TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 06/09 pp643-659
Volume 21, Number 6, December 2016

Robots Exclusion and Guidance Protocol

Dajie Ge and Zhijun Ding*

Abstract: With the rapid development of the Internet, general-purpose web crawlers have increasingly become unable to meet people's individual needs as they are no longer efficient enough to fetch deep web pages. The presence of several deep web pages in the websites and the widespread use of Ajax make it difficult for general-purpose web crawlers to fetch information quickly and efficiently. On the basis of the original Robots Exclusion Protocol (REP), a Robots Exclusion and Guidance Protocol (REGP) is proposed in this paper, by integrating the independent scattered expansions of the original Robots Protocol developed by major search engine companies. Our protocol expands the file format and command set of the REP as well as two labels of the Sitemap Protocol. Through our protocol, websites can express their aspects of requirements for restrictions and guidance to the visiting crawlers, and provide a general-purpose fast access of deep web pages and Ajax pages for the crawlers, and facilitates crawlers to easily obtain the open data on websites effectively with ease. Finally, this paper presents a specific application scenario, in which both a website and a crawler work with support from our protocol. A series of experiments are also conducted to demonstrate the efficiency of the proposed protocol.

Key words: deep web; Ajax; crawler; protocol

1 Introduction

Given the rapid growth of Internet applications in recent years, the inefficiencies of general-purpose web crawlers in fetching deep web pages and Ajax pages have had a considerable impact on user satisfaction, especially because such crawlers have become largely unable to satisfy individual requirements of the users. Though major data sites and, in practice, search engines have various levels of adopted methods to address this problem, the requirements of manual intervention in such methods do not support the autonomous functionalities of the general-purpose web crawlers.

Deep web pages are those pages that can no longer be reached via pure links, and are often hidden behind the search forms. Users are granted access to such pages only after they input a series of keywords in the form of queries. In recent years, studies have revealed that a large portion of the web pages on the Internet actually belong to deep web pages[1]. One of the major requirements of an effective search engine is to extract the hidden deeper resources, which has been proven to be lacking in existing crawlers.

Nowadays, many websites interact with users by dynamic scripts. Ajax is the most widely implemented technology in such dynamic interaction applications. However, the requirement of asynchronously updating the web pages has resulted in traditional crawler mechanisms loosing efficiency in search engines. Specifically, web crawlers cannot directly fetch the contents generated from Ajax dynamic script technology—an inability that seriously affects the search results of the search engines. The reason why general-purpose web crawler cannot fetch Ajax pages is that their source codes are not often composed with the complete data content. In general, a complete Ajax

• Dajie Ge and Zhijun Ding are with Department of Computer Science and Technology, Tongji University, Shanghai 201804, China. E-mail: 1xiaojdge@tongji.edu.cn; zhijun_ding@outlook.com.
* To whom correspondence should be addressed.
Manuscript received: 2016-07-11; revised: 2016-09-15; accepted: 2016-10-03

engine consists of HTML, CSS, and JavaScript codes, and the complete content of a given web page must be executed by multiple calls to an Ajax server constituted by the response data. General-purpose web crawlers are not efficient enough to fetch the Ajax pages, which is another major challenge posed to web crawlers.

In addition, existing general-purpose web crawlers suffer from chaotic page extraction strategies, with the current Internet specifications for crawlers becoming more obsolete and imperfect. The Robots Exclusion Protocol (REP)[2], also known as the Robots Protocol, is an Internet standard that allows websites to insist a list of web pages (i.e., those that can or cannot be crawled) to the visiting crawlers. The Robots Protocol requires a website to create a text file named "robots.txt" at the root directory of the server, with a series of commands written to insist the list of resources that cannot be fetched by a crawler. The original version was invented in 1994, and the second edition[3] with extended command functionalities was rolled out in 1997. At present, two versions of this protocol are currently being practiced. However, the second version of the Robot Protocol has yet to have any updates after its only extension in 1997, and no authoritative agency is maintaining the Protocol. Instead, some major search engine companies extended the Robots Protocol independently in a scattered fashion and in accordance with their own requirements. The most famous extension is the Sitemap Protocol[4], which has been proposed by Google to enable webmasters to indicate the availability of web pages to the search engines. With no unified protocol version being maintained, many scattered extensions look unfamiliar to web users, and such versions are gradually abandoned, such as the command "Visit-time". In fact, such command line features may allow websites to raise their unique requests, which could be used as good communication and interaction standard among websites and search engines.

A healthy crawler pattern should avoid frequent visits to the websites of the search engines, as this causes a strong impact on the web servers. In this context, search engines can have better user experiences if websites reduce unnecessary visit restrictions on crawlers. In recent years, the rapid development of the Internet has given rise to a number of search engines—a phenomenon that has resulted in great demands for an effective crawler mechanism for enriched user experiences in accessing websites. To this end, the REP can still be improved and the Sitemap Protocol, an important expansion of the Robots Protocol, can be enhanced to achieve perfect agreements between crawlers and websites. Moreover, an improved Robots Protocol can enable the general-purpose crawlers to fetch both the deep web pages and Ajax pages effectively.

In this paper, a Robots Exclusion and Guidance Protocol (REGP) is proposed based on the original REP. The proposed protocol integrates various independent expansions of the Robots Protocol developed by major search engine companies as well as expands the file format and command set of the REP and two labels of the Sitemap Protocol. Through our proposed protocol, websites can express their requirements for restrictions and guidance to the visiting crawlers, thus facilitating a general-purpose fast access to deep web pages and Ajax pages for the crawlers. In this way, crawlers can easily obtain the open data on websites effectively with ease.

The rest of the paper is organized as follows. Section 2 presents the related works. Section 3 describes the REGP. Section 4 presents an application scenario, and our experiments are presented and discussed in Section 5. Section 6 concludes this paper and presents directions for our future work.

2 Related Work

2.1 Deep web solutions

In recent years, three different solutions from different search engines have been implemented to fetch deep web pages. The first and the most direct way is to allow crawlers to parse the search pages. In this approach, crawlers first find the entrance to deep web pages, generate a list of accessible pages, and submit these to the websites. Then, crawlers can track the return results to fetch those deep web pages. This approach can be viewed as a simulation of artificial operations. Xian et al.[5] proposed a quality-based data source selection for web-scale deep web data integration. Noor et al.[6] presented a latent dirichlet allocation based semantic clustering of heterogeneous deep web sources, whereas Raghavan and Garcia-Molina[7] and Cope et al.[8] proposed two different strategies of finding query interface of the forms, for the purposes of understanding and modeling the query interface in web pages. Liu et al.[9] presented a kind of deep web entry automatic discovery method. In this method, the information of specific field deep web is obtained to establish domain

ontology, after which web forms can be judged through the process of topic crawler crawling in the web, and the appropriate pages can be downloaded. Saissi et al.[10] proposed another vision of a deep web virtual integration system using a mediated schema that is built with a relational schema describing each deep web source. To match the query interface via data mining, Shao et al.[11] presented a whole pattern-matching method for discovering complicated matches between pluralities of modes. Zheng and Cui[12] proposed a selection strategy for keywords in a single attribute interface. Pusdekar and Chhaware[13] used the visual clues concept for extracting the main data from deep web pages. However, all these approaches that either look for deep web pages or generate the return forms require expertise in a particular industry. Hence, these studies are more likely to be applied to vertical search engines, instead of general-purpose crawlers.

