/JS的AJAX程序框架，传统的HTML标记识别方法无法完成动态页面主体内容及其所含超链接网络地址的提取工作。导致网络爬虫很难在微博网站上爬行和抓取数据，抓取的内容是不完整的，这对于搜索引擎是不利的。

## 相关概念

在本论文中，我们使用下列定义。

### 4.2.1 页面状态与状态切换

在传统的Web应用程序中，每一个URL都有一个页面与之对应，页面状态可以由这个URL和它指向的页面表示，页面状态的变化通常是由URL的改变引起的。但是，在AJAX应用中，一个URL通常对应多个页面，状态变化并不能通过页面的URL体现出来，而是通过DOM树结构的动态变化来展现的。一个状态是指一个AJAX页面或者同一个页面在不同时间段的一个镜像，从一个状态到另一个状态通常表现为一个跳转．通过“上一页”和“下一页”，状态可以实现前驱和后续的变化。AJAX技术使用DOM树动态存储AJAX页面的状态信息．来管理AJAX页面的前驱和后续状态。

### 4.2.2 clickable元素

在客户端，用户可以通过浏览器与网页进行交互，例如：点击页面上的一个元素，双击一个元素，将鼠标悬停在某一元素上，等等。这些动作有可能触发事件，引起页面状态的变化。我们称这些可引起页面状态变化的DOM元素为clickable元素[20]。方便起见，在本论文中我们用点击事件代表其他的事件类型，对于其他的事件类型的处理方法与之类似。

## AJAX建模

### 4.3.1 事件模型

AJAX使用JavaScript与用户动态交互，响应客户端的发出的事件，例如：click，doubleClick，mouseOver等。

一个绑定了事件监听器的DOM元素可用一个二元组<t，x>表示：

（1）t为此元素的事件类型：click, doubleClick, mouseOver等。

（2）x是一个XPath表达式，用于表示此元素在文档中的位置路径。

上一章中表3-3中第3行的元素可以表示为<click,//div[1]>。

### 4.3.2 AJAX页面模型

通过上述所提，一个AJAX页面可以由其中的页面状态和状态之间的迁移构成，我们使用一个状态转换图记录这些状态及状态迁移，其定义如下：

一个AJAX网页A的状态转换图G是一个规范的有向状态图，是节点与边的集合，每一条边都有一个标签，用一个四元组<r,V,E,L>表示如下：

（1）r是根节点（初始节点），代表网页A被浏览器全部加载完成后的初始状态。

（2）V是节点的集合。每一个v∈V代表A中的某个运行时刻DOM树的镜像状态。

（3）E是节点之间有向边的集合。对于任意(v1,v2)∈E，当且仅当通过触发DOM元素节点c绑定的事件，可由状态v1到达v2，我们说元素c连接了这两个状态。

（4）L是有向边上的标签，标注了这条边代表的元素c的事件模型。



图4－4 状态转换图

图4-4是一个简单的状态转换图，显示了用户通过点击网页中的“prev”和“next”按钮，触发相应的click事件，引起页面状态的变化。其中状态1是初始状态节点，节点之间的有向边上的标签表明了要引起状态转移需要哪个DOM元素节点以及触发什么类型的事件，这里用上一节提到的事件模型表示，如<click,//DIV[1]/A[2]>表示点击路径为//DIV[1]/A[2]的元素将引起网页由状态1变为状态2。



图4－5 简化的状态转换图

在图4-4中，状态1可以通过点击“next”到达状态2，在状态3点击“上一页”也可以到达状态2，即在不同状态，或相同状态下通过不同的事件触发可能导致相同状态。为了避免重复状态产生，本研究通过消除冗余边形成一个有向无环图，如图4-5，是消除了“prev”边的简化后的状态转换图。

### 4.3.3 AJAX网站模型

相比于传统网站，AJAX网站既有静态内容也有动态内容，每一个页面也都有指向其他页面的超链接，如图4-6所示。传统网站只是一系列通过超链接联系的页面跳转图，而AJAX网页与传统网页的最大区别就是用户可以在同一个页面下触发事件（比如“prev”与“next”），产生不同的状态[29]。为了便于区分，将不改变URL的事件驱动跳转，定义为AJAX Links[19]。

图4－6 AJAX网站模型

## 系统架构

本系统将传统网络爬虫工作过程中的URL提取和下载网页两个过程分离开，成为两个独立的功能模块：URL抽取模块和网页抓取模块。URL抽取模块从一个初始URL开始，循环抓取网页中符合特定格式的链接地址，并将它们放入URL资源库中。网页抓取模块的核心是Webkit浏览器，只采集URL资源库中链接地址导向的网页信息，也就是说爬虫的爬行范围完全由URL资源库限定，是受控的，因此，我们称之为“定址”的网络爬虫。

用户通过配置独特的URL抽取策略，指导URL抽取模块在工作过程中过滤哪些无效链接及保存哪些格式的有用链接，形成个性化的URL资源库。URL资源库指导网络爬虫采集其中的URL地址页面。根据URL资源库中URL的覆盖范围，可将爬虫划分为针对整个Web的信息采集、针对目标站点的信息采集和针对符合特定网页结构的页面信息采集等。当抽取的URL覆盖整个Web时，网络爬虫的行为同通用爬虫（General-purpose Web Crawler）一样，是一种基于整个Web的信息采集。当抽取的URL都是属于同一站点的URL时，爬虫只爬行属于该站点的网页。当抽取的URL的网页结构都相似时（如这些网页都是个人微博主页），是针对特定网页结构的信息采集。除此之外，本研究设计的网络爬虫既可用于静态页面的信息采集，也可应用于AJAX动态页面的采集。

本研究主要聚焦于相似网页结构的AJAX动态信息采集。通过样本页面和脚本生成模块，自动生成匹配此类网页结构的翻页脚本。爬虫在爬行过程中，浏览器与翻页脚本相互协作，实现分页数据（如用户评论内容）的采集。

综上所述，本研究设计的网络爬虫具有较高的灵活性与适用性。图4-7为本系统架构图。

图4－7 支持AJAX的网络爬虫系统架构图

其中各个组件的介绍如下：

Webkit浏览器：Webkit浏览器访问URL，加载、解析并渲染网页，执行对应此网页结构的JavaScript翻页脚本，获取分页内容。

链接抽取模块：链接抽取模块用来抽取AJAX网站模型（图4-6）中不同页面之间实现跳转的超链接。链接抽取模块按照一定的字符串过滤规则抽取网页中匹配特定格式的超链接，在抽取过程中使用广度优先搜索方法，在完成当前层次的搜索后，才进行下一层次的搜索。当抽取的URL到达指定数量后停止抽取。

URL资源库：URL资源库是一个文本文件，用来存储链接抽取模块抽取到的链接。一个URL资源库存储的URL有着相似的网页结构，通常属于同一网站。

脚本生成模块：通过用户在网页内容上的标注及相关的参数设置，脚本生成模块自动生成针对某一[样本页面](http://www.gooseeker.com/cn/node/document/terms/sample)的翻页脚本文件，由于样本页面的结构代表了同一主题下的所有页面结构，所以，为样本页面定义的翻页规则可以普遍用于所有同一主题的页面翻页任务。

翻页脚本：翻页脚本是一段JavaScript代码，以JS格式保存为一个独立的文件。这段代码在Webkit浏览器中执行，用来在网页中定位clickable元素并模拟点击操作，从而实现翻页的效果。

Cookie管理：Cookie管理模块负责Cookie的管理工作，主要包括读取、保存和清除操作。

cookies：用于将Cookie保存在本地。

本系统主要开发了以下三个包，对这几个包和其中的类具体介绍如下：

core包中主要包含以下几个类：

1）HttpGet为链接抽取模块的核心代码，按照一定的字符串过滤规则抽取网页中匹配特定格式的超链接，在抽取过程中使用广度优先搜索方法，在完成当前层次的搜索后，才进行下一层次的搜索。当抽取的URL到达指定数量后停止抽取。

2）JsGenerator为脚本生成模块的核心代码，根据用户在网页内容上的标注及相关的参数设置，脚本生成模块自动生成针对某一[样本页面](http://www.gooseeker.com/cn/node/document/terms/sample)的翻页脚本文件；

3）MyCookieJar为Cookie管理模块的核心代码，对Cookie进行管理，主要包括从Cookie文件读取Cookie到浏览器内存、保存Cookie至本地文件和清除Cookie等操作；

