基于自动化信息抽取技术的垂直网络爬虫设计与实现

张建宇，王洪波

（北京邮电大学网络与交换技术国家重点实验室，北京 1000876）

摘要： 随着垂直搜索引擎搜索范围的扩大，如何自动化高效地完成数据爬取任务成了一个重要的问题。目前大多数的网络爬虫使用人工定义规则来完成对数据的抽取工作，效率低下。本文首先对自动化信息抽取网络爬虫进行了框架设计和优化，然后针对爬虫抓取问题详分析了开源网络爬虫框架 Scrapy 并且给出了优化方案；针对信息自动化抽取问题分析了自动模板生成算法 RoadRunner 算法并且给出了优化方案；针对爬取 Ajax 网页问题分析了 Ajax 爬取工具 Scrapyjs。最后对基于自动化信息抽取技术的网络爬虫从爬取效率和抽取准确率两个方面进行了测试，给出了测试结果和分析。

关键词： 信息抽取；RoadRunner；网络爬虫；Scrapy

中图分类号： TP39

Web Crawler based on automatic information extraction technology

ZHANG Jianyu, WANG Hongbo

(State Key Laboratory of Networking and Switching Technology, Beijing University of Posts & Telecommunications, Beijing 100876)

Abstract: With the expansion of the scope of vertical engines, how to accomplish data crawling task automatically and efficiently become a important issue. Most web crawling spiders use artificial defined rules to complete the data extraction work and it is inefficient.The paper carry out a framwork desgin and optimization for the web crawler based on automatic information extraction technology, then analyzes the details of the open source web crawler framework Scrapy and gives its optimization according to the crawling problem, analyzes the automatic template generation algorithm RoadRunner and gives its optimization according to the automatic information extraction problem, analyzes the Ajax crawling tool Scrapyjs accroding the crawling Ajax problem. Finally, the performane of this web crawler is tested on crawling efficiency and extracting accuracy and presented

Key words: Information extraction;RoadRunner;Web Crawler;Scrapy

0 引言

在互联网上的信息呈爆炸式增长的今天，大量而冗杂的信息以网页的形式呈现在用户眼前，而垂直搜索引擎能够帮助用户在海量信息中快速、便捷地找到真正需要的信息。垂直搜索引擎是对某一个特定的行业或领域的专业搜索引擎，是搜索引擎的细分和延伸，通常是对行业内或领域内的网站信息进行深度整合，最后向用户提供专业的、全面的以及准确的搜索结果[1]。

垂直搜索引擎的完美搜索结果离不开大量的精确的行业数据的支持，而这些数据的获取则是网络爬虫的功劳。垂直网络爬虫是一个能够自动按照先行确定的规则下载网页的程序，作为搜索引擎中重要的组成部分，网络爬虫爬取数据的效率和质量已经直接影响到搜索引擎

作者简介： 张建宇（1989-），男，硕士研究生，主要研究方向：计算机应用

通信联系人： 王洪波（1975-），男，副教授，主要研究方向：云计算与数据中心网络等

搜索的结果[2]。网络爬虫会按照一定的抽取规则有选择的抽取页面中的信息以及链接，并将抽
取的结构化信息存储至数据库，以供垂直搜索引擎使用。大多的网络爬虫都是文本爬虫，即
爬虫只能下载页面的 HTML 代码，同时使用人工定义的规则进行抽取工作，这就导致了网络爬虫获取信息不完整以及扩展效率低等问题。

本文设计和实现了一个基于自动化抽取技术的网络爬虫，将抽取规则生成方式转变为自动，同时将爬虫框架和 Ajax 爬取工具相结合，提高了网络爬虫的扩展能力和获取信息的完整性。