The second method of deep web solution is the Open Application Programming Interface (API), as demonstrated by majority of today's data websites. Open API, also known as open platform, is a common application in service-oriented websites. Websites encapsulate their web services and data into a series of APIs and avail such APIs for third-party developers. Although this is an effective way of obtaining data, APIs have their own limitations; for instance, the query returns often contain Top-*N* data each time. Accessing more resources with limited APIs is the focus of several studies. Pei and Ye[14] summarized the development status and trends of global Open API, whereas Kim and Kim[15] presented semantic annotation methods on acquired Open API data from target websites based on the data mining technology. Jung et al.[16] discussed key technologies for the automatic generation of new mash-up service using open APIs, in which the automatic service mash-up method uses the ontology. Zhang and Cao[17] designed and implemented a third-party developer auditing system based on Open API, whereas Wang et al.[18] proposed a secondary development method of Open API. However, general-purpose web crawlers cannot fetch deep web data for all websites in a limited API mode, because the type and quantity of query interfaces are obviously different with each website.

The third method is the Onebox strategy, first proposed by Google[19], in which special query results are typically listed separately on the uppermost portion of the search pages for a given search in a search engine. Some other similar concepts exist, such as Aladdin Plan[20], box computing[21], and so on. Here we collectively call them the Onebox strategy. To realize this strategy, search engines should make their Onebox interfaces on the web master open platform available to the owners of deep web data, and present them with the list of users who wish to place their data on search engines, write codes, and then submit the configuration information that meets the interface standards. As a result, search engines directly show their related applications or results instead of the traditional page links in search pages after review. Such a strategy ensures that the search engines can obtain deep web data from the websites. However, website developers should exert tremendous effort for every search engine, because each search engine is unique with its own set of requirements. Despite such approaches, there remains a lack of a universal approach of crawler solutions for data websites.

2.2 Ajax solutions

Generally, two types of methods are used for fetching Ajax pages. The first method includes a manual analysis. For a web page, crawler developers seek those URLs containing useful data and then connect these URL links to traditional web crawlers. Obviously, such a process cannot be applied to general-purpose web crawlers because of the high degree of human intervention required.

The second approach is a browser-analog mode, which is used to build the entire Ajax page. A DOM tree initialized by an HTML source code is completely built by multiple modifications based on Ajax calls. In recent years, many Ajax page fetch algorithms based on state conversion diagrams have been proposed[22–24]. Shao and Li[25] put forward a dynamic page information extraction algorithm based on a tree model. Ma et al.[26] also proposed an advanced Ajax crawler based on DOM. Xia et al.[27] presented a complete crawler system that is enabled to interpret and execute JavaScript codes. Meanwhile, Li et al.[28] proposed a dual de-duplication strategy that can effectively reduce the time cost of such algorithms. However, such methods require complex control flow and may suffer from extremely low efficiency levels. Such a crawl rate is absolutely unacceptable for general-purpose web crawlers.

3 Robots Exclusion and Guidance Protocol

3.1 Access method

The search engine of our proposed protocol follows the access method of the original REP. All the commands of this protocol must be listed in a text file named "robots.txt", and here, following the name "robots" instead of "crawlers" or any other names to meet users' habits is an important consideration. In fact, they have the same meaning in the next content. This file must be placed at the root of the web server and be accessible via HTTP by every visitor. Some examples of URLs for sites and URLs for corresponding "robots.txt" are shown in Table 1.

Every crawler must try to visit the "robots.txt" before visiting other URLs on this website. If this visit succeeds (HTTP 2xx Status Code), the robot must read the inner content, parse it, and follow all the commands applicable to this robot. If this visiting fails (HTTP Status Code 404) because no "robots.txt" presents on the website, then the crawler can assume that no restrictions and guidance are available, thus, it can visit the site with complete freedom.

The current version of this protocol does not provide a definition of other possible responses for a server when visiting the "robots.txt". Here are some recommendations. The crawler should stop visiting the website when encountering the HTTP Status Code 401 or 403. However, the site should try to make its "robots.txt" accessible. The crawler should try to visit the "robots.txt" later when meeting the HTTP Status Code 5xx. When encountering HTTP Status Code 3xx, the crawler should also follow the redirect instructions until such a time that a resource that can be considered as "robots.txt" is found.

3.2 Format

In our proposed protocol, "robots.txt" contains multiple command lines. The composition of each line is listed in Table 2.

In the proposed protocol, the specific command line is divided into two portions: the restriction line and the guidance line. Restriction command means that the command must strictly be followed by the visiting crawlers, and the guidance command is a command suggested by the website for the visiting crawlers. All <Field> for specific commands are listed in Table 3.

In the proposed protocol, the contents of "robots.txt" consist of three different fields (the protocol header, command sequence section, and a global command section), each separated by a blank line. The protocol header consists of two command lines: the "Robot-version" line, which is used to describe the version of the protocol, and the "Last-modified" line, which is used to show its last modified time for the visiting crawlers.

The command sequence section consists of several command sequences separated by blank lines. Each command sequence consists of a command target segment and a command content segment. A command target segment consists of "User-agent" command lines, "Ip-allow" command lines, and "Ip-disallow" command lines. This segment is used to inform the visiting crawler whether the respective command sequence is applicable to the crawler. Visiting crawlers should check each command target segment in the command sequences from top to bottom until it matches a

Table 1 Examples of URLs for "robots.txt".

URLs for web site	URLs for "robots.txt"
http://www.example.com/	http://www.example.com/robots.txt
http://www.example.com:8080/	http://www.example.com:8080/robots.txt

Table 2 Form of command lines.

Name	Format	Explanation
Specific command line	<Field>":" <value>	<Field> is a command name. <value> is the command content.
Left brace line	{	The first non-blank character is a left brace.
Right brace line	}	The first non-blank character is a right brace.
Blank line		The role of blank line is to divide two command section. There should be no comments here.
Comment line	"##"<value>	Any text after a comment tag.

Table 3 List of <Field> for specific commands.

<Field>	Explanation	Type
Robot-version	Version of the protocol	Restriction
Last-modified	Last modified time of the ”robots.txt”	Restriction
User-agent	Name tokens of the visiting crawler	Restriction
Ip-disallow	IP addresses forbidden to access	Restriction
Ip-allow	IP addresses allowed to access	Restriction
Disallow	Resource path forbidden access	Restriction
Allow	Resource path allowed access	Restriction
Crawl-delay	Access interval of a crawler with same name token	Restriction
Ip-delay	Access interval of a crawler with same IP addresses	Restriction
Request-rate	Access rate of a crawler with same name token	Restriction
Ip-rate	Access rate of a crawler with same IP addresses	Restriction
Visit-time	Time period allowed to access	Restriction
Time-forbidden	Time period forbidden to access	Restriction
Language	Languages required to support	Restriction
Encoding	Encodings required to support	Restriction
Charset	Charsets required to support	Restriction
Cookie	Cookie required to set	Restriction
Mirror-site	The mirror site can be visited instead of this website	Restriction
Sitemap	Site map to guide crawlers	Guidance
Host	Master domain of the mirror site	Guidance
Index-page	Pages hoped to be indexed	Guidance
Change-always	Pages always updated	Guidance
Change-hourly	Pages updated each hour	Guidance
Change-daily	Pages updated each day	Guidance
Change-weekly	Pages updated each week	Guidance
Change-monthly	Pages updated each month	Guidance
Change-yearly	Pages updated each year	Guidance
Change-never	Pages that will never update	Guidance

command target segment; then, it should parse the command content segment of this command sequence and follow the corresponding instructions to fetch the site.

In the global command sequence, all command contents are applicable to any visiting crawler. Left brace lines and right brace lines should appear in pairs, and all the contents between them are dependent on the contents of the brace lines. The commands in brace lines are used to describe the commands in detail. Brace lines should be placed within a command content segment. The contents of the inner brace lines consist of a command target segment and a command content segment, similar to the command sequence. In addition to the three command lines of the commend sequence, “Disallow” command lines and “Allow” command lines are placed in command target segment and are used to describe the commands for a different resource path. Brace lines can be used in multiple, parallel, or nested modes.

