面向动态网络的爬虫的设计与实现

# ABSTRACT

This paper describes the design and implement of a crawler against the dynamic Web pages. As the Web technology develops, AJAX has been widely nested into various Web pages as to improve the user experience and provide a friendly interactive interface.

# 1.简介

网络爬虫（也称为网络机器人，网络蜘蛛，网络蠕虫，网络漫步者）几乎与网络本身一样古老。 第一个网络爬虫，马修·格雷的网络漫步者是1993年春天写的，与NCSA Mosaic浏览器的第一个版本大致相符。 在起初的两个万维网会议上提出了几篇关于网络爬行的论文。 然而，当时网络比现在的网络要小两到三个数量级，所以这些系统并没有解决当今网络爬网所固有的扩展问题。而当时更是没有AJAX技术，因此动态网页尚且无从谈起，面向动态网页的爬虫也并未诞生。

显然，所有流行的搜索引擎都使用能够覆盖到大部分网络的爬虫。 然而，由于搜索引擎业务的竞争性，这些爬虫的设计尚未被公开描述。Google搜索引擎是一种使用多台机器进行抓取的分布式系统。 爬虫由运行在不同进程中的五个功能组件组成。 URL服务器进程从文件中读取URL并将其转发到多个爬虫程序进程。 每个爬虫进程在不同的机器上运行，是单线程的，并且使用异步I / O并行从多达300个网页服务器获取数据。 爬虫程序将下载的页面发送到单个存储服务进程，该进程压缩页面并将其存储到磁盘。 然后通过索引器进程从磁盘读回页面，该进程从HTML页面中提取链接并将其保存到不同的磁盘文件。 URL解析器进程读取链接文件，使其中包含的URL绝对化，并将绝对URL保存到URL服务器读取的磁盘文件中。

淘宝，京东，微信朋友圈都是着名的AJAX网站。 他们的目标是通过在浏览器中运行客户端代码来增强用户体验，而不是需要用户始终刷新网页，并尽量减少服务器流量。 传统的网页应用程序，如淘宝，京东还开始在传统的页面中包含AJAX内容，以便为用户提供更高的交互性。 AJAX应用程序是一个给定URL的动态网页应用程序，通常基于Javascript，它向用户呈现用户通过UI事件更改的不同状态。 采用AJAX应用程序可以无缝地呈现应用程序的所有状态，而无需更改URL。 这导致与类似Google的呈现结果由URL唯一标识的搜索引擎的当前搜索模型不匹配，因而为搜索引擎对其内容进行索引带来了很多麻烦。

目前的搜索引擎忽略AJAX内容，因为爬行AJAX有以下困难和缺陷：

（1）没有缓存/预抓取。 当前的搜索引擎预先缓存网页并在本地进行爬行，但AJAX事件无法缓存，因而很难进行缓存并对页面建立索引。

（2）大量重复状态。 在一个页面中，往往几个事件可以导致相同的状态，因为它们使用相同的底层Javascript函数渲染内容。 因此，对状态进行去重对于优化和减少应用程序视图的大小至关重要。

（3）事件的粒度很细。 这可能导致一大堆非常相似的状态。

（4）需要加载很多资源。对于普通文档的爬行仅需要加载HTML文档和指定的多媒体文档，而Javascript文件可以说好无意义，而这对于AJAX页面来说却是不可或缺的部分。页面的加载速度往往和其中需要加载的资源数量有很大关系，加载一份资源需要有HTTP的请求和响应，对于数量很多而每部分内容很少的资源来说，HTTP的请求头和响应头所占的比重会非常大，这份工作的I/O开销也随之变得巨大。

本文就将面向动态网络（以下对动态网络中的页面简称为AJAX页面）的爬虫的设计，实现和优化进行研究和汇报。

# 2.背景与相关工作

我们知道，网络正在经历一个重大的变化。在Web 2.0的支持下，一个技术得到了飞速发展，它就是AJAX（Asynchronous JavaScript And XML），其中JavaScript和文档对象模型（DOM）操作的巧妙组合，以及异步服务器通信实现了高度的用户交互性。

网络爬虫按照系统结构和实现技术，可分为通用网络爬虫（General Purpose Web Crawler）、聚焦网络爬虫（Focused Web Crawler）、增量式网络爬虫（Incremental Web Crawler）、深层网络爬虫（Deep Web Crawler）。其主要应用场景是为搜索引擎采集信息数据。而由此目的制作的网络爬虫大多仅需要抓取静态页面文档，因此，现今有关动态网页，即由表单和AJAX动态加载的文档信息的网络爬虫仍然有待完善和发掘。

随着Web 2.0的发展，页面中的AJAX也越来越多。由于传统爬虫依靠静态分析，不能准确的抓取到页面中的AJAX请求以及动态更新的内容，已经越来越不能满足需求。基于动态解析的Web 2.0爬虫应运而生，通过浏览器内核解析页面源码，模拟用户操作，能有效解决上述问题。虽然动态网页爬虫的应用前景尚不明朗，但随着AJAX技术的成熟和完善，越来越多的文档以异步，动态的方式加载。如果忽视动态加载的文档信息，爬虫所获取的数据将有很大一部分缺失。

在传统的Web应用程序中，每个页面都有唯一的URL，而在AJAX应用程序中，每个状态都不能由唯一的URL表示。 特定的URL可以具有许多具有不同内容的状态。 动态内容通过Javascript添加到DOM（文档对象模型）。在AJAX应用程序中，客户端事件触发网页的DOM结构的变化。 为了爬行特定页面中的众多状态，需要调用这些客户端事件。 如果我们只考虑点击事件。 首先，我们需要识别绑定了点击事件的HTML元素。 然后必须在这些元素上调用click事件。而AJAX页面可以由状态机表示。 因此，AJAX驱动的网页的导航模型可以被表示为状态机。爬行的第一步是加载初始状态，然后等待后台Javascript执行（这将处理使用onLoad事件进行AJAX调用的情况）。 然后找到处于初始状态的所有可点击元素，并调用该事件。如果有DOM更改，则状态机会更新。 爬行以广度优先策略完成。 首先，找到源自初始状态的所有状态。 然后以类似的方式爬取每个状态。但AJAX爬虫的效率是很低的。即便并发执行多个任务，AJAX 爬虫的爬行效率都比相应的普通Web 爬虫差很多。除了实验和环境因素，AJAX 爬虫解析Javascript脚本并更新DOM树的操作将会产生不小的时间开销，同时其I/O开销更是普通爬虫的数万倍。当关闭动态页面的支持时，AJAX 爬虫的速度将与普通Web爬行器几乎没有差别。但是相对于普通爬虫，AJAX 爬虫的主要缺点是需要更多的计算资源。而这一缺点是致命的，也是各大搜索引擎不对AJAX加载的内容进行索引的原因。

本论文将会就动态网页爬行的设计，实现，以及优化进行分析和描述。第三节将会详细描述动态网页爬虫各个模块功能的设计。第四节将会就设计过程中分析到的技术难题和问题进行分析和讨论。第五节将会就动态网页爬虫的各个模块的具体实现进行详细描述。

