ICTCLAS 分词系统研究（一） 收藏

ICTCIAS 分词系统是由中科院计算所的张华平、刘群所开发的一套获得广泛好评的分词系统，难能可贵的是该版的 Free 版开放了源代码，为我们很多初学者提供了宝贵的学习材料。

但有一点不完美的是，该源代码没有配套的文档，阅读起来可能有一定的障碍，尤其是对 C/C++不熟的人来说.本人就一直用 Java/VB 作为主要的开发语言,C/C++上大学时倒是学过,不过工作之后一直没有再使用过,语法什么的忘的几乎一干二净了.但语言这东西,基本的东西都相通的,况且 Java 也是在 C/C++的基础上形成的,有一定的相似处.阅读一遍源代码,主要的语法都应该不成问题了.

虽然在 ICTCLAS 的系统中没有完整的文档说明,但是我们可以通过查阅张华平和刘群发表的一些相关论文资料,还是可以窥探出主要的思路.

该分词系统的主要是思想是先通过 CHMM(层叠形马尔可夫模型)进行分词,通过分层,既增加了分词的准确性,又保证了分词的效率.共分五层,如下图一所示:

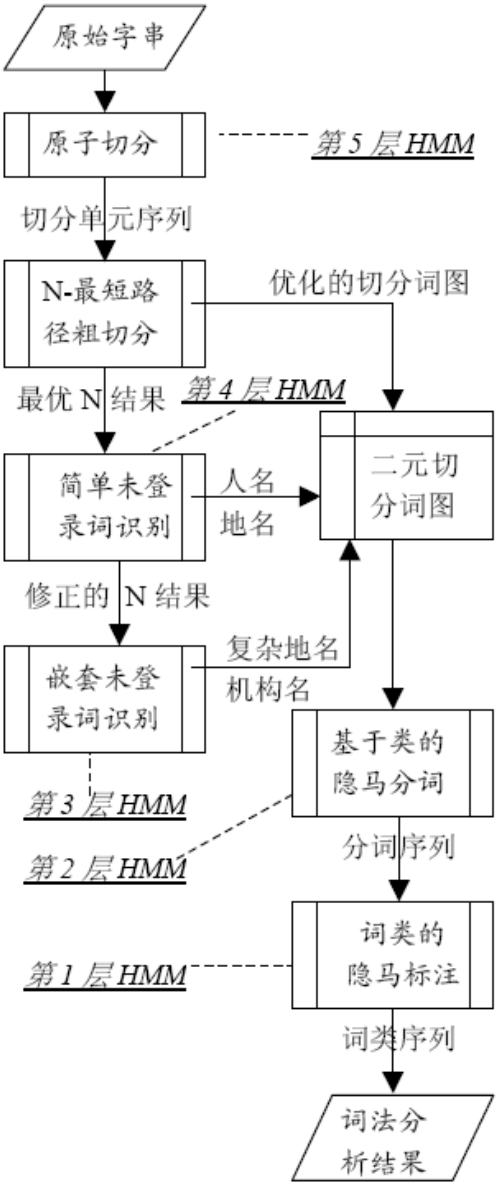


图 1. 基于 CHMM 的汉语词法分析框架

基本思路:先进行原子切分,然后在此基础上进行 N-最短路径粗切分,找出前 N 个最符合的切分结果,生成二元分词表,然后生成分词结果,接着进行词性标注并完成主要分词步骤.

下面是对源代码的主要内容的研究：

1.首先，ICTCLAS 分词程序首先调用 CICTCLAS_WinDlg::OnBtnRun()开始程序的执行.并且可以从看出它的处理方法是把源字符串分段处理。并且在分词前，完成词典的加载过程，即生成 m_ICTCLAS 对象时调用构造函数完成词典库的加载。关于词典结构的分析，请参加分词系统研究（二）。

```
void CICTCLAS_WinDlg::OnBtnRun()
{

   ......
```

```
//在此处进行分词和词性标记

  if(!m_ICTCLAS.ParagraphProcessing((char *)(LPCTSTR)m_sSource,sResult))
        m_sResult.Format("错误：程序初始化异常！");
   else
       m_sResult.Format("%s",sResult);//输出最终分词结果

    ......

}
```

