支持 Ajax 的 Deep Web 爬虫研究与设计①

周 杨

(军事经济学院 基础部计算机教研室，武汉 430035)

摘 要：随着互联网的迅速发展，网络资源日益丰富，如何从 Web 尤其是 Deep Web 中获取信息成为人们关注的焦点，以 Ajax 为基础的新一代网页信息抓取问题也逐渐成为研究热点。通过分析支持 Ajax 的 Deep Web 爬虫关键技术，提出了支持 Ajax 的 Deep Web 爬虫的体系结构，阐述了一种自动爬行 Ajax 网站的算法，为该爬虫的总体框架设计奠定了基础。

关键词：Deep Web；爬虫；Ajax；搜索引擎

Study and Design of an Ajax-Supported Deep Web Crawler

ZHOU Yang

(Computer Teaching and Researching Section, Military Economics Academy, Wuhan 430035, China)

Abstract: With the rapid development of Internet, the network resources are getting more and more abundant, how to extract information from network, especially from Deep Web has been focused on. A new generation of Ajax-based web information extraction has become a hot topic. By analyzing the key technology of the Ajax-supported Deep Web Crawler, this paper puts forward the architecture of the Ajax-Supported Deep Web Crawler, and illustrates an algorithm to crawl the Ajax-supported Deep Web automatically, which lay the foundation for the design of the overall framework of an Ajax-supported Deep Web Crawler.

Key words: deep Web; crawler; Ajax; search engine

随着 Internet 的飞速发展，Web 信息量日益增大，互联网发展报告多次显示当前最重要的网络应用是搜索引擎。这意味着，如何快捷、高效地从互联网上获取所需信息已经成为用户关注的焦点。作为搜索引擎的重要组成部分，网络爬虫起着举足轻重的作用，它是一个自动提取网页的程序，专为搜索引擎从互联网上下载网页。

传统网络爬虫一般通过页面源码中的超链接标签漫游于互联网中，从初始网页开始，不断从当前页面抓取新的链接地址放入队列，直到满足一定的条件时终止操作。而如今的网页大都使用了 Ajax 技术且包含大量的 JavaScript，其页面源码中的超链接远远少于页面呈现中的超链接。因此，传统网络爬虫抓取的内容往往并不完整，这对于搜索引擎的实现极为不利。针对传统网络爬虫的不足，需要设计一种新型爬虫，它能够分析页面的后加载内容，即通过分析页面中的 JavaScript 文件或 JavaScript 代码模拟浏览器的行为，并能够执行相关的 DOM 操作，从而获取真正完整的页面信息[1]。

1 Deep Web及其搜索技术

Web 按其信息蕴含的深度可分为“表层网”(Surface Web)和“深层网”(Deep Web)。Surface Web 是指 Web 空间中由超链接连接的静态网页、文件等资源，是能够被传统搜索引擎索引的页面。而 Deep Web 则是指那些普通搜索引擎由于受技术限制而不能或不作索引的页面或文件，其中的大部分内容无法通过静态链接获取。与 Surface Web 相比，Deep Web 蕴藏着

① 收稿时间:2011-05-24;收到修改稿时间:2011-06-24

更加丰富的信息。

Bright planet 于 2000 年 7 月对 Deep Web 进行了一次较为全面的宏观统计，指出整个 Web 上约有 43000-96000 个 Web 数据库，其中包含的可访问公共信息容量是 Surface Web 的 400-500 倍[2]。2004 年 4 月，UIUC 大学对整个 Deep Web 做了一次较为准确的估算[3]，推测整个 Web 上有 307000 个提供 Web 数据库的网站、450000 个 Web 数据库，而且，Deep Web 中所包含信息的质量更高，对人们更有价值。传统网络爬虫在处理 Deep Web 时难免会遇到诸多问题，因此，对 Deep Web 爬虫的研究是进一步提高互联网信息获取数量与质量的有效途径。

目前，对 Deep Web 进行信息搜索的方法主要分为两类：一类是由数据库所有者向搜索引擎提供后台数据，搜索引擎对提供的数据直接进行抓取；另一类则是通过一定的策略发现网络中存在的 Deep Web 信息资源，并对其进行搜索与抓取。后者因无需从数据库所有者处获取后台数据而具有较好的应用前景，是目前广大研究者所关注的方向。具体而言，Deep Web 搜索涉及的主要技术包括：