4）BrowserMainWindow负责控制Webkit浏览器的行为和执行流程，如通知Webkit访问URL，当网页加载完毕后，保存网页文件并执行对应此网页结构的JavaScript翻页脚本，获取分页内容。

ui包中主要包含以下几个类：

1）HttpGetPage类为链接抽取模块的界面类。

2）JsCreatePage类为脚本自动生成模块的界面类。

3）CrawlPage类实现了批量抓取界面。

4）ConfigPage类为浏览器和Cookie的配置与管理界面。

5）Crawler类为主界面类，提供了一个界面框架，引用了以上四个界面。

6）CrawlerImpl类为启动类，包含了程序的入口方法main函数。

utils包中主要是文件操作的实用工具类。

## 系统流程

本文所设计的网络爬虫能够在AJAX页面的网络爬行过程中提取运行时的页面内容。从页面DOM表示中识别用于页面导航的页面元素，自动触发页面元素上的事件，生成并提取新页面的内容，循环工作直到获取AJAX站点中全部页面内容并写入本地静态HTML文件，形成AJAX站点的本地镜像。具体的系统流程如图4-8所示：



图4－8 支持AJAX的网络爬虫系统流程

爬虫整体工作流程描述如下：

首先，爬虫读取某AJAX网站的种子URL，并将其交由Webkit浏览器处理。Webkit浏览器加载此URL指向的页面，解析JavaScript代码并构造DOM树结构，完成页面中动态内容的获取后，将此页面内容写入本地静态HTML文件。

然后，爬虫调用抽取链接模块，后者将按照一定的字符串过滤规则抽取此页面中匹配特定格式的超链接，并将抽取出的新的链接放入URL资源库中。接着，Webkit浏览器在其自身环境中运行一段由脚本生成模块生成的JavaScript代码，这段JavaScript代码用来在页面中查找clickable元素并模拟鼠标点击的操作，从而实现翻页的效果，此后Webkit浏览器会继续加载分页内容，并按照保存分页内容、抽取链接、放入URL资源库、执行JavaScript代码的流程循环执行，直至到达尾页，表示翻页结束，此时爬虫从URL资源库中取出下一个URL，循环工作直至满足一定的停止条件，例如URL资源库中已经没有未爬取的URL或爬取的URL已经到达一定数量。

最终，爬虫通过上述工作，得到反映该AJAX网站模型（图4-6）的本地镜像，即一系列静态HTML文件，使网站模型中的每一个页面状态都有一个静态HTML文件与之对应。

## 本章小结

本章在第三章中网络爬虫系统的研究基础之上，对系统进行了设计。

首先，本章从一个AJAX网站出发，分别介绍了本系统的相关设计概述、问题模型，包括事件模型、AJAX页面模型及AJAX网站模型。

然后，本章大致介绍了支持AJAX的爬虫系统的流程和架构，包括各个模块、包和类的功能与情况等。

第五章 爬虫系统的实现

通过上一章节对支持AJAX的网络爬虫的系统设计，本章节将详细介绍系统架构中各个模块功能的具体实现及主要算法。

## WebKit渲染引擎

浏览器主要由以下三个部分组成：

* 渲染引擎

渲染引擎（Rendering Engine，俗称浏览器内核），也称浏览器的排版引擎，是浏览器最重要或者说最核心部分，它负责取得网页的内容（HTML、XML、图像等）、整理信息（例如加入CSS等），以及计算网页的显示方式，然后会输出至显示器或打印机。所有网页浏览器、电子邮件客户端以及其它需要编辑、显示网络内容的应用程序都需要渲染引擎。渲染引擎决定了浏览器如何显示网页的内容以及页面的格式信息。不同的浏览器内核对网页编写语法的解释也有不同，因此同一网页在不同渲染引擎的浏览器里的显示效果也可能不同。

* JavaScript引擎

JavaScript引擎是一个专门处理JavaScript代码的部件，JavaScript是一种广泛用于客户端Web开发的脚本语言，常用来给HTML网页添加动态功能，比如响应使用者的各种操作。

* UI&Plugins

UI提供使用者方便的界面达成查看或编辑的基本功能，如复制、粘贴、上一页、下一页等，Plugins则为第三方开发者对浏览器进行功能扩充的程序，如Macromedia Flash[30]播放器、Java运行时环境（JRE）。

### 5.1.1渲染引擎分析

知名的渲染引擎有Trident、Gecko、Webkit、Presto等。

（1）Trident

Trident（又称为MSHTML）[31]，是微软的视窗作业系统（Windows）搭载的网页浏览器——Internet Explorer的排版引擎，它的第一个版本随着1997年10月Internet Explorer第四版释出，之后不断的加入新的技术并随着新版本的Internet Explorer释出。在Internet Explorer第七版中，微软对Trident排版引擎做了重大的变动，除了加入新技术之外，还增加了对网页标准的支持。

Trident引擎被设计成一个软件组件（模块），使得其它软件开发人员很容易的将网页浏览的功能加到他们自行开发的应用程序里，其内核接口设计相当成熟，以致出现了很多用IE内核而非IE的浏览器（如Maxthon、腾讯 TT、遨游、360安全浏览器等），但是Trident只能用于Windows平台。至今Trident仍然是市场占有率最高的排版引擎。

（2）Gecko

Gecko[32]是用C++语言编写的、开放源代码的、跨平台的网页渲染引擎。原本由Netscape公司开发，现在由Mozilla基金会维护。Gecko是跨平台的内核，能在Windows、Linux、BSD和Mac OS X等主要操作系统上运行。由于Gecko开放源代码，因此受到世界各国程序员的青睐，其中很多都可以为其编码，增加功能。导致其功能很强，但很庞大，由于开源，市场占有率迅速提高，成为流行程度仅次于Trident的排版引擎。目前最主流的以Gecko为内核的桌面浏览器是Mozilla Firefox，由于Firefox的出现，IE的霸主地位逐渐被削弱。

以Gecko作为内核的网页浏览器有：Firefox，Netscape（6到9）、Minimo、Fennex、SeaMonkey、Songbird等。

（3）Webkit

Webkit[33]是以C++语言编写的、开放源代码的页面渲染引擎，最初的代码来自KDE（K Desktop Environment）的KHTML和KJS（它们均为开放源代码，都是自由软件，在GPL协议下授权）。该项目小组每天都会上传最新版本的源码，更新速度非常快。苹果当年在比较了Gecko和KHTML后，选择了后者，因为它拥有清晰的源码结构、极快的渲染速度。苹果将KHTML发扬光大，推出了装备KHTML改进型Webkit引擎的浏览器Safari。

传统上，Webkit包含一个排版引擎WebCore和一个JavaScript解析引擎JavaScriptCore，它们分别对应的是KDE的KHTML和KJS。

Webkit渲染引擎高效稳定，兼容性好，且源代码结构清晰，易于维护。以开源、强大的页面渲染效果和高效的脚本执行而著称。可以移植到Windows、Linux、Mac OS等操作。

Google所推出的Chrome浏览器以及Chrome OS还有Android上内建的网页浏览器也是用Webkit作为排版引擎。

（4）Presto

Presto[34]由Opera Software公司开发，是Opera网页浏览器的排版引擎。Presto取代了旧版Opera4至6版本使用的Elektra排版引擎，包括加入动态功能，例如网页或其部分可随着DOM及Script语法的事件而重新排版。

Presto实际上是一个动态内核，与Trident、Gecko等内核的最大区别就在于脚本处理上，Presto有着天生的优势，页面的全部或者部分都能够在回应脚本事件时等情况下被重新解析。该款渲染引擎的特点就是渲染速度的优化达到了极致。此外该内核在执行JavaScript的时候有着最快的速度，根据同等条件下的测试，Presto内核执行同等JavaScript所需的时间仅有Trident和Gecko内核的约1/3，然而代价是牺牲了网页的兼容性。它是一个商业引擎，很大程度上限制了它的发展。

Presto也是手机网页浏览器的排版引擎，目前Opera Software推出了两款手机网页浏览器：Opera Mobile和Opera Mini。在各种测试中Presto和Webkit的结果都不分上下。然而Presto没有开放源代码，由Opera公司维护和升级。

### 5.1.2 选择Webkit的理由

为了我们的研究目的，我们必须针对以下几点，分析四大渲染引擎：

（1）开发难易度

（2）跨平台支持度

（3）许可证问题

（4）网页渲染速度

表5－1 渲染引擎比较表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trident | Gecko | Webkit | Presto |
| 开发难易度 | 普通 | 容易  （有丰富的程序开发API） | 容易  （有丰富的程序开发API） | 普通 |
| 跨平台支持度 | Windows | Windows/Linux  /BSD/Unix  /Mac OS X | Windows/Linux  /BSD/Unix  /Mac OS X | Windows/Linux  /BSD/Unix  /Mac OS X |
| 许可证 | Proprietary  /MS-EULA | MPL/GNU  GPL/GNU  LGPL | LGPLv2  /BSD-style | Proprietary |
| 网页渲染速度 | 最慢 | 第三快 | 最快 | 第二快 |