2.在 OnBtnRun()方法里面调用分段分词处理方法 bool CResult::ParagraphProcessing(char *sParagraph,char *sResult)完成分词的整个处理过程，包括分词的词性标注.其中第一个参数为源字符串，第二个参数为分词后的字符串.在这两个方法中即完成了整个分词处理过程，下面需要了解的是在此方法中，如何调用其它方法一步步按照上图所示的分析框架完成分词过程.为了简单起见，我们先不做未登录词的分析。

```
//Paragraph Segment and POS Tagging
bool CResult::ParagraphProcessing(char *sParagraph,char *sResult)
{

    ........

   Processing(sSentence,1); //Processing and output the result of current sentence.
  Output(m_pResult[0],sSentenceResult,bFirstIgnore); //Output to the imediate result

   .......

}
```

3.主要的分词处理是在 Processing()方法里面发生的，下面我们对它进行进一步的分析.

```
bool CResult::Processing(char *sSentence,unsigned int nCount)
{

......

 //进行二叉分词

m_Seg.BiSegment(sSentence,
m_dSmoothingPara,m_dictCore,m_dictBigram,nCount);

......

 //在此处进行词性标注

m_POSTagger.POSTagging(m_Seg.m_pWordSeg[nIndex],m_dictCore,m_dictCore);

......
```

}

４.现在我们先不管词性标注，把注意力集中在二叉分词上，因为这个是分词的两大关键步骤的第一步.

参考文章:

1.<<基于层叠隐马模型的汉语词法分析>>,刘群 张华平等

2.<<基于 N-最短路径的中文词语粗分模型>>,张华平 刘群

ICTCLAS 分词系统研究（二）--词典结构 收藏

ICTCLAS 的词典结构是理解分词的重要依据，通过这么一个数据结构设计合理访问速度高效的词典才能达到快速准备的分词的目的。

通过阅读和分析源代码，我们可以知道，是程序运行初，先把词典加载到内存中，以提高访问的速度。源代码在 Result.cpp 的构造函数 CResult（）内实现了词典和分词规则库的加载。如下代码所示：

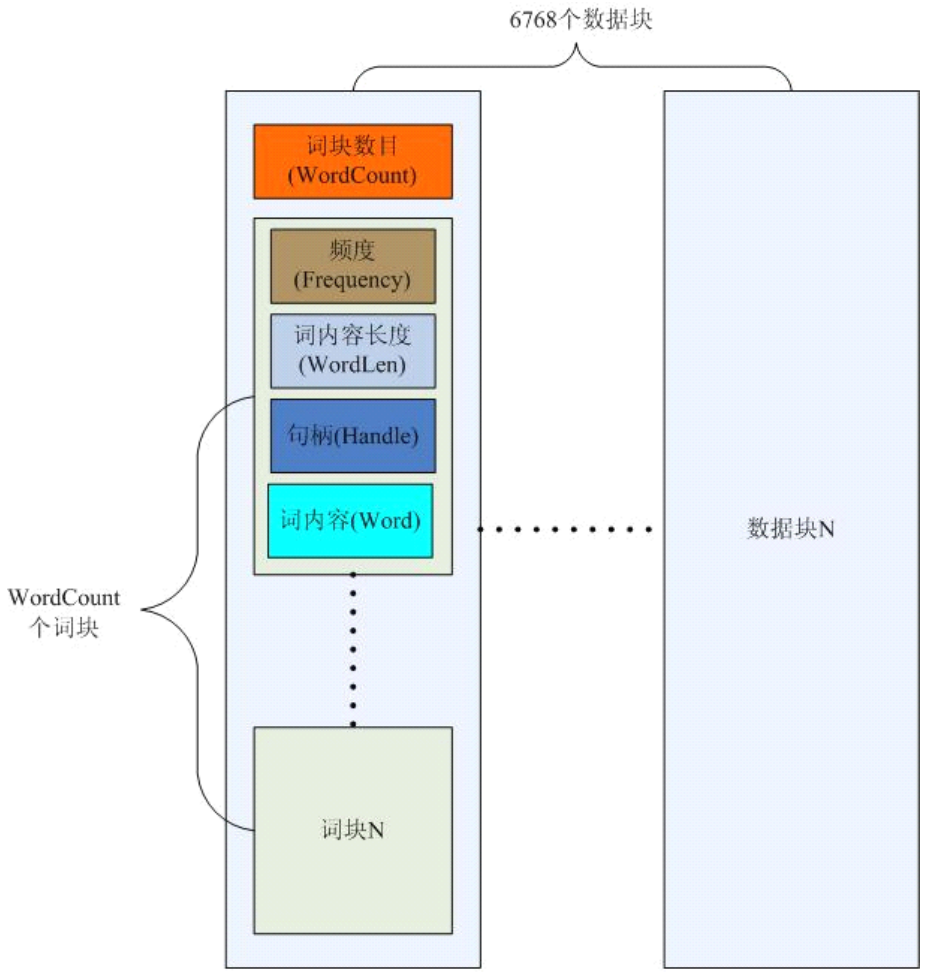
```
CResult::CResult()
{
 ……

 m_dictCore.Load("data\\coreDict.dct");
 m_POSTagger.LoadContext("data\\lexical.ctx");

……

}
```

