A new method of page standardization based on DOM

Weicheng Ma[1, a], Yong Fan[2,b], Wenqian Shang[3,c]and Fengyan Wu[4,a]

[1]School of Computer, Communication University of China, Beijing, China

[2] School of Computer, Communication University of China, Beijing, China

[a]hikariTGpass@gmail.com, [b] fanyong19932006@163.com, [c]shangwenqian@163.com

Keywords: DOM Tree; unreadable codes; charset; page standardization

Abstract. With the rapid development of the Internet, information as well as websites boomed. And, being differentiated in style, structure or content, it is unable to get the information from different pages using the same model, while it is really a waste of time to search each line of the page to find useful information because of noises. That makes arranging all the information from a page to build a DOM tree for search a wise choice firstly because it raises the possibility of searching accurately. What is more, converting a web page into a tree helps identify the main frame of the page. On the other hand, unreadable codes, which are caused by invalid transformation between languages, is a barrier separating people apart from information on websites of other districts of the world. Our work is aimed at solving the listed problems to make information from all around the world accessible while convenient to extract.

Introduction

With the advance of technology, the Internet has become a universal tool for people all around the world. By the end of 2011, the number of Internet users worldwide has reached 2 billion according to HamadounTouré, ITU secretary general. [1] Meanwhile, the number of websites has also skyrocketed since the year of 1995, meeting the number of 698000000 in June, 2012, with around 200 million of them active. The data is from the June 2012 web server survey of netcraft. [2] It is clear that we are in a time of information explosion, and that the search for techniques for extracting useful parts of information from such an avalanche of different websites is indispensable.

Previous work has been done for getting pages according to given URLs. As an example, HTTPClient [3], an open-source tool written in JAVA, can satisfy the need of downloading web pages automatically if given the addresses of them. However, simply using traditional web crawler for getting pages and then analyze them line by line for needed information may not provide good efficiency, since each page has its own structure, and the expected data may be put somewhere different from any other page, making it necessary to go through all the codes for just a few lines of them. That is why we choose to use DOM tree for standardization.

There have already been products transforming HTML pages into DOM trees. Early works such as JTidy [4] and HTMLCleaner [5] can meet the needs of pairing incomplete tags and replacing improper tags according to the standard of XML. Also, there are still researches concerning about DOM tree building method now, hoping to come up with new ways of web page standardization which would fit most of the pages on the Internet or would help raise the efficiency of information extraction. For example, ZHANG Rui-xue et al. [6] have come up with an algorithm of traversing the HTML pages conversely to build DOM trees and finding out blocks of main information by matching the sub-structure of blocks.

We match tags using a simplified decision tree. There are now already some famous decision tree algorithms like C4.5 and ID3. Since it is not respected to do that complex work, the decision tree in our system is modified to meet the needs of deciding the type of tags and of automatically adding unknown tags into it. Similar works include that of Li Ya et al. [7] and Li Guangxia et al. [8].

Another problem in DOM tree building now is that of unreadable codes. People from all around the world, and using different languages as well, can now meet on the Internet more often, making the problem of unreadable codes caused by wrong encoding is growing a severe problem. Products are already there aimed at solving this problem. Researches are done to improve the method of getting the

Published by Atlantis Press, Paris, France.

right charsets of pages. Li Xiaoxin [9] deals with pages by getting the charset by first analyzing the information in the message heads of pages and then searching for <meta> tags for the charsets of pages if the former step fails to get a valid charset. Li Xiaoxin's method is in some extent similar to ours.

To sum up, our system is a DOM tree builder which realizes the importation as well as exportation of DOM trees and can wipe out the barrier of different encoding of pages. In this paper, the structure and overview of our system in the System Description section, the demonstration of each module in Detailed Explanation section, and experiments in the next. Conclusion of this paper will be put at the end of this paper.

System Description

System Flow Chart.

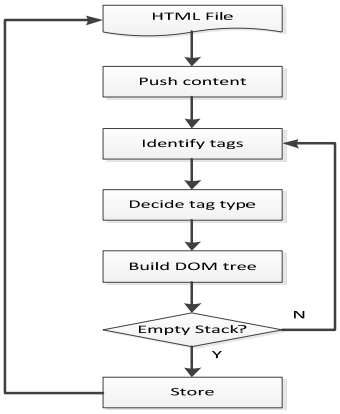

Figure 1. System Flow Chart

General Statement of System Process. As is shown in Fig. 1, our system starts work by reading a HTML page into memory for analysis. The files are read character by character and the characters are pushed into a stack for further use. We choose stack as buffer for identifying the tags from the end of file. The reason why we build DOM trees reversely is that it is easier for the system to detect unpaired tags and add the missing part of them more easily, since if a beginning tag is found without its ending tag behind it, the tag is not correctly matched and should be paired. The position for the tags to be added is decided by the types of them. If the missing tag is not a "container tag" such as , <table>, <td>, <tr>, it will be inserted right behind its beginning cousin and the text they contain. Otherwise the missing tag would be put in front of ending tag of its father block. Also, building the DOM tree reversely helps avoid the wrong division of tags in afraid of '<' or '>' signs in the text between tags.