Comments begin with a “##” tag; thus, all contents after the tag are considered as comments, which are ignored by crawlers. Therefore, comments should follow a command line or appear as single lines. As used herein, “##” is the tag of a comment instead of original “#”, because the “#” is often used as a character in URLs with the popularity of JavaScript. The BNF-like syntax to describe our proposed protocol appears in Appendix. Below is a simple example of a “robots.txt”.

Robot-version: 3.0
Last-modified: 30 Oct 2014 04:31:17 UT

User-agent: WebCrawler
Disallow: /data
Allow: /data/open

User-agent: infoseeker
User-agent: wiseRobot
Allow: /info
Allow: /news
{
User-agent: infoseeker
Allow: /info/hot

Crawl-delay: 5
}

Sitemap: http://www.example.com/documents/example_sitemap.xml
Visit-time: 1:00-16:00 UT ##8:00-17:00 Beijing Time is not allowed
Language: zh-cn, zh

In this example, the first two lines are the protocol header, which describes the version number and the last modification time. The last three lines belong to the global command section, and show the location of the Sitemap, setting time, and allowed language, respectively. The middle portion consists of the two command sequences. The first sequence describes the resource path that can be used by the "WebCrawler" to access the websites. The second sequence first shows that crawlers named "infoseeker" or "wiseRobot", which can access all the resources with path of "/info" or "/news" path, and then uses a pair of brace lines to inform "infoseeker" to visit resources using the path "/info/hot" at a minimum interval of five seconds.

3.2.1 Robot-version

This command line represents the version number of the "robots.txt" written in the first line in three different values of "1.0", "2.0", and "3.0", respectively. Here, "1.0" means the original 1994 version, which only supports "User-agent" and "Disallow"; "2.0" represents the improved 1997 version, which supports "User-agent", "Disallow", and "Allow"; and "3.0" represents the latest version of the protocol. Each "robots.txt" supporting our protocol should begin with "Robot-version: 3.0". Of course, a higher version number may be used in the future.

This command line has been gradually abandoned, precisely because some major search engine companies have independently added scattered extensions in accordance with their own needs, thus making it difficult to identify the "robots.txt" with a simple uniform version number. The proposed protocol has been integrated with all the useful commands, so the version number can be reused.

Command type: Restriction
Default value: 1.0
Example:
Robot-version: 3.0

3.2.2 Last-modified

This command line represents the last modified time of the "robots.txt" written in the second line. After extension, the "robots.txt" may be updated frequently. The value of the last modified time is represented using the conventions of RFC822. Note that the value of the year can be presented either in two digits or in four digits; for example, "15" and "2015" both express the year 2015.

Crawlers should resolve the last modification time and update the "robots.txt" it finds with a new time value in the "robots.txt" file of this website that has been previously saved. If no such command is found in the "robots.txt", crawlers should consider the last modified attribute of the file provided by operating system. Crawlers must fetch a website listed in the commands of its latest "robots.txt". The "Last-modified" command is mainly presented for those websites adopting a dynamic generation approach in response to the "robots.txt" file. For other sites, this command can be ignored. When the value of the last modified attribute of the file provided by the operating system and the command line are different, crawlers should consider the value presented in the "Last-modified" command line.

Command type: Restriction
Default value: The last modified attribute of the file provided by the operating system
Example 1:
Last-modified: 30 Oct 2014 04:31:17 UT
Example 2:
Last-modified: Thu, 30 Oct 2014 04:31 UT
Example 3:
Last-modified: 30 Oct 2014 12:31 +0800

These three examples insist the same time, 12:31 on October 30th, 2014, Beijing Time. Example 1 insists an extra second.

3.2.3 User-agent

This command line follows the definition of the original REP. However, after the extension, this command is no longer the only one mark to identify crawlers. Every crawler should choose a name token for itself, and send it as part of the HTTP User-agent header when visiting a web page. Name tokens should be short and must meet the requirements of the 26 case-insensitive letters, the 10-digit requirement, and case underscores. The name token of commercial crawlers must be well documented and maintained.

This command line is part of the command target segments; it is used to identify to which specific crawlers the command content segment applies. The command is valid for a crawler when the crawler's name token contains the value of a "User-agent" line. Then, the crawler must strictly follow all the commands with the restriction type listed in the corresponding

command content segment.

Multiple "User-agent" lines can exist in a command target segment. If no "Ip-allow" line or if an "Ip-disallow" line exists, then the command target segment is matched only if a "User-agent" match is found. Specially, if a "User-agent" line has a "*" value, any crawler can be matched. Crawlers should also try to match a command target segment from top to bottom until they find a match. If no match is made, access to the corresponding site is unlimited apart from possible commands in the global command section.

We must note that the "robots.txt" will be visited only by the crawlers, so the command "User-agent" with a value of a browser name is ineffective.

Command type: Restriction
Default value: none
Example 1:
User-agent: infoseeker

For this value, matching and non-matching crawlers are listed in Table 4.

Example 2:
User-agent: infoseeker
User-agent: wiseRobot

For this value, crawlers matching and non-matching are listed in Table 5.

Example 3:
*User-agent: **

Any crawler is matched.

3.2.4 Ip-allow and Ip-disallow

The "Ip-allow" and "Ip-disallow" command lines are part of the command target segments. They are used to identify to which crawlers the command content segment applies, based on the IP address. At present, these crawlers are often deployed as distributed systems. These two commands give websites another method to identify crawlers with a smaller granularity. The values of the IP address are presented with the conventions of RFC791, using the format of four decimal numbers between 0 and 255 separated by dot. Each number can be replaced with the wildcard "*" to match any number. In particular, "*" can be used to indicate any IP address.

Table 4 List of crawlers' name tokens with matching status of the command in Example 1.

Crawlers name token	Match
Infoseeker	Yes
InfoSeeker	Yes
BobbyInfoSeeker	Yes
Info_seeker	No
InformtionSeeker	No
Infoseek	No

Table 5 List of crawlers' name tokens with matching status of the command in Example 2.

Crawlers name token	Match
Infoseeker	Yes
Wiserobot	Yes
Inforobot	No

A command target segment may have multiple "Ip-allow" command lines and "Ip-disallow" command lines. All these commands can be assumed as an IP token group. Crawlers must traverse all the commands in an IP token group from top to bottom to learn whether or not they can access the website. If more than one lines are matched, the last one should be considered as a perfect match. When the first command field of an IP token group is "Ip-allow" but the value is not "*", then the crawlers can assume that another "Ip-disallow: *" line exists before the current line. In contrast, when the first command field of an IP token group is "Ip-disallow" but the value is not "*", crawlers can assume that there exists another "Ip-allow: *" line before it.

IP token group commands should be used with "User-agent" commands. If a command target segment consists of IP token group commands but not the "User-agent" commands, crawlers can assume that a "User-agent: *" line exists in it. Crawlers should parse the "User-agent" commands first. At this point, a command target segment is matched if and only if "User-agent" commands in the crawlers are matched. If a crawler matches the "User-agent" commands, the IP token group commands inform the crawler that it is disallowed from accessing the websites. Hence, the crawler can stop parsing next the contents in "robots.txt" and stop accessing any resource in the corresponding website simultaneously.

Command type: Restriction
Default value: none
Default commands of IP token group: Ip-allow: *
Example 1:
*Ip-disallow: 222.69.212.**
Example 2:
Ip-disallow: 222.69..**
Ip-allow: 222.69.212.14
Ip-allow: 222.69.212.15
Example 3:
*Ip-allow: **

3.2.5 Allow and disallow commands

These two commands follow the definition of the original REP, and are described as follows. These two

commands are the most important commands in the command content segments as they indicate whether accessing a URL matching the corresponding path is allowed or disallowed. Their values are case-insensitive, and the corresponding paths normally begin with using the conventions of RFC1808. Wildcards "*" and "$" are allowed: "*" matches the zero or more repetitions of any character, and "$" matches the character End of Line. The addition of these wildcards can facilitate the description of an address.