# 3.设计

本论文致力于面向动态网页的爬虫的设计和实现。如同一般的通用爬虫，动态网页爬虫也必须要有URL队列，页面爬行，链接提取，内容入库，文档处理等功能。这些功能相互独立，因此，模块化将是一个明智的选择。

为了实现模块化，就要引入软件工程中的一些思想和原则，各个模块之间应该具有较低的耦合度，也就是尽量少的依赖其他模块，而模块内部应该拥有较高的内聚度，每个模块的功能应该明确并独立。为了实现这一需要，这里引入Pimple这一由SensioLabs公司开发的开源第三方容器，以实现依赖注入和控制反转。以模块化的思想开发，使得本爬虫非常易于维护和扩展，也使得二次开发变得非常轻松。

动态网络爬虫有别于普通爬虫的地方就是页面爬行部分，普通爬虫抓取静态页面后即进行后续步骤，而通过Javascript事件触发的改变文档内容的部分则会被忽略。动态网页爬虫将会模拟诸如下拉页面等操作，尽可能加载Javascript事件，以获取尽量全面的静态文档。那么，在模块化的基础上，只需要修改页面爬行模块即可，这样大大降低了各功能实现间的耦合度，也更便于实现和调试。

然而，无论如何优化，爬取动态页面的开销仍然是巨大的。下载一个静态页面的平均时间开销是10ms级，然而下载一个动态页面的平均时间开销是10^3ms级。因此，对于动态网页的爬取必须是定向的。试想对一个明显是静态页面的网页，比如百度首页，仍然去进行DOM建模，Javascript事件模拟等操作，无疑是对资源的一种极大的浪费。

## 3.1.架构

网络爬虫最基础的算法是，输入一个URL种子列表，然后重复以下步骤，从URL列表中取出一个URL，对该URL发起请求，下载对应的HTML文档，从下载的文档中提取URL链接，对每个链接进行绝对化，然后将其插入待下载的URL列表，并对URL列表进行去重，以确保待下载URL列表不会形成一个环。如果有需要的话，也可以对下载的HTML文档进行更深一步的处理，比如对其建立索引，搜索引擎会需要这一步。为了完成上述算法，需要以下几个模块：

1. 用以存储待下载的URL的容器，这里使用队列，以下称URL队列；
2. 根据协议下载URL所定位的文档的模块；
3. 从下载的文档中提取其中所包含的URL链接的模块；
4. 处理下载文档的模块，可以从下载的文档中提取所需的数据；
5. 将文档信息保存入数据库的模块，以便于管理和去重；

首先，需要从待下载URL队列中取出一个绝对URL，绝对URL是由策略（如“http”，“https”）或者“//”开头的，策略表明了下载该文档所需使用的网络协议。在PHP和phantomjs中，策略的选取是由运行环境自动检测的，虽然可以自行指定，但是对于大多数的页面，http协议都可以满足需求，即便是https网页，如百度，依然可以通过http访问，然后会收到一个带有Location: https://www.baidu.com的返回头。

在选择了合适的网络协议后，抓取方法将会被调用，其会向取出的绝对URL发出一个请求，然后根据响应头中的状态码进行对应操作，如果是200，也就是响应成功，就会将响应体下载下来，而如果是302，则会根据响应头中的Location进行跳转，如果是40X，则说明请求失败，服务器无法响应或者拒绝响应。

构造请求头不是一个轻松的任务，现今的大量网站都有一定的反爬虫策略，而绕过反爬虫这一课题，本身就可以展开为一个单独的课题，本文将不予详细描述。构造请求头对于绕过反爬虫有着举足轻重的作用，其对于合适的User-Agent，Referer，Connection，Cookie等的选择都可以对反爬虫策略造成干扰。在这里，因为本课题的重点并不是反反爬虫，仅仅使用一种简单的，模拟用户浏览器的请求头就可以满足需求。

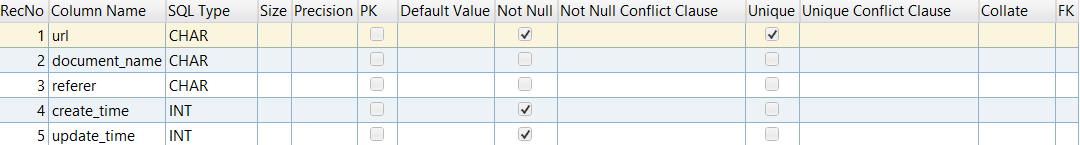
当文档被下载下来以后，将被写入文件，以便于多线程或者分布式工作单元进行下一步处理。在写入文件的过程中，文件的命名建议用对URL或者文档内容的HASH进行，在数据表中，将URL与文件名关联以便于查找和更新。

在本爬虫中，仅对a标签中的href属性中的URL进行提取，即可以认为仅对HTML文档进行抓取，而不抓取多媒体资源。对于下载下来的文档，在写入文件的同时进行链接提取，a标签中的href属性中的URL有几种特征。根据RFC标准，URL由<scheme>://<user>:<password>@<host>:<port>/<url-path>构成，根据URL命名规则使用正则匹配进行链接提取。但是要注意，由于大量可以省略的部分，URL可以是省略scheme的，由//开头的，也可以是只有url-path的，省略域名的相对链接。相对链接是不可以脱离上下文独立存在的，因此对于相对链接要进行去相对化。然后查询数据库判断该URL是否已被请求，为了避免对资源的浪费，更重要的是避免URL形成环，仅近期未被请求过的URL才会被录入数据库并且加入URL队列。

上述步骤就是一个基础的通用爬虫所遵循的，当然，其中省略了很多实现细节。设计一个能够有效处理大量条目的数据结构和数据表并不是一个简单的工作。其目标在于平衡时空开销。下一节将对数据表进行介绍。

## 3.2.数据表

对于处理过的URL和已下载的文档，需要进行高效的管理，也就是查询和修改。虽然可以自己设计一个数据结构用以进行该目的，但是使用数据库无疑是更方便高效的选择。既然要使用数据库，设计一个合理简约的数据表就是至关重要的。这张数据表必须能够将URL和根据URL下载的HTML文档关联起来，存储该链接的引用地址，文档创建时间和更新时间。也就是如下所示。



表中存储的是HTML文档名而不是直接存储HTML文档内容，以免HTML文档内容过多而对数据库造成不必要的负担。而因为URL中有部分字符不符合文件系统的命名规范，不能直接用URL作为文件名，这里的处理方法是对URL进行MD5加密，得到的是一个固定长度的，符合文件系统命名规范的哈希字符串，而因为该字符串并不包含文档内容的信息，无法直接通过该字符串进行检索，因为要将URL与文档名也就是该字符串进行关联，同时URL字段需要设置为Unique。

有些网站会对请求头做REFERER检查，只有通过某些页面的引用才能访问该页面，因此就需要伪造REFERER请求头，因此存储REFERER是必不可少的。

使用数据库来存储爬虫爬取的文档信息有另一个得天独厚的优势，因为字段名可以设置为唯一的，因此URL去重在试图存储的过程中就自然而然的完成了。因为网站的维护者可能会在一定时间后更新页面内容，因此需要保存更新时间，以便在一定时间以后重新下载该URL对应的HTML文档。