When met the end of file, the tag-identification and DOM-tree-builder modules are activated. The characters pushed in are now popped out and the tags are identified by proper pairing of '>' and '<' signs and the text between them are extracted and made into tags named <text> with the value of the extracted text. Specially, the text between <script></script> tag pairs are made into <! CDATA [[ ]]/> tags and that in <style> blocks into <! SDATA [[ ]]/> tags since they are functional texts and may be used in different ways from normal texts. The tags are connected according to the relationships among them to become a tree and then stored after the stack of characters is popped up to be empty. The rules of building it are explained in detail in section 4.

Detailed Explanation

In the former section, our system has been roughly introduced. The main function of it has been explained and the duty of each module declared. In this section, we would divide the system into several main functional modules and introduce them one by one in the sequence of which the data flows. Experimental certification of the time bounds is placed in the next section.

Encoding-detecting Module. This module is aimed at getting the charset of a certain page. Clearly it is the most secure way of doing it by analyzing the HTML file line by line to get the value of attributes “charset”. However, since the <meta> tags in which the “charset” attributes are included may appear many times in a file, with most of them useless and the fact that searching each line for certain words costs remarkable time, it is not worthwhile to have the charsets of every page attained in this way. On the other hand, getting the charset of a page using its TCP/IP message header works a lot faster the problem of it is that it performs poor accuracy, with a success rate of only around 40 per cent. In consideration of both the accuracy and time bound of getting the charset, we have chosen a way of detecting the charset firstly from the TCP/IP message header and then from analyzing the page if the former way fails to make a goal.

When a HTML page is first got, the module tries to get charset from the TCP/IP message header using the URL of the page and the class java.net.URL. If the charset is got successfully, the gotten charset would be returned and a message shown. Otherwise, the HTML file would be opened using the charset of UTF-8 and read lines for the search of tags <meta>. It is safe to do this because the attributes are always in English and would by no means be unreadable when encoded UTF-8. Once the expected tag is found, the module will search further for the attribute “charset” inside of the <meta> tag and the charset is gotten and returned if found, or it would drop the line and go for the next if not found. The charset would be set “UTF-8” if the attribute “charset” is never found until met the end of file.

HTML File Reader. The file reader module of our system completes a relatively simple work of opening HTML files using the detected charset and traverses the whole page character by character until the end of file. Once gotten, the characters are pushed into a stack since our system identifies and matches the tags reversely so that the files would be dealt with from behind. The empty lines in the files are removed for the benefit of further analysis.

Tag Identifier and Classifier. The duty of this part of our system is to combine the characters pushed in the stack into tags and decide the type of them using a simplified decision tree algorithm.

The border of tags is no doubt the paired ‘<’ and ‘>’ signs. In order to eliminate these signs in the text parts of a page, we have come up with a set of rules deciding whether a ‘<’ or ‘>’ sign is part of the border of a tag. The rules are listed as following.

Rule 1 :’<’ or ‘>’ signs in quotation marks or brackets or braces are not borders.

Rule 2: The priority of being considered as part of a border is raised if the ‘<’ sign has a ‘/’ sign immediately followed or a ‘/’ right in front of a ‘>’ sign.

Rule 3: In adjacent ‘<’ or ‘>’ pairs, the former ‘>’ or the latter ‘<’ is considered to be part of a border.

The principles are based on the consideration that texts containing the two signs are often quoted in quotation marks, brackets or braces, and that the texts usually exceeds the tags in length. And it is clearly working better compared with simply matching the adjacent ‘<’ and ‘>’ pairs.

Once the tag is bordered, it would be sent to the classification tool for the type of it. In that process, the tag would be firstly decided whether it is a beginning tag, an ending tag, or a single tag. This could be judged easily by whether the ‘<’ sign is followed by a ‘/’ or whether the ‘>’ has a ‘/’ sign in front of it. This step is important for the building of the tree and the insertion of missing tags. The result of the first step would be recorded using pre-decided codes, like we use 0 to present beginning tags, 1 for ending tags and 2 for single tags.

Afterwards, the tag will be divided into a name part and an attribute part. The former part of it is judged about whether it can be the border of a block of things, can hold only texts, or are simply stylizers by matching them to the nodes of a decision tree. In our work, the decision tree is simplified so as to meet our needs of simply deciding the expected things between a pair of tags. For most of the time, the tree stays static. Only when a new tag appears repeated and repeated again would it be inserted to the tree.