由表5-1我们可以看出Webkit是开源的，符合标准的，程序取得修改容易，而且跨平台支持度高，网页渲染速度快，Google公司将Webkit作为Chrome的排版引擎，因此我们选择Webkit作为本研究的引擎。

### 5.1.3 WebKit框架结构

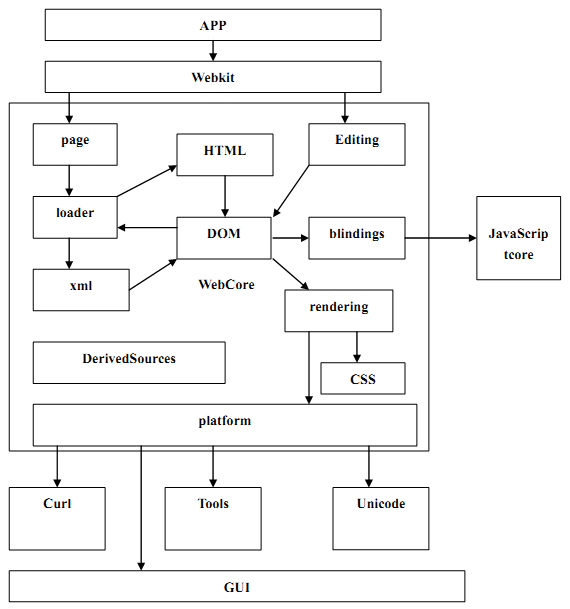


图5－1 Webkit框架结构

Webkit内核的体系结构如图5-1所示：Webkit内核的主要部分是WebCore和JavaScriptCore。WebCore主要负责页面的加载、解析以及布局渲染页面，而JavaScriptCore主要负责解析JavaScript代码，目的是为了扩展基本的HTML功能，为Web增加动态效果[35]。从图5-1中可以看到DOM模块是WebCore的核心。

### 5.1.4 WebKit解析流程

作为浏览器的核心，Webkit渲染引擎的作用是输入一个HTML文档，输出一个Web页面。解析流程如下[36]：

1.用户在手机浏览器界面输入URL地址向服务器发送页面请求，页面的URL或者本地文件名被发送到服务器。

2.如果URL没错，服务器返回给本地HTML/XHTML源数据。

3.浏览器通过W3C规范，把接受到的HTML/XHTML代码用分析器分析，解析成一棵DOM树，然后加载HTML代码。

4.如果HTML里面引用外部CSS文件，浏览器发出CSS文件请求，服务器返回这个CSS文件。

5.浏览器继续加载HTML代码，使用CSS和DOM树解析成Render树。布局管理器Layout对DOM的可视元素排版，分配位置、高度、宽度，并由CSS规定输出的特性如字体的颜色，大小等。再由Render进行视觉化表达，这个是一个动态的过程。

6.布局器根据Render树中各组件指派位置和大小。最后把Web页面显示给用户。

### 5.1.5 Qt WebKit

Qt是一个跨平台的C++图形使用者界面应用程序架构。它提供给应用程序开发者建立美观、视觉化的图形用户界面所需的功能。它亦是Linux桌面环境KDE的基础，而KDE亦是所有主要的Linux发行版的一个标准组件。从图5-2可知，它可建立在大部分的作业系统环境，拥有良好的跨平台特性，并提供丰富的功能模块、API等于应用程序做沟通。

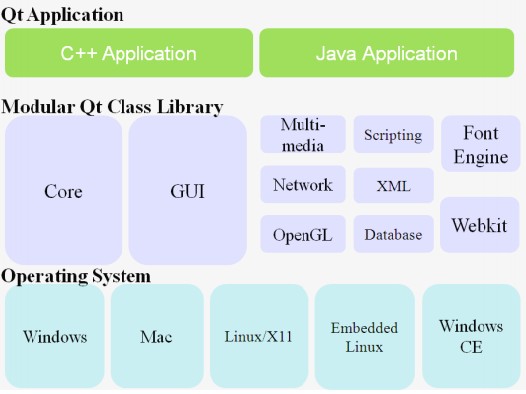


图5－2 Qt跨平台架构图

在Qt源码中，Qt WebKit在Qt Framework当中的地位等同于Qt4其他模块。图5-3中，WebKit以Qt Core Module本身的函数库模块、Qt Network Module网络模块与Qt GUI Module图形化界面模块三个模块，实现出对HTML格式的资料接收与解析，并做编辑排版的工作。Qt应用得以直接通过Qt WebKit嵌入Web页面。

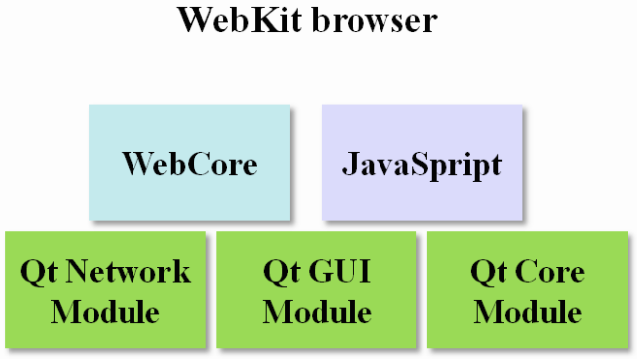


图5－3 Qt Webkit概念图

本研究使用Qt Webkit实现的浏览器界面如图5-4所示。



图5－4 Webkit浏览器界面

## 链接抽取模块

链接抽取模块从给定的种子URL开始，沿着网页中的链接，按照广度优先搜索策略提取链接，将抽取到的链接放入URL资源库。

### 5.2.1过滤条件

链接抽取模块在抽取链接过程中，会根据用户设置的过滤串过滤链接，最终得到的URL集合代表着同一种网页结构。

过滤链接共有四种模式，分别为：

（1）必包含串

对于必包含串S1、S2，待抽取串S，当且仅当S.contains(S1)为真，并且S.contains(S2)也为真时，我们说串S是符合要求的目标串。其中S.contains(S1)表示在串S中查找子串S1，若找到，contains方法返回真；否则返回假。即必包含串之间的关系同逻辑与（&）。

（2）可包含串

对于可包含串S1、S2，待抽取串S，当且仅当S.contains(S1)为真，或者S.contains(S2)为真时，我们说串S是符合要求的目标串。即可包含串之间的关系同逻辑或（|）。

（3）不包含串

对于不包含串S1、S2，待抽取串S，当且仅当S.contains(S1)为假，并且S.contains(S2)也为假时，我们说串S是符合要求的目标串。即不包含串之间的关系同逻辑非（~）。

（4）可展开串

对于可展开串S1、S2，待抽取串S，当且仅当S.contains(S1)为真，或者S.contains(S2)为真时，爬虫将访问S所指向的页面，然后从新页面中继续提取URL。

以京东商城为例，我们想要抓取商品信息页面（图5-4）及分页，链接抽取模块负责抽取商品信息页的URL地址，分页抓取交由翻页脚本处理。通过分析发现，所有的商品信息页的URL都有着相似的结构，如表5-2所示。其中1、2条是京东图书的URL样式，3、4条是京东商城普通商品的URL样式。

表5－2 京东商城商品信息页URL示例

|  |
| --- |
| 1. http://book.360buy.com/11140353.html 2. http://book.360buy.com/14623453.html 3. http://www.360buy.com/product/352688.html 4. http://www.360buy.com/product/234663.html |

从表中可以看出，这些链接都包含“html”，因此我们把“html”添加到必包含串中，如图5-5所示。

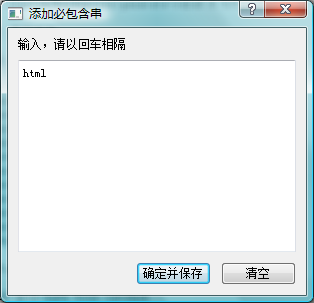


图5－5 添加必包含串

京东图书页面均包含字符串“http://book.360buy.com/”，而普通商品页包含字符串“http://www.360buy.com/product/”，因此我们把这两个字符串添加到可包含串中，如图5-6所示，即链接抽取模块抽取到的目标URL或是普通商品页，或是图书页面。