## 3.3.AJAX内容抓取

这里提出一种理论化的方法，也就是AJAX抓取的本质和内涵，然而在具体实现的过程中会有所不同。

当Javascript脚本被调用的时候，页面将会对用户事件做出响应，如点击，双击，鼠标悬停等等。一个AJAX页面不仅仅是一个由URL标识的简单页面，而是一系列状态，事件和转移。这就是普通页面和AJAX页面的主要区别。AJAX页面模型是对页面中所有状态的审视。准确的说这事一种页面自动化，一个状态转移图。状态转移图包含了页面中所有的状态，事件和转移，它由以下内容构成：

1. 节点，节点是页面的各种状态。一个页面状态是由DOM树表示的，它在应用程序的每个阶段包含当前DOM与所有相应的属性。
2. 边，边是状态之间的转移。当元素上的事件被触发的时候，状态转移就会应用于目标元素，使得它们的属性发生变化。

与传统页面不同，AJAX网页不仅包含静态内容，也包含动态内容，而传统页面仅需要一个页面和超链接的图就可以刻画。当普通网页触发Javascript事件的时候，并不会引入外部链接指向的内容，想要获取外部链接的内容你必须点击并跳转到一个新的页面，而AJAX网页却可以使当前页面进入一个新的状态，而这一状态的页面中可以包含外部链接的内容。这些通过AJAX技术动态加载的外部链接被称为AJAX链接。

AJAX爬行的流程的第一步是从给定的URL所标识的HTML文档中读取初始的DOM。接下来就是AJAX爬行特有的部分，包括运行HTML文档body标签中的onLoad事件，所有支持Javascript技术的浏览器都会在初始化的时候调用这个函数。在初始状态被构建之后，针对动态页面的爬行才会开始。具体的算法是：将Javascript中的事件响应函数绑定到DOM中的节点上作为节点的属性，然后按照一定的顺序遍历DOM节点，这里使用广度优先搜索，在遍历过程中，对节点上绑定的事件响应函数直接执行。当DOM节点上绑定的事件响应函数执行的时候，如果DOM有所变化，比如某个节点的属性改变，或者多了一个节点，则建立一个新的状态用来存储当前的DOM，并且将相对应的转移添加到AJAX页面模型中。转移是由事件信息标识的，事件信息包括源代码，触发条件，动作和改变等等。当爬行完一个新的状态以后，爬虫将会恢复到初始的DOM，这样就能够触发初始状态中的下一个事件。在所有的事件都被触发以后，一个完整的AJAX页面模型，也就是也就是一个状态转移图也就生成了。这其中必须要注意，必须避免生成先前出现过的状态，也就是去重。这是一个传统爬虫也可能遇到的问题。然而，传统爬虫可以将指定页面的URL与已爬行过的URL进行对比，这是一个开销较小的操作。AJAX爬虫却不能简单的使用这种方法避免重复爬行，因为一个页面中所有的AJAX状态都拥有同样的URL。这就需要去定义一种高效的相似度计算函数用来区别两个不同的状态。哈希算法是一种高效的方法，如果两个状态的DOM具有相同的哈希值，那么这两个状态的AJAX页面模型就是重复的。也就没有必要进行AJAX页面建模，以避免浪费开销了。

## 3.4.反爬虫策略

常用的反爬虫策略之一是爬虫陷阱。爬虫陷阱是一组可能有意或无意地用于引起网页抓取工具或搜索漫游器无限请求，或导致构建不良的抓取工具崩溃的网页。 网页爬虫也成为网页蜘蛛，爬虫陷阱这一名称就是由此而来。 爬虫陷阱可能被创建为捕获垃圾邮件或其他爬虫，浪费网站的带宽。 它们也可能在无意之中由动态页面的日历创建，链接不断指向下一天或年份。常用的爬虫陷阱有：

* 创建一个深度无限的目录结构，比如：

http://a.com/b/a/b/a/b/......。

* 由动态页面产生的无穷的文档。比如链接引用日历。
* 由大量无意义字母组成的文档，可能导致爬虫的语法解析模块崩溃。

对于爬虫陷阱，目前尚无有效的算法来检测，只能人为检测异常的爬虫进程并终止。除此之外，能加以改善的就是针对反爬虫策略的侦测方法。要让反爬虫策略无法检测到我们的爬虫，首先要明白为什么一些网站要制定反爬虫策略。当爬虫对一个HTML文档进行爬取的时候，会对文档的内容进行链接提取，并将提取后的链接放入待爬行队列。由于很多爬虫，尤其是初学者编写的爬虫只是简单的使用一个先入先出队列作为存储待爬行URL的容器，而一个网页中大多数的链接肯定也是指向和该网页相关的页面，也就是处于同一个根域名下的页面。这样，当对一个页面进行链接提取并插入URL队列的时候，属于相同根域名的URL密度会很高，当抓取模块对URL爬行的时候，会瞬间对目标网站的服务器造成巨大负荷，尤其是当抓取模块是多线程的情形下，这很容易导致一些服务器配置不高的网站崩溃并无法对后续的请求进行响应。而且多数爬虫无法对网站产生商业价值，因此大多数爬虫都是不受欢迎的。了解了网站屏蔽爬虫的原因，也就有了针对它的方式，让爬虫看起来象能为网站产生商业价值的客体，比如普通用户，或者搜索引擎的爬虫，其具体策略有：

* 降低对于同一根域名下网页的搜索频率，由于普通用户点击页面速度有限，为了更逼近普通用户的真实操作，并且也不对服务器造成灾难性的负荷，访问频率应该有所限制。限制访问频率可以使用一种简单的策略，在每次爬行之后调用系统的sleep函数，将自身挂起一定时间，但是这样无疑是对性能极大的浪费。另一种较为睿智的方式则是使用下文5.3节所描述的循环队列字典这种数据结构来作为待下载URL的容器。
* 在请求头中使用合适的USER-AGENT，很多网站都会对URSER-AGENT进行识别，因为不同的浏览器在对前端页面的渲染效果上会有些许的不同，而倘若请求头中不带有USER-AGENT，或者带有不合适的USER-AGENT，则服务器有可能不予以合适的响应。各大搜索引擎的爬虫有自己的USER-AGENT，比如百度的是Baiduspider，Google的是Googlebot，微软的是Bingbot等等，大部分网站都是希望被搜索引擎索引的，因此他们不会拒绝这些搜索引擎的爬虫。
* 在请求头中包含REFERER，真实的用户在访问网站的时候肯定是通过点击超链接，而不是直接在地址栏输入URL的方式浏览网页的，因此在爬虫构造的请求头信息中包含REFERER字段可以让爬虫和真实用户的操作模式更为相似。

除此之外，一般在网站的根目录下会有一个robots.txt文件，用以说明网站对于爬虫的规范，当然，这只是一个君子协议，是否遵守依赖于编写爬虫的作者。虽然文明的爬取网页，遵守爬虫规范是被提倡的，但是大多数网站仅接受搜索引擎的索引，如果完全遵循这些规范，那么你的爬虫可能一无所获。

