# 有效的爬行Ajax页面的网络爬行算法

李华波，吴礼发，赖海光，郑成辉，黄康宇

(解放军理工大学指挥信息系统学院 南京 210007)

【摘要】Ajax页面的生成和页面导航需要执行客户端的JavaScript代码，传统网络爬行算法无法获取Ajax页面全部内容。分析了Ajax的工作方式，阐述了爬行Ajax网页所面临的主要问题，提出并实现了一种有效爬行Ajax页面的网络爬行算法。该算法可控制客户端浏览器动态生成页面内容和完成页面导航，为爬行过的页面分配标识编号并生成相应静态页面。实验结果表明，提出的算法所爬行的Ajax页面数量明显多于传统方法，同时，采用的双重消重策略可有效减少算法的时间耗费。

关 键 词 Ajax; 爬行算法; 消重策略; 搜索引擎
中图分类号 TP393.08 文献标志码 A doi:10.3969/j.issn.1001-0548.2013.01.024

# Efficient Algorithm for Crawling Ajax Web Pages

LI Hua-bo, WU Li-fa , LAI Hai-guang, ZHENG Cheng-hui, and HUANG Kang-yu

(College of Command Information System, PLAUST Nanjing 210007)

**Abstract** The generation of Ajax web pages and the Ajax page navigation must execute the client JavaScript, thus it is impossible to extract the complete content of an Ajax page through the traditional crawling algorithms. In this paper, the working mode of Ajax is analyzed, the problem of crawling Ajax web pages is elaborated, and an effective algorithm for crawling Ajax pages is proposed. The algorithm can realize the dynamic generation of Ajax web contents in client browser and the navigation of Ajax web pages, and also it can assign identification number for the crawled pages whose static pages can be generated. Experimental result shows that the number of Ajax pages crawled by the proposed algorithm is obvious bigger than the traditional ones', and the presented replicas-detecting policies can effectively reduce the time consumption of the algorithm.

**Key words** Ajax; crawling algorithm; replicas-detecting policy; search engine

近年来，Ajax 技术以其良好的用户体验、优良的设计模式、较快的响应速度，在 Web 开发领域得到广泛的认可和应用。Google 的系列服务 Google Suggest、Google Group、Gmail 等，以及新浪和网易的个人博客等都是 Ajax 技术的典型应用。但是，当前 Ajax 技术开发的网页却无法被 Google、Yahoo 等主流的搜索引擎完整地检索，其原因在于 Ajax 页面是通过执行客户端浏览器中脚本代码动态生成的，多个页面可具有相同 URL 地址，导致上述搜索引擎的网络爬虫无法爬行 Ajax 页面内容，无法提供可被索引的 Ajax 页面资源。因此有必要对适用于 Ajax 的爬虫技术进行研究，以支持搜索引擎检索 Ajax 页面。

本文针对传统网络爬行算法应用于Ajax站点时存在的问题，提出一种Ajax站点的网络爬行算法dAjaxCrawling。该算法利用Web自动测试工具控制IE浏览器提取Ajax页面内容；通过引入消重策略，避免重复爬行相同页面，减少Ajax网络爬行算法的时间耗费；引入静态页面生成算法，生成Ajax站点的本地镜像；引入页面导航算法，自动触发页面事件，生成下一个待爬行页面。最后，以新浪博客为实验对象对dAjaxCrawling算法的性能进行了评估。

## 1 相关工作

文献[1]提出了一种基于事件驱动的爬行 Ajax 页面的设想，自动控制浏览器触发页面元素上的事件以生成新页面，分析浏览器中的 DOM 树结构以提取页面内容，分析 JavaScript 代码以提取链接地址。文献[2]将 Ajax 应用程序视为一系列程序状态、事件组成的状态变迁图，图中节点表示通过执行客户端 JavaScript 代码生成的程序状态，边表示程序中事件引起的状态变迁。该方法通过解析和执行 JavaScript 生成 Ajax 页面内容，使得爬行过程较为

收稿日期：2011－04－12； 修回日期：2012－09－18
基金项目：江苏省自然科学基金(BK2010132)
作者简介：李华波(1981－)，男，博士，主要从事网络安全方面的研究.