A command content segment may have multiple "Allow" command lines and "Disallow" command lines. All the consequent commands can be assumed as a resource path command group. Crawlers must traverse all the commands in the resource path command group from top to bottom to learn which path they can access. If more than one lines are matched, the last one should be considered as a perfect match. When the first command field of a resource path command group is "Allow" but the value is not "*" or "/", then the crawlers can assume that there exists another "Disallow: /" line before it. In contrast, when the first command field of a resource path command group is "Disallow" but value is not "*" or "/", crawlers can assume that another "Allow: /" line exists before it. Note that "/robots.txt" is always allowed access and should not appear in the resource path command groups.

Command type: Restriction
Default value: none
Default commands of resource path command group: Allow:/
Example 1:

Disallow: /tmp

For this command, paths being allowed or disallowed access are listed in Table 6.

Example 2:

Allow: /tmp/

For this command, paths being allowed or disallowed access are listed in Table 7.

Example 3:

Disallow: /news/
Disallow: /info
Allow: /info/open

Table 6 List of resource paths with the access status in Example 1.

Resource path	Allow
/tmp	No
/temp	Yes
/tmp1	No
/tmp.html	No
/tmp/tmp0001.html	No

Table 7 List of resource paths with the access status in Example 2.

Resource path	Allow
/tmp	No
/temp	No
/tmp1	No
/tmp.html	No
/tmp/tmp0001.html	Yes

For these commands, paths being allowed or disallowed access are listed in Table 8.

3.2.6 Crawl-delay and Ip-delay

The "Crawl-delay" command, first proposed by Yandex[29], is integrated into our proposed protocol. These "Crawl-delay" and "Ip-delay" commands should be used in the command content segments to indicate the minimum interval between two consecutive visits of crawlers with the same name tokens or with the same IP addresses. Their values can be an integer or a decimal, with time presented in seconds. For a distributed crawler, if the command is "Ip-delay", two different crawler nodes can fetch the website separately. These two commands can be used when a website wants to restrict access frequency for crawlers.

Command type: Restriction
Default value: 0
Example:

Crawl-delay: 5

3.2.7 Request-rate and Ip-rate

These two commands are used in the command content segments to indicate the minimum access frequency, which is granted for the crawlers having the same name tokens or the same IP addresses. Compared with the "Crawl-delay" and "Ip-delay", the "Request-rate" and "Ip-rate" commands provide a more general crawl frequency limit method. The format of the values is an integer or a decimal, followed by "/" as well as an optional integer and a letter. The first integer or decimal indicates the number of access times, whereas the second optional integer indicates the time, which

Table 8 List of resource paths with the access status in Example 3.

Resource path	Allow
/news/news0001.html	No
/info/infoShow.html	No
/info/latest/info0001.html	No
/info0001.html	No
/info/open/info0001.html	Yes
/tmp/tmp0001.html	Yes

is defaulted to 1. The last letter can be "s", "m", or "h", indicating the units of time, second, minute, and hour, respectively. The "Request-rate" and "Ip-rate" commands can be used when a website wants to restrict the access frequency for crawlers.

Command type: Restriction
Default value: No limit
Example:
Request-rate: 500/h

Note that when a website requires some degree of technical limitations according to the commands, the time period for counting the access times should begin with the first visit of a crawler instead of a fixed period. Hence, if a crawler first visits the site at 5:12, the site should prove that the crawler can successfully access for another 499 times before 6:12.

3.2.8 Visit-time and time-forbidden

These two commands should be used in the command content segments to indicate whether access for the crawlers at a given time is allowed or disallowed. The format of values is a time expression, followed by "-" and another time expression. Time expression uses the conventions of RFC822 similar to those mentioned in Section 3.2.2. The first field indicates the beginning time of a period, and the second field indicates the ending time. If only a single time expression contains the time zone, crawlers can assume that another website also adopts this time zone.

In general, "Visit-time" command is used for daily restrictions, so the date is often not specified. In this case, if the second time is earlier than the first one, crawlers can assume that the second time is the next day following the first time field. On the contrary, "Time-forbidden" is often used for a specific period, such as during a website activity. To reduce the temporary higher loads on the web server, websites can add this command into "robots.txt" to prohibit crawlers from accessing the websites temporarily. In this case, the date is specified generally.

Command type: Restriction
Default value: No limit
Example 1:
Time-forbidden: 30 Oct 2014 00:00:00 UT-2 Nov 2014 23:59:59 UT
Example 2:
Visit-time: 01:00 +0800-06:00 +0800

3.2.9 Language, encoding, and charset

These commands should be used in the command content segments to indicate which language, encoding, and charset crawlers should support when accessing the websites. Their values are a series of language, encoding, and charset using the conventions of RFC2616. Generally, crawlers exchange such information with websites via HTTP when visiting one page. Here, these commands provide websites a description method with a larger granularity.

Command type: Restriction
Default value: No limit
Example:
Language: en-us,en
Encoding: gzip,deflate
Charset: gb2312,utf-8

3.2.10 Cookie

This command should be used in the command content segments to indicate which cookies crawlers should put into the HTTP header during a resource request. The value of this command consists of a series of key-value pairs using the conventions of RFC6265. This command is helpful when websites want to use cookies to confirm crawlers.

Command type: Restriction
Default value: No limit
Example:
Cookie: visitor=crawler

3.2.11 Mirror-site

This command should be used in the command content segments to indicate which mirror-site crawlers should visit instead of this corresponding website. The value of this command is a host using the conventions of RFC1738. Websites with multiple mirror sites can use this method to balance the crawler access pressure for each mirror sites.

Command type: Restriction
Default value: No limit
Example:
Mirror-site: www.example.com

3.2.12 Sitemap

This command is first proposed by Google[4], and is integrated into our proposed protocol. This command should be used in the command content segments to indicate where the Sitemap of the site is located. A Sitemap file's format can be XML, XML index, or TXT. Among them, XML format and XML index format follow the Sitemap Protocol. TXT format is used to list all the data pages, and ignores all the description tags of the data presented in the XML format. We suggest the webmasters to compose the deep web pages with those pages that cannot be accessed through links

by integrating traditional general-purpose web crawlers into the Sitemap, rather than composing the static files under the server or pages obtained by a crawler from the own website. The reason is that the latter pages listed in the Sitemap can be easily fetched by traditional crawlers without a significant contribution from the Sitemap. In fact, those deep web pages that are hidden behind a search form should be listed in the Sitemap. The following options are suggested for the webmasters while listing the webpages in order to accumulate the deep web pages.

(1) List one or more sets of Top-*N* proprietary data pages according to the knowledge provided in the professional field, and add them into the Sitemap. Every once in a while, list Top-*N* pages again and put those new pages into the Sitemap.

(2) List *M* proprietary data pages that users have visited recently, and add them into the Sitemap. Every once in a while, list and add new *M* pages into the Sitemap.

(3) List all proprietary data pages visited in a time period, and add them into the Sitemap. List and add all new pages visited in another consequent time period into the Sitemap.

(4) Whenever a proprietary data page is accessed, add it to the Sitemap.

When webmasters think an entire page may not be recognized by traditional crawlers, a corresponding crawler page can be designed exclusively for the crawlers. The content of this crawler page should be consistent with the original page, and such a page should not adopt any asynchronous access technology, thus ensuring that crawlers can fetch complete contents through a single request. By this way, fetching can be more effective under the following scenarios.

