Design and Realization of the Xml Parser based on Parsing Approach of Delaying Extension and Reducing Redundancy

Xiaoxia Sun[1], Hui Zhao[2] and Wenjun Meng[3]

**[1] School of Mechanical Engineering, Taiyuan University of Science and Technology
Taiyuan Shanxi, 030024, China**

**[2] China Mobile Communications Corporation
Taiyuan Shanxi, 030024, China**

**[3] School of Mechanical Engineering, Taiyuan University of Science and Technology
Taiyuan Shanxi, 030024, China**

Abstract

DOM parsing approach will consume a lot of memory size when it parse large XML document. This paper proposes an improved method of DOM parsing approach--- parsing approach of delaying extension and reducing redundancy. This method reduces the size of the object created by delaying expanded document, whose purpose is to reduce the memory size used. At the same time it improves the performance of the system by reducing the redundancy of the string stored. After analysing the new algorithm, improvement on it by Hash table is used. It reduces process time and increases parsing efficiency of system further. This paper describes the new algorithm based on this method above and programs it using Delphi6.0. Seven different sizes of XML document were tested based on the new algrithom and DOM parsing approach. The test results demonstrate this algorithm is feasible and effective.

Keywords: *XML Parser, DOM, delaying extension, reducing redundancy，Hash table*

1. Introduction

DOM is a standard API developed by W3C for browsing the XML document. It not only provides a complete representation for XML document stored in the memory, but also provides the method to access to the entire document randomly. The user can regard document as a structural information tree by DOM. The parser of the XML document designed by DOM approach enable developers to use the document Information repeatedly. However, the consumption of memory size is very impressive when the document is very large. In order to reduce the memory size, we must improve existing DOM parsing approach.

When parse an XML document using DOM parsing approach, all documents node are regarded as an expanded tree structure in memory. And usually the TEXT part of the node accounted for a lot of memory size in the XML document. Therefore we should study the TEXT part of the node in order to reduce the memory size. The approach taken in this paper is the TEXT part of the node is not expanded in the tree structure in memory, instead some index. These indexes are the index number of the TEXT part of the node in an array. At the same time we use the method of reducing data redundancy, which use the same index when the contents of the node are the same.

2. Design Idea and Realization of Algrithm

2.1 Design Idea

During the scanning process, each part of the document is expanded into a document tree in memory (except for the TEXT part of the document), which is similar to the DOM parsing approach. The TEXT part of the document is saved in an dynamic array, and the index number of the TEXT part in dynamic array is recorded in the document tree. In the dynamic array the same TEXT value only occupies one memory space. The same TEXT value only use an index number in the document tree expanded, which reduce the data redundancy and the memory size.

Example 1: An XML document

```
<?xml version="1.0"?>
<a>
  <b>China</b>
  <c>America</c>
 <d>Britain</d>
 <e>
   <f>China</f>
```

```
    <g>America</g>
    <h>France</h>
    <i>America</i>
  </e>
    </a>
    </xml>
```

The result of parsing the document above using new algorithm is as follows:

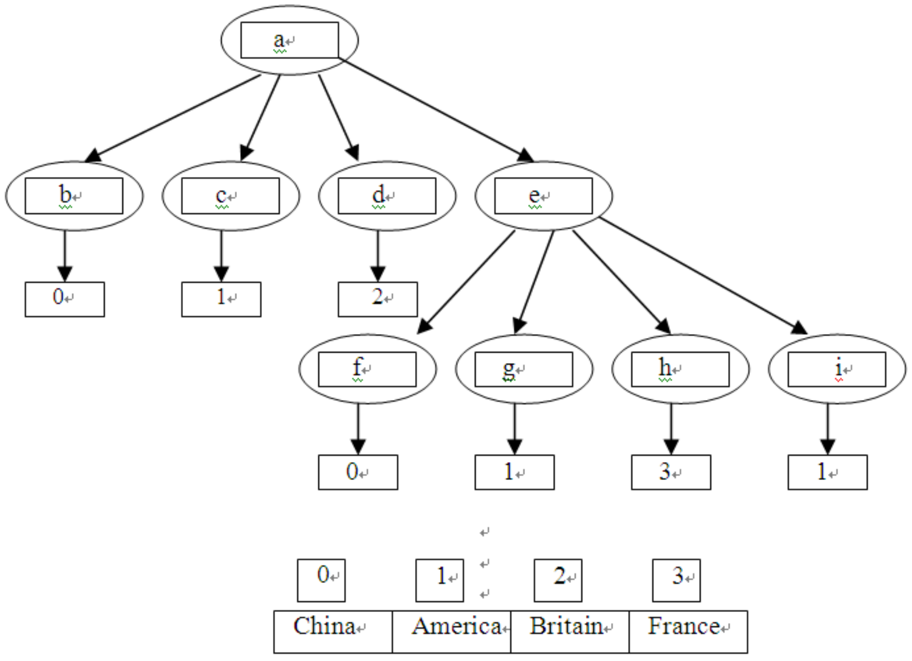

Fig.1 Expanded tree and dynamic array for document 1 in memory

2.2 Implementation of Algorithm

(1)Parsing algorithm of the XML document
①Create a dynamic array, and parse on the basis of DOM parsing methods.
②When the parsing program scans TEXT part, allocating a space for dynamic array to storage TEXT value. Then index number of array obtained is added to the DOM tree.
③When it scans the next TEXT part, firstly in dynamic array the parsing program queries whether the TEXT value has the same value as one of array. If so, the index number of array obtained is added to the DOM tree. If not, go to step 2.

LEX and YACC are design tools for the compiler and interpreter. They are used in this paper in order to it easier for realization of parsing algorithm.
The corresponding procedure is as follows:

Procedure 1: LEX program (Lexical analysis):

```
%%

  [\t ]+        ;
  \n       lineno:=lineno+1;
  \<        return(ZKH);
  [a-zA-Z]+  begin
             s:=yytext;
             return(ZF);
             end;
  \>        return(YKH);
  \/        return(XG);
%%
```

Procedure 2: YACC program (Grammatical analysis):

```
%token  ZKH ZF YKH XG
%%
  arule: rule1 arule rule3
  |arule rule1 arule rule3
  |arule rule1 rule2 rule3
|rule1 rule2 rule3
 ;
 rule1: ZKH ZF YKH
        {new(w);
        w^:=Txml.create;
        w^.name:=s;
        w^.left:=NIL;
        w^.right:=NIL;
        w^.parent:=NIL;
        if s='xml' then p:=w
```

```
            else if p^.left=NIL then
            begin
            p^.left:=w;
            w^.parent:=p;
             p:=p^.left;
            end
            else begin
            h:=p^.left;
            while(h^.right<>NIL) do
            h:=h^.right;
            h^.right:=w;
            w.parent:=p;
            p:=w;
            end;
            };
rule2: ZF
          {if length(arry)>0 then
            begin
            for i:=0 to high(arry) do
            if arry[i]=s then
            begin
            flag:=1;
            p^.data:=inttostr(i);
            break;
            end;
            if flag=0 then
            begin
            count:=count+1;
            SetLength(arry,count);
            arry[count-1]:=s;
            p^.data:=inttostr(i);
            end;
            end;
            else begin
            count:=count+1;
            SetLength(arry,count);
            arry[count-1]:=s;
            p^.data:=inttostr(i);
            end;
            };
rule3: ZKH XG ZF YKH
          {if p^.parent<>NIL then p:=p^.parent;}
            %%
```

(2) Query algorithm of XML document

Procedure 3: query algorithm

Input (a, s): “a” is pointer of pointing at root node of document tree generated after parse. “s” is the name of the document element to query.