复杂，控制较为困难，并且基于页面内容哈希值识别重复页面的工作方式效率不高，会重复执行相同的 JavaScript 代码、生成相同页面内容。文献[3]将 Ajax 站点表示为状态流程图，Ajax 站点中的页面视为流程图中的状态，页面间的联系视为图中的状态变迁。文献系统地提出了一种爬行 Ajax 站点的算法，但类似于文献[2]，该算法基于页面内容哈希值识别重复页面，运行效率不高，会重复生成相同页面内容，并耗费大量爬行时间。文献[4]扩展了传统网络爬虫的功能，使其能够解释和执行 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。该方法采用的仍然是解析和执行 JavaScript 代码的工作方式，控制流程复杂，局限于依据 URL 地址爬行页面，爬行 Ajax 站点时页面覆盖率不高。文献[5-6]分别利用开源 JavaScript 引擎 SpiderMonkey、Rhino 解析和执行动态页面中的 JavaScript 代码，获取页面包含的所有 URL 地址，完成爬行任务。该方法限于提取超链接 URL 地址，只适用于爬行普通的动态网页，还无法直接用来爬行 Ajax 页面。

## 2 dAjaxCrawling算法

传统网络爬虫无法爬行 Ajax 页面，其原因在于：Ajax 页面源码并没有包含全部页面内容，它包含的是 HTML 和 CSS 代码以及对 JavaScript 代码组成的 Ajax 引擎，真正的页面内容则是通过执行对 Ajax 引擎的 JavaScript 调用，获取服务器响应数据，利用文档对象模型(document object model，DOM)操作动态生成的。传统网络爬行算法直接根据页面源码提取页面内容的工作方式显然不适用于爬行 Ajax 页面。另外，不同 Ajax 页面可具有相同 URL 地址，因为新页面的生成不需要载入一个完整的页面，局部更新原有页面即可，在此过程中 URL 地址保持不变。传统网络爬行算法根据 URL 获取新页面的工作方式不适用于爬行 Ajax 页面。

基于上述分析，本文提出了用于爬行 Ajax 页面的网络爬行 dAjaxCrawling 算法，它能够分析并提取运行时的页面内容，从页面 DOM 表示中识别导航元素，为其目标页面分配页面编号，以代替 URL 地址标识页面，从而避免重复爬行相同页面，自动触发导航事件，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。

### 2.1 算法假定

为降低 dAjaxCrawling 算法复杂度，作如下假定：

1) Ajax 站点中页面无输入表单，避免爬行需要提交表单数据以完成页面导航的 Ajax 站点；

2) Ajax 页面是基于文本的，不考虑爬行页面是基于图像的 Ajax 站点，以避免出现状态爆炸现象而无法完成爬行任务；

3) JavaScript 函数调用具有不变性，即在爬行过程中，对 Ajax 引擎发出的相同 JavaScript 函数调用将产生相同的结果。

### 2.2 基本定义

为了算法描述的方便，做如下定义：

**定义 1** 导航元素：若触发页面元素上的事件后形成新页面，称该页面元素为导航元素，称生成的新页面为导航元素的目标页面。导航元素可用一个五元组〈eID, eXPATH, navEvent, navInfor, tpNum〉表示，记为 navElement。其中，eID 表示页面元素的 ID 属性，eXPATH 表示页面元素在页面 DOM 树结构中的 XPATH 表达式，navEvent 表示绑定的导航事件类型，navInfor 表示导航信息(如页面元素的 href、onclick 以及 onmouseover 属性值)，tpNum 表示目标页面编号。

**定义 2** 导航关系：对于两个页面 $P_i$ 和 $P_j$，若触发页面 $P_i$ 中某一导航元素 navElement 上的导航事件后可生产页面 $P_j$，称 $P_i$ 与 $P_j$ 之间存在一个导航关系 navRelation，用二元组<pNum1, navElement>表示。其中，pNum1 表示页面 $P_i$ 的页面编号，navElement 表示导航元素。