(1) When pages contain many asynchronous presentation methods, such as Ajax, for user experience, webmasters can generate their static snapshots as crawler pages.

(2) When pages contain many pictures, video, flash, or other non-text contents designed for user experience, webmasters can replace the links in such resources with related description texts.

(3) When pages contain many CSS codes that may not help crawlers, webmasters can remove those contents in the crawler pages altogether.

In addition to the labels defined in the Sitemap Protocol, the proposed protocol expands two labels under label <url> as its child node. The first one is <type> to insist how to fetch a page for crawlers with optional values of "data", "list", and "other". Value "data" indicates that the respective page is a proprietary data page, and crawlers just need to fetch this page without following any links in that page. This value indicates that the page is a list of a series of proprietary data pages, such as a search result page, and crawlers must follow all the links in this page to fetch proprietary data pages. Value "other" indicates that there exists no suggestion for fetching this page, and crawlers can fetch them in a traditional way. The other label is <srcloc>, which is used to inform the URL of the original page to a crawler page. For example, if the URL of the original page is "http://www.example.com/news/hot.asp?date=1030&id=8" and the URL of the crawler page for it is "http://www.example.com/crawler/news/20141030008.html", then this item can be written in the Sitemap as follows:

```
<url>
 <loc>http://www.example.com/crawler/news/20141030008.html</loc>
 <srcloc>http://www.example.com/news/hot.asp?date=1030&id=8</srcloc>
 <lastmod>2014-10-30</lastmod>
 <changefreq>daily</changefreq>
 <priority>0.8</priority>
 <type>data</type>
</url>
```

Command type: Guidance

Default value: None

Example:

```
Sitemap: http://www.example.com/sitemap.xml
```

3.2.13 Host

This command, first proposed by Yandex[29], is integrated into our proposed protocol. This command should be used in the command content segments to specify the preferred master domain whenever a website owns multiple mirrors. This command can help search engines organize the relationship between a website and its master domain. The value of this command is a host that uses the conventions of RFC1738 similar to the format used in Section 3.2.14.

Command type: Guidance

Default value: None

Example:

```
Host: www.example.com
```

3.2.14 Index-page

This command, first proposed by 360[30], is integrated into our proposed protocol. This command should be used in the command content segments to indicate which pages should be indexed. The value of this command is a case-insensitive corresponding path normally beginning with "/" and uses the conventions of RFC1808. Similar to the format used in Section 3.2.5,

wildcards "*" and "$" are allowed in this command. The "Index-page" command can be used when web pages often update their contents, hoping that crawlers visit and fetch them frequently.

Command type: Guidance
Default value: None
Example 1:
Index-page: /news/newslist.html
Example 2:
Index-page: /news/
Example 3:
Index-page: /news/news.html$*
Index-page: /news//list.html*

3.2.15 Change-always, change-hourly, change-daily, change-weekly, change-monthly, change-yearly, and change-never

These commands are extended by the Sitemap Protocol[4] to provide a way with which to describe the update frequency of the pages with a larger granularity. This command should be used in the command content segments. The value of this command is a case-insensitive corresponding path normally beginning with "/" and uses the conventions of RFC1808. Similar to the format used in Section 3.2.5, wildcards "*" and "$" are allowed in this command. These commands can be used when websites hope crawlers visit and fetch some pages with a specific frequency.

Command type: Guidance
Default value: None
Example 1:
Change-hourly: /news/newslist.html
Example 2:
Change-daily: /news/hot/.html*

4 Application

Given that both website and crawler must abide by this protocol, we present in this section an application scenario from both sides with an illustrated example.

4.1 For websites

Here we build a simple news website at http://www.example.com/. The architecture of the site is shown in Fig. 1.

The website includes

http://www.example.com/ — Homepage;

http://www.example.com/login.do — Login page;

http://www.example.com/reg.do — Registration page;

http://www.example.com/news.do — News search page;

http://www.example.com/newsItem.do?id=9, etc. — Specific news page;

http://www.example.com/html/news9.html, etc. — Static snapshot of specific news page;

http://www.example.com/newsRelease.do — News Release Page;

http://www.example.com/robots.txt—The only correct path of robots.txt;

http://www.example.com/sitemap.xml—The website's sitemap.xml.

The homepage contains links to the login page, registration page, news search page, and news release page. The news search page contains links to the latest specific news pages and a search form. All the search results, including links to these eligible specific news pages, are returned in the news search page. The news release page can be visited after a user logs in as an administrator. In specific news pages, users can see the existing comments and post new comments after logging in. The comments are loaded asynchronously by Ajax technology after the news text is completely loaded.

The website sets up a module to automatically maintain sitemap.xml and generates snapshots for

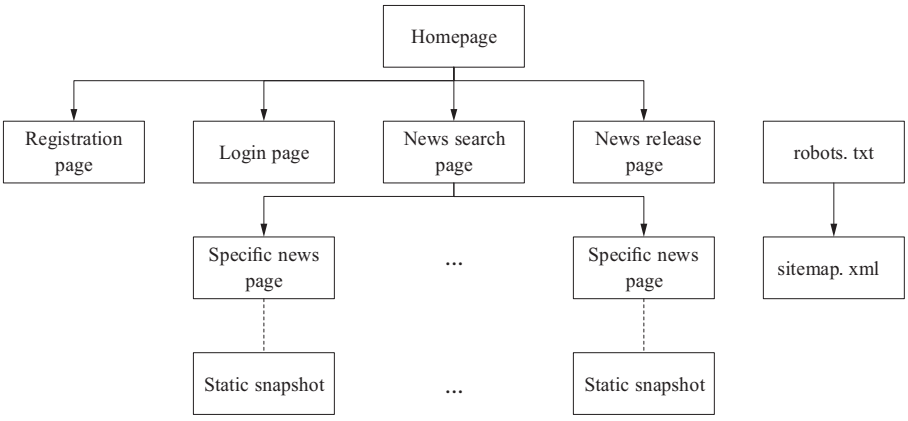

Fig. 1 Website architecture.

specific news pages. Whenever a news piece is released, a static web page snapshot is automatically generated and maintained at http://www.example.com/html/, and the sitemap.xml is also updated automatically. A new metadata item corresponding to the news page is added into the sitemap.xml, where the value of <loc> tag is the path of the specific news page, the value of <srcloc> tag is the path of the static snapshot, and the value of <type> tag is the "data". Whenever a new comment is generated, the static snapshot of the news page is regenerated, and the value of corresponding <lastmod> tag is updated in the sitemap.xml. Here, if a website has a larger size compared with others, it can update the static snapshots at an interval instead of generating them immediately after users post new comments. The workflow processes are shown in Figs. 2 and 3.

4.2 For crawlers

The crawler system normally has a scheduling thread and several crawler threads in either stand-alone or distributed fashion. To crawl a website, each task is assigned to a crawler thread via the scheduling thread. The workflow of a crawler thread is shown in Fig. 4.

Whenever a crawler thread receives a new site task, it first attempts to access its "robots.txt". If this fails, the crawler begins to crawl the whole site with the traditional method, because the access failure means that no restriction or guidance exists in this site. Otherwise, the crawler analyzes the "robots.txt" and records all the commands to its memory. All these parsed commands can help subsequent crawling of the same website. The main workflow to parse the "robots.txt" is shown in Fig. 5.

