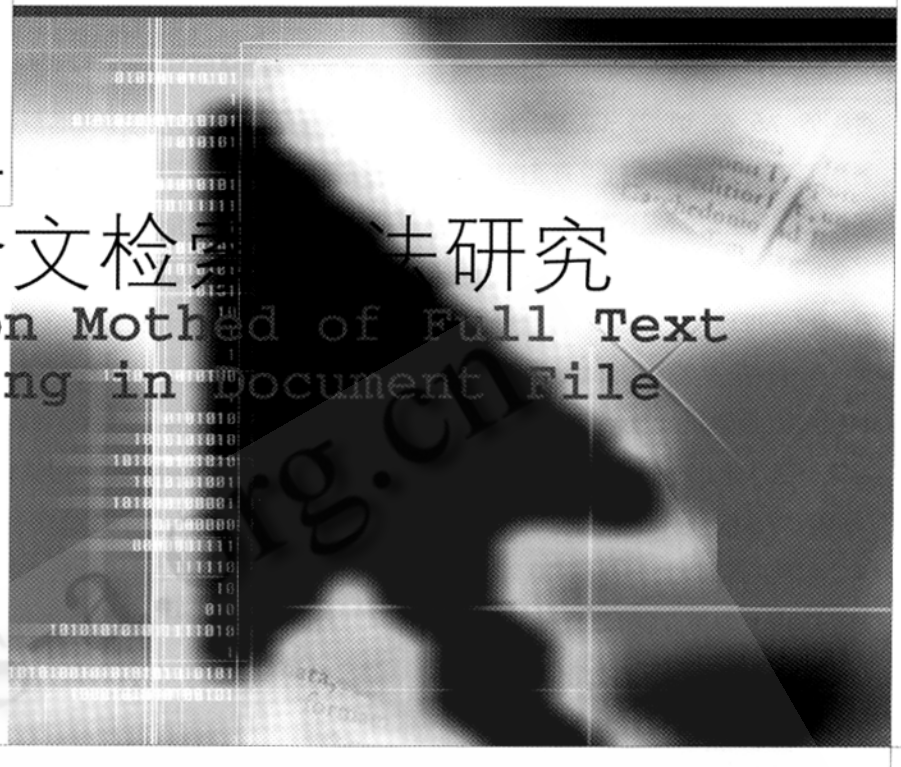
Word文件快速全文检索方法研究

Study on Mothed of Full Text Searching in Document File



宣士斌

（南宁广西民族学院现代教育技术中心　530006）

摘要：在分析Word6.0文件、Word97及Word2000文件、rtf格式文件、Uniocde文本文件等文件的存储结构的基础上，找到了这些文件中字符流的存储开始位置及字符流的长度存储单元及字符存储方式，实现了对上述文件的底层快速检索。

关键词：Word文档　RTF格式　字符流　存储结构

目前出现的检索工具主要是针对网页，其实网页是一种字符流式文件，即一般文本文件。对于一般文本文件的检索可以用任何一本《数据结构》中介绍的方法和一般文件操作即可。在日常生活中用得更多的可能是Word文字编辑软件。可是这类文件是有一定的结构，如何在这类有结构文件中进行快速检索就成了本文讨论的对象。本文主要讨论Word6.0文件、Word97及Word2000文件、rtf格式文件、Uniocde文本文件等文件的底层二进制检索。

1 Word6.0文件

一般来说，文件类型标识符都在文件偏移量0处，即文件开头处。所以首先必须将文件开头的文件类型标识符读出来，进行识别才能确定是什么类型的文件。Word6.0文档文件的文件类型标识符为{D0 CF 11 E0 A1 B1 1A E1}（为表达方便本文全部用十六进制表示）。