### 2.3 设计思想及描述

算法 1 给出了 dAjaxCrawling 算法的主要步骤。首先导航至 Ajax 站点起始页面，将其页面编号 0 加入全局页面编号队列 num$Q$。num$Q$ 非空时，取出队头编号 cpNum，其目标页面就是下一个待爬行的页面。cpNum 非 0 时，调用页面导航算法 PNBN(page navigating by number algorithm)导航至 cpNum 的目标页面。根据当前页面 DOM 表示，调用页面编号分发算法 PND(page number distributing algorithm)，提取导航元素集 navElementSet，为其中每个导航元素目标页面分配页面编号并加入 num$Q$，然后调用静态页面生成算法 SPG(static page generating algorithm)，生成该页面对应的本地静态页面。最后 Ajax 站点的本地镜像形成，爬行 Ajax 站点的过程结束。

算法 1 AjaxCrawling 算法

initURL: Ajax 站点起始页面地址; cpNum: 当前

页面编号; num*Q*: 全局页面编号队列; dom: 当前页面 DOM 表示; navESet: 当前页面的导航元素集。
initPage(initURL) //导航至 Ajax 站点起始页面
while num*Q* 非空{
  cpNum = num*Q*.deHead(); // cpNum 出队列
  if(cpNum != 0){
    PNBN(cpNum); // 导航至 cpNum 目标页面
  }
  navElementSet=PND(dom, cpNum);//提取导航元素集
  SPG(dom, navESet);//生成静态页面
}

### 2.4 页面编号分发算法PND

#### 2.4.1 导航元素提取

dAjaxCrawling 算法首先解析页面内容，然后依据页面元素标签类型提取导航元素，触发导航事件后即可生成更多可被爬行的新页面。算法限定提取绑定 onclick 或 onmouseover 类型事件的导航元素和指向本站点中页面的超链接(导航事件类型视为 onclick)。

#### 2.4.2 消重策略

页面编号分发算法运行过程中，如不采取任何限制策略，将为每个导航元素的目标页面分配不同页面编号，但这些导航元素可能会具有相同目标页面，从而导致相同页面被重复分配不同页面编号，该页面将被重复多次爬行。为消除重复分配页面编号的问题，需引入消重策略[2-3]。在已有的消重策略中，页面编号的分发必须在页面生成后才能进行，对于每个导航元素，都需要触发导航事件生成目标页面，然后确认该页面是否已分配页面编号，这将导致重复生成相同页面，增加爬行算法时间复杂度。因此，需要改进消重策略。

在 Ajax 站点中，页面的生成需要执行页面指令，可认为页面指令与目标页面一一对应，而且如果两条页面指令完全相同，则它们的目标页面必定相同。本文正是根据这些特点，通过分析页面指令，在页面生成前预先分配页面编号，改进消重策略：每次提取一个导航元素后，不执行相应的页面指令生成目标页面，只形成该页面指令的表示，将它与先前已分析过的所有页面指令的表示进行比对。若不存在匹配，先前未分析过该页面指令，为其目标页面分配页面编号；否则，已分析过该页面指令，无需为其目标页面新分配页面编号。页面指令通过导航信息表示：若导航信息为 URL 地址，直接用 URL 地址表示；若导航信息通过 onclick 或 onmouseover 属性值所得，页面指令为 Ajax 指令，由 Ajax 引擎地址 eurl 和 JavaScript 函数调用 funcall 两部分组成，表示形式为 eurl:funcall。为减小存储已分析过的页面指令表示的开销、加快页面指令比对速度，引入页面指令哈希表 pIstrhTable，Key 存储利用 ELFhash 哈希算法[8]计算得到的页面指令摘要，Value 存储目标页面的页面编号。

不过，单纯通过分析页面指令分配页面编号还不够完善，原因在于两条不同的页面指令却有可能形成相同的页面，仍将导致为相同页面重复分配不同页面编号。对于这种情形，可借鉴单重消重策略的做法，维护一个页面内容哈希表 phTable。对于页面编号为 cpNum 的页面，计算页面内容摘要 cpfp 并查询 phTable，判断是否存在 Key 为 cpfp 的表项。如不存在，表明该页面只有唯一的页面编号 cpNum，不可能是两条不同页面指令的目标页面，向 phTable 中添加表项<cpfp, cpNum>；如存在，表明该页面还有一个与 cpNum 等价的页面编号 dpNum，而且存在两条不同页面指令，执行它们后都将生成该页面，该页面先前已被爬行，后续过程中均通过 dpNum 标识，需更新导航关系表和页面指令哈希表，即将目标页面编号为 cpNum 的导航关系的目标页面编号更新为 dpNum；将页面指令哈希表中 Value 为 cpNum 的表项的 Value 更新为 dpNum。