我们再跳进 Load 方法具体分析它是怎样读取数据词典的,看 Load 的源代码:

```
bool CDictionary::Load(char *sFilename,bool bReset)
{
   FILE *fp;
   int i,j,nBuffer[3];

    //首先判断词典文件能否以二进制读取的方式打开
   if((fp=fopen(sFilename,"rb"))==NULL)
    return false;//fail while opening the file

  //为新文件释放内存空间
   for( i=0;i<CC_NUM;i++)
 {//delete the memory of word item array in the dictionary
  for( j=0;j<m_IndexTable[i].nCount;j++)
   delete m_IndexTable[i].pWordItemHead[j].sWord;
  delete [] m_IndexTable[i].pWordItemHead;
 }
    DelModified();//删除掉修改过的,可以先不管它

  //CC_NUM:6768,应该是 GB2312 编码中常用汉字的数目 6763 个加上 5 个空位码
   for(i=0;i<CC_NUM;i++)
   {

  //读取一个整形数字(词块的数目)
    fread(&(m_IndexTable[i].nCount),sizeof(int),1,fp);
      if(m_IndexTable[i].nCount>0)
```

```
            m_IndexTable[i].pWordItemHead=new
WORD_ITEM[m_IndexTable[i].nCount];
    else
    {
     m_IndexTable[i].pWordItemHead=0;
     continue;
    }
      j=0;

  //根据前面读到的词块数目,循环读取一个个词块
    while(j<m_IndexTable[i].nCount)
    {

       //读取三字整数,分别为频度(Frequency)/词内容长度(WordLen)/句柄(Handle)
        fread(nBuffer,sizeof(int),3,fp);
        m_IndexTable[i].pWordItemHead[j].sWord=new char[nBuffer[1]+1];

       //读取词内容
        if(nBuffer[1])//String length is more than 0
        {
          fread(m_IndexTable[i].pWordItemHead[j].sWord,sizeof(char),nBuffer[1],fp);
        }
       m_IndexTable[i].pWordItemHead[j].sWord[nBuffer[1]]=0;
       if(bReset)//Reset the frequency
         m_IndexTable[i].pWordItemHead[j].nFrequency=0;
       else
            m_IndexTable[i].pWordItemHead[j].nFrequency=nBuffer[0];
        m_IndexTable[i].pWordItemHead[j].nWordLen=nBuffer[1];
        m_IndexTable[i].pWordItemHead[j].nHandle=nBuffer[2];
        j+=1;//Get next item in the original table.
    }
   }
   fclose(fp);
   return true;
}
```