1 网络爬虫系统设计

1.1 系统设计

基于自动化信息抽取技术的网络爬虫逻辑上由抓取模块、数据处理模块、规则生成模块、Ajax 处理模块组成，系统架构图如图 1 所示。

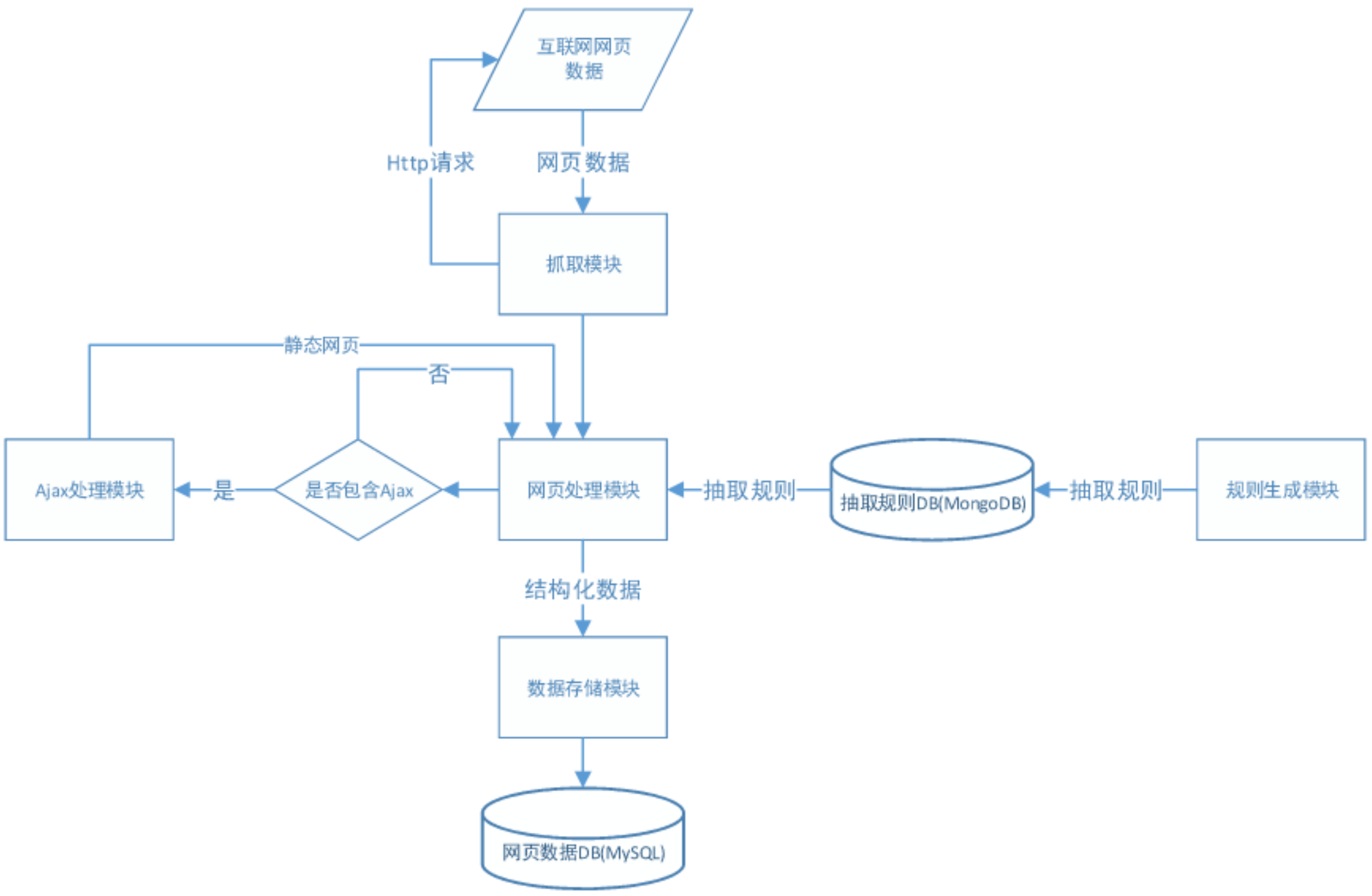

图 1 网络爬虫系统架构

各个模块的功能描述如下：

抓取模块：模块主要负责根据 URL 队列中的 URL 来抓取互联网上的网页信息，并将抓取到的网页信息传递给数据处理模块。

数据处理模块：模块主要负责根据数据库中的抽取规则抽取抓取成功的网页，并对抽取后的信息做初步的验证，最后将通过验证的数据写入后台数据库。

规则生成模块：模块主要负责生成同一类网页的抽取规则，并将生成的规则写入数据库以供数据处理模块使用。

Ajax 处理模块：解析抓取成功网页中的 Ajax 部分。

网络爬虫分成了这样的几个较大的模块，在实际实现中，抓取模块、数据处理模块是基

于 Scrapy 框架进行优化和修改的，规则生成模块是基于 RoadRunner 算法进行改进，Ajax 处理模块是使用了 Scrapyjs 工具进行处理。

1.2 系统关键技术

1.2.1 爬虫框架 Scrapy

上述模块描述中的爬取模块和数据处理模块是基于开源爬虫框架 Scrapy 进行的修改和优化。Scrapy 是用 Python 开发的一个高效、功能齐全的爬虫框架，它的主要用途是抓取 web 站点并从网页中抽取结构化的数据，同时 Scrapy 框架还是一个开源的框架，这允许我们根据需求方便的修改内部代码。Scrapy 在处理网络通讯上使用了 Twisted 这一异步处理框架，实现了高效的网络下载效率[3]。

Scrapy 体系架构清晰，其大体架构如图 2 所示，其中包含了 Scrapy 的主要组件和数据处理流。图中各个组件作用如下：

① 图中位于中心的 Scrapy Engine 是 Scrapy 框架中引擎组件，是整个 Scrapy 爬虫框架的核心，这个组件控制着整个爬虫的数据处理流程，并且管理着整个框架的事务触发处理；

② Scheduler 是 Scrapy 框架中调度组件，这个组件主要功能是负责从 Scrapy 引擎接收其他组件发起的请求，并将这些请求按照一定的抓取策略进行排序之后加入代抓取的队列中；Scheduler Middlewares 是 Scrapy 框架中调度中间件，这个中间件介于 Scrapy 引擎和 Scheduler 组件之间，负责对从 Scrapy 引擎发送出来的请求和响应进行处理，这个部分可以根据需求对相应代码进行修改，以扩展 Scrapy 的调度组建的功能；