综上分析，改进后的消重策略可视为一种基于页面指令与页面内容的消重策略：通过分析页面指令，在页面生成前预先分配页面编号；通过分析页面内容，消除多个页面编号标识相同页面的情况。称改进后的策略为双重消重策略(double replicas detecting policy，DRDP)，改进前的策略为单重消重策略(single replicas detecting policy，SRDP)。与 SRDP 相比，DRDP 的主要优点是耗时更少，但由于需要额外维护一个页面指令哈希表，空间耗费将比 SRDP 大一些。采用 DRDP 的页面编号分发 PND 算法的详细步骤如算法 2 所述。

算法 2　PND 算法

cpNum:当前页面编号; dpNum:cpNum 的等价页面编号; cpfp:当前页面内容指纹摘要; phTable:页面内容哈希表; pIstrhTable: 页面指令哈希表; navRelTable: 导航关系表; navESet: 导航元素集; elementList:当前页面中的页面元素列表; pIstr:页面指令表示; npNum:pIstr 目标页面编号; elementList: 当前页面中的页面元素列表; Counter:全局计数器;

navEle:通过 npNum 构建的导航元素; navRel:通过 navEle 生成的导航关系。

```
dpNum=queryhTable(cpfp); //查询页面内容哈希表
if(dpNum != null){
  update pIstrhTable and navRelTable;
  navESet=null;
}
else{
  add <cpfp,cpNum> to phTable;
  foreach e in elementList{
  pIstr=genpIstr(e);
  if(isValid(pIstr)){ //判断是否为站内指令
    pIstrfp=ELFhash(pIstr);// 页面指令指纹摘要
    npNum=querypIstrhTable(pIstrfp);
    if(npNum==null){//判断页面指令是否已被分析
      npNum=Counter+1;//生成目标页面编号
      numQ.add(npNum);//目标页面编号入队列
      add <pIstrfp,npNum> to pIstrhTable;
    }
    // 构造导航元素、导航关系
    generate navEle,navRel from e and npNum;
    add navEle to navESet;
    write navRel to navRelTable
}}}
  return navESet;
```

对于 cpNum 标识的当前页面，算法首先查询页面内容哈希表 phTable 中是否存在 Key 为当前页面内容摘要 cpfp 的表项。若存在，表明页面已被爬行，获取等价页面编号 dpNum，依据 dpNum 更新页面指令哈希表 pIstrhTable 和导航关系表 navRelTable，导航元素集设置为空；否则，表明页面未被爬行，向 phTable 中添加表项<cpfp, cpNum>。然后根据页面标签提取页面元素列表 elementList，对于其中每一个页面元素 *e*，形成页面指令表示 pIstr。若 pIstr 为站内指令，计算摘要 pIstrfp，查询 pIstrhTable；若查询结果为空，表明该页面指令还未被分析，为它的目标页面分配页面编号 npNum 并加入全局页面编号队列，同时向页面指令哈希表添加表项<pIstrfp, npNum>。然后根据页面元素 *e* 和页面编号 npNum，构造导航元素 navEle 和导航关系 navRel，将 navEle 加入导航元素集 navEleSet，将 navRel 写入导航关系表 navRelTable。算法运行结束后，输出导航元素集 navEleSet。

#### 2.4.3 生成导航关系

对于 cpNum 标识的当前页面，页面编号分发算法从中提取导航元素 navElement 后，将为 navElement 的目标页面分配页面编号 tpNum，然后生成一个导航关系并添加至导航关系表。若 navElement 的目标页面为 Ajax 站点内页面，新生成的导航关系为〈–1, navElement〉；否则，新生成的导航关系为〈cpNum, navElement〉。