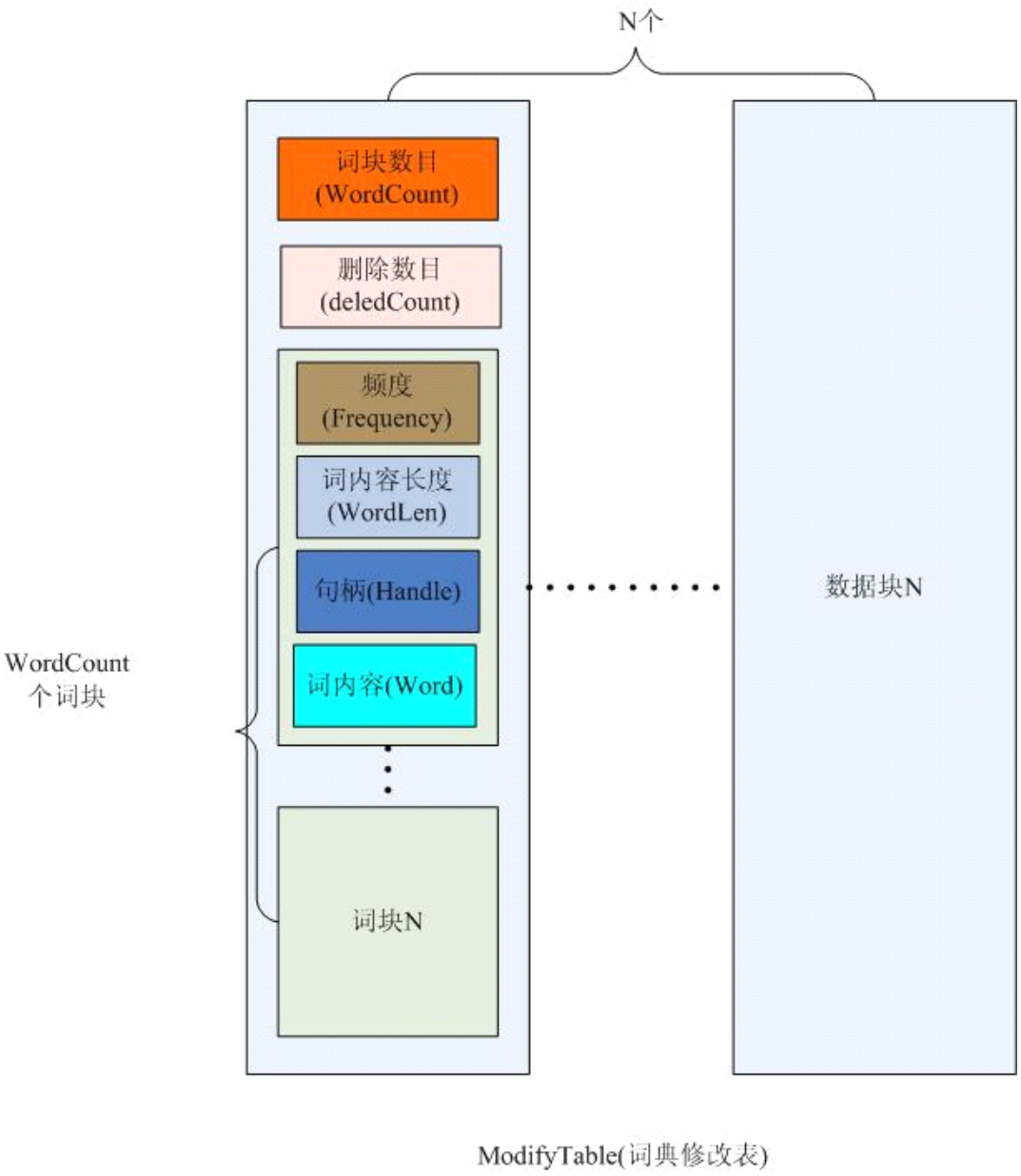
看完上面的源代码,词典的结构也应该基本清楚了,如下图一所示:



词典结构

图一

修改表的数据结构和上图差不多,但是在词块数目后面多了一个 nDelete 数目,即删除的数目,数据结构如下图二所示:



图二

GB2312(1980 年)一共收录了 7445 个字符，包括 6763 个汉字和 682 个其它符号。汉字区的内码范围高字节从 B0-F7，低字节从 A1-FE，占用的码位是 72*94=6768。其中有 5 个空位是 D7FA-D7FE。词典库图一所示的 6768 个块即对应 GB2312 编码中的这个 6768 个区位.图一中每一个大块代表以该字开头的所有词组,括号内的字为区位码对应的汉字,词典表中并不存在,为了说明方便才添加上去的.如下所示:

块 6759
count:5
wordLen:2 frequency:0 handle:24832 word:(黯)淡
wordLen:2 frequency:1 handle:24942 word:(黯)淡
wordLen:2 frequency:3 handle:31232 word:(黯)然
wordLen:6 frequency:0 handle:27648 word:(黯)然神伤

wordLen:6 frequency:0 handle:26880 word:(黯)然失色

块 6760

count:1

wordLen:2 frequency:0 handle:28160 word:(齖)鼠

块 6761

count:2

wordLen:4 frequency:0 handle:28160 word:(齸)鼠皮

wordLen:2 frequency:0 handle:28160 word:(齸)獾

对修改后如何保存的源代码进行分析:

```
bool CDictionary::Save(char *sFilename)
{
  FILE *fp;
  int i,j,nCount,nBuffer[3];
  PWORD_CHAIN pCur;
  if((fp=fopen(sFilename,"wb"))==NULL)
   return false;//fail while opening the file

 //对图一中所示的 6768 个数据块进行遍历
  for(i=0;i<CC_NUM;i++)
  {
   pCur=NULL;
   if(m_pModifyTable)
   {

   //计算修改后有效词块的数目

    nCount=m_IndexTable[i].nCount+m_pModifyTable[i].nCount-m_pModifyTable[i].
nDelete;
    fwrite(&nCount,sizeof(int),1,fp);
    pCur=m_pModifyTable[i].pWordItemHead;
    j=0;

   //对原表中的词块和修改表中的词块进行遍历,并把修改后的添加到原表中
    while(pCur!=NULL&&j<m_IndexTable[i].nCount)
    {

    //如果修改表中的词长度小于原表中对应位置的词的长度或者长度相等但 nHandle 值比原表中
的小,则把修改表中的写入到词典文件当中.

     if(strcmp(pCur->data.sWord,m_IndexTable[i].pWordItemHead[j].sWord)<0||(str
cmp(pCur->data.sWord,m_IndexTable[i].pWordItemHead[j].sWord)==0&&pCur->dat
a.nHandle<m_IndexTable[i].pWordItemHead[j].nHandle))
     {//Output the modified data to the file
```

```
nBuffer[0]=pCur->data.nFrequency;
      nBuffer[1]=pCur->data.nWordLen;
    nBuffer[2]=pCur->data.nHandle;
          fwrite(nBuffer,sizeof(int),3,fp);
if(nBuffer[1])//String length is more than 0
      fwrite(pCur->data.sWord,sizeof(char),nBuffer[1],fp);
          pCur=pCur->next;//Get next item in the modify table.
 }

//频度 nFrequecy 等于-1 说明该词已被删除,跳过它
 else if(m_IndexTable[i].pWordItemHead[j].nFrequency==-1)
 {
  j+=1;
 }

 //如果修改表中的词长度比原表中的长度大或  长度相等但句柄值要多,就把原表的词写入的词典文件中
 else
if(strcmp(pCur->data.sWord,m_IndexTable[i].pWordItemHead[j].sWord)>0||(strcmp(pCur->data.sWord,m_IndexTable[i].pWordItemHead[j].sWord)==0&&pCur->data.nHandle>m_IndexTable[i].pWordItemHead[j].nHandle))
 {//Output the index table data to the file
nBuffer[0]=m_IndexTable[i].pWordItemHead[j].nFrequency;
nBuffer[1]=m_IndexTable[i].pWordItemHead[j].nWordLen;
nBuffer[2]=m_IndexTable[i].pWordItemHead[j].nHandle;
fwrite(nBuffer,sizeof(int),3,fp);
if(nBuffer[1])//String length is more than 0
  fwrite(m_IndexTable[i].pWordItemHead[j].sWord,sizeof(char),nBuffer[1],fp);
j+=1;//Get next item in the original table.
 }
}

//把原表中剩余的词写入的词典文件当中
if(j<m_IndexTable[i].nCount)
{
 while(j<m_IndexTable[i].nCount)
 {
  if(m_IndexTable[i].pWordItemHead[j].nFrequency!=-1)
  {//Has been deleted
 nBuffer[0]=m_IndexTable[i].pWordItemHead[j].nFrequency;
 nBuffer[1]=m_IndexTable[i].pWordItemHead[j].nWordLen;
 nBuffer[2]=m_IndexTable[i].pWordItemHead[j].nHandle;
 fwrite(nBuffer,sizeof(int),3,fp);
 if(nBuffer[1])//String length is more than 0
   fwrite(m_IndexTable[i].pWordItemHead[j].sWord,sizeof(char),nBuffer[1],fp);
```

```
        }
       j+=1;//Get next item in the original table.
       }
      }
      else////原表已到尾部但修改表还没有遍历完,把修改表中剩余的词写入到词典文件当中
       while(pCur!=NULL)//Add the rest data to the file.
       {
      nBuffer[0]=pCur->data.nFrequency;
           nBuffer[1]=pCur->data.nWordLen;
        nBuffer[2]=pCur->data.nHandle;
                fwrite(nBuffer,sizeof(int),3,fp);
      if(nBuffer[1])//String length is more than 0
        fwrite(pCur->data.sWord,sizeof(char),nBuffer[1],fp);
                pCur=pCur->next;//Get next item in the modify table.
       }
     }

    //不是修改标记,则把原表的数据全部写入到词典文件当中
     else
     {
      fwrite(&m_IndexTable[i].nCount,sizeof(int),1,fp);
      //write to the file
           j=0;
      while(j<m_IndexTable[i].nCount)
      {
     nBuffer[0]=m_IndexTable[i].pWordItemHead[j].nFrequency;
       nBuffer[1]=m_IndexTable[i].pWordItemHead[j].nWordLen;
     nBuffer[2]=m_IndexTable[i].pWordItemHead[j].nHandle;
             fwrite(nBuffer,sizeof(int),3,fp);
     if(nBuffer[1])//String length is more than 0
       fwrite(m_IndexTable[i].pWordItemHead[j].sWord,sizeof(char),nBuffer[1],fp);
      j+=1;//Get next item in the original table.
      }
     }
    }
    fclose(fp);
    return true;
}
```