图5－6 添加可包含串

通过以上过滤条件的设置并不能保证抽取到的URL全部符合要求，例如：“http://www.360buy.com/products/1320-5019-000.html”符合以上设置的过滤要求，但此链接并不是真正的商品信息页面；“http://c.fa.360buy.com/adclick?sid=2&cid=232&aid=3101&bid=0&unit=32946&advid=39294&guv=&url=http://www.360buy.com/product/1002521932.html”同样符合过滤要求，但目标URL——“http://www.360buy.com/product/1002521932.html”——只是在此链接的参数中出现，因此它也不是真正的商品页面。为了确保链接抽取模块能够过滤掉以上两种不符合要求的URL，我们可以设置如图5-7所示的不包含串。

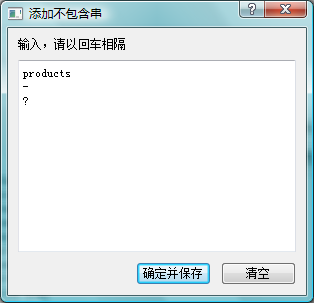


图5－7 添加不包含串

通过以上三种过滤条件的设置，已经可以保证链接抽取模块抽取到的URL集全部是正确的URL。最后为了让爬虫能够循环抽取网页中的链接，需要设置可展开串，如图5-8所示，爬虫将沿着所有包含“360buy.com/”的链接地址深入进去，以广度优先搜索方式循环抽取网页中的链接。链接抽取模块运行结果如图5-9所示。

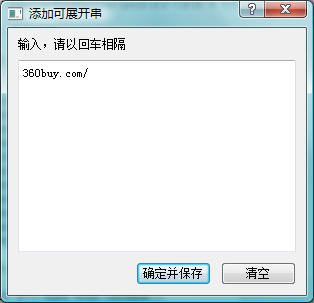


图5－8 添加可展开串

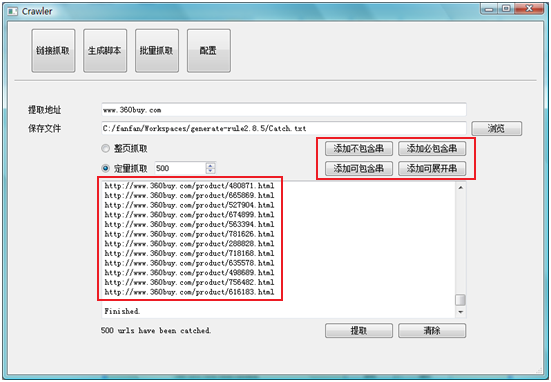


图5－9 链接抽取结果

### 5.2.2 Robots协议

Robots协议是一个纯文本文件，网站通过robots协议限定网络爬虫对该网站的访问范围，即通过一些规范规定本网站不允许哪些爬虫程序访问和哪些文件不允许被爬虫程序访问。这就是网络爬虫排斥机制（Robots Exclusion Standard），简称RES[37]。

当网络爬虫访问一个网站时，它会首先检查该网站的根目录下是否存在一个名为robots.txt的文件，如果存在，爬虫程序就会按照该文件中的内容来确定访问的范围；如果该文件不存在，所有的爬虫将能够访问网站上所有没有被口令保护的页面。

该文件的内容共有三个域组成：

（1）User-agent（用户代理）字符串行

该域的值主要用来表示网站排斥的搜索引擎的爬虫程序，即网站不允许这些爬虫程序访问。以百度的爬行器Baiduspider为例，有User-agent：Baiduspider。如果网站限制多个蜘蛛程序访问，在“robots.txt”中就会有多条User-agent记录。如果User-agent的值为\*，则互联网上所有蜘蛛程序在访问该网站时必须遵守该协议。

（2）Disallow字符串行

该域的值用来表示不希望被访问的链接URL，这些URL可以是一条完整的路径，也可以其中的一部分，所有以Disallow标示的URL都不会被蜘蛛程序访问到。例如“Disallow: /admin”就表示该网站中所有URL中含有“/admin”的网页均不会被访问。而“Disallow: /admin/”则允许爬虫访问/admin.html，而不能访问/admin/index.html。如果“/robots.txt"文件中Disallow记录为空，则说明该网站的所有部分都允许访问。

（3）Allow字符串行

该域的意义与Disallow相反，表示允许爬虫程序访问的目录。

“/robots.txt”文件的格式如下：

User-agent: …

Disallow: …

Allow: …

网站通过Robots协议来管理网络爬虫程序对网站的访问，例如，可以通过实时修改robots.txt文件来限制网络爬虫的访问次数，从而避免网站服务器负担过重；可以通过robots.txt文件来规定拒绝哪些网络爬虫程序的访问，规定网站的哪些文件不允许被访问等，来保护网站的信息。

所以，本文设计的网络爬虫为了避免因robots协议而浪费资源，固在爬虫访问网页时严格遵守robots协议，以robots协议中规定的路径字符串作为过滤条件，不抽取不允许访问的URL，从而避免了不必要的爬行，节省了网络爬虫爬行时间，提高了网络爬虫的工作效率。

### 5.2.3 URL消重策略

由于多个网页可能嵌入了对同一网页的链接，当我们对爬虫访问的网页内容进行解析获取URL时，不可避免的会得到已经提取过的链接。如果不对提取到的链接进行消重的话，不仅会使网络爬虫重复抓取，还会浪费大量的存储资源，因此需要设计一种URL消重策略来避免这种问题。

目前URL消重已经出现了很多相关技术，比如由Burton Howard Bloom提出的Bloom Filter[38]算法，由Ronald L. Rivest提出的MD5算法[39]，Mercator提出的散列表法[40]，Polytechnic Crawler提出的R-B树检索法[41]等，通过对这些方法的研究，决定采用Bloom Filter实现URL消重。

Bloom Filter是一种空间效率很高的随机数据结构，它利用位数组（实际应用中可利用一维数组或向量来实现）很简洁地表示一个集合，并能判断一个元素是否属于这个集合。Bloom Filter的这种高效是有一定代价的：在判断一个元素是否属于某个集合时，有可能会把不属于这个集合的元素误认为属于这个集合（false positive）。因此，Bloom Filter不适合那些“零错误”的应用场合。而在能容忍低错误率的应用场合下，Bloom Filter通过极少的错误换取了存储空间的极大节省。

Bloom Filter算法的基本思想为：

初始状态时，Bloom Filter是一个包含m位的位数组，每一位都置为0，如图5-10所示。

o_bf1

图5－10 Bloom Filter位数组的初始化

为了表达S={x1,x2,…,xn}这样一个n个元素的集合，Bloom Filter使用k个相互独立的哈希函数，它们分别将集合中的每个元素映射到{1,…,m}的范围中。对任意一个元素x，第i个哈希函数映射的位置hi(x)就会被置为1（1≤i≤k）。在图5-11中，k=3，且有两个哈希函数选中同一个位置（从左边数第五位），其演算法如表5-3所示。

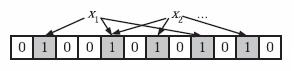


图5－11 X1 , X2的映射图

表5－3 Bloom Filter储存算法

|  |
| --- |
| Algorithm 1: Bloom Filter’s Storing |
| input: *n* URLs, each *b* bits long   1. for each url *m* do 2. url *m* hash coded into distinct bit addresses *a1*,*a2*,…,*ad* 3. set bit addresses *a1* through *ad* to 1 4. end |

在判断y是否属于这个集合时，我们对y应用k次哈希函数，如果所有hi(y)的位置都是1（1≤i≤k），那么我们就认为y是集合中的元素，否则就认为y不是集合中的元素。下图中y1就不是集合中的元素。y2或者属于这个集合，或者刚好是一个误报。

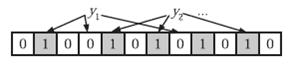


图5－12 在位数组中查找Y1 , Y2是否属于集合

Bloom Filter算法使用较小的计算时间来进行查询及储存，然而Bloom Filter所面临的缺点则是误报率的发生和不易将已存储的元素进行删除。

### 5.2.4抽取流程

链接抽取模块的具体工作流程如下：

（1）根据用户设置的过滤字串初始化链接抽取模块。

（2）从给定的种子URL开始，沿着网页中的链接，按照广度优先搜索策略提取链接。

（3）对提取到的URL进行判重，过滤掉重复的URL。

（4）将抽取到的链接放入URL资源库。

（5）抽取到的URL达到一定数量，模块停止工作。

## 脚本生成模块

通过用户在网页内容上的标注及相关的参数设置，脚本生成模块自动生成针对某一[样本页面](http://www.gooseeker.com/cn/node/document/terms/sample)的翻页脚本文件。图5-13为脚本生成界面。



图5－13 脚本生成界面

### 5.3.1样本页面

如果需要抓取网页内容中的分页数据，需要为目标网页定义翻页规则，在很多网站上，大量的页面使用相同的HTML模板，例如，论坛网站上的所有的讨论主题都使用一个网页模板写成。在这种情况下，只需要为其中一个网页定义翻页规则即可，这个网页就是样本页面。