仅仅知道文件标识符，只能识别该文件是否是Word文件，关键是要知道Word文档文件的存储格式。非常遗憾的是Microsoft公司没有公开Word的存储格式，并且每次更新版本，存储格式都有所变化。虽然Microsoft在MSDN中提供了一些API接口函数，但操作比较麻烦，速度不可能快，对速度要求很高的检索来说，不很适用，这方面的例子很多，最好的方法应该是直接在底层打开文件，然后直接检索。我们知道Word文档有两个流，一个是控制流，另一个是字符流。由于本文只讨论对文本类文档的检索（内容可包括一般正文、表格、页眉页脚、批注、脚注尾注等），对象、链接等复杂结构不在考虑之列。由于Word文档的字符流与控制流是分开存放，用一般方法获取的文件长度并不是文件中字符流的实际长度，为了准确检索，获取文件实际文本长度很重要，所以我们只要找到字符流的开始位置及存储方式并且知道字符流的长度就可以实现对Word文档以二进制方式进行查询。由于Word6.0和Word97及Word2000的存储格式不同，所以我们要分开介绍。

Word6.0的字符流长度存放地址及开始位置随着字符流长度变化有所不同，字符流长度用4个字节存放，当字符流长度较短时，字符流长度存放地址在偏移地址081C；字符流开始位置的偏移地址0AFE+[0AFD]，当字符流长度较长时，字符流长度存放地址在偏移地址181C；字符流

的开始位置的偏移地址1AFE+[1AFD]，判断的方法是首先检查偏移地址0800处的8个字节值是否是{99、A6、65、00、23、C0、09、04}，若是则为短字符流文件，否则按长字符流处理。对长字符流，字符流有时不是连续存储的，当字符流长度小于0EF8时，字符流按顺序存储；当字符流长度大于0EF8而小于1EF8时，字符流分成两段，第一段存放在偏移地址1AFE+[1AFD]到2A00，第二段存放在偏移地址0800到1800。当字符流长度大于1EF8时，字符流分成三段，第一段存放在偏移地址1AFE+[1AFD]到2A00，第二段存放在偏移地址0800到1800，第三段存放在偏移地址2A00直至结束。

Word6.0的字符编码也不是一般的ANSI编码，不管是半角字符或是全角字符都用2个字节表示。具体情况是：对半角字符第一个字节取该字符的ASCII，第二个字节设为0；对全角字符或汉字第一个字节取该全角字符的ANSI编码的第二个字节，第二个字节取该全角字符的ANSI编码的第一个字节。

2 Word97及Word2000

Word97及Word2000文件的文件类型标识符与Word6.0相同，但字符流长度存放地址及开始位置和字符编码与Word6.0有很大的区别。Word97及Word2000文件的字符流长度存放地址及开始位置是固定的都在偏移地址021C处，但由于Word97及Word2000是采用压缩存储，所以在Word97及Word2000中字符流长度实际上有两个，一个是实际字符数（一个全角算一个字符），另一个是字符流实际占用空间长度。偏移地址021C处存放的是字符流实际占用空间长度，而在偏移地址024C处存放的是字符流实际字符数。我们只要比较这两个数就知道该文档是否有压缩存储。Word97及Word2000文件的字符流的开始位置是固定的都从偏移地址0600开始。在具体存储时，以200个字节为一段，根据具体情况有些段以压缩形式存储，有些段以非压缩形式存储。具体说，当一段内全是半角字符是用压缩形式存储，当某段内有全角字符（或汉字时）时，采用非压缩形式存储。由于Word控制流用复合形式存储，无法直接得到某段是否压缩存储的标志，但我们根据实际情况还是可以识别的。由于压缩存储是对ASCII半角字符而言，全角字符或汉字不存在压缩存储问题。某段是压缩存储，则该段内的字符必定全是ASCII半角字符，根据实际情况，该段必定是英文或数字，如果是全文，根据英文的书写格式，单词间用空格或标点符号分隔，则该段内肯定有空格或标点符号；若有数字，则必定有小数点、加号、减号。这样我们就可以根据半角标点符号和小数点、加号、减号及回车和换行符0D和0A的下一个字节是否是零就可以判断该段是否是压缩存储，如果是零则是非压缩存储，否则为压缩存储。但本文后面给出的程序只考虑非压缩存储形式，对压缩存储形式可以按上面所叙方法进行识别。

