An Analysis of Characters and Structures of Web Pages Based on Regular Expressions

Lei Xu
Faculty of Physics and Electronic Science
Hubei University
Wuhan, China
e-mail: ceifei@hubu.edu.cn

***Abstract*—This paper introduces a method to analyze characters and structures of web pages via regular expressions. From encoding to HMTL elements, characters in Web pages are counted one by one. The effectiveness of this tool is proven in experiments with more than one hundred real-world web pages. All work can be ready for massive web information extraction.**

information extraction; HTML; regular expressions

I. INTRODUCTION

Web pages are plain text documents written in HTML. The syntax follows the W3C (World Wide Web Consortium) HTML technical documents, which latest version is HTML 5[1]. Web information extraction requires well-formed HTML documents, but there are lots of syntax errors and structural defects in real-world pages because of the loose nature of the HTML syntax itself. So it is very important to web information extraction to preprocess web pages.

Regular expressions is the key to powerful, flexible, and efficient text processing. Regular expressions themselves, with a general pattern notation almost like a mini programming language, allow you to describe and parse text [1]. It is the mainstream extraction rule type of tools for information extraction (IE). About 80 percent (15 of 19) adopts regular expressions to parse semi-structured inputs, especially template-based pages [2].

Our focus is on the flexibility of extraction rules. The existing proposals work on one or more input web document and search for repetitive structures that hopefully identify the regions where the relevant information insides [3]. But the structures of documents varies enormously in a real-world application. Extraction rules must be adjusted to the target pages.

In this paper, we introduce regular expressions based on Unicode character properties, which are very suited to process multiple languages and multiple domains. The details of structures of Web pages are discussed in Section II. Through experiments in Section III, the effectiveness of this method will be clear.

II. HTML DOCUMENT STRUCTURE

HTML is the core language of Web and HTML documents are written in HTML. According to the W3C specification, an HTML document must consist of the following parts, in the given order:[2]

1. Optionally, a single U+FEFF BYTE ORDER MARK (BOM) character.
2. Any number of comments and space characters.
3. A DOCTYPE.
4. Any number of comments and space characters.
5. The root element, in the form of an html element.
6. Any number of comments and space characters.

Here BOM is actually a Unicode character and can be detected by regular expressions via the property in Unicode Character Database (UCD)[5]. After removing BOM and redundant space characters, there are only DOCTYPE, elements and comments left in a document.

Note that all these components are wrapped by a pair of angle brackets (<>).

A. DOCTYPE and Comments

A DOCTYPE determines the type of the document and ensures that the browser makes a best-effort attempt at following the relevant specifications.

One document only can have one DOCTYPE, which must come in front of an html element. The content of a DOCTYPE must begin with an exclamation mark (!) and then follow the string "DOCTYPE" (usually capitalized). These is no non-ASCII characters in a DOCTYPE. The regular expression of a DOCTYPE can be designed like this: `<!DOCTYPE(.*?)>`.

Unlike a DOCTYPE, a comment can be inserted anywhere in a document and any character can be accepted in a comment except for another comment. The content of a comment must be wrapped by two continuous hyphens (--) and do not accept the string "-->" in order to prevent nested comments. The regular expression of a comment should be like this: `<!--(.*?)-->`.

B. Elements

Elements in HTML can be divided into normal ones and void ones. Most elements are normal, which means children

[1] http://www.w3.org/TR/html5/

[2] http://www.whatwg.org/specs/web-apps/current-work/multipage/syntax.html#writing

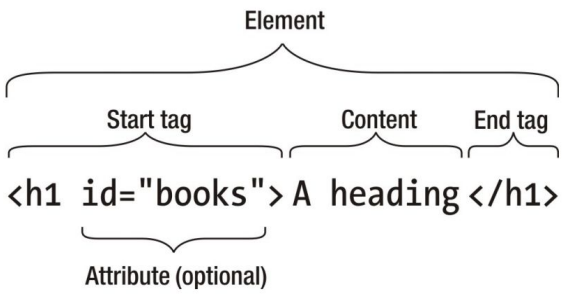

Figure 1. HTML element structure.

and end tags existed. A typical example of normal HTML element is shown in Fig. 1.[6]

A normal HTML element usually consist of four parts: start tag, attributes, content and end tag, of which attributes and content are optional. The start tag (also called open tag) contains the name of the element, surrounded by angle brackets.