下一节将会就动态网页爬虫遇到的技术上的难点和待解决的问题进行描述和分析。对于动态网页爬虫的设计更多的是关乎本质和原理，而在具体实现的过程中难免会遇到技术上难以越过的鸿沟，同时并不会从原理入手，从零起步搭建很多工具，而是使用现有的工具，并对其进行改良和整合。

# 4.面临的问题

首要的问题就是DOM树的建立和Javascript的解释。在上一节的描述中，我们把重点放在了AJAX页面模型的建立上，从一个抽象的层面上描述了待解决的问题以及一种解决方案，然而在实现的过程中诸多的问题将会浮出水面，第一个也是最难解决的问题就是DOM树的建立和Javascript的解释。Javascript具有复杂的，灵活的以致于为人诟病的语法，并且Javascript具有诸多的库和框架，比如Jquery，AngularJs，并且这些库和框架还在迭代更新，这使得Javascript拥有类似于方言的很多不同的语法。诚然，对于Jquery或者AngularJs，我们可以引用Jquery和AngularJs的源文件，它们可以看作是用Javascript语言写出的一门新的语言，那么引用了它们的源文件就相当于取得了它们的解释器。然而每次访问一个页面都需要重新引用加载一次库和框架的源文件，显然对系统产生了很多冗余的开销。我们再回到上一个问题，编写一个DOM树和Javascript解释器的复杂程度已经可以拆分出独立的几篇论文了。因此，自己编写Javascript解释器显然是不太现实的。

而进行AJAX爬行所需要进行的工作就是尽可能多的获取页面的状态，每一个状态对应的都是一个DOM树，其中的每个节点上都绑定了Javascript事件。每一个事件对页面内容的改变都是细粒度的，这对于DOM树的剪枝来说十分重要。在转移图中，一个状态可以在一系列事件的触发以后被重复搜索。在AJAX抓取的情况下，每页的多个单独事件导致获取网络内容。 连接的问题更大。 传统的搜索引擎通过预先缓存Web并在本地进行抓取来处理此问题。 在AJAX的情况下，这个过程只是部分可能的，因为动态内容不断地从服务器获取。 然而，缓存的问题也不容忽视。在爬行的AJAX页面应用中，页面的结构通常是稳定的，并且包含例如所有状态中存在的菜单和动态部分。 例如，京东网站页面包含每个页面几个链接到下一页或直接跳转到其中一个连续的页面。 这导致频繁的情况：使用菜单项从服务器重新加载相同的内容，并导致已经达到的状态。 在从服务器再次检索内容之后，抓取工具可以识别出在比较两种状态时达到的状态。 我们通过识别相同的状态但是并不获取内容来处理这个问题。 在传统的情况下，可以使用单个URL检查两个页面是否相同，而在AJAX的情况下，URL不会更改。并且现今互联网上存在的文档已经有上亿的数量级，倘若对每个文档都进行AJAX页面模型建立，这个数字可能还要大数十倍，这一方法的时间和空间开销都是无法接受的。

因此，在此AJAX页面建模的思想之上，另一种进行AJAX爬行的方法被提出了。既然AJAX页面模型建立是一种页面自动化方式，而且绝大多数面向用户的AJAX事件也只是下拉加载，并不用遍历全部的DOM节点，触发所有的Javascript事件，接下来需要做的就是找到一个页面自动化工具，对页面模拟用户的下拉操作，以加载完整的页面内容。恰好，PhantomJs和Selenium这两种开源免费的无头页面自动化工具可以完美的满足本爬虫的需求。在PhantomJs的基础上，只需要简单的Javascript脚本，就可以对页面进行自动化，模拟用户的操作来获取完整的页面内容。

然而，即便如此，AJAX爬行依然有一个致命的问题，这也就是当下搜索引擎无一支持对AJAX内容索引的原因，AJAX爬行的时间开销太大了，爬取一个普通页面的时间开销是十毫秒级别，而爬取一个AJAX页面的时间开销在秒级别上。抓取中出现的问题是获取网页所需的网络时间。 在AJAX抓取的情况下，每页的多个单独事件导致获取网络内容。 连接的问题更大。 传统的搜索引擎通过预先缓存Web并在本地进行抓取来处理此问题。 在AJAX的情况下，这个过程只是部分可能的，因为动态内容不断地从服务器获取。 然而，缓存的问题也不是微不足道的。 在传统的情况下，可以使用单个URL检查两个页面是否相同，而在AJAX的情况下，URL不会更改。在爬行的AJAX应用程序中，应用程序的结构通常是稳定的，并且包含例如所有状态中存在的菜单和动态部分。 例如，YouTube页面包含每个页面几个链接到下一页或直接跳转到其中一个连续的页面。 这导致频繁的情况：使用菜单项从服务器重新加载相同的内容，并导致已经达到的状态。 在从服务器再次检索内容之后，抓取工具可以识别出在比较两种状态时达到的状态。 我们通过识别相同的状态但是不获取内容来处理这个问题

试想用户打开一个网页，浏览器该如何得知页面已被加载完毕呢？事实上浏览器不知道，很多时候用户打开一个网页以后，即便大多数的页面内容已经被渲染出来，页面加载的图标仍然在旋转，这意味着依然有请求响应在进行，这一过程可能持续数秒，而对于一个异步加载的页面，这一过程可能持续数十秒，而什么时候加载完毕，用户并不知道。所以对于一个AJAX页面，需要等待加载的时间本身就是个未知数，这一则取决于爬虫端的网络状态，二则取决于服务器和中继的网络状态。由于下载这一I/O开销过于巨大，当进行AJAX爬行的时候，爬取模块如果能够做到多线程异步回调的模式是最好的，nodejs语言本身具有这种特性，PHP的Swoole扩展也具有这种功能。对于网页下载时间的估计，一种思想是，可以尝试对下载线程进行I/O流量监控，在开始下载时给予500毫秒的下载时间，时间到了以后回调该线程，并查询线程当前的I/O状态，如果下载速度大于一个阈值比如100KB/s，则再分配两倍的时间，即再过1000毫秒以后再查询该线程的状态，以此类推，每次将分配上一次两倍的下载时间，同时，在下载时间超过一个阈值比如31秒以后，强行终止该线程，因为它可能遇到了一个故障，或者不正常的服务器。

还有一个很重要的优化AJAX爬行的策略，不去加载页面中的CSS和各种image文档，在一个页面中，加载一个image文档占用的I/O资源往往非常大，而且一个页面中往往有数以百计的图片资源，即便是初始化这数以百计的链接也需要大量的开销，既然我们的爬行主要针对HTML文档及页面之中的数据，对于无关紧要的图片可以不予加载。在实际测试中，对于京东首页的爬取时，放弃对image文档的加载使得加载速度提升了10倍，加载时间从20000毫秒降低到2000毫秒，当然，这也和网络环境相关，但是可见，这一策略对于AJAX爬行的性能有了质的提升。

另一个可以优化的点在于PhantomJs，和nodejs不同，PhantomJs是一个对REPL支持很差的无头页面自动化工具，每次使用PhantomJs进行页面自动化的时候都要重新初始化整个PhantomJs，这也是一个不小的开销，而且由于PhantomJs对REPL支持很差，在REPL中没有办法进行页面自动化，也就很难模仿数据库连接池，对其进行复用。可以说目前在这个点上没有太好的办法，如果PhantomJs在更新以后对REPL的支持优化了，可以维护一个数据库连接池来保存初始化完成的PhantomJs进程，并对之进行复用。