(1) 数据源的发现与选择。该技术主要解决如何从网络中自动识别可搜索的数据库，并判断该数据库是否与某个搜索请求相关。

(2) 用户接口与查询规约。该技术主要解决如何将不同的可搜索数据库集成为统一接口供用户提交搜索请求，以及如何将用户提交的搜索请求转化为满足各个数据库查询接口所要求的形式。

(3) 查询结果的抽取与聚合。该技术主要解决如何从数据库查询返回页面上自动识别每个结果记录，并将不同数据库所识别的结果记录抽取为统一格式及进行集成处理[2]。

2 支持Ajax的Deep Web爬虫关键技术

传统网络爬虫在抓取 Ajax 应用中的信息时特别困难，主要是由 Ajax 的一些特性所致。第一，所有 Ajax 应用的共同点是拥有一个 JavaScript 引擎，它作为浏览器扩展运行于浏览器与 Web 服务器之间，任何期望处理此类应用的搜索引擎必需支持脚本语言的执行，也势必增加了网络爬虫设计与实现的复杂性；第二，索引传统的 Web 应用需要跟踪链接、检索并保存每个页面的 HTML 源代码，而在 Ajax 应用中，状态的改变由运行时动态更新 DOM 树所呈现，这意味着搜索引擎必需能够获取这些动态呈现模型；第三，Ajax 能够在运行时动态地加载事件至 DOM 树，从而隐藏不仅是超链接才能够形成的另一个状态，对于传统网络爬虫而言，如何识别这些运行时可点击元素是一项举足轻重的任务。

因此，支持 Ajax 的 Deep Web 爬虫必需在传统网络爬虫的基础上进行相关功能的扩展，其关键技术包括：

2.1 JavaScript 解析

传统网络爬虫抓取网页后只分析源码中的<a>标签，然后通过超链接去抓取其它网页，而对于动态页面，除了<a>标签之外还包含大量的 JavaScript，信息与用户之间已不再是单纯的显示与浏览关系。页面源码由 Web 服务器发送至客户端之前无需经过编译，而只将文本格式的字符代码交由客户端浏览器解释执行。支持 Ajax 技术网页源码的更多内容是通过异步调用方式后加载而来，这些内容在页面源码中均无法显示，必需由 JavaScript 解析器进行分析，找出所有的 Ajax 调用，从而执行这些代码获取从服务器返回的数据。JavaScript 解析的主要步骤包括：

(1) 词法分析：识别嵌入在 html 的 JavaScript 脚本程序，并将其作为输入形成单词链表，以便进行语法分析；

(2) 语法分析：对单词链表依照 JavaScript 的语法规则形成中间数据结构；

(3) 执行控制器：以中间代码为输入，负责对语句解释执行的控制；

(4) 语句解释器：完成各类型控制语句的解释执行，该模块可能会调用解释执行器而形成递归调用。

2.2 DOM 操作解析

对于支持 Ajax 技术的网站，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。由于 JavaScript 解析器只能解析纯 JavaScript，对于操作 DOM 树的代码并不支持，因此就必需扩展 JavaScript 解析器，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 支持Ajax的Deep Web爬虫的总体设计

3.1 体系结构

为了提升搜索引擎识别 Ajax 应用的能力，应将一

个支持 Ajax 的 Deep Web 作为输入提供给爬虫，而最终输出一个包含多页面的镜像网站，它显示与 Ajax 应用相同的内容与结构且能够被搜索引擎索引。这里用图 1 描述支持 Ajax 的 Deep Web 爬虫的体系结构：