Output: Display the element name and the element content in the ListBox of the window.

```
procedure find(a:point; s:string);
    begin
    if a<>NIL  then
        {if  a^.name=s  then
        // If the name of the node which the pointer pointed to
is the same as the string to query
             {if a^.data<>' '  then
               { form1.ListBox1.Items.Add('<'+s+'>') ;
                 form1.ListBox1.Items.Add(arry[strtoint(trim(a.da
                 ta))]);
                 form1.ListBox1.Items.Add('</'+s+'>');}
               if a^.data=' '  then
                 { pp:=a; def(a);};
        }
         find(a^.left,s); // Recursion to search the children node
that pointer pointed to;
         find(a^.right,s); // Recursion to search the brother
node that pointer pointed to;
    }
    end;
```

（3）Increase algorithm of XML document

Procedure 4: increase algorithm

Input(a,s,s1,s2): “a” is pointer of pointing at root node of document tree generated after parse. “s” is the name of document element. “s1” is the element name to be added. “s2” is element value to be added.

Output: Add a brother node to element node of document tree specified in memory.

```
Procedure add (a:point;s:string;s1:string;s2:string);
    var i: integer;
    begin
      if a<>NIL then
    { if a^.name=s then
   // If the node name which the pointer pointed to is the
same as the node name to be added after it.
         {new(xin); xin^:=Txml.create;
          xin^.name:=s1; flag:=0;
          if length(arry)>0 then
            { for i:=0 to high(arry) do
         If arry[i]=s2 then {flag:=1;xin^.data:=inttostr(i);
                 break;}
              }
       if flag=0 then
       {count:=count+1; SetLength(arry,count);  arry[count-
         1]:=s2;
              in^.data:=inttostr(count-1);
          }
    }
          xin^.left:=NIL; xin^.parent:=a^.parent;
      xin^.right:=a^.right; a^.right:=xin;ww:=1;
        }
        if ww=0 then
        //Ensure that document only add a node Everytime.
      { add(a^.left,s,s1,s2); // Recursion to add a children node
  that pointer pointed to;
        add(a^.right,s,s1,s2);
      }
      }
   end;
```

（4）Modification algorithm of XML document
Procedure 5: modification algorithm
Input(a,s,s1): “a” is pointer of pointing at root node of document tree generated after parse. “s” is the element name to be deleted. “s1” is altered contents.
Output: Alter element contents to element node of document tree specified in memory.

```
procedure modi(a:point;s:string);
   begin
  if a<>NIL then
   {if a^.name=s then
     {if a^.data<>'' then
           {arry[strtoint(trim(a^.data))]:=trim(
            form1.Edit2.Text); ww:=1;
        }
     }
     if ww=0 then
     {modi(a^.left,s); // Recursion to alter the
children node that pointer pointed to;
       modi(a^.right,s);      }
     }
     end;
```

(5) Serializable output algorithm of XML document
Procedure 6: Serializable output
Input (a): “a” is pointer of pointing at root node of document tree generated after parse.
Output: Export document tree in memory as XML document.

```
procedure abc(a:point);
   begin
      assignfile(text1,'a.txt'); append(text1);
      writeln(text1,'<'+a^.name+'>');
   closefile(text1);
   if a^.left<>NIL then abc(a^.left);
   if a^.left=NIL then
         {assignfile(text1,'a.txt'); append(text1);
          writeln(text1,arry[strtoint(trim(a.data))]);
          writeln(text1,'</'+a^.name+'>');closefile(text1);
      }
     if a^.right<>NIL then abc(a^.right);
     if (a^.right=NIL) then
            {assignfile(text1,'a.txt'); append(text1);
        if a^.parent<>NIL then
                 writeln(text1,'</'+a^.parent^.name+'>');
         closefile(text1);
        }
     end;
```

2.3 Analysis of Algorithm

Although the new algorithm has advantage in memory consumption, but there will be additional costs in parsing process. In parsing process, when we read an element value, first in dynamic array to find out whether them have the same value. If so, we only add index value found in the document tree. If not, we assign a space for dynamic array, and add element value in it. Then the corresponding index number is added in the document tree. When the XML document is very large, dynamic array will be numerous. At this time we need to compare with all dynamic array value to find out whether one of them has the same value as the element value being read. This will be more time consumption.