下一节将会对动态网络的爬虫的实现进行介绍，这一节面临的一些问题和难点虽然有了解决思路，但是由于实现的难度，有一些问题并没有得到解决。

# 5.实现

本爬虫由依赖注入和控制反转的模式实现，以降低各模块之间的耦合度。这里使用一个开源库Pimple作为依赖注入的容器。整个爬虫由文档抓取模块，链接提取模块，文档处理模块，数据库连接模块，数据访问对象模块，配置读取模块，调度模块和文档抓取模块组成。下面将就几个重要的模块详细描述。下图即为自动加载并进行依赖注入的实现代码。

private function autoInjection()

{

$this->dirMap = json\_decode(file\_get\_contents('composer.json'), true)['autoload']['psr-4'];

$stack = array\_map(function ($value) {

return array(

$value => 0,

);

}, array\_keys($this->dirMap));

while ($stack) {

$cwd = array\_pop($stack);

$files = $this->scandir($cwd);

foreach ($files as $key => $file) {

if ($file == '.' || $file == '..' || $file == 'deprecated') {

continue;

} elseif (is\_dir($this->map($file))) {

array\_push($stack, array(key($cwd) => $key + 1));

array\_push($stack, array($file => 0));

break;

} elseif ($sufpos = strpos($file, '.php')) {

$class = substr($file, 0, $sufpos);

$alias = substr($class, strrpos($class, '\\') + 1);

$this->container[$alias] = function ($c) use ($class) {

return new $class($c);

};

}

}

}

}

表1：自动加载并进行依赖注入，这大大简化了整个系统中功能的实现难度

## 5.1.链接提取模块

链接提取模块可以说是爬虫最基础也是最重要的部分，链接提取模块就仿佛用来点击网页上各个链接的鼠标，没有链接提取模块便无法获取当前HTML文档中指向其它页面的URL，爬行也便无从谈起。这一节主要介绍了链接提取模块的功能和实现方式，然而这一最基础最重要的模块却并不复杂。

链接提取的核心功能，也就是寻找a标签中的href属性，由正则匹配完成，如果使用的语言支持正则匹配，只需要两行代码，一行定义正则表达式，一行调用正则匹配函数。在本爬虫中使用一个久经考验的正则表达式

<a\s.\*?href\s\*=\s\*(\"[.#]+?\"|\'[.#]+?\'|[^\s]+?)(>|\s.\*?>)(.\*?)<[\/ ]?a>

该正则表达式匹配带有href属性的a标签，然而会将a标签完整的匹配下来，因此就需要一个剔除href属性之外所有字符的方法来将链接清洗干净。例如<a href=”http://www.a.com” hidden></a>将会被处理为http://www.a.com。这里有一个需要注意的地方，在HTML中单引号和双引号都可以表示字符串，href=后面既可能是单引号包裹的URL，也可能是双引号包裹的URL。

在对URL进行清洗以后，将得到一个包含该HTML文档中所有URL的列表，在该列表中会含有相对链接，而相对链接在脱离了上下文的情况下是毫无意义的，因此要根据RFC中对相对URL的规定，根据相对链接的类型对其进行处理。以‘/’开头的以该网站的根目录为前缀，而不以‘/’开头的相对链接则遵循目录访问的规则，其中‘.’代表当前目录，‘..’代表上级目录。

至此链接提取就完成了，将处理过的列表返回即可。该模块的输入是HTML文档和该文档对应的URL。对其它模块没有任何依赖。

## 5.2.数据库连接模块和数据访问对象模块

数据访问对象模块是依赖数据库连接而模块的，而数据库连接模块的功能相对简单。数据库连接模块是为了适配不同的数据库而设计的，在应用过程中根据用户配置的不同可以返回不同的数据库连接对象。由于PHP语言PDO（PHP Date Object）扩展的强大功能，获取数据库连接对象的获取十分轻松，这也是PHP这门语言在内容处理上的优势。

<?php

namespace Vatel\DAO;

abstract class BaseDAO

{

private $conn;

public final function \_\_construct($container)

{

$this->conn = $container['DBConnection'];

$this->boot();

}

public function insert($fields)

{

$sql = 'INSERT INTO ' . $this->table . ' (' . implode(', ', array\_keys($fields)) . ') VALUES ('

. implode(', ', array\_fill(0, count($fields), '?')) . ')';

return $this->conn->exec($sql, array\_values($fields))->rowCount();

}

public function delete($conditions)

{

$sql = 'DELETE FROM ' . $this->table . (empty($conditions) ? '' : ' WHERE ' . implode(' = ? AND ', array\_keys($conditions)) . ' = ?');

return $this->conn->exec($sql, array\_values($conditions))->rowCount();

}

public function update($fields, $conditions)

{

$sql = 'UPDATE ' . $this->table . ' SET ' . implode(' = ? , ', array\_keys($fields)) . ' = ? WHERE '

. implode(' = ? AND ', array\_keys($conditions)) . ' = ?';

return $this->conn->exec($sql, array\_merge(array\_values($fields), array\_values($conditions)))->rowCount();

}

public function findEquals($conditions = array(), $fields = array())

{

$sql = 'SELECT ' . (empty($fields) ? '\*' : implode(', ', $fields)) . ' FROM ' . $this->table . (empty($conditions) ? '' : ' WHERE '

. implode(' = ? AND ', array\_keys($conditions)) . ' = ?');

return $this->conn->exec($sql, array\_values($conditions))->fetchAll(\PDO::FETCH\_ASSOC);

}

protected function boot()

{

;

}

}

表2：数据访问对象的基类

上图即为数据访问对象基类的代码。数据访问对象模块对数据库连接模块进行了封装，数据访问对象模块是面向数据表的，其自动组装SQL语句，并将之和数据库连接对象封装在一起，这样在外部对数据表的访问不需要进行任何SQL拼写，仅仅需要直接调用数据访问对象中的CURD方法操作数据表。通过这样的封装，不但简化了对于数据库连接对象的调用，也限制了对数据表的访问，不会出现对设计之外数据表的访问，同时数据访问对象还会对SQL语句进行预处理，以防止可能出现的SQL注入等情况。

同时在基类中将构造方法限制为final的，这样子类就不能覆写构造方法，这是PHP语言的特性，这对容器的依赖注入有着很关键的作用，如果在子类中也实现了一个构造方法，并且没有给依赖注入留出接口，那么就会产生令人迷惑的异常。同时，为了给子类自己的初始化留出空间，提供了一个boot方法，子类可以覆写boot方法，这样一样可以起到在构造过程中将自定义的初始化内容完成的效果。在子类中，如果没有针对数据表的特殊操作，只需要给定table属性即可，table用以指明操作的数据表名。

## 5.3.调度模块

如同第三节中所描述的，从文档中抓取到的链接将被放入URL容器中。URL容器是一种存放了所有待下载URL的数据结构。大多数爬虫都是根据广度优先搜索的顺序，从种子URL集合开始遍历网络的。这种遍历方式可以由一个先入先出队列实现。