The crawler threads of our particular case are divided into two working status: normal and temporary. When a normal-status thread completes a task, it attempts to ask the scheduling thread for a new task. If there are no tasks to schedule, the corresponding thread sleeps for a while and then tries to ask again. The temporary-status is usually opened by a normal-status thread to track the status of the crawlers while crawling a website of low frequency visits. To crawl a website of this kind, the crawler thread should wait longer. As shown in Figs. 4 and 5, several considerations must be made. First, when a crawler thread finds the time point that is not available for visit after parsing the related commands, a countdown timer is triggered. When time elapses, a temporary-status crawler thread is initiated to fetch the site. Second, a crawler thread obtains a minimum period value for visiting the site by calculating related commands. If the calculated value is greater than a threshold value, the site is believed to be more time consuming for crawlers to fetch this site. Thus, a temporary-status crawler thread is initiated to take over this task. Finally, a crawler thread sequentially parses all the sitemap.xml file listed in "robots.txt" and saves all the URLs found in the Sitemap files into a non-crawl queue. If a URL has an <srcloc> tag, crawlers fetch the static snapshot path instead of the origin path, after which they record the corresponding relationships. The workflow of the crawler thread to fetch the pages in a website is shown in Fig. 6.

The workflow to fetch pages is similar to the traditional way. However, before fetching a URL, the crawler thread first checks all kinds of settings to ensure compliance with the protocol.

5 Experiments

In the next sequential experiments, we use the crawler described above to fetch our site and analyze the benefits of the protocol through the results. To facilitate the experiments, we made small modifications to the abovementioned crawler. For instance, the crawler thread will only be terminated after completing the fetching task, rather than while waiting for another task. In order to facilitate the observation of the crawling results, we output some useful information, such as URL and crawling time cost, during the fetching process and then save all the HTML into the output files. The fetching results from the crawler when the website

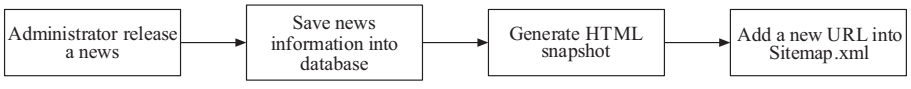

Fig. 2 Website workflow for news releases.

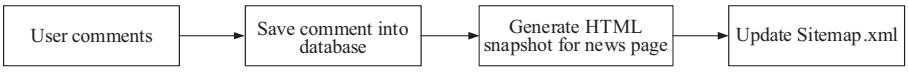

Fig. 3 Website workflow for new comments.

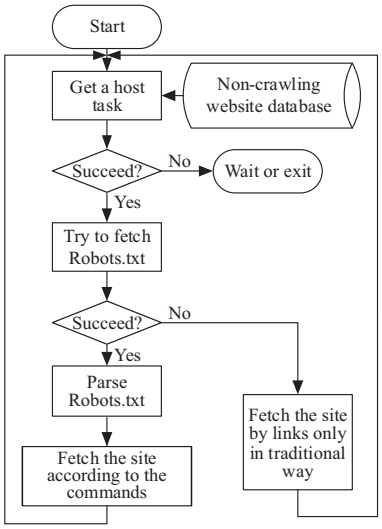

Fig. 4 Workflow of a crawl thread.

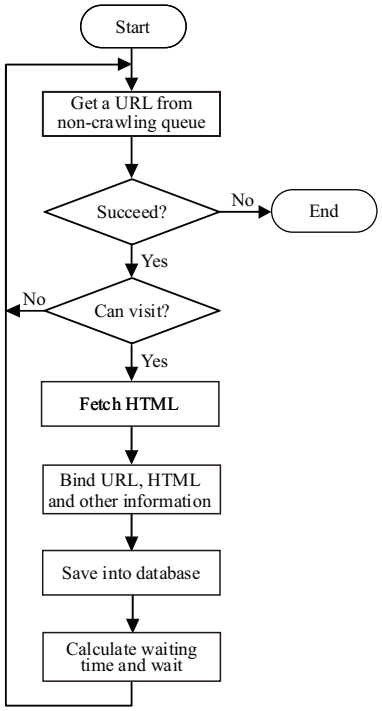

Fig. 5 Workflow to parse robots.txt.

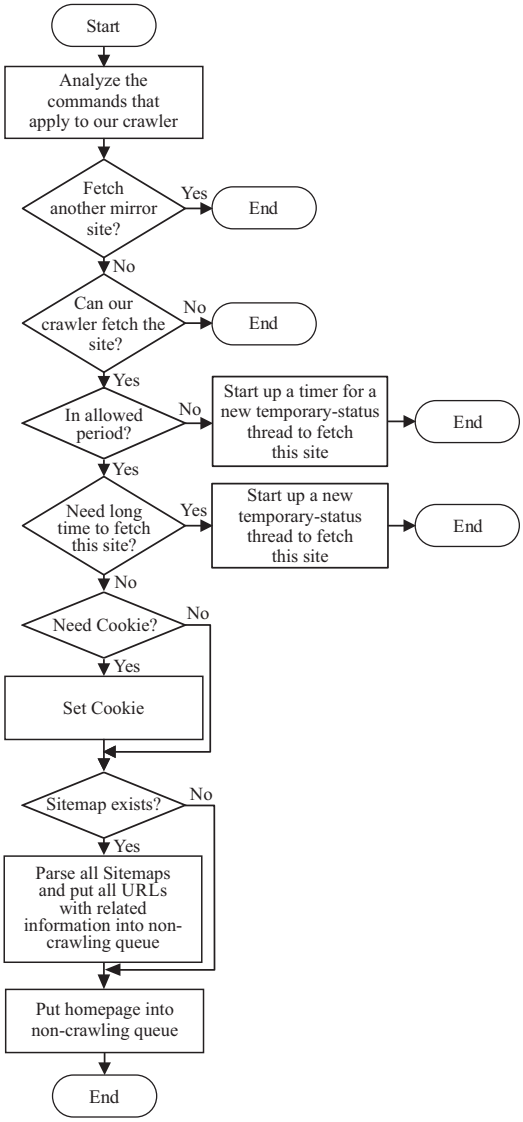

Fig. 6 Workflow to fetch pages.

Table 9 Fetching results without robots.txt.

Resource	Fetch	Crawl delay (s)
Home page	Yes	0
Useless pages (e.g., login page)	Yes	0
Latest news pages	Yes	0
Old news pages	No	0

does not establish any "robots.txt" file are shown in Table 9.

As shown in Table 9, although the crawler can fetch some pages through the links, most specific news pages are still missing. Given that the home page has three links to the three latest news pages, respectively, the crawler can fetch these three pages. Here, we can easily guess that the fetching results of these three news pages have no comment information, because the crawler cannot obtain the paths of these snapshot pages through these links. Though the website has its sitemap.xml, the crawler cannot obtain its path without "robots.txt".

Now, the "robots.txt" is set as shown below.

Robot-version: 3.0
Last-modified: 19 Mar 2015 12:00:00 +0800

*User-agent: **
Disallow: /login.do
Disallow: /reg.do

```
Disallow: /newsRelease.do
Request-rate: 120/m
{
 Allow: /newsitem.do?
 Crawl-delay: 5
}
{
 Allow: /html/
 Crawl-delay: 1
}

Sitemap: http://10.60.149.65:8089/test1/sitemap.xml
Cookie: user=robot
Language: zh-CN
Encoding: gzip, deflate
```

The contents of sitemap.xml are automatically maintained by the website. A URL item is similar to

```
<url>
 <loc>http://10.60.149.65:8089/test1/html/news1.html</loc>
 <srcloc>http://10.60.149.65:8089/test1/newsItem.do?id=
                1</srcloc>
 <lastmod>2015-03-19</lastmod>
 <changefreq>weekly</changefreq>
 <priority>0.5</priority>
 <type>data</type>
</url>
```

The fetching results from the crawler are shown in Table 10.