3.Optimization OF Algorithm

3.1 Optimization Idea

The ideal situation is that we get the record queried only one access without comparison. We should build a certain corresponding relationship f between store location of record and its key words. We can find MAP f (K) to given value K according to the corresponding relationship when we search. If the key word of record is the same as K in structure, the store location must be location of f (K). Consequently, we can get the record queried directly without comparison.

We call the corresponding relationship f for Hash function, and the table created according to the idea for Hash table.

3.2 Construction of Hash Function

For XML document, the element value of document may be long or short. It also may be digit, letter, Chinese characters or combination of them. In this optimization we use resulting “middle square method”. For XML document, the key word we take is divided into three parts: primacy of string, neutral position of string, end position of string. Take out the value of the three parts, and the average value got is regard as the key word. The algorithm of square to the key word is as follows:
Input(SJ , M): “ SJ ” is the element value of document, that is string; “ M ” is node counts of leaves that Documents prior to traverse.
Output(AJ): address AJ of string SJ in Hash table.

(1) Take out the internal code of initiatory 5 bytes of string to SJ_1 , take out the internal code of

intermediate 5 bytes of string to SJ_2, take out the internal code of final 5 bytes of string to SJ_3;

(2) $SJ = \frac{1}{3}\sum_{i=1}^{3} SJ_i$;

(3) Take out Intermediate $\sqrt[2]{M}$ from SJ^2 to AJ.

3.3 Approach of Processing Conflict

We may get the same Hash address for different key words. In general, the conflict only reduces as much as possible. But it cannot avoid completely. So how to deal with conflict is essential to structure hash table.

We use the "linear detection to hash method" in "open address method" in order to resolve conflict. The reference formula is as follows:

H =(H(key)+ d_i)MOD m, i=1,2,⋯,k(k≤m-1)

Among them, "H(key)" is Hash function. "m" is length of Hash table. "d_i" is Incremental sequence.

3.4 Algorithm Analysis with Hash Table in Parsing

The search by Hash table in parsing document may have comparison because of production of conflict. But the comparison of the average search length can be limited within limits. And direct map have been build between most of key words and store location of record. They don't need compare. So Hash table can reduce comparand of search in parsing XML document and reduce the cost of time. We only need to structure function in search by Hash table approach, which have nothing to do with the length of Hash table. It's time complexity is O(1), which is far less than parsing approach of delaying extension and reducing redundancy by dynamic array , whose time complexity is O(n).

4. THE RESULT OF THE EXPERIMENT

In this paper, we design a XML parser based on parsing approach of delaying extension and reducing redundancy. In order to test the merit of new approach, we program DOM parsing approach and improved parsing approach respectively using Delphi6.0, and seven sets of data are contrasted. The results are as follows:

Table1: Parsed numerical results for seven document

Size of the XML document	32.9kb	230kb	1147kb	2304kb	4608kb	9226kb	18432kb
Memory size occupied based on DOM approach	3160kb	3460kb	4912kb	6712kb	10316kb	17520kb	31952kb
Memory size occupied based on new approach	3148kb	3284kb	3968kb	4800kb	6524kb	9944kb	16732kb

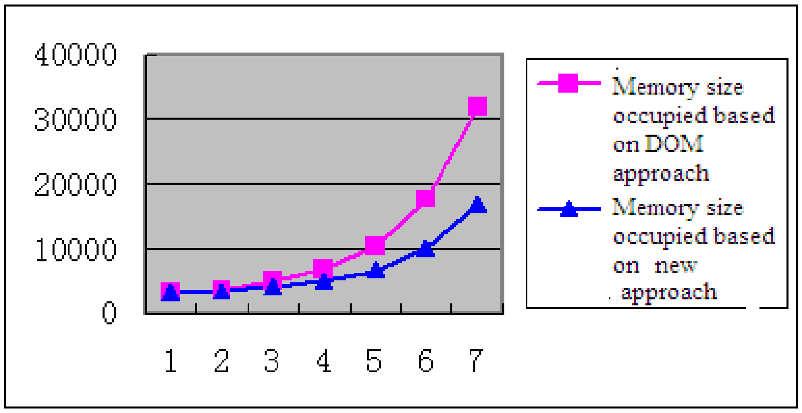

Fig.2 Contrast diagram of two kinds of parsing approach

It can be seen from table 1 and fig.2 that new approach has obvious advantage over DOM approach, especially in the use of memory resources. As is shown in fig. 2, Memory size occupied based on new approach is smaller than Memory size occupied based on DOM approach. When parsing document increases, grow rate of memory size based on new approach is significantly lower than ordinary DOM parsing approach.