然而在并行工作的情形下，由于这是一个先进先出的队列，在一个HTML文档中大部分的链接肯定是指向同一个根域名的URL，因此插入队尾的时候它们的密度会比较大。在弹出元素时，靠近队首的元素一般都拥有相同的根域名，这样会导致本地IP非常频繁的请求一个域名上的服务器，一则可能导致服务器过载，再则可能被被请求服务器封禁IP。因此一个朴素的先入先出队列并不是最合适的URL容器。一种能够有效避免上述情形的数据结构是一个列表的循环字典。其中的每一个元素是一个根域名和URL列表的键值对，一个URL列表中的所有元素都拥有同一个域名。

每次从该数据结构中取出一个URL时，由于这是一个循环字典，它会从当前计数器所对应的列表中取出第一个元素，然后计数器加一，指向下一个列表。而向该数据结构中插入一个URL时，首先判断是否存在该URL的根域名，如果存在，就向根域名所在的列表中插入该URL，否则创建该URL的根域名作为键的新的列表，然后向其中插入URL。

使用这样的数据结构意味着当循环字典中存在大于一个的根域名所代表的列表时，每次请求的根域名都不一样，这样就避免了在一段时间内过于频繁的对一个域名的服务器发起请求，既降低了服务器的负荷，又避免了被服务器限制访问。

在实际的网络爬虫中，爬虫所爬取的页面数量在数亿个网址的大小。 因此，大多数URL必须存储在磁盘上。为了平摊读写磁盘的成本，循环字典队列需要实现在内存中维护固定大小的入队和出队缓冲区。

## 5.4.文档抓取模块

文档抓取模块无疑是本论文最核心的内容，这一模块使得动态网络爬虫在根本上有别于一般的网络爬虫。普通的网络爬虫抓取的都是静态资源，无论是HTML文档，多媒体资源，甚至是CSS，Javascript代码等文档，都是一个请求对应一次抓取操作，对于有绑定AJAX事件的页面，则只能获取首次请求所返回的内容。

如同3.3.节所描述的AJAX爬行，比较理想的方式是对下载的HTML文档进行DOM建模，然后将Javascript所定义的事件监听器绑定到对应的DOM节点，并将所有的事件监听器记录下来，然后模拟触发各个事件。换言之，要对HTML进行DOM建模，并且还要解析Javascript，模拟触发Javascript事件，这基本上就是要实现一个Javascript解释器，还要绑定DOM节点。这一工作量有多大呢？Google的开源社区浏览器Chromium的源代码有21G，这几乎是不可能完成的。好在有nodejs环境，这一环境可以将Javascript语言脱离浏览器独立执行，也就是说通过爬虫爬取的Javascript文档有现成的环境可以解释执行，这无疑大大减少了工作量。然而在nodejs中Javascript是独立运行的，并不会绑定与DOM元素上，这一特性再次使得遍历AJAX页面状态变得复杂了。

一个可行的解决方案是使用PhantomJs。PhantomJs是一个具有Javascript API的，支持脚本的，无头浏览器内核。它对于多种网页标准都有快速和原生的支持，包括DOM操作，CSS选择器，JSON，Canvas，SVG等等。PhantomJs可以用于网站的无头测试，通过运行编写的脚本，它可以在诸如Jasmine，QUnit，Mocha，Capybara，WebDriver等框架下工作，可以进行页面自动化，模拟用户的点击，下拉操作等。而其最为精妙的一点在于，由于PhantomJs内置渲染引擎，它可以对页面进行渲染，并保存为一张图片。同时它也拥有流量监控的API，可以对自身的工作状态进行监测和管理。使用PhantomJs可以极大的简化AJAX爬行的工作，在文档抓取的时候，对于AJAX网站可以交由PhantomJs进行处理，而对于普通网站，则可以使用速度更快的方式，使用CURL进行抓取。

本模块非常重要的一点在于对AJAX爬行性能的优化，由于AJAX爬行这一行为的开销过于巨大，倘若将之直接应用于实际的生产环境中，对于京东首页这种级别的页面应用，可能需要20秒以上的时间进行加载，而对于一个普通页面的爬行只需要几十毫秒，这一I/O开销是无法接受的。因此，如同第四节所描述的性能优化的方式，对于非Javascript文件并不会予以加载。而如果爬行的目的是多媒体文件，则可以对爬行模块的代码进行改写，从而将指定扩展名的多媒体文件进行下载。

# 6.实验结果

PS D:\Workspace\PHP\crawler> php .\start.php

string(18) "http://www.jd.com/"

string(33) "//order.jd.com/center/list.action"

string(19) "http://home.jd.com/"

string(18) "http://vip.jd.com/"

string(16) "http://b.jd.com/"

string(17) "http://www.jd.com"

string(30) "http://cart.jd.com/cart.action"

string(22) "http://miaosha.jd.com/"

string(17) "https://a.jd.com/"

string(18) "http://red.jd.com/"

string(21) "http://paimai.jd.com/"

string(35) "https://channel.jd.com/fashion.html"

string(22) "http://chaoshi.jd.com/"

string(20) "http://fresh.jd.com/"

string(17) "http://www.jd.hk/"

string(17) "http://jr.jd.com/"

string(16) "http://yp.jd.com"

string(29) "http://yp.jd.com/sitemap.html"

string(42) "http://yp.jd.com/1318cfc24cdd91d045c1.html"

string(42) "http://yp.jd.com/9987a9e8feb4498010c4.html"

string(41) "http://yp.jd.com/6705d4212f942afb823.html"

string(39) "http://www.jd.com/pinpai/881-11468.html"

string(38) "http://www.jd.com/pinpai/753-8417.html"

string(38) "http://yp.jd.com/list/737\_752\_757.html"

string(18) "http://yp.m.jd.com"

string(42) "http://yp.jd.com/1315bb300504e61110d2.html"

string(41) "http://yp.jd.com/737e8b975e236352fef.html"

string(41) "http://yp.jd.com/737c0a6f1bf933246b6.html"

string(43) "http://yp.m.jd.com/6705d4212f942afb823.html"

string(44) "http://yp.m.jd.com/9987a9e8feb4498010c4.html"

string(44) "http://yp.m.jd.com/1318cfc24cdd91d045c1.html"

string(44) "http://yp.m.jd.com/1315bb300504e61110d2.html"

string(38) "http://www.jd.com/pinpai/757-4759.html"

string(40) "http://www.jd.com/pinpai/12341-8042.html"

string(35) "http://so.m.jd.com/pinpai/9211.html"

string(41) "http://yp.jd.com/737d567a626cf00dae1.html"

string(42) "http://yp.jd.com/16726553444457618ecd.html"

string(42) "http://yp.jd.com/1672d5e5e084212f35e7.html"

string(42) "http://yp.jd.com/16722c74ef7d64a30a37.html"

string(42) "http://yp.jd.com/16725592aa8ff1612ffc.html"

string(42) "http://yp.jd.com/16727bb826b571c28c68.html"

string(42) "http://yp.jd.com/1672d7de89a591e8635a.html"

string(42) "http://yp.jd.com/1672263fc4108e33b00e.html"

string(42) "http://yp.jd.com/1672396d4bbed6f300bf.html"

string(42) "http://yp.jd.com/1672b8f3ab0a9903e174.html"

string(42) "http://yp.jd.com/1672f414ea336d655013.html"

string(42) "http://yp.jd.com/1672d8710842d9bc1dd3.html"

string(42) "http://yp.jd.com/167206452672e6a5fab2.html"

string(42) "http://yp.jd.com/1672545c7e6bd6978ff0.html"