[样本页面](http://www.gooseeker.com/cn/node/document/terms/sample)的结构代表了同一主题下的所有页面结构，所以，为样本页面定义的翻页规则可以普遍用于所有同一主题的页面翻页任务。当前互联网处于Web2.0时代，典型的网站都是以数据库为后盾，而HTML页面是以特定模板动态生成的，样本页面结构反映了网页模板结构，因此，为样本页面定义的翻页规则并推广应用于同一主题的所有页面的思路是可行的。图5-13使用一个京东图书页面作为样本页面，代表了京东网站上所有的商品信息页面结构。

### 5.3.2关键参数设置

（1）自动滚屏

用浏览器阅读该样本网页上的微博消息时，如果网络速度不是很快，会有一种特殊体验：微博消息很多，这个网页很长，需要拖动右侧滚动条滚屏才能看全所有消息，当网速比较慢时，滚屏时先看到文字，然后是图片，然后是转发数和点评数。这些内容是异步加载的，没有滚动到可见范围就不从服务器上下载这些内容。如果我们采用通常的抓取方法，势必只能抓取到前面几条消息的点评数和转发数；另外，通过第四章的介绍可知，新浪微博采用自动加载分页的机制，只有当多次拖动滚动条到底部时，才会看到分页导航，因此，必须要求爬虫在抓取的时候自动滚屏。在图5-13的“参数”一栏中设置自动滚屏次数。

（2）时延

为了抓取AJAX动态数据，需要设置时延，因为试验发现，转发数和点评数是在网页加载完成后才使用JavaScript/JS程序从服务器异步加载的，所以一定要设置时延。这个参数将影响到数据采集效率，为了等待转发数和评论数等AJAX动态内容完整加载好，需要时延，时延过长，造成采集效率下降，时延太短，可能来不及发送数据加载请求。

在图5-13的“参数”一栏中设置时延。用户可以根据网络速度及爬虫实际运行情况微调时延参数，以便达到最佳的性能和准确度。这个参数的推荐值是2000ms。

（3）模式

传统的网页采用刷新整个页面的方式响应用户的翻页操作，而使用AJAX技术的网页只局部更新页面上的部分内容，为了使爬虫具有良好的通用性，能够处理以上两种情况，在生成脚本时需要选择样本页面的刷新模式。一个简单的判断方法是：刷新整个页面会导致URL改变，而采用AJAX技术局部刷新不会使URL变化。在图5-13的“模式”一栏中设置刷新模式。

### 5.3.3定位clickable元素

由于在网页翻页过程中需要触发绑定在clickable元素上的事件，因此在脚本生成前需要定位clickable元素。用户需要对网页内容进行标注，也就是标注clickable元素的位置，剩下的工作由脚本生成模块自动完成。大部分网站在分页的设计上大同小异，都带有明显的“下一页”链接，如图5-14所示的新浪微博的分页设计。



图5－14 新浪微博分页交互

但是也有部分网站没有明显的“下一页”链接，如图5-15所示的新华博客的分页设计。



图5－15 新华博客分页交互

生成脚本时需考虑上述两种情况。针对第一种有明显“下一页”链接的分页样式，在图5-13的“分页链接样式”一栏中选中

只需简单地标注“下一页”链接即可；针对第二种无明显“下一页”链接的样式，爬虫需要识别当前页的链接和其他未选中页的链接样式。

图5-14所示的clickable元素为“下一页”链接，图5-15所示的新华博客中需要标注第1页和其他任意一页（如第2页）。

标注的clickable元素被系统接收，系统识别出该DOM元素的标签名，id、class等属性及文本内容，并依此生成该页面结构对应的翻页脚本，使翻页脚本能够在网页中定位clickable元素并模拟点击操作，从而实现翻页的效果。

当目标网页结构发生改变时，该技术能够尽可能确保以前定义好的网页翻页脚本有效，使翻页规则受到的影响尽量降到最低。但是，如果目标网站大规模改版，无法避免翻页脚本失效。此时，用户使用脚本生成界面重新生成针对目标网页的翻页脚本即可。

### 5.3.4脚本执行逻辑

通过用户在网页内容上的标注及相关的参数设置，脚本生成模块自动生成针对某一[样本页面](http://www.gooseeker.com/cn/node/document/terms/sample)的翻页脚本文件，脚本执行逻辑如图5-16所示。



图5－16 脚本执行流程图

由于AJAX网站中为DOM元素绑定事件存在多种不同方式（详见3.3.4节），本研究采用一种通用的技术模拟点击操作，触发绑定在DOM元素上的事件。

表5－4 模拟点击操作代码

|  |
| --- |
| //参数说明：g：clickable元素；c：事件类型，如”click”  function simulate(g, c) {  var e = extend(defaultOptions, arguments[2] || {});  var b, f = null;  for ( var d in eventMatchers) {  if (eventMatchers[d].test(c)) {  f = d;  break  }  }  if (!f) {  throw new SyntaxError("Only HTMLEvents and MouseEvents interfaces are supported")  }  //DOM2  if (document.createEvent) {  //创建指定类型的Event对象  b = document.createEvent(f);  if (f == "HTMLEvents") {  //对于HTML Event，使用initEvent方法初始化Event对象  b.initEvent(c, e.bubbles, e.cancelable)  } else {  //对于Mouse Event，使用initMouseEvent方法初始化Event对象  b.initMouseEvent(c, e.bubbles, e.cancelable, document.defaultView,  e.button, e.pointerX, e.pointerY, e.pointerX, e.pointerY,  e.ctrlKey, e.altKey, e.shiftKey, e.metaKey, e.button, g)  }  //调用事件  g.dispatchEvent(b)  } else {  //Internet Explorer  e.clientX = e.pointerX;  e.clientY = e.pointerY;  var a = document.createEventObject();  b = extend(a, e);  g.fireEvent("on" + c, b)  }  return g  }  function extend(a, c) {  for ( var b in c) {  a[b] = c[b]  }  return a  }  var eventMatchers = {  HTMLEvents: /^(?:load|unload|abort|error|select|change|submit|reset|focus|blur|resize|scroll)$/,  MouseEvents :  /^(?:click|dblclick|mouse(?:down|up|over|move|out))$/  };  var defaultOptions = {  pointerX : 0, pointerY : 0, button : 0,ctrlKey : false,altKey : false,shiftKey : false,metaKey : false,bubbles : true,cancelable : true  }; |

## Cookie管理模块

许多大型的站点，必须注册后才能使用它们的服务，并且在登录前和登录后看到的内容数量有差别，例如，访问新浪微博前如果不登录，在网页看到的微博信息条数要少很多，同时，也无法查看下一页的微博内容。所以，Cookie管理模块用于支持网站Cookie，支持网站登录采集，使爬虫系统能够处理需要登录认证才能浏览的目标页面。

Cookie是小段的文本信息，Web服务器将它放在HTTP响应报头中发送到浏览器，之后，在访问同一网站或域时，浏览器又将它原封不动地返回。通过让服务器读取它之前发送给客户程序的信息，站点可以为访问者提供诸多便利[42]：

* 在电子商务会话中标识用户
* 记录用户名和密码
* 定制站点
* 定向广告

本系统使用Cookie的目的是为了记录用户名和密码，使用户不必每次使用系统前要输入用户名和口令。以爬取新浪微博为例，用户在运行爬虫前首先要使用Webkit浏览器访问新浪微博，完成登录。由于登录状态记录在cookie中，即便爬虫运行结束退出了，在一定时间内登录状态还是有效的，在此期间运行爬虫不用再次登录。

Cookie管理模块负责Cookie的管理工作，使用一个Cookie容器存储网络交互过程中的Cookie，主要包括读取、添加、删除和持久化保存等操作，具体工作步骤如下：

步骤一：爬虫初始化过程中，读取本地Cookie文件并构建Cookie对象保存在Cookie容器中，Cookie容器位于浏览器内存中。读取过程中，过期的Cookie对象被移除。

步骤二：浏览器向服务器发送请求前，根据请求的目标URL从Cookie容器中找出domain和path与目标URL相匹配的Cookie对象，将其添加到HTTP请求的报头中，发往Web服务器的目标URL。

步骤三：浏览器得到Web服务器发回的HTTP响应，从响应报头中取出Cookie，更新Cookie容器内的Cookie对象，若有同名的Cookie存在容器中，则旧的Cookie将被覆盖。