③ Downloader 是 Scrapy 框架中下载组件，这个组件负责从互联网上抓取相应的网页并将网页数据交给 Spider 组件，下载组件是基于 Twisted 这个高效异步模型的，因而代码不是很复杂但下载效率很高；Downloader Middlewares 是 Scrapy 框架中下载中间件，这个中间件是 Scrapy 引擎和下载组件之间的钩子框架，它可以以轻量级的改变 Scrapy 框架处理请求和响应的方式；

④ Spider 是 Scrapy 框架中爬取处理组件，这个组件负责根据规则处理抓取的网页，在这个组件中用户可以定义特定的抓取和解析规则以满足实际需求；Spider Middlewares 是 Scrapy 框架中爬虫中间件，这个中间件介于 Scrapy 引擎和爬取处理组件间的钩子框架，主要是处理爬取处理组件的响应输入和请求输出，用户可扩展这个中间件以达到过滤等特殊目的；

⑤ Item Pipeline 是 Scrapy 框架中数据输出组件，这个组件将 Spider 组件从网页中抽取的信息清晰高效地存储，组件会由一个或多个过滤处理组件组成，用户可以自定义过滤和处理的规则，信息会在管道中被逐步过滤和处理直至流程结束。

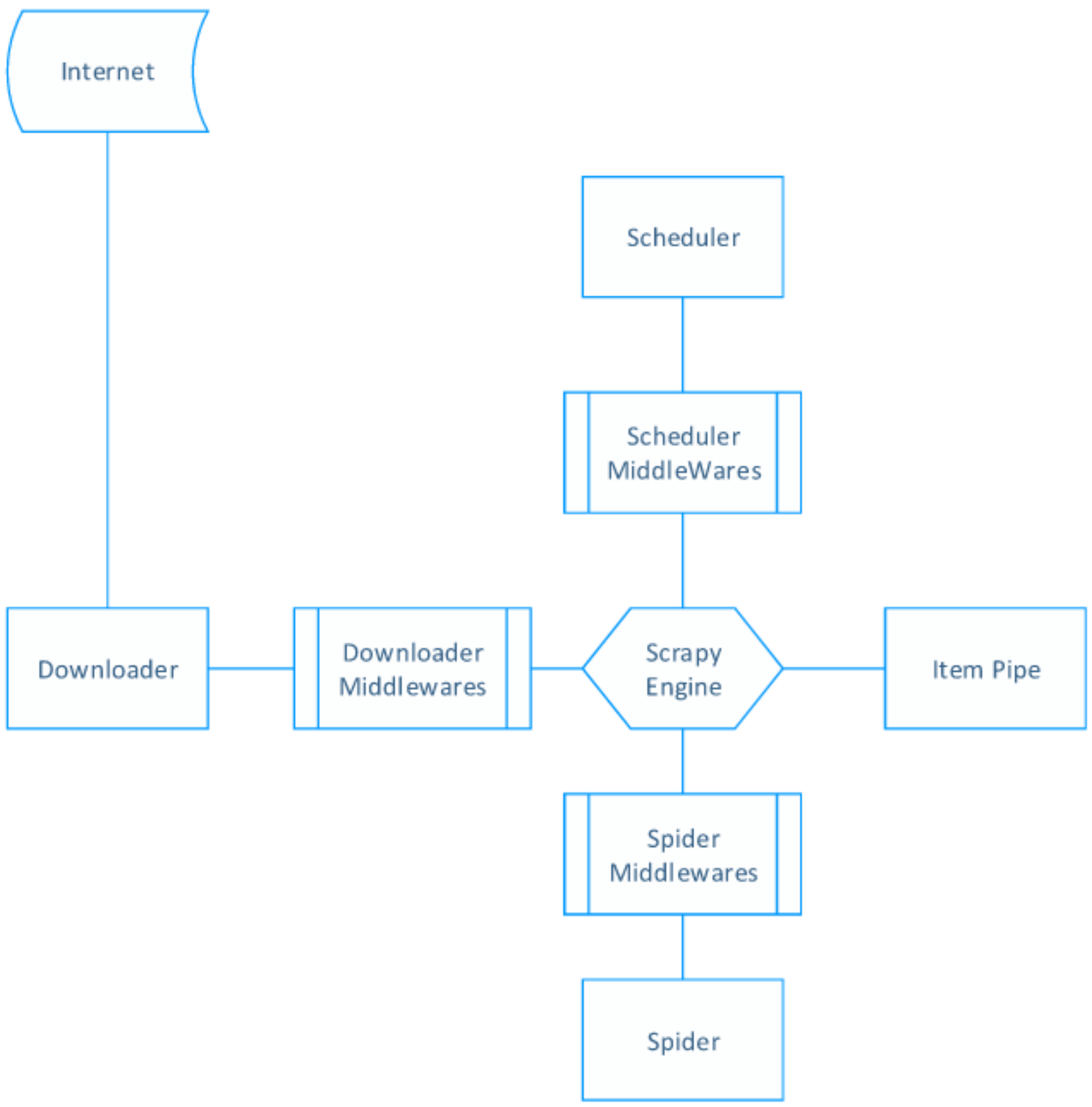

图 2 Scrapy 组件架构图