First, owing to restrictions on visiting login page, registration page, and news release page in "robots.txt", respectively, the crawler has filtered out these addresses and no longer crawls these useless pages. Second, the site gives a "sitemap.xml". The crawler has fetched all the news pages based on the metadata in the "sitemap.xml", and most of the fetched pages here are unavailable to traditional crawlers. Furthermore, we can see that the home page has been crawled within 0.5 s. That is because that "robots.txt" commanded the crawler to set the visiting frequency at 0.5 s per URL in general. We can also observe that all news pages are crawled within 1 s. The crawler did not allow these URLs to match the "/newsltem.do?" but allowed for the matching of the "/html/" and sets their access gap to 1 s. The reason is that the crawler found the <srcloc> tags of these URLs in the "sitemap.xml" and tried to fetch the static snapshots of the news pages provided by the website, instead of the original addresses of these pages, and the paths of those static snapshot pages actually matched the "/html/". In comparison, if we use the old REP, we can only restrict whether resources are allowed access to. The "robots.txt" will be something like

```
User-agent: *
Disallow: /login.do
Disallow: /reg.do
Disallow: /newsRelease.do
```

The fetching results from the crawler are shown in Table 11.

Obviously, it can help prevent the crawler from accessing those useless pages. However, it has limited functionality. Crawlers cannot fetch deep web contents or Ajax pages like those old news pages, and the crawler cannot know the required access frequency of the site; hence, the crawler will put a lot of pressure to the site while working. As we expand the commands, the REGP can apparently achieve more tasks. Now we add a "Visit-time" command in the global command paragraph to insist the crawlers to visit the site after 21:00 every day as "Visit-time: 21:00:00 +0800-08:00:00 +0800".

Suppose that the time to fetch the site is set to 21:00. We could easily predict that the normal-status crawler thread would find it, but that it should wait for a long time to fetch the site after parsing the "robots.txt". A countdown timer would now be triggered by the thread. When the timer expires, a new temporary-status crawler thread is initiated to take over the task.

Finally, we change the language requirements to French in the "robots.txt" into "Language: fr". As the crawler does not support French, it would have no access to any pages of the site. It should be noted here that this is just an example; all commands in the "robots.txt" must be set according to the actual situation under real circumstances.

6 Conclusion

The existence of numerous deep web pages in websites and the widespread use of Ajax make it difficult

Table 10 Fetching results with robots.txt.

Resource	Fetch	Crawl delay (s)
Home page	Yes	0.5
Useless pages (e.g., login page)	No	0
Latest news pages	Yes	1.0
Old news pages	Yes	1.0

Table 11 Fetching result with the old REP.

Resource	Fetch	Crawl delay (s)
Home page	Yes	0
Useless pages (e.g., login page)	No	0
Latest news pages	Yes	0
Old news pages	No	0

for general-purpose web crawlers to fetch information quickly and efficiently. In this paper, an REGP is proposed by using the original REP as basis and by integrating the independent scattered expansions of the protocols developed by major search engine companies. Our proposed protocol expands the file format and command set of the REP and the two labels of the Sitemap Protocol. Through our protocol, websites can express their aspects of requirements for restrictions and guidance to the visiting crawlers as well as ensure rapid access of general-purpose crawlers to deep web pages and Ajax pages. Our proposed protocol also enables the crawlers to easily and effectively obtain the open data from various websites. Finally, this paper describes a specific application scenario, in which both a website and a crawler work together with support from our protocol. We then conducted a series of experiments to demonstrate the efficiency of our protocol.

However, similar to the original REP, our protocol also encountered a problem when the crawlers do not comply with the protocol and fetch websites indiscriminately, thus making the protocol itself and the objective website. This is because our protocol is only used for websites to provide a description of crawling requirements and guidance. This issue will be addressed in our future work, along with the growing maturity of different Internet-based applications.

Appendix

BNF-like Syntax

The BNF-like syntax used in our proposed protocol is similar to the description of the original REP and is reproduced in our protocol. The syntax used in our protocol is a BNF-like description using the conventions of RFC 822, except that "|" is used to designate alternatives in the original protocol. Briefly, literals are quoted with " " and parentheses [(" and ")] are used to group elements. In addition, optional elements are enclosed in [brackets], and elements may be preceded with <n>* to designate *n* or more repetitions of the following element, with *n* defaults to 0. Moreover, "#" is used to describe the encoding, language and charset, as defined in RFC 2616. The BNF-like syntax is described below.

robotstxt = *blankcomment | *blankcomment head 1*commentblank record *(1*commentblank 1*record) 1*commentblank globalrecord *blankcomment blankcomment = 1*(blank | commentline)

Blank line or comment line, possibly used in beginning or end of the file

commentblank = *commentline blank *(blankcomment)

Blank lines; one pure blank line is necessary, used to separate two command part.

commentline = comment CRLF

comment = *space "##" anychar

blank = *space CRLF

space = 1*(SP | HT)

CRLF = CR LF

head = versionline *commentline lastmodline

record = robotrecord *commentline rule

globalrecord = *(commentline | ruleline)

versionline = *space "Robot-version:" *space version [comment] CRLF

lastmodline = *space "Last-modified:" *space date-time [comment] CRLF

robotrecord = (agentrecord | iprecord | agentrecord iprecord)

agentrecord = agentline *(commentline | agentline)

iprecord = ipline *(commentline | ipline)

agentline = *space "User-agent:" *space agent [comment] CRLF

ipline = (ipdisallowline | ipallowline)

ipallowline = *space "Ip-allow:" *space ipaddress [comment] CRLF

ipdisallowline = *space "Ip-disallow:" *space ipaddress [comment] CRLF

rule = 1*ruleline *(commentline | ruleline) *innerrule

ruleline = (disallowline | allowline | sitemapline | crawldelayline | ipdelayline | requestrateline | iprateline | visittimeline | hostline | indexpageline | changealwaysline | changehourlyline | changedailyline | changeweeklyline | changemonthlyline | changeyearlyline | changeneverline | languageline | encodingline | charsetline | mirrorsiteline | timeforbiddenline | cookieline | extension)

innerrule = lparenthesisline [robotrecord] *commentline rule *commentline

lparenthesisline = *space "{" [comment] CRLF

rparenthesisline = *space "}" [comment] CRLF

disallowline = *space "Disallow:" *space rpath [comment] CRLF

allowline = *space "Allow:" *space rpath [comment] CRLF

sitemapline = *space "Sitemap:" *space httpurl [comment] CRLF

crawldelayline = *space "Crawl-delay:" *space int [comment] CRLF

ipdelayline = *space "Ip-delay:" *space int [comment] CRLF

requestrateline = *space "Request-rate:" *space rate [comment] CRLF

iprateline = *space "Ip-rate:" *space rate [comment] CRLF

visittimeline = *space "Visit-time:" *space date-time [comment] CRLF

hostline = *space "Host:" *space host [comment] CRLF

indexpageline = *space "Index-page:" *space httpurl [comment] CRLF

changealwaysline = *space "Change-always:" *space rpath [comment] CRLF

changehourlyline = *space “Change-hourly:” *space rpath [comment] CRLF

changedailyline = *space “Change-daily:” *space rpath [comment] CRLF

changeweeklyline = *space “Change-weekly:” *space rpath [comment] CRLF

changemonthlyline = *space “Change-monthly:” *space rpath [comment] CRLF

changeyearlyline = *space “Change-yearly:” *space rpath [comment] CRLF

changeneverline = *space “Change-never:” *space rpath [comment] CRLF

languageline = *space “Language:” *space language [comment] CRLF

encodingline = *space “Encoding:” *space encoding [comment] CRLF

charsetline = *space “Charset:” *space charsetval [comment] CRLF

mirrorsiteline = *space “Mirror-site:” *space host [comment] CRLF

timeforbiddenline = *space “Time-forbidden:” *space date-time [comment] CRLF

cookieline = *space “Cookie:” *space set-cookie-string [comment] CRLF

extension = *space token “:” *space value [comment] CRLF

version = 1*DIGIT “.” 1*DIGIT

agent = token

ipaddress = (IPv4address | IPv6address)