5. Conclusion

(1)This paper proposes an improved method of DOM parsing approach--- parsing approach of delaying extension and reducing redundancy. It can resolve the question of consumption of a lot of memory size and reduce redundancy of system, which increase parsing efficiency of system.

(2)After analysing the new algorithm, improvement on it by Hash table is used. It reduces process time and increases parsing efficiency of system further.

(3)We program DOM parsing approach and improved parsing approach respectively using Delphi6.0, and seven sets of data are contrasted. The test results demonstrate this algorithm is feasible and effective.

6. Acknowledgment

This research was supported by the Shanxi Provincial Natural Science Fund under Grant No. 2011011019-3 and Foundation of School Younger Teachers in Taiyuan University of Science and Technology under Grant No. 20123001.

REFRENCES:

[1] ZhangWei, Van Engelen, Robert A, “High-performance XML parsing and validation with permutation phrase grammar parsers”，Proceedings of the IEEE International Conference on Web Services, ICWS 2008, pp. 286-294.
[2] Stanislav STANKOVIC, Jaakko ASTOLA, “XML Framework for various Types of Decision Diagrams for Discrete Functions”, IECE TRANS. INF. &SYST, Vol. e90-D, No. 11, 2007, pp. 1731-1741.
[3] Xubing, Liqiyan, Zhuqia, “Application analysis of XML parser”, Computer Systems & Applications, No. 1, 2002, pp. 30-32.
[4] Sunyizhong, “Basic theory and application of XML”, Press of Beijing University of Posts and Telecommunications, 2000.
[5] POleg Kiselyov, “A Better XML Parser through Functional Programming”, Proceedings of the 4th International Symposium on Practical Aspects of Declarative Languages, 2008, pp. 294-308.
[6] W. Zhang and R. van Engelen, “A Table-Driven Streaming XML Parsing Methodology for High-Performance Web Services”, Proceedings of the IEEE International Conference on Web Services, pp.197—204.
[7] Aswatha Kumar M., Selvarani R., T V Suresh Kumar, “An XML Parser of Efficient Updates for a Binary String: A Case, Study Proceedings of International Conference on Advances in Computing, Vol. 174, 2012.
[8] Haihui Zhang, Xingshe Zhou, Yang Gang, Xiaojun Wu, “An Index-Based XML Parser Model”, Network and Parallel Computing, Vol. 3779, No. 11, 2007, pp. 65-71.
[9] Caofenghua, “Study and Realization of an XML parser technology”, Microcomputer & Its Applications, Vol. 30, No. 11, 2011, pp. 6-10.
[10] Liuying, Wangsiliang, Xieyuemei, “Realization and study of XML parser C++ oriented”, Computer Application and Rearch, Vol. 24, No. 12, 2007, pp. 268-271.

Xiaoxia Sun born in 1979. She received her M.S. degree from North University of China in 2005, she is working towards a Ph.D. degree in Taiyuan University of Science and Technology. She is a lecturer at Taiyuan University of Science and Technology. Her main research interests include computer algorithm and conveying machinery.

Hui Zhao born in 1978. He received his M.S. degree from Shanxi University in 2005. He is working for China Mobile Communications Corporation.

Wenjun Meng born in 1963. He received his M.S. degree from Taiyuan University of Science and Technology, and his Ph.D. degree from Beijing Institute of Technology in 1990 and 2005 respectively. Currently, he is a professor at Taiyuan University of Science and Technology. His main research interests include Electromechanical integration and Continuous conveying machinery.