图 3 中包含了 Scrapy 框架的数据流向，Scrapy 框架中的数据流程由 Scrapy 引擎组件控制，主要的运行方式[3]：1.引擎访问种子 URL，Spider 组件解析 URL；2. 引擎从 Spider 组件获取需要爬取 URL，然后将 URL 组成请求在调度中调度；3.调度将爬取请求发送给下载组件，下载组件下载网页；4.下载组件将下载完成网页对应的响应通过下载中间件发送给引擎；5.引擎将受到的响应通过 Spider 中间件发送给 Spider 组件，Spider 组件处理响应并返回抽取信息以及给引擎发送新的请求；6.引擎将抽取到的信息放到 Item Pipeline 中，信息最后被存储；7.重复 2 之后动作，直至调度队列中无请求，爬取过程结束。

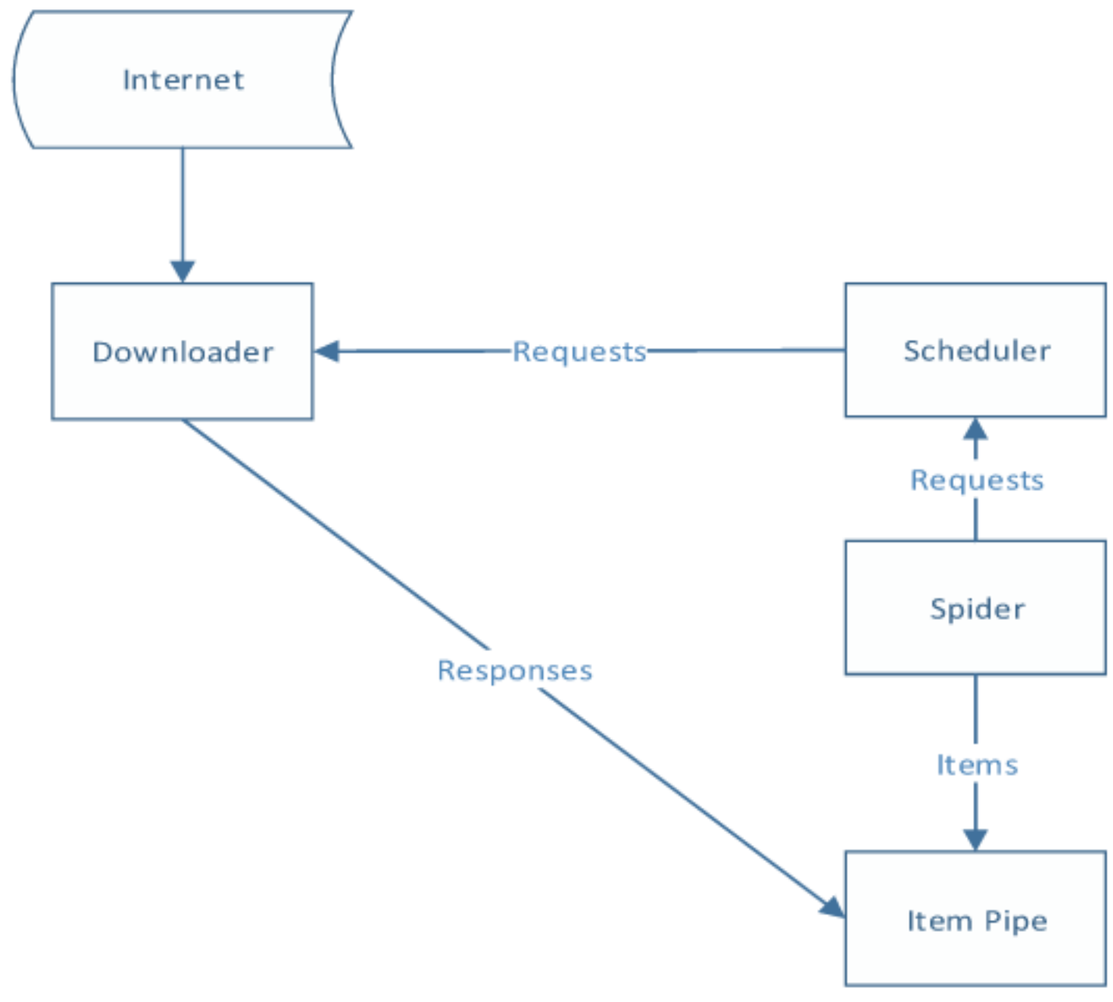

图 3 Scrapy 组件数据流

1.2.2 抓取模块和数据处理模块

抓取模块和数据处理模块都是基于 Scrapy 框架的，抓取模块利用了 Scrapy 框架的调度组件和下载组件，并对 Scheduler Middlewares 和 Downloader Middlewares 进行了特定的扩展；数据处理模块利用了 Scrapy 框架的爬虫组件和数据处理组件，并对两个组件进行了修改和优化。