### 2.5 静态页面生成算法SPG

SPG 算法根据 PND 算法输出的导航元素集生成当前页面对应的本地静态页面。静态页面的生成并不是简单地将页面内容写入到本地 html 文件，需要将页面元素的导航信息更新为其他静态页面名。这样使得在本地镜像中即可浏览与 Ajax 站点中相同的页面内容，并且保持页面之间的链接关系。静态页面生成 SPG 算法的主要步骤如算法 3 所述。

算法 3 SPG 算法

dom: 当前页面 DOM 表示; navESet: 导航元素集; content: 更新后的页面内容; cNum: 当前页面编号; element: 页面元素; navInfo: 导航信息.

```
dom.setDesignMode(true);//页面 DOM 置为可编辑状态
foreach nav in navESet{
  element=locateElement(dom,nav);//定位页面元素
  if(element != null){
    navInfo=nav.tpNum+".html";
    updatenavInfo(element,navInfo);//更新导航信息
}}
Dom.removeScript(); //删除脚本代码
saveToHtml(content,cNum); //新内容写入本地页面
```

为更新页面元素的导航信息，算法首先将当前页面 DOM 表示设置为可编辑状态。对于导航元素集中每个元素，根据其 ID 属性或 XPATH 表达式定位相应的页面元素 element。利用其目标页面编号生成新导航信息 navInfo。若 element 为超链接，更新 href 属性为 navInfo；否则，将其替换为 href 属性为 navInfo 的超链接。为消除浏览静态页面时脚本代码的影响，还需要删除页面源码中的 Ajax 引擎以及其它脚本代码，提取更新后的页面内容，写入到本地静态页面中。

### 2.6 页面导航算法PNBN

PNBN 算法通过查询 PND 算法中形成的导航关系表，获取导航至下一个待爬行页面的导航路径。PNBN 算法读取页面编号队列头编号，查询导航至目标页面的最短导航路径，并导航至目标页面。若目标页面为传统页面或 Ajax 主页面，根据 URL 地

址就可直接完成页面导航，最短导航路径中只包含一条导航信息为目标页面 URL 地址的导航关系；若目标页面为 Ajax 从页面，无法直接根据 URL 地址完成页面导航，需首先导航至 Ajax 主页面，然后通过广度优先搜索策略获取从 Ajax 主页面到 Ajax 从页面的最短导航路径，逐步触发导航事件，导航至目标页面。PNBN 算法的主要步骤如算法 4 所述。

算法 4　PNBN 算法

nextNum: 页面编号队列头编号; navPath: 最短导航路径; navRel: navPath 中下一条导航关系; pNum: navRel 的目标页面编号.

```
navPath=getShortestPath(nextNum);//查询最短导航路径
navRel=navPath.getHeadRel();//取第一条导航关系
while(navRel!=null){
  pNum=navRel.tpNum;
  if(pNum==–1) //根据 URL 导航至 Ajax 主页面
    navToPage(navRel.navInfo);
  else triggerEvent(navRel.navElement);//触发导航
  navRel=navPath.getNextRel();
}
```

PNBN 算法首先查询到达页面编号队列头编号 nextNum 目标页面的最短导航路径 navPath，从中选择下一个导航关系 navRel。navRel 非空时，取目标页面编号 pNum。若 pNum 为–1，表明 navRel 中包含 Ajax 主页面或传统页面的地址 url，直接根据该地址导航至所标识的页面；否则，自动触发 navRel 中导航元素上的导航事件。重复运行算法直到 navPath 中导航关系提取完毕，最终将导航至 nextNum 目标页面。

# 3　dAjaxCrawling算法性能评估

本节通过实验评估和分析 dAjaxCrawling 算法的性能：1) 对比分析采用双重消重策略、单重消重策略后爬行算法的性能，以表明双重消重策略可有效地节省爬行时间；2) 对比分析传统网络爬行算法、dAjaxCrawling 算法在爬行 Ajax 站点时的性能，以检查 dAjaxCrawling 算法能否爬行更多页面。