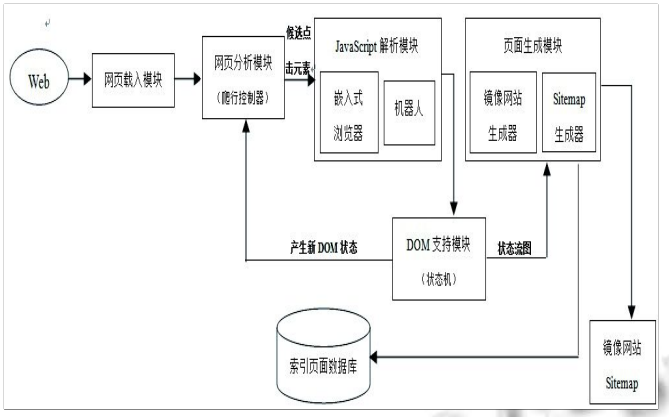

图 1 支持 Ajax 的 Deep Web 爬虫的体系结构

从图 1 可以看出，该体系结构主要包含网页载入模块、网页分析模块、JavaScript 解析模块、DOM 支持模块以及页面生成模块，各模块的主要功能如下：

(1) 网页载入模块用于载入爬行的初始页面，对页面的 HTTP 状态代码进行判断，载入成功则传给后续模块，载入失败则抛出异常；

(2) 网页分析模块用于分析网页载入模块打开的正确页面，分析的内容由网络爬虫的配置文件决定，主要是页面上的标签；

(3) JavaScript 解析模块主要由嵌入式浏览器和机器人组成。嵌入式浏览器是用来测试 Web 的一套独立的 Java API，具有执行 JavaScript 的能力，而且支持 Ajax 必需的技术，如 DOM，XMLHttpRequest 等，可以很容易地将它们导入到具体的开发工具中使用；而机器人则用于模仿嵌入式浏览器中真实的用户点击和输入，从而识别出所有的候选可点击元素并触发其上的事件与响应；

(4) DOM 支持模块时刻关注被爬行网站 DOM 状态的变化，根据 JavaScript 解析模块的运行结果分析 DOM 树所发生的变化，通过更新状态机维护所有的 DOM 状态，生成新的状态流图。同时，新产生的 DOM 状态将继续交由网页分析模块进行分析；

（5）页面生成模块主要由镜像网站生成器和 Sitemap 生成器组成，负责生成由状态机传递过来的状态流图。

3.2 自动爬行模块

在本模块中，爬虫将自动爬行支持 Ajax 的网站，找出所有可能的可点击元素，然后执行相应的操作。这里给出爬虫自动扫描（Crawler Auto Scan）的算法，将该爬虫简称为 ASC，并结合算法阐述本模块的具体实现步骤[4]。

算法 1

```
procedure begin(url,Set tags)
browser←initialBrowser(url)
robot←initialRobot()
stateMachine←initialStateMachine()
crawl(stateMachine,tags)
generateMirrorSite(stateMachine)
generateSiteMap(stateMachine)
end procedure
procedure crawl(StateMachine stateMachine,Set tags)
cs←stateMachine.getCurrentState()
Set C←getCandidateClicks(cs.getDom(),tags)
foreach c∈C do
robot.click(c)
dom←browser.getDom()
if editDistance(cs.getDom(),dom)!=0 then
ns←State(c,dom)
stateMachine.addState(ns)
stateMachine.changeState(ns)
crawl(stateMachine,tags)
stateMachine.changeState(cs)
if browser.history.canback then
browser.history.goBack()
else
browser.reload()
clickThroughTo(cs)
end if
end if
end foreach
end procedure
```

3.2.1 确定候选点击

为了使 ASC 能够自动爬行，必需为它确定爬行入口，通常为网站的 URL，同时应指定需要扫描的 HTML 标签元素名，如<a>、 等，拥有这些标签名的元

素将全部被视为候选点击。完成运行环境的初始化后，ASC 爬行程序开始递归执行，DOM 树中符合条件的元素将被添加到候选点击列表。为了从候选列表中找出确实可点击的元素，ASC 将指示机器人对每个候选元素执行一次点击命令。