Scrapy 框架的调度组件初始只是实现了基本的 FIFO 调度方法，但是在垂直搜索引擎中，网页爬取是需要时效性的，所以需要对 Scheduler Middlewares 进行扩展以符合需求。抓取模块在 Scheduler Middlewares 中首先获取了网页的头部，这个头部包含了网页的生成时间，然后按照网页生成时间排序后来进行 URL 调度，这样就可以保证搜索引擎中网页的时效性。

在 Scrapy 的下载组件中，我们修改了 Scrapy 的 Downloader Handler 的代码安装了 Scrapyjs 服务，以满足爬取 Ajax 网页的需求。其中修改的 Downloader Handler 的基本代码如下：

```
def process_request( self, request, spider ):
    if 'renderjs' in request.meta:
        webview = self._get_webview()
        webview.connect('load-finished', self.stop_gtk)
        webview.load_uri(request.url)
        gtk.main()
        ctx = jswebkit.JSContext(webview.get_main_frame().get_global_context())
        url = ctx.EvaluateScript('window.location.href')
        html = ctx.EvaluateScript('document.documentElement.innerHTML')
        return HtmlResponse(url, encoding='utf-8', body=html.encode('utf-8'))
```

数据处理模块使用了 XPath 这一 XML 路径语言， XPath 基于树形结构，可以提供在网页 DOM 树中查找确定节点的功能[4]。数据处理模块主要是对爬虫组件进行了修改，当获取到抓取的网页后，数据处理模块会根据网页 URL 的根域名去数据库中查找相应的抽取规则，然后按照规则中 XPath 来获取网页中的信息，最后将这些信息交由 Scrapy 的数据处理组件。在数据抽取的过程中，若数据库中没有该域名的抽取规则，数据处理模块会触发规则生成模块的运行，并终止此次的信息抽取进程。在 Scrapy 数据处理组件上，按照抓取数据的规格设置了对爬取数据质量检测的检查点，当抽取数据进入数据处理组件之后，会逐一经过这些质量检测点，若完全合格数据会被直接写入后台数据库，以备索引程序使用，若不合格则会直接抛弃相应的数据。

1.2.3 自动模板生成算法 RoadRunner

目前垂直网络爬虫信息抽取技术主要是人工抽取和自动抽取。人工抽取是利用人工定义一类网页的抽取规则，然后利用这些规则进行信息抽取，这种方法优点是数据抽取准确性高，但是工作效率不高、过度依赖人力则是根本缺点。自动抽取则是利用一类网页之间的关系，通过对比等方法抽取信息规则，然后进行信息抽取[5]。虽然 Internet 上网页存在异构性、动态性，但是垂直网络爬虫的爬取目标是特定的网站的网页，这些网页具有同一网站的网页结构数量为可数，网页之间只有数据不同的特点，因而我们可以根据网页之间相同的规则来定义怎样去抽取相应的信息。RoadRunner 算法则是一个可以自动对比相同模式网页并生产相应模板的算法。

RoadRunner 算法首先提出了一种介于嵌套类型和无并正则表达式之间的新型正则语言——UFRE 表达式，这种正则语言可以规范的定义 HTML 网页中的抽象结构，而且它可

以覆盖所有网页中出现的各种分支情况，这样就允许算法同过对比有限页面推导一类相似网页的模板[6]。而基于这种正则表达式和嵌套类型，RoadRunner 算法提出了栅格理论（lattice theory）：

#PCDATA：定义网页中出现的与结构无关的文本信息数据；

∑：定义了不包含#PCDATA 的网页上所有的信息；

∑∪{#PCDATA, . , + , ? , (,)}：定义网页的 URFE 表达式的元素集合。这个集合中的元素通过以下规则生成：

① 空字符串 ε 和∑∪{#PCDATA, . , + , ? , (,)}中所有的元素都是 URFE 表达式；

② 如果字符串 a 和 b 是 URFE 表达式，则 a·b，(a)+，(a)?都是 URFE 表达式，其中 a·b 表示字符串 a 与字符串 b 相连接，(a)+表示重复 a 的字符串，(a)?则是(a|ε)的缩写，同样可以使用(a)*作为((a+))?的缩写；

RoadRunner 算法接受若干个 HTML 字符串 s1, s2, s3,…sk，对应的源数据集 i1, i2, i3, …, ik(i1, i2, …, ik 为嵌套类型)；RoadRunner 算法提出了可以把一个 URFE 当作 Wrapper 来解析上述若干个 HTML 源串以及重构对应的源数据集。因此，算法需要寻找能够包含所有 HTML 字串的 UFRE 表达式 σ，即 σ=LUB(s1, s2, s3, …, sk)（LUB()：上确界），而在实际使用中一般都是两个网页，即 σ=LUB(s1, s2)。