步骤四：当Cookie容器中的Cookie发生改变时，将其保存至本地Cookie文件，其中，会话Cookie不会被保存。

## 模块交互与关键算法

为了更清晰地说明各个模块之间的交互与协作，本节使用如下时序图展示模块之间的调用关系与消息传递。



图5－17 模块之间的交互

（1）Webkit浏览器进行初始化，包括加载Cookie容器。

（2）浏览器从URL资源库中读取一条URL，访问此URL，加载并解析HTML网页，在与Web服务器的通信过程中，更新Cookie容器中的cookie对象。若URL资源库中不存在未访问的URL，跳至（5）。

（3）浏览器在渲染完成的网页文档中执行JavaScript翻页脚本。

（4）翻页脚本在执行过程，通知浏览器保存网页内容，并在其中查找clickable元素，若找到，触发点击事件，浏览器执行翻页操作，跳至（3）；反之，通知浏览器翻页结束，跳至（2）。

（5）爬行结束，浏览器释放Cookie容器，执行相关清理工作，关闭窗口。

具体算法如表5-5所示：

表5－5 爬行算法

|  |
| --- |
| Algorithm 1: Crawling AJAX |
| input: URL,clickable c   1. Function main(URL) 2. global *browser* = initEmbeddedBrowser() 3. global *linker* = initHyperlinkExtractor () 4. Set *U* = *linker*.getHyperlinks(*URL*) 5. for *url* ∈ *U* do 6. *dom*=*browser*.getDOM(*url*) 7. *s* = state(*dom*) 8. crawl(*s*) 9. end for 10. end Function 11. Function crawl(State *s*) 12. *s*.saveAsHTML() 13. *xe* = getXpathExpr(*c*) 14. if (findClickables(*xe*)) then 15. *type* = getEventType(*c*) 16. *browser*.fireEvent(*c*, *type*) 17. waitForDynamicContentToLoad() 18. *newDom* = *browser*.getDOM() 19. *s\_1*= state(*newDom*) 20. crawl(*s\_1*) 21. end if 22. end Function |

算法在第2、3行进行Webkit浏览器和链接抽取器的初始化，随后链接抽取器以指定的种子URL为起始URL，以广度优先搜索方式循环抽取直至获得一定数量的URL后停止，形成URL集U（第4行）。对于U中的每一个URL，浏览器访问此URL并加载、解析网页内容，然后对其进行翻页抓取（第8行）。

在crawl函数中，浏览器首先保存当前网页状态，然后在DOM结构中查找clickable元素c，若找到，则触发元素c绑定的事件，引起DOM树的变化而产生新的网页状态，针对新到达的网页状态，递归执行crawl函数；若没有找到，则翻页结束，浏览器访问URL集中的下一个URL。

## 系统实现

本系统建构在以微软操作系统为基础的实验环境上，采用My Eclipse 8.5作为Java开发平台。在外部资源的引用上，本系统采用qtjambi（Qt为Java开发的Eclipse插件）作为Webkit浏览器；在压缩JS脚本的部分则通过yui compressor来完成。

本系统采用Bally等人（1997）提出的原型法作为本系统的开发方法，运用原型法的主要特色是边做边学（Learning by Doing），并且可以随时依据使用者需求做不同的功能设计的调整等优点，不断反复在系统开发中学习与验证系统可行性以增加系统完整性与适用性。图5-18是本研究的系统开发流程图。

采用原型法系统开发方法的优点如下（曹中天2004）：

1.通过使用者在操作原型时，可能发现新的系统需求。

2.原型设计及制作成本较低。

3.借助可操作的模型，系统分析师容易与使用者沟通。

4.允许系统分析验证逻辑设计规格与技术设计规格。

5.使用者参与全部需求分析的过程，能找出真正需求。

6.对于重大任务、攸关生命安全或不易实地测试的系统，如EIS及专家系统更是适合采用原型法。



图5－18 系统开发流程图

## 本章小结

本章详细介绍了支持AJAX的网络爬虫系统架构中各个模块功能的具体实现及主要算法，主要包括Webkit浏览器、链接抽取、脚本生成、Cookie管理这些模块的具体实现及关键技术。

第六章 爬虫系统实验与结果分析

## 6.1实验环境

由于爬虫系统的性能不仅同系统运行所在机器的处理器及内存有关系，还依赖于具体的网络环境和目标网站的Web服务器的处理速度，因此，实验环境（尤其是网络速度）对实验结果存在较大的影响，本实验的各种结果及评估均是在表6-1所示的实验环境产生的。详细的实验环境规格如表6-1所示。

表6－1 实验环境

|  |  |
| --- | --- |
| 元件 | 详细资料 |
| 处理器 | Intel(R) Core(TM)2 Duo CPU P8400 @2.26GHz |
| 内存 | 4.00GB |
| 操作系统 | Windows Vista(TM) Business |
| 网络速度 | 2.4MB/秒 |

## 6.2 评估方法

本研究使用的评估方法为查全率（Precision）、准确度（Accuracy）与抓取速率（Crawl Time）。查全率是考察爬虫系统采集AJAX动态数据的能力，测试其是否能够正确爬行并获取网页上完整的AJAX动态内容；准确度表示爬虫抓取下来的网页中，正确网页所占的百分比；抓取速率可表示为单位时间抓取到的网页大小，抓取速率越高，表示爬虫系统的性能越强，抓取网页所消耗的时间越短。分别表示成下面的公式：

准确度（Accuracy）= 式(6-1)

抓取速率（Crawling Rate）= 式(6-2)

其中Total Pages表示总共抓取了多少个网页，Correct Pages表示其中正确的网页个数；DOM Size为保存的HTML静态文件的总大小，Crawling Time为总的抓取时间。

## 6.3 实验设计及结果分析

为了测试爬虫系统的查全率、准确度及性能，本实验将爬虫系统应用于表6-2所列的6个真实的网站（C1-C6）：

表6－2 实验用例

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Case | AJAX site | Clickable Elements |
| Electronic  Commerce | C1 | http://www.360buy.com/ | <a class="next" href="#comments-list">下一页</a> |
| C2 | http://www.tmall.com/ | <a class="page-next" data-val="2" data-act="page" href="#">下一页</a> |
| Twitter | C3 | http://weibo.com/ | <span action-data="id=34..&page=2" action-type="feed\_list\_page">下一页</span> |
| C4 | http://t.163.com/ | <li class="js-btn js-next">  <a href="javascript:;">下一页</a></li> |
| Blog | C5 | http://blog.sohu.com/ | <a class="txt" action="page" href="javascript:;" data-page="next">下一页</a> |
| C6 | http://blog.sina.com.cn/ | <a href="javascript:Ui.Pagination.showPage('commentPaging',2);"> 下一页</a> |

实验选取了三类典型的Web2.0时代的AJAX网络应用，分别是电子商务网站、微博和博客，并且在每个类型中选择两个网站作为代表，列出了它们的clickable元素。实验中选择这些网站用于测试的原因是他们使用AJAX改变网页的状态，使用JavaScript为DOM元素绑定事件，与Web服务器异步通信获取数据，并动态地修改DOM树结构。

表6-2中的网站C1是京东商城，京东商城是中国最大的B2C综合网络[零售商](http://baike.baidu.com/view/154748.htm)，是中国电子商务领域最受消费者欢迎和最具有影响力的电子商务网站之一。通过分析发现京东商城页面中呈现的大部分内容（评论、推荐等）没有在源代码里出现。C2是天猫商城，是[淘宝网](http://baike.baidu.com/view/1590.htm)打造的[B2C](http://baike.baidu.com/view/16398.htm)[购物平台](http://baike.baidu.com/view/2492459.htm)。

微博类网站中选择了新浪微博C3和网易微博C4，微博网站是需要登录认证才能浏览的网站；C5（搜狐博客）和C6（新浪博客）是博客类网站中比较流行的两大网站。

### 6.3.1查全率

查全率是考察爬虫系统采集AJAX动态数据的能力，测试其是否能够正确爬行并获取网页上完整的AJAX动态内容，在测试这个指标时我们并不考虑抓取耗时，因此我们设置足够大的时延，以确保所有的AJAX动态内容都已从服务器异步发送到浏览器，并被浏览器加载并显示。

如何判断爬虫抓取的网页内容是否完整是一个难题，因为我们无法得到内容完整的网页。从桌面浏览器中将呈现的网页保存至本地磁盘，保存的内容只是网页的源代码，由4.1节得知网页源代码中的内容不是完整的，因此，无法通过编程的方式比对两个文件内容，只能通过人工肉眼比对抓取的网页内容和浏览器中呈现的网页内容。由于爬虫抓取的网页数量众多，无法实现人工一一比对，因此我们采取随机抽样的方法，在每10个抓到的网页中随机抽出一个，与浏览器中呈现的网页比对，若内容一致，则认为这10个网页均为内容完整、正确的网页。