Word97及Word2000的字符编码与Word6.0的字符编码又不一样，对半角字符的非压缩存储与Word6.0相同，而压缩存储直接存储其ASCII；对全角字符或汉字以UNICODE编码形式存储，但在Windows中一般输入用的是ANSI编码，为了实现对Word文档的查询，必须实现两种编码之间的转换，为此我们建立了两种编码的字符对照表"anstouni.idx"。其实这种字符对照表很容易建立，方法是：按《GB80-2312》规范编一程序生成一文本文件记为a1.txt存放ANSI编码的所有字符，再用附件中的记事本将其打开，将所有字符复制到一Word空文件记为a1.doc中，并将该Word文件保存起来，再编一程序不断地分别从a1.txt和a1.doc中各取两个字节组成一个记录保存到"anstouni.idx"文件中，并将该文件按每条记录的第一个字段进行排序，即可得所需的编码字符对照表。

3 RTF格式文件

RTF文件有多种形式，但大体上可将其分为三大类，对应三种不同的文件类型标识码，它们分别是：1、{7B 5C 72 74 66 31 5C 61 6E 73 69 5C 61 6E 73 69 63 70 67 39 33 36}，2、{7B 5C 72 74 66 31 5C 61 6E 73 69 73 69 20 5C 64 65 66 30}，3、{7B 5C 72 7466 31 5C 61 6E 73 69 5C 64 65 66 66 30}。RTF文件的文件类型标识符在文件开始处。RTF格式文件是一种字符流式文件，所有的控制符号及字符都以ASCII码字符形式存储。所以字符流存储的开始位置和字符流长度都不好确定。但通过对文本进行分析，我们还是可以找到办法。对第一类文件从偏移地址0A00处找字符串"＼langfenp2052"，其后即为字符流的开始位置。由于RTF文件的控制符含在字符流中，所以在析取字符流时必须能识别哪些是控制符哪些是字符。其实并不难识别，在"＼f0"后面的一般是半角ASCII字符，在"＼'"后面的两个字符是全角字符码的一个字节码。由于RTF文件所有的控制符号及字符都以ASCII码字符形式存储。所以全角字符的存储码也必须要转换成ASCII码字符存储。例如字符串"文件"的一般ANSI存储码为"CE C4 BC FE"，而在RTF文件中的存储码为"63 65 63 34 62 63 66 65"，这正好是字符串"CEC4BCFE"的ASCII码。因此我们必须将这些ASCII码转换成对应全角字符或汉字的ANSI存储码。虽然RTF格式文件字符流长度不好确定，但却能找到字符流结束的标志符"＼par }}"且可以通过C语言提供的获取文件长度的函数获取文件长度，字符流的结束标志也在文件结束处，所以我们可以不通过结束标志判断，而直接通过文件长度判断是否该结束查找。对于第二类文件从偏移地址0900处找字符串"＼f6"或"＼f0"，其后即为字符流的开始位置。由于RTF文件的控制符也含在字符流中，所以在析取字符流时必须能识别哪些是控制符哪些是字符。在"＼f6"后面的一般是半角ASCII字符，在"＼'"后面的两个字符是全角字符码的一个字节码。由于RTF文件所有的控制符号及字符都以ASCII码字符形式存储。所以全角字符的存储码也必须要转

换成ASCII码字符存储。判断文件结束方法同上。对于第三类文件从偏移地址0200处找字符串"＼fs21"，其后即为字符流的开始位置。由于RTF文件的控制符也含在字符流中，所以在析取字符流时必须能识别哪些是控制符哪些是字符。在字符串"＼fs21"后面还没有出现字符"＼"、"{"、"}"前所有字符都是原文件中的半角ASCII字符。在"＼'"后面的两个字符是全角字符码的一个字节码。由于RTF文件所有的控制符号及字符都以ASCII码字符形式存储。所以全角字符的存储码也必须要转换成ASCII码字符存储。判断文件结束方法第一类。