然后 RoadRunner 算法利用 ACME（Align, Collapse under Mismatch, and Extract）匹配算法对两个网页进行匹配[6]，其中 ACME 算法中①Align 为对齐网页数据、②Collapse under Mismatch 为在模板中按一定规则清除不匹配、③Extract 为抽取信息，ACME 算法每次会接收两个处理好的页面作为输入，以其中一个页面作为样本，以另一个作为包装器，然后通过相互比较、不断找到 mismatch 消除 mismatch 的过程，最终若能消除所有的 mismatch，那么就会得到一个能够适用这两个网页公用的信息抽取模板。

由上述过程生成的信息模板是没有任何语义信息的，需要对模板和样例中的匿名信息进行分析以进行语义标注。因为网页是展现给用户观看的，所以关键数据项的周围会出现一些帮助用户理解的信息，而 RoadRunner 算法则根据这一特点进行了大致的语义标注。

RoadRunner 算法极大程度避免了人为选择网页和特定网页数据结构的干扰，生成过程中不需要任何交互，是一种全自动的抽取规则生成算法。但是 RoadRunner 算法也存在着一些问题：1.算法默认接收的网页都是 XHTML 格式的，但是互联网上的网页有可能是极不规整的；2.RoadRunner 算法默认输入两个页面是相似，但是判断两个页面为相似页面也是一个实际问题；3.无噪声处理，极大影响了模板生成的效率和准确率；4.算法生成了一个包装器，利用这个包装器可以抽取网页中信息，但是却没有将这些信息按特定的语义组织成结构化信息。

1.2.4 规则生成模块

由于 RoadRunner 算法有上述几点缺点，因而在实际开发规则生成模块时需要对其进行修改和扩充。这一模块可以被分解成为三个小模块，分别是：网页预处理模块、wrapper 生成模块、规则生成模块。总体流程如图 3 所示。

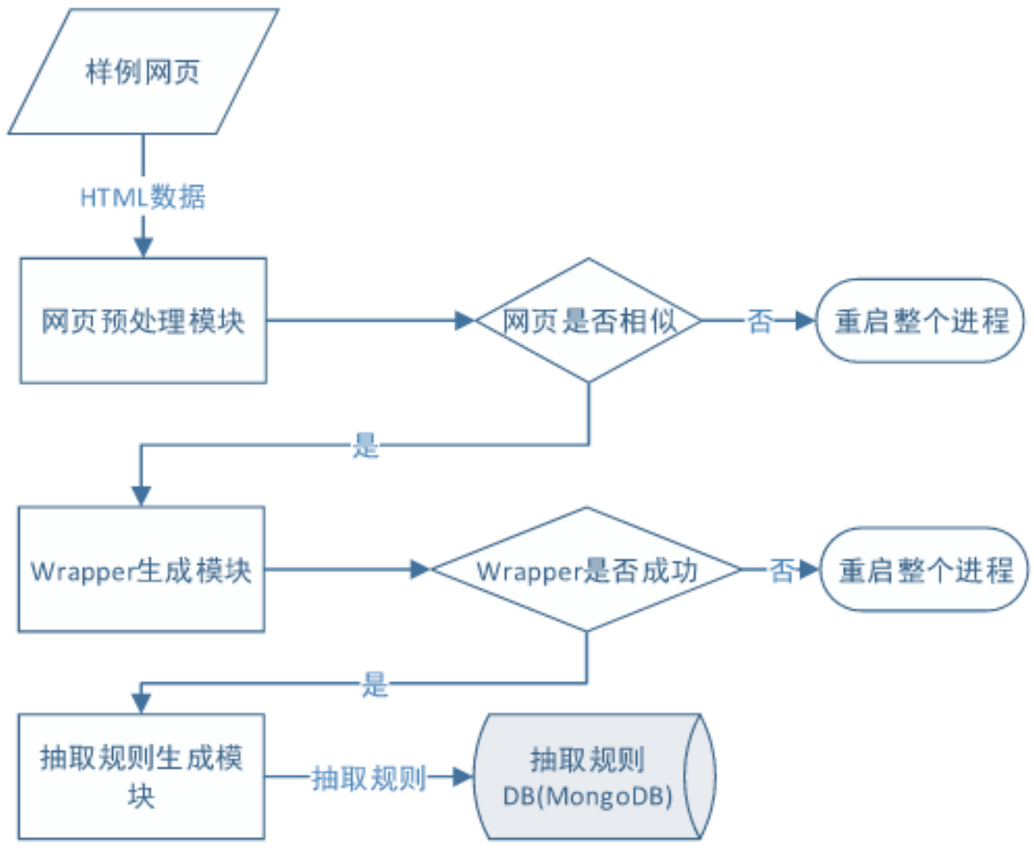

图 3 规则生成模块

网页预处理模块主要负责对提供的样本网页进行预处理，以达到能够 RoadRunner 算法输入标准。预处理模块首先将下载的网页从 HTML 转换成 XHTML；然后通过计算两个网页编辑距离来计算网页相似度，计算编辑距离伪代码如下：

```
Edit-Distance(page1, page2)   //page1, page2 分别为处理后的 token list
{
    int lenPage1 = page1.size();int lenPag2=page2.size();
    int d[0…lenPage1, 0…lenPage2];int i, j, cost;
    for i 0 ← lenPage1:
        d[i, 0] = i;
    for j 0 ← lenPage2:
        d[j,0] = j;
    for i 1 ← lenPage1:
        for j 1 ← lenPage2:
            if page1.get(i) == page2.get(j): cost = 0;
            else cost = 1;
        d[i, j] = min(d[i–1, j]+1, d[I, j-1]+1, d[i-1, j-1]+cost);
    return d[page1, page2];
}
```