## 3.1　实验环境及数据

新浪博客中每篇文章由多页组成，每页的评论均采用 Ajax 技术加载到页面中，因此本文选定新浪博客作为实验用的 Ajax 站点。传统网络爬行算法只能爬行该站点中每篇文章第一页的内容，无法获取其他页的内容。采用 dAjaxCrawling 算法，可爬行站点中所有页面内容。实验数据总共 5 组，分别取自新浪某博客(http://blog.sina.com.cn/u/1189729754)的前 50、150、300、500 篇以及全部 725 篇文章，基本情况如表 1 所示。可以看出，该 Ajax 站点中大部分文章包含多个页面。要爬行该站点中的页面，采用 dAjaxCrawling 算法是很有必要的。

**表1　实验数据基本情况表**

| 编号 | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ |
|---|---|---|---|---|---|
| 文章总数 | 50 | 150 | 300 | 500 | 725 |
| 页面总数 | 381 | 1063 | 1416 | 2037 | 2764 |

## 3.2　双重消重策略性能分析

本节从选定的 Ajax 站点中抽取前 20、40、60、80、100 篇文章，构造 5 组采样数据用于实验，从爬行时间耗费、页面吞吐率两个方面对采用单重消重策略、双重消重策略的 Ajax 网络爬行算法性能进行对比和分析。

### 3.2.1　消重效果

利用 dAjaxCrawling 爬行 5 组采样数据，对比爬行过程中分析过的页面指令总数和最终生成的静态页面总数，如图 1 所示，爬行过程中分析过的页面指令总数远大于生成的静态页面总数。原因在于 Ajax 站点中每篇文章包含大量重复页面指令，采用双重消重策略后，dAjaxCrawling 算法会判断页面指令是否重复，不重复时才为其目标页面分配页面编号，并最终爬行该页面编号标识的页面和生成相应的本地静态页面。

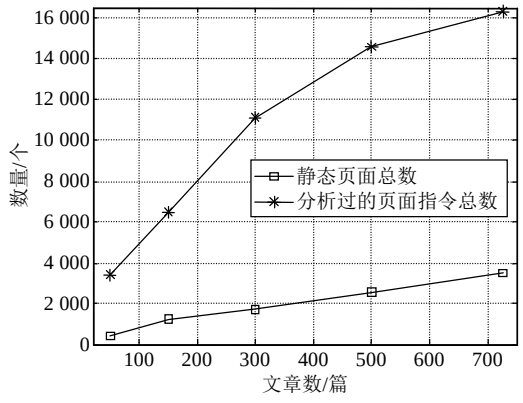


图 1　双重消重策略消重效果图

### 3.2.2　爬行时间对比

dAjaxCrawling 算法可避免重复执行相同的页面指令，减少重复生成相同页面的次数，从而减少爬行时间，其效果可从图 2 所示的时间对比看出。

由图可知，dAjaxCrawling 算法耗时明显少于

sAjaxCrawling 算法，表明双重消重策略能有效减少爬行算法耗时。原因在于 dAjaxCrawling 算法会在页面生成之前，通过比对页面指令摘要判断它是否是重复页面，不需要等到页面生成后再根据页面内容来判断，从而节省爬行时间。

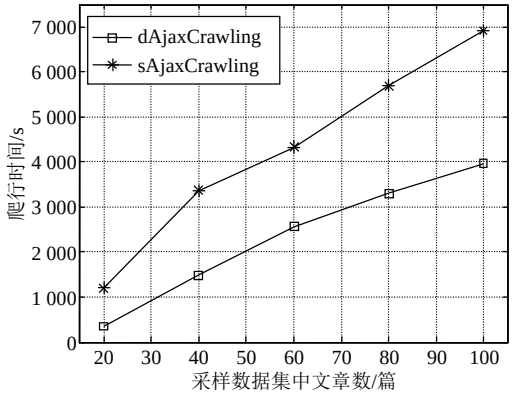


图 3 爬行时间对比

### 3.3 与传统网络爬虫的性能比较

本节分别利用传统网络爬虫 Heritrix、WebSPHINX 与基于 dAjaxCrawling 算法实现的 Ajax 网络爬虫爬行表 1 中 $S_1$、$S_2$、$S_3$、$S_4$、$S_5$ 共 5 组数据，对比传统网络爬行算法和 dAjaxCrawling 算法的性能。其中，Heritrix 采用多线程的工作模式，WebSPHINX 选择单线程工作模式。3 种爬虫爬行 5 组数据的时间耗费、爬行页面数量对比如表 2 所示。