3.2.2 比较 DOM 树

在传统的多页面 Web 应用中，每个状态都由一个 URL 表示，而在支持 Ajax 的 Deep Web 中，呈现给用户的往往是单个页面，每个状态的改变都需要由 DOM 树内部结构的改变来表示。通常用状态流图记录那些导致 DOM 树改变的途径，从而建立导航模型。一个 Ajax 网站 W 的状态流图由一个三元组<I,V,E>构成，起点 I（Initial）表示浏览器完全载入 W 后的初始状态；顶点集 V（Vertices）表示所有状态，每个 v∈V 代表 W 的一个运行时状态；边集 E（Edges），每个 e∈E 代表一个可点击事件，状态 v1 通过执行 e 可以到达状态 v2[5]。

为了确定某次点击是否导致了新状态的产生，必需对点击前后的两棵 DOM 树进行比较，使用相关算法计算两棵 DOM 树之间的 Edit distance。若 DOM 树发生了变化，则表明有一个新状态产生，状态机会将该状态加入状态流图中，并且将当前状态指向这个新加入的状态。当某个状态经过有效点击产生新的状态后，新状态中的新内容称为差异更新，由相应的差异算法来发现。此时，为了从差异更新中找到新的候选点击，ASC 的爬行程序将被递归调用。

3.2.3 状态导航

Ajax 的特点决定在一个支持 Ajax 的 Deep Web 中进行导航并不像在传统 Web 中那么简单。一个动态创建的 DOM 状态不会自动被记录在浏览器历史引擎中，因此触发浏览器的后退按钮并不能返回前一个状态，于是增加了爬行 Ajax 的复杂性。若一个 Ajax 应用具有浏览器历史支持，对于浏览器中发生的状态改变，此时只需调用浏览器的后退方法，如算法 1 中的 browser.history.goBack()；若浏览器历史不被支持，返回前一个状态的唯一办法就是重新载入初始状态，然后跟随可点击路径到达目标状态。因此，一个可点击元素应该具备 ID 属性或能够被发现的 XPath 路径，只有这样，当重新载入 Ajax 应用致使页面内容发生改变时，才能确保找到正确的可点击元素，从而通过点击到达目标状态。所有可点击元素的 ID 或 XPath 都应保存在状态机中。对于如下的一段 HTML 代码：

```
<body>
<div>
<span id='myspan1'>"<a id='a1'>test</a></span>
</div>
<div id='mydiv1'></div>
</body>
```

编写相关算法可获取页面元素的 XPath 表达式：“myspan1” 的 XPath 为 “/HTML/BODY[1]/DIV[1]/SPAN[1]”，“a1”的 XPath 为“/HTML/ BODY[1]/DIV[1]/SPAN[1]/A[1]”,“mydiv1” 的 XPath 为 “/HT ML /BODY[1]/DIV[2]”。

3.3 镜像网站生成模块

自动爬行过程一旦执行完毕，所创建的状态流图将被传递给镜像网站及 Sitemap 生成组件。

3.3.1 DOM 转换为 HTML 集

为了使搜索引擎能够发现所有生成的状态，需要为状态流图中的每个 DOM 对象建立链接，检查可点击元素的类型，若元素为超链接，则更新其 href 属性；若元素为其它类型，则将其转换为一个超链接元素。最终，所有元素的 href 属性都将指向即将生成的静态页面。链接建立完毕后，状态流图中的每个 DOM 对象将被转换为相应的 HTML 字符串，保存在一个专用文件夹中，所生成的文件描述了 ASC 在爬行 Ajax 应用过程中所记录的每个状态，包括样式、结构及其具体内容。本过程需要一个处理 HTML 文件的 DOM 解析器，在转换过程中还应对 HTML 的格式进行严格修饰。

将生成页面上传至服务端时，必需确保 CSS 文件和图片信息等不被破坏，使镜像网站与原 Ajax 应用内容一致。原 Ajax 网站可以与镜像网站链接形成搜索引擎的入口点，镜像网站也可以通过各种方式链接至原 Ajax 网站的初始状态。

3.3.2 生成 Sitemap

Sitemap 是一个静态的 XML 文件，最初由 Google 提出，用于站点管理员向搜索引擎爬虫公布站点可被抓取的页面。每个 Sitemap 文件包含一个以上的 URL，每个 URL 则包含更新的周期和时间、该 URL 在整个站点中的优先级等，Sitemap 可以帮助搜索引擎更加有效地对网站内容进行抓取。