上述代码会计算出两个网页的相似节点数，进而判断两个网页是否相似。最后将网页中的噪声根据网络上的先验规则过滤掉，系统中采用了 Ad Block 这个开源软件中的规则进行过滤。

wrapper 生成模块主要是基于 RoadRunner 算法实现的，主要负责接收预处理模块提供的网页，然后经过 ACME 算法比较后，最后生成一个符合所有页面的 wrapper。

上述生成的 Wrapper 只是直接指出了网页中所有的结构和文本信息，但是这些信息依旧是半结构化的信息，也就不能直接提供给垂直搜索引擎使用。虽然 RoadRunner 算法中包含了语义标注，但是其语义标注有些时候并不能满足实际需求，因此需要根据对 wrapper 进行进一步的处理。抽取规则生成模块主要接收生成好的 wrapper 文件，然后根据数据库先验数据信息以及网页上的特定信息点，对 wrapper 进行语义标注，最后将生成的抽取规则存入数据库中备用。

1.2.5 Ajax 处理工具 Scrapyjs

互联网进入 web2.0 时代之后，越来越多的网页选择使用 Ajax 作为展示数据的工具，这些网页虽然对用户界面友好，但是却给网络爬虫的爬取工作带了不小的坏处[7]。现在大多的数网络爬虫都是基于文本爬取，即爬虫抓取的网页数据实际上是该网页的 HTML 代码，而网页运用 Ajax 技术之后，网络爬虫抓取的网页信息和用户实际观看的网页是不尽相同的，这样就会导致一些关键信息就不能够被爬虫很好的抓取，进而导致搜索引擎数据的不齐全。而在实际网络爬虫爬取的工作中，爬取包含 Ajax 页面信息的方法主要分成以下几种：1.人工模拟 Ajax 请求逻辑，以获取服务器返回的信息；2.调用浏览器内核做 Ajax 解析，这个方法还分成调用有界面和无界面两个方法；3.结合开源 JS 引擎，自己实现一个轻量级的浏览器[8]。其中第三种方法最为适合但也最为困难；第一种方法最为简单但是人工模拟费时费力，效率不高；第二种方法是比较流行的一种解决方案，这个种方案经常使用的工具有 selenium，phantomjs，casperjs，scrapyjs 等，本文选用的是 Scrapyjs 这一工具。

Scrapyjs 是由 scrapinghub 公司开发的一个开源的模拟 JS 事件请求的工具，现在托管在 github 上，现在这个代码仓库的维护者也是 Scrapy 框架的开发者之一。Scrapyjs 通常有两种模式[9]：1.修改 Scrapy 框架的 Downloader Middlewares 实现，这种模式不要修改 Scrapy 框架内部的代码，但是会阻塞 Twisted 异步框架的运行；2.同 Scrapy 框架中 Downloader Handler 实现，将 Scrapyjs 代码添加到 Scrapy 框架中，这种模式需要修改 Scrapy 框架中 Twisted 的内部代码，但是不会阻塞异步通讯，效率很高。

Scrapyjs 使用方法不管使用上述两种模式的哪一种，都必须要将 renderjs 这个键值存放到 Scrapy 框架中 request.meta 这个字典中，当调度组件调度这个 request 时，Scrapy 会选择使用浏览器内核 webkit 来下载，而不是使用一般的下载组件[9]。Scrapyjs 在使用的过程中需要安装 pyhon-gtk2、python-webkit 以及 python-jswebkit 三个软件包。

相对于上述其他几种模拟工具，Scrapyjs 可以是完全异步处理的，它将浏览器内核 webkit 的事件循环和 Scrapy 框架的 Twisted 时间循环整合在一起，而其他的模拟工具基本上是阻塞或者多进程的，相对而言 Scrapyjs 速度上要快的多，但是 Scrapyjs 这个工具目前还不支持定制的请求头部，例如 User-Agent、Accept 等等。

2 网络爬虫系统测试

主要测试网络爬虫在爬取效率和抽取准确率两个方面的能力。测试环境搭建在一台操作系统为 CentOS6.0、内存 8G、CPU 为 Intel(R) Xeon(R)E3-1230@3.3GHz 的服务器上。测试环境搭建的软件有 python2.7、JDK7 以及 pip。

网络爬虫爬取效率的测试在学校网络环境下，由于学校网络存在较大的不稳定性，同时也考虑到目标网站服务器不同时间负载也不尽相同，所以选取了 6 个时间进行了 6 次重复爬取，最后对 6 次爬取的数据进行详细分析，力求得到较为准确的结果。段爬取效率的测试结果如下：