表6－3 查全率实验结果

|  |  |  |  |
| --- | --- | --- | --- |
| Case | Hyperlinks | Missing Pages | Precision(%) |
| C1 | 107 | 0 | 100 |
| C2 | 96 | 0 | 100 |
| C3 | 114 | 0 | 100 |
| C4 | 90 | 0 | 100 |
| C5 | 104 | 0 | 100 |
| C6 | 86 | 0 | 100 |

表6-3为爬虫系统应用于C1-C6的查全率实验结果，其中Hyperlinks是URL资源库中的链接数量，表示爬虫共采集了多少个网页；Missing Pages是抓取的网页中，内容不完整的网页数量；Precision是查全率。实验结果表明，爬虫能够正确爬行C1-C6，并获取网页上完整的AJAX动态内容。

### 6.3.2准确度

准确度表示爬虫抓取下来的网页中，正确网页所占的百分比。正确网页是指网页包含完整的AJAX动态内容，我们在浏览器中看到的内容都可在正确网页中找到。

由之前查全率的实验可知，爬虫系统在时延足够大的情况下，是能够抓取到AJAX网页中的完整内容的，因此，不存在错误网页。但在实际运作过程中，考虑到抓取效率的关系，我们不可能设置太长的时延，而时延的长短会影响准确度的高低，主要表现为以下两种情况：

* 内容不完整的网页。当时延设置过短时，可能导致来不及发送数据加载请求或异步加载的数据还未从服务器端返回，此时保存的网页内容是不完整的。
* 内容重复的网页。当爬虫翻页抓取时，由当前页翻至下一页时，很可能由于时延过短，导致在当前页的内容还停留在网页上，而下一页的内容还未从服务器传送过来时，爬虫便保存网页。此时保存的网页名为下一页的网页，实际内容却是当前页的内容，网页内容与当前页重复，我们称之为重复网页。

以上两种情况下抓取到的网页称为错误网页。根据实验结果中时延与准确度的关系，我们统计出图6-1。

图6－1 时延vs.准确度

### 6.3.3抓取速率

爬虫的抓取速率与时延有着密切的关系，时延过长，造成抓取效率下降，时延太短，造成准确度下降。如果为了追求高准确度，而加大时延，最终得到的较低的抓取效率并不能真实的反应本系统的性能；而在较低的准确度下得到较高的抓取速率也没有意义。因此为了测试爬虫的抓取速率，我们将时延设置为2000ms，透过图6-1可知，此时既能达到较高的准确度97%，2000ms的时延也不至于过长而对抓取速率造成太大的影响。相当于以时间的代价换取了准确度的提升，虽然抓取速度因此受到影响，但也在可以接受的范围之内。

除此之外，爬虫的抓取速率还与是否翻页抓取有关系。Webkit浏览器加载一个全新的网页和在一个已加载的网页上进行翻页，并获取分页数据这两种操作上所消耗的时间存在明显的差别。通常情况下，浏览器加载并渲染整个页面消耗的时间较长，而在一个已渲染出的网页上进行翻页只需很短的时间，而两者运行的结果都导致一个新的网页状态（HTML静态文件）的产生，如果单靠抓取速率公式（式6-2）判断系统性能，会导致爬虫对于分页少的网站和分页多的网站的抓取性能出现较大的差异，分页少的网站抓取速率低，而分页多的网站抓取速率则相对高很多，这将会影响爬虫性能的评估。因此，本研究将分为两种情况进行实验分析，分别是翻页抓取和不翻页抓取。

表6-4为爬虫系统应用于C1-C6的爬行结果，爬行过程中采取不翻页策略，其中Hyperlinks是URL资源库中的链接数量；DOM size为保存的HTML静态文件的总大小；Crawl Time为总的抓取时间，Crawling Rate为抓取速率。

表6－4 不翻页抓取实验结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case | Hyperlinks | DOM size (kb) | Crawl Time(s) | Crawling Rate(kb/s) |
| C1 | 107 | 17,404 | 594 | 29.3 |
| C2 | 96 | 23,375 | 916 | 25.5 |
| C3 | 114 | 19,075 | 568 | 33.6 |
| C4 | 103 | 14,210 | 369 | 38.5 |
| C5 | 104 | 13,208 | 337 | 39.2 |
| C6 | 86 | 28,395 | 207 | 137.2 |

从表6-4中可以看出，针对C1-C6，爬虫爬行URL资源库中的每一个链接，由于是不翻页抓取，最终得到的HTML静态文件与链接数数量相等。爬行C2的速率是最低的，为25.5kb/s，主要是由于天猫商城的网页结构较为复杂，并包含许多样式，Webkit浏览器加载、解析并渲染网页需要消耗较长的时间。

但是由于在翻页抓取过程中单靠抓取速率公式（式6-2）判断系统性能，会导致爬虫对于分页少的网站和分页多的网站的抓取性能出现较大的差异，分页少的网站抓取速率低，而分页多的网站抓取速率则相对高很多，不能准确地评估爬虫性能。因此，本研究采用如下图表反应分页数与速率的关系。

图6－2 分页数vs.速率

图6-2以C1为例，统计了随着分页数的增长，爬虫速率的变化趋势。可以看到随着分页数的增长，爬虫的速率保持较高较平稳的速率。

从以上的实验数据和分析中，我们可以得出以下结论：

1）本研究设计的爬虫系统能够正确抓取AJAX网站内容及分页数据。

2）在翻页抓取过程中，时延的设置会影响系统的准确度和抓取速率。时延设置过短，可能来不及发送数据加载请求，造成重复状态或数据缺失，系统准确度下降；时延过长，则会造成抓取速率下降。

## 6.4本章小结

本章在支持AJAX的爬虫系统的研究、设计与实现的基础上进行了相应的测试实验，并对实验结果进行了深入的分析。主要工作内容包括：

首先，介绍了系统实验环境。

其次，介绍了评估系统的两个指标。

最后，介绍了系统的实验进行及对实验结果展开了分析。包括对本文所提出的支持AJAX的爬虫系统进行了实验，并给出了实验结果。实验结果证明本研究设计的爬虫系统能够准确抓取AJAX网站中的动态内容及分页数据。

第七章 结束语

针对本系统实验之所得结果进行归纳分析，第一节总结本研究所做的工作；第二节则针对研究未尽完善的部分，提出未来延伸研究与探讨的议题。

## 7.1 结论

随着网络的蓬勃发展，数以万计的网站林立，这些网站除了有公司企业、门户网站，还有相当数量的个人网站以及各种社交网站，它们拥有着众多但却杂乱无章的信息，如果我们把互联网看成一个非常庞大的资料库，那么这些网站则成为了互联网主要的资料来源。

人们使用搜索引擎在如此浩瀚、复杂的巨大资料库中快速、准确地查找需要的资料，而网络爬虫作为搜索引擎的重要组成部分，其抓取页面的数量和质量对搜索引擎的查询结果影响很大。

传统网络爬虫获取页面源码并提取URL的工作方式不适用于爬行AJAX页面。本研究提出了一种支持AJAX的网络爬虫，它能够在AJAX页面的爬行过程中提取运行时的页面内容，采用Webkit浏览器加载HTML网页并解析其中的JavaScript代码，从页面DOM表示中识别用于页面导航的页面元素，自动触发页面元素上的事件，生成并提取新页面的内容，循环工作直到获取AJAX站点中全部页面内容并写入本地静态HTML文件，形成AJAX站点的本地镜像。依据本系统的实验结果与分析后得到以下结论：

1.本研究设计的爬虫系统能够准确抓取AJAX网站动态内容。

2.本研究开发的爬虫系统能够对相似网页结构的目标网页进行分页数据的采集。

本研究的主要工作包括以下几个方面：

1.分析了AJAX爬虫的技术难点，并针对AJAX问题领域进行建模。

2.设计并实现了支持AJAX的定址网络爬虫系统，详细介绍了系统中各个模块功能的具体实现与主要算法。

2.设计并实现了支持AJAX的定址网络爬虫系统。

3.为测试系统准确度与性能设计了详尽的实验，包括针对三大类共六个真实网站做测试，并对实验结果进行讨论。

虽然本研究的目标定位于解决AJAX网站的动态数据采集问题，但是本文提出的方法及实现的系统同样可以应用于传统的静态网页的数据采集。

## 7.2 未来方向

本文对支持AJAX的网络爬虫系统进行了较为深入和全面的研究，并给出了相应的系统实现。且目前系统已经应用到具体项目中去并取得了较好的成果。但通过对实验结果集合的分析与统计，发现仍有许多有待完善之处，概括为以下几点：