ASC 遵守 Sitemap 协议，每次爬行会话后自动生成一个有效的协议实例，包含了爬行过程生成的所有静态

页面的 URL。在 Sitemap 生成器中，ASC 用 Sitemap Schema 创建新的 URL 入口，基于 Schema 生成相应的 Java 对象并将 Java 对象序列化为相应的 XML 实例文档。

3.4 实验结果分析

为了验证支持 Ajax 的 Deep Web 爬虫的可行性与实用性，选择一些网站进行实验，在实验过程中选用 ASC 与 JSpider 分别对这些网站进行信息抓取，并对两者抓取的结果进行比较，从而分析 ASC 的优缺点。

实验的硬件环境为：Pentium(R) Dual-Core CPU E5300 2.60GHz，2GB 内存，软件环境为：Windows XP SP3 、 MySQL5.0 、 MyEclipse6.0 ， 以 http://www.265.com 作为爬行的起始页。实验所用 JSpider 是一个用纯 Java 语言编写的、可配置且高度可扩展的 Web 爬虫引擎，直接在 MyEclipse 中运行主程序，目标网站中能够被识别的资源及状态全部以文件形式保存至本地；对 ASC 进行相关的参数设置后开始进行网页抓取，在运行过程中不断出现对抓取页面进行分析与点击的操作提示，程序运行完毕，所有被抓取的页面、该网站的状态流图及 Sitemap 将自动保存到本地相关文件夹中。用表 1 反应实现结果，分析该表可以得到如下结论：

表 1 ASC 与 JSpider 网页爬行实验结果对比

爬虫类别	爬行标签	候选点击数	实际点击数	识别链接数	爬行用时（ms）	生成文件用时（ms）
ASC	SPAN、DIV、A	628	26	29	107807	559
JSpider	—	—	—	26	13035	0

第一，与 JSpider 相比 ASC 可以获取更多的链接数，很明显是由于网站上某些页面运用了 Ajax 技术，调用执行了 Javascript，从而在页面上生成了新内容，形成了新的状态，ASC 能够识别出这个状态，而 JSpider 却无法识别。第二，与 JSpider 相比 ASC 的爬行效率较低，主要原因在于：抓取更多的页面数决定 ASC 需要花费更多时间；ASC 将所有设定的标签都看成候选可点击元素，而很多候选元素实际上并不会成为点击元素，设定的标签类别越多，爬行所需的时间就越长；ASC 的所有点击行为都在嵌入式浏览器中进行，如果点击行为有效，则会产生一个新的 DOM 状态，而状态的载入又由网速快慢决定，这是程序无法控制的客观因素；Ajax 的网站后退机制导致初始状态不断地被重新载入，致使整个爬行所用时间相应增加；ASC 爬行算法设计过程中难免存在不足，对最终的爬行时间也将产生一定的影响。

4 结语

本文在介绍 Deep Web 及其主要搜索技术的基础上，针对支持 Ajax 的 Deep Web 爬虫所面临的困难，分析了支持 Ajax 的 Deep Web 爬虫的关键技术，并构建了该爬虫的总体框架，从一定程度上解决了爬虫对支持 Ajax 的 Deep Web 信息量的获取问题。而对于 Deep Web 本身的研究内容而言，本文仅取得了一些阶段性成果，尚不完善。伴随超媒体技术及对等网络技术的广泛应用，有关 Deep Web 的许多关键技术均有待进一步突破，各种基于网络的多媒体爬虫技术也将成为最新的研究方向。

参考文献

1 杨丽萍,马继涛,张虹霞.网络搜索引擎分类与发展.情报学报,2006,25:422.

2 Bergman MK. The Deep Web:Surfacing Hidden Value. http://www.brightplanet.com/resources/details/deepweb.html.

3 He H, Meng WY. Automatic integration of Web search interfaces with WISE-integrator. VLDB Journal, 2004, 13(3): 269.

4 罗兵.支持 AJAX 的互联网搜索引擎爬虫设计与实现[硕士学位论文].杭州:浙江大学,2007.

5 郑冬冬,崔志明.Deep Web 爬虫爬行策略研究.计算机工程与设计,2006,27(17):3156.