表3：爬行过程中命令行的输出

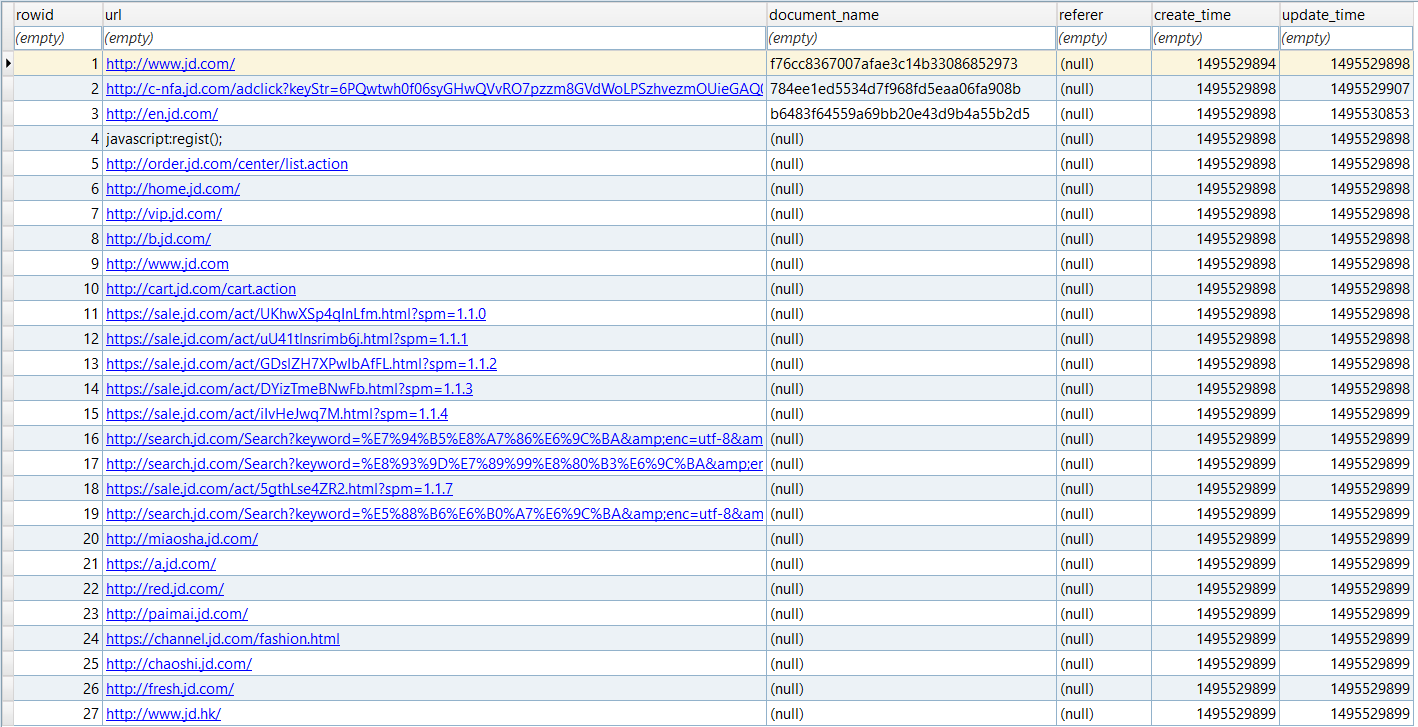


图1：爬行过程中的数据表

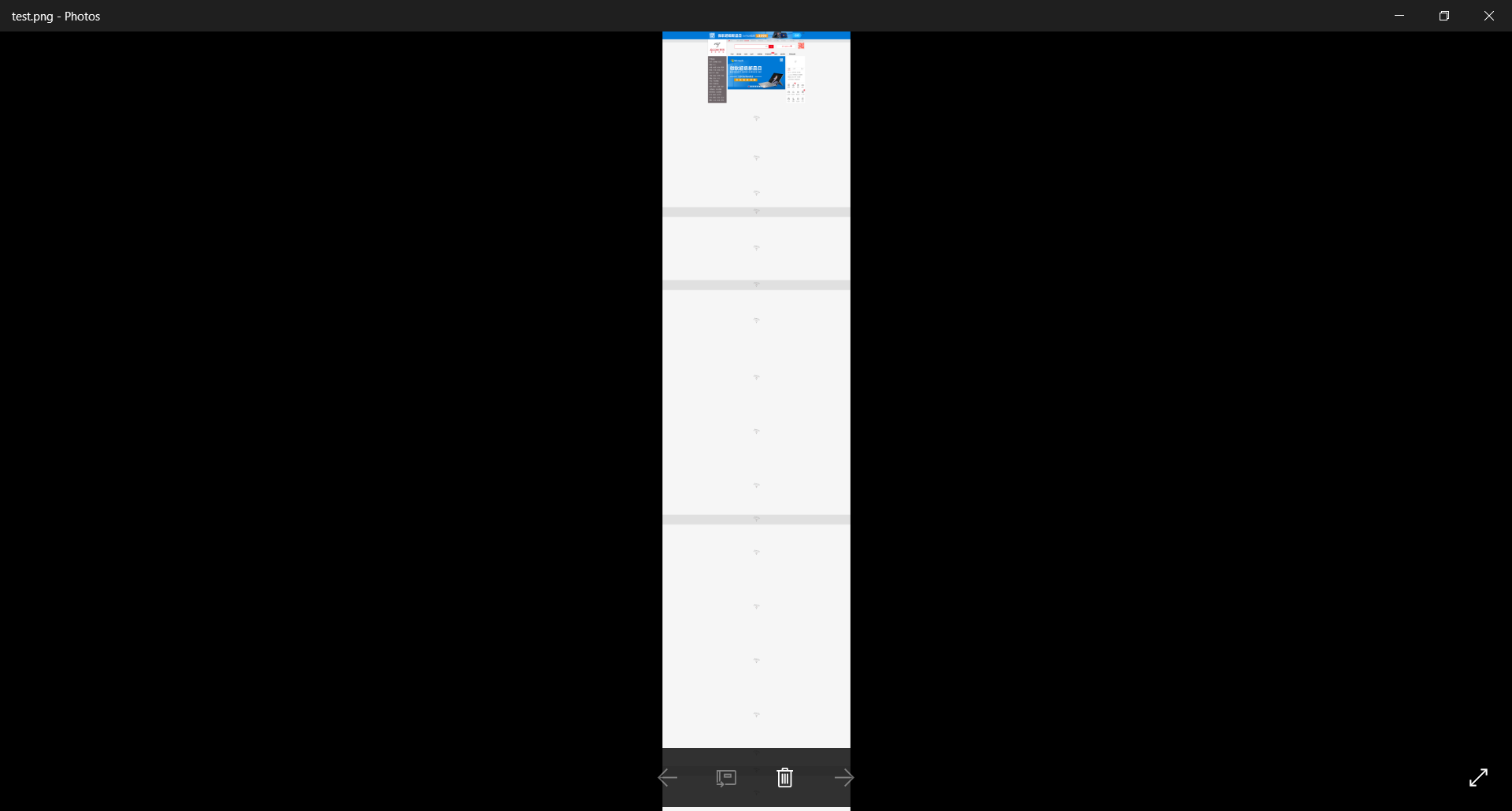


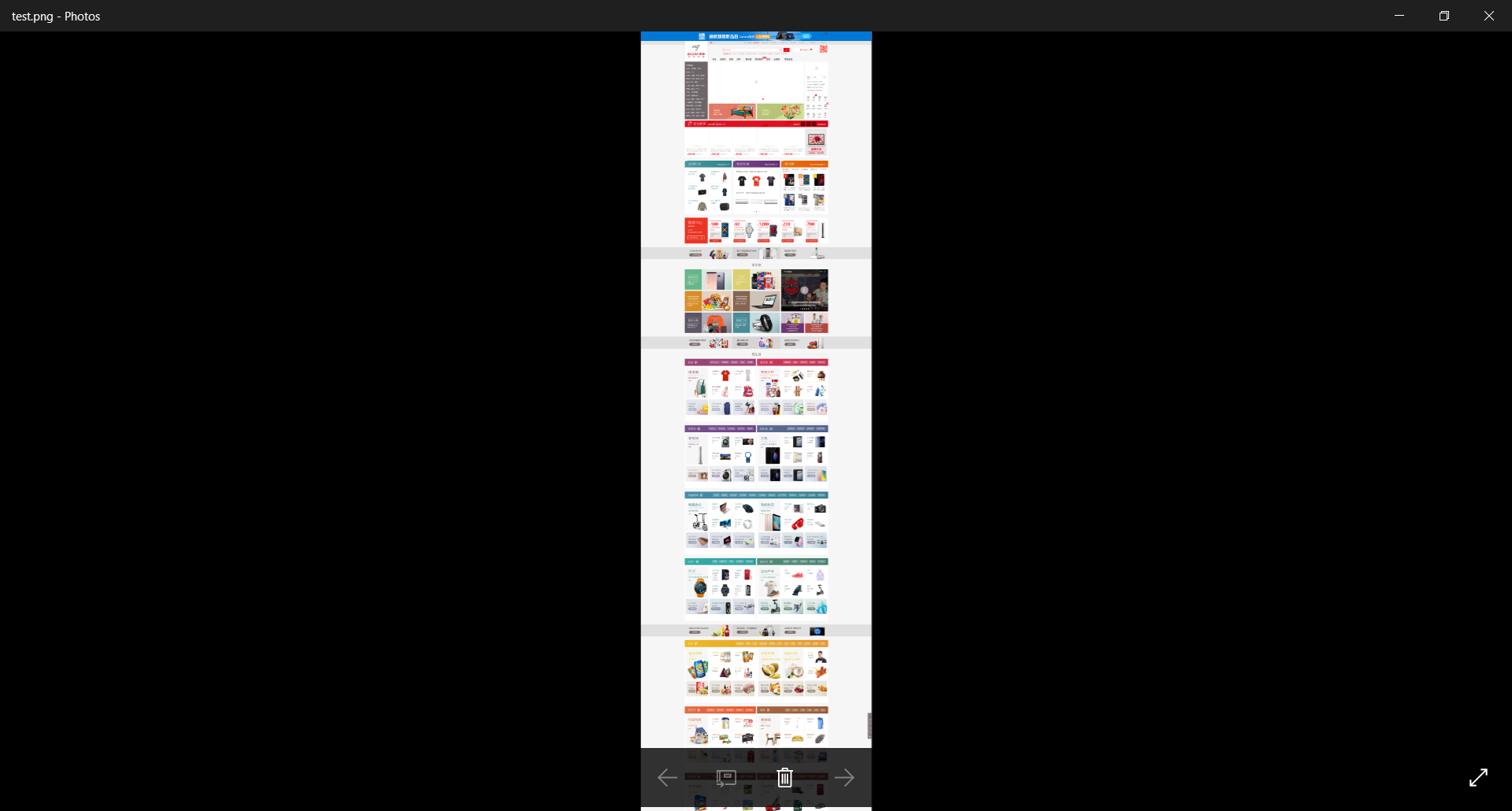
图2：不使用AJAX技术加载京东首页

图3：使用AJAX技术下拉动态加载京东首页

如图所示，即为爬虫爬行的部分结果，倒数第二张图是不使用动态加载获得的页面，最后一张图是动态加载的，可以看到其效果的差别十分显著，非动态加载的页面经由渲染以后有大面积的内容缺失，或者说仅仅由一张京东LOGO占位；而通过动态加载的页面获得了页面的完整信息。

对于最后两张图，为了展示和比较的方便，在结果中将图片也进行了加载，并且渲染成了可视化的图片。实际操作为了提升性能中并不会加载多媒体文件和CSS，也不会对获得的HTML文档进行渲染。

# 7.引用第三方类库

本软件使用了Pimple这一由SensioLabs公司开发的开源第三方类库，其功能是为依赖注入和控制反转提供一个可用类似数组的访问和赋值方式的容器。

# 致谢

非常感谢我的导师王兴起老师，王兴起老师耐心和精辟的指点对我的研究具有十分重要的指导意义。

# 参考文献

[1]Allan Heydon, Marc Najork。“Mercator: A scalable, extensible Web crawler”

[2]Cristian Duda #1, Gianni Frey #2, Donald Kossmann #3, Reto Matter #4, Chong Zhou ∗5。“AJAX Crawl: Making AJAX Applications Searchable ”