（1）提升浏览器的性能

本研究采用基于Qt Webkit组件实现的浏览器加载并解析HTML网页，能够很好地处理客户端的JavaScript代码，但由于Qt Webkit对Webkit内核实现了较为全面的封装，使得我们想直接操作Webkit内核代码较为困难，下一步的工作需要替换Qt Webkit组件，直接采用Webkit源码实现浏览器，这样可以只完成网页的加载与JavaScript解析，而去除网页的布局和绘制过程，从而提升爬虫的性能。

（2）摒弃时延

在传统的Web页面中，使用window对象的onload方法监听页面加载完成事件，或使用document.onreadystatechange的方法来监听状态改变，然后用document.readyState==“complete”判断是否加载完成。但是由于AJAX与服务器进行数据交换时采用异步通信的方式，局部刷新网页内容的时候并不会触发onload或onreadystatechange等事件，因此无法准确判断AJAX动态内容加载完成的准确时间。

目前本研究的解决方案是设置时延等待异步AJAX数据的加载。但时延的设置会影响系统的准确度或性能。时延设置过短，可能来不及发送数据加载请求，造成重复状态或数据缺失，系统准确度下降；时延过长，则会造成抓取速率下降。因此下一步的工作是摒弃时延，找出能准确判断AJAX动态内容加载完成的时间的方法。

（3）基于多线程的网络爬虫的实现

目前本论文实现了基于单线程的网络爬虫，为了提升搜索效率，下一步的工作是改进目前的系统，使之成为基于多线程的爬虫程序。多个线程并发地在网络上协同工作，既可以充分利用网络宽带，又能够利用多线程的并发性来降低Intemet高延迟的副作用，并发抓取网页。

由于本文作者的水平有限，部分问题的研究仍然有待完善与深入。文中所述难免有不当之处，恳请各位专家、同行给予批评指正，在此本人表示衷心的谢意。

参考文献

1. http://web2.socialcomputingmagazine.com, Dion Hinchcliffe
2. Garrett, J.: Ajax: A new approach to web applications. Adaptive path, 2005. <http://www.adaptivepath.com/publications/essays/archives/000385.php>
3. 曾伟辉. 支持AJAX的网络爬虫系统的设计与实现[D]. 中国科技大学
4. Google Mail. <http://www.gmail.com>.
5. Yahoo! Mail. http://mail.yahoo.com.
6. Google Docs. <http://docs.google.com>.
7. Amazon.com. http://www.amazon.com.
8. YouTube. <http://www.youtube.com>.
9. M K Bergman: The Deep Web: Surfacing Hidden Value. 2001, http://www.press.umich.edu/jep/07-01/bergman.html
10. D Florescu, A Levy, A Mendelzon: Database techniques for the World-Wide-Web: a survey, SIGMOD Record, 1998,27(3):59-74
11. C Sherman: The Invisible Web. 2001, <http://www.freepint.com/issues/08060.htm>
12. Alvarez M., Raposo J., Pan A., et al: Crawling the Content Hidden Behind Web Forms[J]. Lecture Notes in Computer Science, 2007, 4702:322-333
13. 罗兵. 支持AJAX的互联网搜索引擎爬虫设计与实现[D]．浙江大学，2007
14. 王映,于满泉,李盛韬. JavaScript引擎在动态网页采集技术中的应用[J]．计算机应用，2004．24(2)：33-36
15. 金晓鸥，钟宝燕，李翔．基于Rhino的JavaScript动态页面解析研究与实现[J]．计算机技术与发展，2008，18(2)
16. 王晶,陈卫卫. AJAX搜索引擎研究.电脑知识与技术2009-07.5(19):5124-5127
17. SHAH S.: Crawling Ajax-driven Web 2.0 applications[R]. 2007
18. FREY G.: Indexing Ajax Web applications[D]. Zurich: Swiss Federal Institute of Technology, 2007
19. Matter, R.: AJAX Crawl: Making AJAX Applications Searchable. Master’s Thesis. ETH, Zurich. (2008)
20. Mesbah, A., Deursen, A. v., and Lenselink, S.: Crawling Ajax-based Web Applications through Dynamic Analysis of User Interface State Changes. ACM Transactions on the Web, 2012
21. ECMA <http://www.eema-international.org>
22. W3C Document Object Model(DOM). http://www.w3.org/DOM/ (2005)
23. XPath
24. 周立柱，林玲.聚焦爬虫技术研究综述.计算机应用.2005(9):1965-1969
25. 关于搜索引擎页面分析中的javascript处理的2个思路[EB/OL]．2006-09-03．http://blog.csdn.net/DanceFire/archive/2006/09/03/1163683.aspx
26. Ajaxian Community. Ajax tools usage survey results[EB/OL].(2007). [2008-10-09]. http://ajaxian.com/archives/2007-ajax-tools-usage-survey-results/
27. jQuery http://jquery.com/
28. 微博七日通
29. 钱程，阳小兰.一种支持Ajax框架的网络爬虫的设计与实现.计算机与数字工程.2012.40(4):69-71
30. Macromedia Flash. <http://www.adobe.com/products/flashplayer/>
31. Trident[EB/OL]. http://en.wikipedia.org/wiki/Trident\_(layout\_engine)
32. Gecko[EB/OL]. <http://zh.wikipedia.org/wiki/Gecko>
33. The Webkit open source project[EB/OL]. http://www.webkit.org
34. Presto[EB/OL]. <http://zh.wikipedia.org/wiki/Presto>
35. 杜娇.基于Webkit手机浏览器的研究与实现[D].南京邮电大学.2012
36. 赵经纬,周余,王自强等. 基于Webkit的嵌入式浏览器的研究与实现[J].电子测量技术 2009年 34期, pages 135-137
37. A Standard for Robot Exclusion．http://www.robotstxt.org/we/robots.html
38. Bloom, B． H．: Space time trade-offs in hash coding with allowable errors. Communications of the ACM[J], 1970
39. Rivest R.: The MD5 Message-Digest Algorithm[J]. RFC 1321, 1992
40. Marc Najork，Allan Heydon．High-performance web crawling[C]. Handbook of Massive Data Sets, Kluwer Academic Publis-hers Inc, 2001:25-45
41. Dustin Boswell. Distributed High-performance web crawlers: A survey of the state of the art[EB/OL]. http://www.cs.ucsd.edu/-dboswell/PastWork/Web Crawling Survey.pdf
42. jsp和servlet核心编程
43. 个性化的Web信息采集技术研究

致谢

本论文得以顺利完成，不是单单只倚靠个人的辛苦以及努力，而是要归功于许多人的指导、协助。在此，希望以我最诚挚的心意，向你们说声“谢谢”。

首先感谢我的指导教授吴国仕教授，恩师总是不辞辛劳地予以教导，而敦厚的品德与对研究学问的严谨态度，更是让我在生活与待人处事上获益良多。在这两年半的日子中，我学习到很多事情，不论是做事情的态度、专业的技能还是面对挑战的能力，都在这两年的日子中经过扎实的磨练。在此向恩师致上最深敬意。同时也感谢评审小组的教授给予我在论文写作上宝贵的意见与指教，使我的论文能更加的严谨与完善。

撰写论文的过程是一种体验，体验研究内容所能展现应用价值的自豪感。不仅让本人对于研究课题有了更深度的了解，更使得个人在技术上得到充分的锻炼与提高。

研究室中温馨且和睦的气氛，彼此互通有无的协助，都是令大家不断进步且努力付出的重要条件。两年的日子里，实验室里共同的生活点滴，不论是技术的研究、论文写作的指导、学术上的讨论、言不及义的闲扯、赶进度的革命情感、共同休闲时的欢乐，感谢实验室里的众位同学：林哲、魏晶晶、刘杨、任建新、范捷琦、刘博轩、肖英男、郭琴。你们的陪伴让两年的研究生活变的绚丽多彩。

最后，一定要感谢一直支持我、关心我的家人与朋友，让我能够在无后顾之忧下，全心完成硕士学位。谨以此论文献给所有关心我与支持我的人，愿你们与我一同分享这份荣耀与喜悦。

攻读学位期间发表的学术论文

[1]Guoshi Wu，Fanfan Liu.Web crawler for event driven crawling of AJAX-based web applications被2012年信息系统、计算及管理国际学术会议**（**ICM2012）录用，大会论文集由Springer旗下的Lecture Notes in Electrical Engineering(LNEE, ISSN：1876-1100)期刊出版，并提交EI Compendex检索。杭州2012年12月8日。论文编号：ICM368.