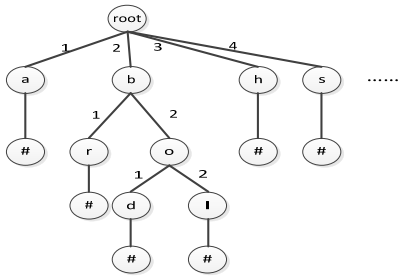

Figure 2. Example of a Decision Tree

TABLE I. EXAMPLE OF PATH-TYPE TABLE

Path	Name	Type
1	a	text
21	br	style
221	body	block
222	bold	style
3	html	block
4	style	text

We use hash set for storing the tree to meet the needs of quick search, insertion and removal. The ASCII value of each character is used as the key to calculate the hash value of them. The system is insensitive to uppercase of lower case characters, and all of them are transferred to lower case when their ASCII codes used. The hash function is as following:

$$f(key) = (key - 48) \bmod 39 \quad (1)$$

Since the only elements of tag names are English characters and numbers, the function is made as simple as possible while can be proved to have no collisions among the elements.

DOM Tree Builder. The DOM Tree Builder module accomplishes the core of our system. The main job of it is to get the tags from the former module and insert them to the tree according to some rules. The intact process is as following:

- A special node containing no parent node and a name of “root” is made and activated.
- A tag is received.
- If the tag is an ending tag, insert it to the present active node as a child and set it as active.
- Else if the tag is a single one, simply make it as a child of the present active node.
- Else, make a judge of whether it matches the name of the present active node. If true, activate the parent node of the now-active one. Otherwise, make it as the child of the active node now and then set it as active. Then begin the completion process of it. Specially, if the current active node meets the beginning tag of its parent node, it should be ended and its parent node activated.
- Completion process: if the tag holds a block, set it as the parent of presently all the other children of its parent node. Otherwise, include the adjacent text node into it. After that, activate its parent node
- Execute the whole process in a loop until the end of one file, and send the tree for exportation.
- Wait for the next file to come in.

Since it goes through each path twice, the time bound of this module can be proved to be O(2n).

Exportation and Importation Module. The Exportation and Importation Module is the destination of a certain file. As implied by the name, this module contains two functional parts. The exportation part is for the store of XML page file, and the importation part is left as an interface for other programs to use for reading the XML file in as a DOM tree.

To realize the exportation of XML file, a process as following is designed:

- Create an empty file which gets its name according to the URL of the original HTML file.
- Write a line of XML information to mark it as an XML file.
- Traverse the tree depth-firstly and write each tag into the file as a single line. Write one more ‘\t’ sign in front of the line if the tag in the previous line comes to be the beginning tag of the parent node of the present one. Otherwise, when the cursor comes back to the parent node of the previous

one, decrease one of the '\t' signs used. If all the children of a node are traversed, write the ending tag of it and move the cursor to the parent node of it.

- If the cursor comes back to the root node, save the file and wait for the next input.

For the importation part, the process is reversely done and the number of the '\t' signs shows the layer each tag is in. The nodes are stored in a hash set to benefit the search for tags, and they are linked together using the indexes of them in the adjacency list of the hash set. The key is chosen as the name of tags, so that tags of the same type are stored together to make the further extraction of information easier.

Conclusion

In this paper, we have come up with a new method for building DOM trees. Our system makes the goal of achieving the building of DOM trees in an innovative way of combining the encoding-getting, tag classification and DOM tree exportation and importation together. It is proved by the experiments that our system works efficiently compared with others' works. The encoding-getting is also of high rate of success as well as optimistic time bound. So generally speaking, our work can meet the needs of solving the problem of unreadable codes and of standardizing HTML files effectively.

Acknowledgment

This work was supported by the project "Undergraduate Scientific and Technological Innovation Project of Communication University of China (201210033063)", supported in part by the project "National Science and Technology Support Program (2009BAH40B04)" and partly supported by the project "48th Postdoctoral science foundation of China" (20100480357).

References

[1] http://www.news.com.au/technology/worldwide-internet-users-reaches-2bn/story-e6frfro0-1225995328284

[2] http://news.netcraft.com/

[3] http://hc.apache.org/httpclient-3.x/

[4] http://jtidy.sourceforge.net/

[5] http://htmlcleaner.sourceforge.net/

[6] R.X. Zhang, M.Q. Song and Y.L. Gon, Parsing DOM Tree Reversely and Extracting Web Main Page Information

[7] Y. Li and J.N. Guo, Research on Pruning Algorithm of Decision Tree

[8] G.X. Li, F. Zhu, S.L. Zhang and Z. Cui, Research on Model of Multiple Decision Trees Fusion Based on Genetic Algorithm

[9] X.X. Li, Designition and realization of Web Parser XiaoQBot