表 1 网络爬虫爬取效率

爬取开始时间	爬取 URL（个）	爬取时间（小时）	爬取速率（个/小时）
3A.M.	53593	6	8932.2
10A.M.	43616	5	8723.3
12A.M.	21943	3	7314.3
3P.M.	25271	3	8423.6
9P.M.	28845	4	7211.3
11P.M.	41263	5	8252.6

由上表可以看出网络爬虫在上午 12 点和晚上 9 点左右爬取效率明显下降，其可能的原因是由于上午 12 点是目标网站负载量最大的时间，网络爬虫获得返回信息较慢；而晚上 9 点是目标网站负载较大且校园网负载也较大的时间，网络爬虫发出和获得信息都较慢。而其余时间，网络爬虫的效率基本相同，差别不是很大。可以计算出 6 次爬取的平均数 σ=(53593+43616+21943+252271+28845+41263)/(6+5+3+3+4+5)=8249.5 个/小时。若剔除网络较为拥塞的数据，则最大爬取速度为(53593+43616+25271+41263)/(6+5+3+5)1=8618.1 个/小时。至此得到了爬虫系统爬取速度的峰值和平均值，由于普通网站的网页数量大致在几万到十几万之间，所以可以估算完整遍历一个网站的时间基本在几个小时至一天以内，但是平时爬取时，URL 数量基本维持在 5000-10000 条，即爬取时间大致在 0.5-1 小时，所以网络爬虫爬取的时间效率是可以接受的。

抽取准确率的测试主要是将自动化抽取的结果和人工标注的结果进行比对，由于人工标注中也会出现二义性情况，所以随机选择了互联网上 10 组不同的网页，每个组网页包含一条信息数据，测试过程中每个抽取信息中包含了 50 个不同的信息点。信息抽取准确率测试结构如下：

表 2　抽取信息准确率

自动抽取信息点（个）	自动抽取准确率%	人工标注信息点（个）	人工标注准确率%
39	78	48	96
40	80	49	98
39	78	50	100
35	70	45	90
33	66	43	86
37	74	49	98
34	68	47	94
45	90	49	98
30	60	40	80
39	78	49	98

由上表可以看出除了个别的数据外，其余数据中自动抽取准确率和人工准确率是联动的，即同时增长或同时减小，这是因为人工标注抽取中，人工会用到网页上额外的信息来判断抽取信息，而这些信息同样也被自动化抽取系统所使用。在上述实验数据中，可以看出有两次实验出现了自动抽取准确率偏高或偏低的现象，其中准确率偏高的原因是自动抓取系统中的先验数据恰好覆盖所有的信息点，而准确率偏低的原因是因为该组网页的相似度不高，导致 Wrapper 质量不高。所以，人工标注平均准确率为 93.8%，而自动化抽取平均准确率为(39+40+39+35+33+37+34+39)/(50*8)=74%，两者相比可知自动抽取准确率接近人工标注准确率，可以通过后期的人工修正来提高自动化抽取的准确率。

通过测试可知，本系统爬取速率基本可以满足日常爬取需求，而自动化抽取的结果也能达到较为合理的正确率。

3　总结

首先本文讨论了自动化信息抽取爬虫在网络爬取中的重要性，其次设计并实现了一个可以自动获取信息的网络爬虫，此系统可以通过网络爬取、规则生成、数据处理等可以高效、准确获取垂直搜索引擎需要的信息，并且在爬虫运行过程中大部分为自动化工作，只需要前期人工配置即可。

但是本系统还有许多需要改进的地方，比如网络爬虫如何更好地爬取 Ajax 网页、如何进一步提高信息抽取的准确率等等，有待进一步研究。

[参考文献] (References)

[1] 刘金红, 陆余良. 主题网络爬虫研究综述[J]. 计算机应用研究, 2007, 24(10): 26-29.

[2] 周德懋, 李舟军. 高性能网络爬虫: 研究综述[J]. 计算机科学, 2009, 36(8): 26-29.

[3] Scrapy developers.Scrapy 0.24 documentation[OL]. [2014]. http://doc.scrapy.org/en/latest/

[4] 刘玮玮. 搜索引擎中主题爬虫的研究与实现[D]. 南京: 南京理工大学, 2006.

[5] 孙立伟, 何国辉, 吴礼发. 网络爬虫技术的研究[J]. 电脑知识与技术, 2010, 6(15): 4112-4115.

[6] Crescenzi V, Mecca G, Merialdo P. Roadrunner: Towards automatic data extraction from large web sites[A]. VLDB. In VLDB[C].Roma, Italy: Proceedings of the 27th VLDB Conference, 2001.109-118

[7] Olston C, Najork M. Web crawling[J]. Foundations and Trends in Information Retrieval, 2010, 4(3): 175-246.

[8] Cui L J, He H, Xuan H W, et al. Design and Development of an Ajax Web Crawler[J]. 2013, 17：6-10.

[9] Scrapinghub.Scrapyjs - Scrapy+Javascript integration[OL]. [2013]. https://github.com/scrapinghub/scrapyjs