Except for headings (h1-h6), the name of elements is made of pure Latin letters and is less than 10 characters. So the accurate regular expression for element names is as follow: `h[1-6]|[a-z]{1,10}`.

The end tag is similar to the start one and must have a same name with the start if it exists. To distinguish the end tag from the start tag, the end tag has a slash after the opening angle bracket (in the form </element>). There are no attributes in the end tag. In the case of excluding attributes and children, the normal elements can be expressed as follow:

```
<(?P<name>h[1-6]|[a-z]{1,10})>
([^<>]*)
</(?P=name)>
```

The regular expression "`(?P=name)`" in the end tag is a back-reference via captured name, meaning the same to the part of "name".

Unlike normal elements, void elements, also called empty elements, have no children and end tags. Even though accepted in HTML 5, self-closing slashes in the start tag are not necessary. They are not many, so can be inserted into a regular expression directly: `<(area |base |br |col |command |embed |hr |img |input |keygen |link |meta |param |source|track|wbr)/?>`. This regular expression will match exactly void elements without considering attributes.

C. Attributes

The structure of attributes is complicated. Attributes only can exist in the start tag, separated by spaces. Each attribute have a name and may have a value that may be in apostrophes (') or double quotes ("). There are four possible forms of an attribute:

```
1. id="books"
2. id='books'
3. disabled
4. border=1
```

The last form is often considered as valid, but actually should not be used. Because, if so, the boundary between names and values is not clear and it is difficult to exactly split the previous value and the next name if used. All attributes in Form 4 will be converted to Form 1 by default in Section III.

For Form 1, the value of an attribute itself could not contain any double quotes, that is the following is not allowed: `id="I met a man, named "Sherlock Holmes"."`.

On the same, the value in Form 2 could not contain any apostrophes. In the last form, both double quotes and apostrophes are not allowed as well as spaces. To summarize, the regular expression for an attribute can be coded like this:

```
([\x20][a-z][\w]*(=("[^"]*"|'[^']*'))?)*
```

D. Children and Content

Elements can be and usually be nested by each other. One element (normal or void), even a span of text could be a child of another normal element.

Elements should be nested properly so that overlapping does not occur. Not surprisingly, the most difficult thing in parsing HTML documents is arbitrary nesting. A powerful feature in regular expressions is recursive expressions, which is designed to matching nested constructs arbitrarily.

The sequence "`(?R)`" means "recursively apply the entire expression at this point," while "`(?num)`" sequence means "recursively apply the sequence within the number ed set of capturing parentheses at this point." The named-capture version of the latter uses a "`(?P>name)`" notation. [1]

With recursive expressions, arbitrary nested elements can be indicated as follow. Attributes are removed for the sake of clarity:

```
<(?P<name>h[1-6]|[a-z]{1,10})>
      ((?R)*)
</(?P=name)>
```

E. Foreign Elements

HTML is an open language and other available languages can be inserted as a part of documents. For example, an SVG image can be embedded between the <svg> and </svg> tags like this [6]:

```
<svg xmlns="http://www.w3.org/2000/svg">
   <rect
   stroke="black"
   fill="blue"
   x="50px"
   y="50px"
   width="300px"
   height="150px"
   strokewidth="2">
</svg>
```

In the case like this, names excluded in HTML5 may be used in element names, so the regular expression should evolve to like this: `[a-z][\w]*`. It may be not accurate, but more compatible and practical.

F. Special Elements

Two kinds of elements are so special that we have to an extra discussion about them [1]. A <script> section is important because it may have raw '<' and '>' within it, and so is the element of style. In principle, other elements cannot contain raw angle brackets in their content. Scripts and styles in HTML documents can be represented as follow:

```
<(script|style)([^>]*)>(.*?)</\1>
```

At last, An complete regular expression for elements can be described as follow:

```
<(script|style)([^>]*)>(.*?)</\1>
|
<(area|base|br|col|command|embed|hr|img|
input|keygen|link|meta|param|source|trac
k|wbr)
(([\x20][a-
z][\w]*(=("[^"]*"|'[^']*'))?)*
)/?>
|
<(?P<name>[a-z][\w]*)
(([\x20][a-
z][\w]*(=("[^"]*"|'[^']*'))?)*
)>((?R)*)</(?P=name)>|[^<>]+
```

III. EXPERIMENTS

To test the regular expressions in Section II, we sampled a set of pages from a variety of websites home and abroad. To be broadly representative, multiple languages, such as Chinese, English, Japanese, French, German, Spanish, Italian, Russian, Arabic and Sverige, are included. On the other hand, multiple domains are also involved such as education, government, news, video, sports, SNS, software, stock etc.