[3]Jayant Madhavan, Google Inc., jayant@google.com; David Ko, Google Inc., dko@google.com; Lucja Kot, Cornell University, lucja@cs.cornell.edu; Vignesh Ganapathy, Google Inc., vignesh@google.com; Alex Rasmussen, University of California San Diego, arasmuss@cs.ucsd.edu; Alon Halevy, Google Inc., halevy@google.com。“Google’s Deep-Web Crawl ”

[4]S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda. Lazy Query Evaluation for Active XML。 “SIGMOD“

[5] Z. Bar-Yossef, I. Keidar, and U. Schonfeld. Do not Crawl in the Dust: Different URLs with Similar Text。 “WWW“

[6] K. Chellapilla and A. Maykov. A Taxonomy of JavaScript Redirection Spam。 “AIRWeb“

[7] A. Mesbah and A. van Deursen. Migrating Multi-page Web Applications to Single-page AJAX Interfaces。 “CSMR“

[8] C. Shah. YouTube Crawling: A VidArch Year “Retrospect。 http://www.ils.unc.edu/vidarch/vidarch-annrep-2008.pdf“

[9] W. Wu, A. Doan, and C. Yu. WebIQ: Learning from the Web to Match Deep-Web Query Interfaces。 “IICDE”

[10] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms。 “SIGIR”