在以上三类文件中都有一相同点，即是字符的控制符都是通过"＼"加上ASCII字符组成，用一对花括表示一串结构相同的字符序列。所以在RTF文件中对于原文件中的字符"＼"、"{"、"}"必须特别处理，处理方法是"＼＼"、"＼{"、"＼}"分别对应原文件中的"＼"、"{"、"}"。识别时必须专门判断。

4 Unicode文本文件

Unicode文本文件的处理其实很简单，它的文件类型标识符也在文件开始处，标识符内容为{FF FE}，标识符后即为字符存储的开始位置，所有字符按非压缩Unicode编码存储。具体方法和Word97及Word2000的处理方法一样。

5 总结

本文只是为文本检索分析了几种文件的部分存储格式，并给出了一般处理方法，实践表明，该方法十分有效。文后附有部分程序源代码，用VC++编制。

6 程序源代码

```
//Word97、Word2000处理方法
lstr=stringexchange1(tab,stri,pstring);//将待查串stri转成unicode串pstring
// 求出字符流长度
fp.Seek(0x21c,CFile::begin);fp.Read(lchar.s,4);filelength=lchar.l-0x400;
fp.Seek(0x600,CFile::begin);// 将文件指针移到字符流开始处
for(bts=0,i=0;bts<filelength;bts=bts+1024)
{ btst=fp.Read(&buff [i],(filelength-bts>1024)?1024:filelength-bts);
buff [btst+i] ='＼0';//从字符流中读取字符
i=lstr;
if(index(buff,btst+i,pstring,lstr)!=1)//在所读取字符串中查找
strcp(buff,&buff [btst-lstr+i],lstr);//若未找到将本次中尾lstr个字符
// 复制到串前以备下次查找
else
return 1;// 若找到则返回1
}
return 0;// 若全文中都未找到则返回0
// 第二类rtf文件处理方法
// 先将文件读入tb所指向的内存内
k=0;kt=0;
while(k<filelength-0xa00)
{if(tb [k++] =='＼＼')
{if(tb [k] =='f')
{k++;if(tb [k]=='0')
{k++;if(tb [k] ==0x20) // 判断是否是字符串"＼f0"
{ k++;// 以下进行ASCII码字符转存
while(1)
{if(tb[k]=='＼＼'&&(tb[k+1]=='{'||tb[k+1]=='}'||tb [k+1] =='＼＼'))
{k++;tb [kt++] =tb [k++];}// 识别是否是字符"＼"、"{"、"}"
else if(tb [k]!='＼＼'&&tb [k]!='}'&&tb [k]!='}'&&tb [k]!=0x0d&&tb [k]!=0x0a)
tb [kt++] =tb [k++];// 识别是否是一般字符
else if(tb [k] ==0x0d||tb [k] ==0x0a)// 识别是否是回车换行
{k++;continue;}
else break;
}}}}
else if(tb [k] =='＼")// 识别是否是全角字符码的开始标志
{k++;
for(i=0;i<2;i++) b0 [i] =tb [k++];
b0 [2] ='＼0';t=strtochar(b0);tb [kt++] =t;}// 全角字符码转换
else if(tb [k] =='＼＼'||tb [k] =='{'||tb [k] =='}') // 识别是否是字符"＼"、"{"、"}"
tb [kt++] =tb [k++];
else k++;}}
tb [kt] ='＼0';}
else return 0;
//以下程序完成在经转换的字符表tb中查找字符串变量pstring中的字符串
for(k=0,l=0;k<kt&&l<lstr;)
{if(tb [k] ==pstring [l])
{k++;l++;}
else{k=k-l+1;l=0;}}
if(l>=lstr) return 1;// 若找到则返回1
else return 0;// 若没有找到则返回0
```


参考文献

1 刘成，客户程序自动读写Word文档的实现[J]，计算机应用，2001，21(3):96-96。

2 李秀芹，朱跃龙，Microsoft Word文件转换器的设计与实现[J]，计算机应用，2002,22(1):37-41。

3 艾晓明，周定康，引用Microsoft Word对象的技术及实现[J]，计算机与现代化，2002(2):50-56。