These documents may not be encoded by UTF-8, but all translated into it at the very beginning. After cleaning and standardization, the characters of the documents are analyzed in detail. And then the accuracy of recognition to tags and elements is computed by particular programs. In these experiments, all programs are written in PHP. Fig. 2 shows the basic flow of the test.

A. Data

For an empirical study, we collected 113 web documents from 104 websites[3], which cover the most common domains of lives. Our collection includes some famous search engines, such as Google and Baidu. Instead of boring home pages, we chose the pages of search results by the keyword "apple" for these search engines. Other documents are all home pages.

B. Procedure

1) Encoding Translation

All documents are detected by the function `mb_detect_encoding()`[4] with an encoding list "UFT-8, CP936, GB18030, ISO-8859-1" and strict mode. All non-UTF-8 documents are translated into UTF-8 by the software of Notepad2 because of supporting to substantial encodings.

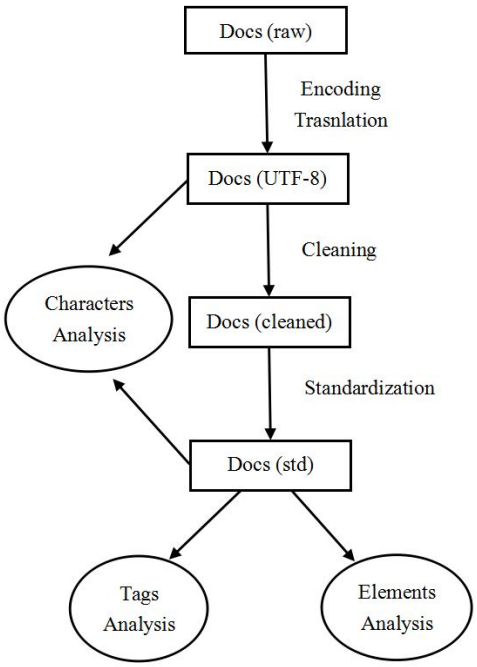

Figure 2. Flow of experiments.

The number of characters of documents will keep the same after the conversion, but the bytes will change.

2) Cleaning

Remove control characters and separators. Carriage returns (\r), line feeds (\n) and vertical tabs (\t) in Category C and spaces (\x20) in Category Z in UCD need to be reserved temporarily, and other characters in these two Categories will be removed. Notably, there is a modifier '`u`' in the regular express, meaning that pattern strings are treated as UTF-8[5]. Otherwise, symbols such as "`\pL`" could not be recognised.

Process Newlines and Vertical Tabs. Generally Speaking, newlines and tabs are meaningless in an HTML document, but it does not mean they can be removed directly. Consider the case that there are not spaces but newlines or tabs existed between the tag name and an attribute. If removing these control characters, the name of the tag will change and cause unexpected errors. But if not remove, as a matter of fact, these codes can be interpreted correctly by any real-world browser. So it is necessary to convert newlines and tabs to spaces before further processing.

Remove redundant spaces. According to the HTML specification, two or more continuous blank spaces are totally equivalent to just one in effect, so need to be replaced. Meanwhile, the leading and trailing whitespaces in a document are also useless, should removed.

3) Standardization

[3] http://ms.n.blog.163.com/blog/static/18595352014087593237/

[4] http://cn2.php.net/manual/en/function.mb-detect-encoding.php

[5] http://cn2.php.net/manual/en/reference.pcre.pattern.modifiers.php

The looseness of HTML makes Web pages different from XML documents, good for designers and bad for programmers. The number of syntax errors in real-world HTML documents are so huge that we have to fix them up as possible. Most errors are unexpected and often come from the last failure.

The most sophisticated structure brings the most difficult problem. Attributes are misused hardly in every document we collected and most problems are related with them.