rpath = “/” path int = 1*DIGIT

rate = 1*DIGIT “/” *DIGIT (“s” | “m” | “h”)

encoding = 1#(codings [“;” “q” “=” qvalue])

language = 1#(language-range [“;” “q” “=” qvalue])

charsetval = 1#((charset | “*”)[“;” “q” “=” qvalue])

value = <any CHAR except CR or LF or “##”>

anychar = <any CHAR except CR or LF>

CHAR = <any US-ASCII character (octets 0 - 127)>

CTL = <any US-ASCII control character (octets 0 - 31) and DEL (127)>

CR = <US-ASCII CR, carriage return (13)>

LF = <US-ASCII LF, linefeed (10)>

SP = <US-ASCII SP, space (32)>

HT = <US-ASCII HT, horizontal-tab (9)>

Where, the syntax for “token” is defined in RFC 1945. The syntaxes for “IPv6address” and “IPv4address” are defined in RFC 2373. The syntax for “date-time” is defined in RFC 822. The syntaxes for “httpurl” and “host” are defined in RFC 2373. The syntaxes for “codings”, “qvalue”, “language-range”, and “charset” are defined in RFC 2616. The syntax for “set-cookie-string” is defined in RFC 822. The syntax for “path” is defined in RFC 1808.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (Nos. 61672381 and 90818023).

References

[1] C. C. Chang, B. He, C. Li, M. Patel, and Z. Zhang, Structured databases on the web: Observations and implications, *ACM SIGMOD Record,* vol. 33, no. 3, pp. 61–70, 2004.

[2] M. Koster, A standard for robot exclusion. NEXOR, http://www.robotstxt.org/rig.html, 1994.

[3] M. Koster, A method for web robots control, http://www.robotstxt.org/orobots-rfc.txt, 1997.

[4] Google, Sitemaps XML format, http://www.sitemaps.org/protocol.html, 2008.

[5] X. F. Xian, P. P. Zhao, W. Fang, and J. Xin, Quality-based data source selection for web-scale data integration, in *Proc. 2009 International Conference on Machine Learning and Cybernetics*, Baoding, China, 2009, pp. 427–432.

[6] U. Noor, A. Daud, and A. Manzoor, Latent dirichlet allocation based semantic clustering of heterogeneous deep web sources, in *Proc. 5th International Conference on Intelligent Networking and Collaborative Systems*, Xi’an, China, 2013, pp. 132–138.

[7] S. Raghavan and H. Garcia-Molina, Crawling the hidden web, in *Proc. 27th International Conference on Very Large Data Bases*, Roma, Italy, 2001, pp. 129–138.

[8] J. Cope, N. Craswell, and D. Hawking, Automated discovery of search interfaces on the web, in *Proc. 14th Australasian Database Conference*, Adelaide, Australia, 2003, pp. 181–189.

[9] G. Liu, K. Liu, and Y. Dang, Research on discovering deep web entries based on topic crawling and ontology, in *Proc. 2011 International Conference on Electrical and Control Engineering*, Yichang, China, 2011, pp. 2488–2490.

[10] Y. Saissi, A. Zellou, and A. Idri, Extraction of relational schema from deep web sources: A form driven approach, in *Proc. 2014 Second World Conference on Complex Systems*, Agadir, Morocco, 2014, pp. 178–182.

[11] X. Shao, Y. Liu, and L. Liu, Research on overall pattern matching method in deep web, (in Chinese), *Acta Scientiarum Naturalium Universitatis Nankaiensis*, vol. 45, no. 5, pp. 24–31, 2013.

[12] D. Zheng and Z. Cui, Research on strategy of crawling deep web strategy, (in Chinese), *Computer Engineering and Design*, vol. 27, no. 17, pp. 3154–3158, 2006.

[13] S. J. Pusdekar and S. P. Chhaware, Using visual clues concept for extracting main data from deep web pages, in *Proc. 2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies*, Maharashtra, India, 2014, pp. 190–193.

[14] S. Pei and X. Ye, Development status and trend of open API abroad, (in Chinese), *Information Science*, vol. 27, no. 12, pp. 1896–1900, 2009.

[15] S. Kim and H. Kim, Ontology modeling for provision of semantic based open API information, in *Proc. 15th International Conference on Advanced Communication Technology*, Rajampet, India, 2013, pp. 664–667.

[16] W. Jung, S. I. Kim, and H. S. Kim, Ontology modeling for rest open apis and web service mash-up method, in *Proc. 2013 International Conference on Information Networking*, Bangkok, Thailand, 2013, pp. 523–528.
[17] J. Zhang and S. Cao, Design and implementation of a third-party developers auditing system based on open API, in *Proc. 6th IEEE International Conference on Software Engineering and Service Science*, Beijing, China, 2015, pp. 803–806.
[18] L. Wang, Z. Li, Y. Tan, and Z. Liu, Research on UG/Open API for secondary development of UG, (in Chinese), *Development & Innovation of Machinery & Electrical Products*, vol. 19, no. 5, pp. 105–106, 2007.
[19] Google, Google Onebox, https://www.google.com/work/gsa/onebox.html, 2015.
[20] T. Zhou, Baidu pushes Aladdin Plan for searching deep web, (in Chinese), *China Securities News*, B3, Dec 19, 2008.
[21] Baidu, Box computing, http://boxcomputing.baidu.com/, 2009.
[22] H. Guo, Y. Lu, and J. Liu, Ajax crawling algorithm based on a kind of state transition diagram, *Application Research of Computers,* vol. 26, no. 11, pp. 4266–4269, 2009.
[23] G. Frey, Indexing ajax web applications, master degree dissertation, Dept. Computer Science, Institute of Computational Sciences, ETH, Zurich, Switzerland, 2007.
[24] A. Mesbah, E. Bozdag, and A. D. Van, Crawling Ajax by inferring user interface state changes, in *Proc. 8th International Conference on Web Engineering*, New York, NY, USA, 2008, pp. 122–134.
[25] H. Shao and F. Li, Research and implement on information extraction of dynamic pages based on tree model algorithm, (in Chinese), *Computer Applications and Software*, vol. 24, no. 10, pp. 99–100, 2007.
[26] W. Ma, X. Chen, and W. Shang, Advanced deep web crawler based on Dom, in *Proc. 5th International Joint Conference on Computational Sciences and Optimization*, Harbin, China, 2012, pp. 605–609.
[27] B. Xia, J. Gao, T. Wang, and D. Yang, An efficient effective pages fetching method for websites with dynamic script, (in Chinese), *Journal of Software*, vol. 20, no. suppl., pp. 176–183, 2009.
[28] H. Li, L. Wu, H. Lai, C. Zheng, and K. Huang, Effective web crawling algorithm for Ajax webpages, (in Chinese), *Journal of University of Electronic Science and Technology of China*, vol. 42, no. 1, pp. 115–121, 2013.
[29] Yandex, Using robots.txt, https://yandex.com/support/webmaster/controlling-robot/robots-txt.xml, 2015.
[30] 360, Robots exclusion support, http://www.so.com/help/help_3_2.html, 2015.

Zhijun Ding received the MS degree from Shandong University of Science and Technology, Taian, China, in 2001, and PhD degree from Tongji University, Shanghai, China, in 2007. Now he is a professor of Department of Computer Science and Technology, Tongji University. His research interests are in mobile internet, services computing, formal engineering, and Petri nets. He has published more than 80 papers in domestic and international academic journals and conference proceedings.

Dajie Ge received the MS degree from Tongji University, Shanghai, China, in 2016. Now he is a game developer at Shanda Games, Inc. His research interest is in crawler.