实验结果表明，Ajax 网络爬虫爬行的页面数量远大于传统网络爬虫。原因在于传统网络爬行算法不支持 Ajax 引擎的运行，无法通过执行对 Ajax 引擎的 JavaScript 调用来生成页面内容，只能爬行每篇文章第一页的内容。对于每个页面的平均爬行耗时，dAjaxCrawling 算法耗时为 WebSPHINX 的 5 倍多，为 Heritrix 的十几倍，主要原因在于 dAjaxCrawling 算法运行过程中需要开启、关闭浏览器，提取导航元素，自动触发导航事件，执行页面指令，并且工作在单线程模式，每次只能分析和处理一个页面。

**表2 3种网络爬虫的爬行性能对比表**

| 编号 | DAjaxCrawling | | | Heritrix | | | WebSPHINX | | |
|---|---|---|---|---|---|---|---|---|---|
| | 爬行的页面数 | 总时间/s | 每页平均耗时/s | 爬行的页面数 | 总时间/s | 每页平均耗时/s | 爬行的页面数 | 总时间/s | 每页平均耗时/s |
| $S_1$ | 381 | 2 077.595 | 5.453 | 50 | 15.262 | 0.305 | 50 | 47.309 | 0.946 |
| $S_2$ | 1 063 | 5 546.079 | 5.217 | 150 | 41.887 | 0.279 | 166 | 163.192 | 0.983 |
| $S_3$ | 1 416 | 7 715.137 | 5.449 | 300 | 83.009 | 0.277 | 300 | 270.225 | 0.901 |
| $S_4$ | 2 037 | 10 648.165 | 5.227 | 500 | 146.369 | 0.293 | 500 | 447.884 | 0.896 |
| $S_5$ | 2 764 | 15 528.475 | 5.618 | 725 | 208.934 | 0.288 | 725 | 679.325 | 0.937 |

## 4 结 束 语

本文在深入分析 Ajax 工作方式的基础上，阐述了爬行 Ajax 网页所面临的问题，设计实现了一个用于爬行 Ajax 页面的网络爬行算法。实验结果表明，该算法能够有效爬行 Ajax 页面内容，消重策略可有效减少算法时间耗费。下一步的研究工作包括进一步提高 Ajax 网络爬行算法性能，基于该算法实现一个 Ajax 网页搜索引擎等。

## 参 考 文 献

[1] SHAH S. Crawling Ajax-driven Web 2.0 applications [EB/OL]. [2011-01-18]. http://www.infosecwriters.com/texts.php?op= display&id=539.

[2] FREY G. Indexing AJAX Web applications[D]. Zurich, Switzerlang: Swiss Federal Institute of Technology, 2007.

[3] MESBAH A, BOZDAG E, DEURSEN. VAN A. Crawling AJAX by inferring user interface state changes[C]// Proceedings of the 8th International Conference on Web Engineering. New York, USA: [s.n.] , 2008.

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

LUO Bing. The design and implement of AJAX-enabled internet search engine crawler[D]. Hangzhou: Zhejiang University, 2007.

[5] 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.
WANG Ying, YU Man-quan, LI Sheng-tao. Extracting dynamic URLs using JavaScript engine[J]. Journal of Computers Applications, 2004, 24(2): 33-36.

[6] 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): 1-4.
JIN Xiao-ou, ZHONG Bao-yan, LI Xiang. Research and implementation of interpreting JavaScript dynamic Web page based on Rhino engine[J]. Computer Technology and Development, 2008, 18(2): 1-4.

[7] 张世永. 网络安全原理与应用[M]. 北京: 科学出版社, 2006.
ZHANG Shi-yong. Network security principle and application[M]. Beijing: Science Press, 2006.

[8] 李晓明, 凤旺森. 两种对URL的散列效果很好的函数[J]. 软件学报, 2004, 15(2):179-184.
LI Xiao-ming, FENG Wang-sen. Two effective functions on hashing URL[J]. Journal of Software, 2004, 15(2): 179-184.

编 辑 张 俊