4) Characters Analysis

As for characters, what we want to know is the variation after cleaning and standardization. According to General Category property, Every character in documents is counted by codes like this:

```
$str=preg_replace('/\pL/u','',$str,-1,$n);
```

5) Tags & Elements Analysis

Based on the standard version of documents, we grouped tags by name and counted them separately. After replacing all tags, the rest string was examined by a generic regular expression (`<[^<>]+>`), which can make sure that there are no angle brackets left.

The analysis of elements are very similar to the previous except including children or content. All normal elements are considered as a whole that can contain another element or some raw text.

C. Results

Tab. I reports on the character compositions of collected data in experiments. The criteria of classification are character properties in UCD. Surprisingly, there are some other characters in documents, which are not realized by people before. They are unprintable and meaningless to the structure of a document. It is impossible to typewrite them via a keyboard. The existence of these characters indicates that some programs could accidentally make pieces of normal characters, which are ignored by programmers. The significance of cleaning and standardization is shown in Tab. II. All newlines and vertical tabs and 56.74% whitespaces are removed. In total, about 13 percent of characters are saved.

TABLE I. STATISTICS OF CHARACTERS

Total	Letter	M	Num	Punc	Sym	\n	\r	\t	Space	Other
12165509	7003418	107	850496	1858625	774666	152432	60695	151624	1311359	2087

TABLE II. COMPARISON BETWEEN RAW AND CLEANED

	\n	\r	Tabs	Spaces	Other	Total
Raw	152432	60695	151624	1311359	2087	12165509
Cleaned	0	0	0	567320	0	10546593
Cut-off(%)	100	100	100	56.74	100	13.31

TABLE III. STATISTICS OF TAGS

pages	DT	Comment	Raw	Empty	Normal	Left	Acc. (%)
113	111	3982	2487	7542	201985	267	99.877

Tab. III shows the performance of regular expressions while extracting tags in documents. The first column lists the number of documents. The second column reports on the number of DOCTYPEs (DT) followed by comments (C), raw elements (R, including script, style and textarea), Empty elements (E), normal elements (N) and unavailable elements (L). The last column shows the accuracy (Acc.) of extraction, which is calculated in (1).

$$Acc = \frac{DT + C + R + E + N}{DT + C + R + E + N + L} \quad . \tag{1}$$

IV. CONCLUSION

A new Web information extraction method based on regular expressions is proposed to extract the basic structure of HTML documents including characters, tags and elements. The accuracy is 100% for characters, more than 99% for tags and more than 98% for elements. Considering solvable problems existed, there is still room for the accuracy to advance.

The extraction rules in the method are not unalterable. On the contrary, more different rules are encouraged to use when facing different tasks. In addition, two third-party tools can function together: HTML tidy [3] and HTML Parser [7]. The former is a proposal that is intended to preprocess web documents by fixing their HTML code and converting it into XHTML. And the latter is a famous open source project to extract information from a Web page. The study will remain deeply involved in the future.

ACKNOWLEDGMENT

Thanks to Jeffrey Friedl, his outstanding book inspiring me. Thanks to my collogues Song Su, Zhonghan Zhou, and my student Jiqiang Li.

REFERENCES

[1] J. E. F. Friedl, Mastering Regular Expressions, 3rd ed., Sebastopol: O'Reilly Media, p. 1, 2006

[2] C. H. Chang, M. Kayed, M. R. Girgis, K. Shaalan, "A servey of Web information extraction systems", IEEE Transactions on Knowledge and Data Engineering (TKDE), pp. 1411-1428, October 2006.

[3] H. A. Sleiman, R. Corchuelo, "TEX: A efficient and effective unsupervised Web information extractor", Knowledge-Based Systems, pp.109-123, 2013

[4] J. Spolsky, "The absolute minimum every software developer absolutely, positively must know about unicode and character sets", http://www.joelonsoftware.com/articles/Unicode.html, October 2008

[5] J. D. Allen, The Unicode Standard Version 6.2, the Unicode Inc., 2012

[6] L. F. Sikos, Web standards: mastering HTML5, CSS3 and XML, Apress, p. 67, 2011

[7] X. Ji, J. Zeng, S. Zhang, C. Wu, "Tag tree template for Web information and data extraction", Expert Systems with Applications, pp.8492-8498, 2010