增加一个词条目:

```
bool CDictionary::AddItem(char *sWord, int nHandle,int nFrequency)
{
```

```
char sWordAdd[WORD_MAXLENGTH-2];
int nPos,nFoundPos;
PWORD_CHAIN pRet,pTemp,pNext;
int i=0;

//预处理,去掉词的前后的空格
if(!PreProcessing(sWord, &nPos,sWordAdd,true))
 return false;

 //查找词典原表中该词是否存在
if(FindInOriginalTable(nPos,sWordAdd,nHandle,&nFoundPos))
{//The word exists in the original table, so add the frequency
 //Operation in the index table and its items
    if(m_IndexTable[nPos].pWordItemHead[nFoundPos].nFrequency==-1)
 {//The word item has been removed
    m_IndexTable[nPos].pWordItemHead[nFoundPos].nFrequency=nFrequency;
     if(!m_pModifyTable)//Not prepare the buffer
 {
   m_pModifyTable=new MODIFY_TABLE[CC_NUM];
   memset(m_pModifyTable,0,CC_NUM*sizeof(MODIFY_TABLE));
 }
 m_pModifyTable[nPos].nDelete-=1;
 }
 else
 m_IndexTable[nPos].pWordItemHead[nFoundPos].nFrequency+=nFrequency;
 return true;
}

//如果修改表为空,为它初始化空间

if(!m_pModifyTable)//Not prepare the buffer
{
 m_pModifyTable=new MODIFY_TABLE[CC_NUM];
 memset(m_pModifyTable,0,CC_NUM*sizeof(MODIFY_TABLE));
}

//在修改表中查询该词是否存在,如果存在增加该词的频度
 if(FindInModifyTable(nPos,sWordAdd,nHandle,&pRet))
{
 if(pRet!=NULL)
  pRet=pRet->next;
 else
  pRet=m_pModifyTable[nPos].pWordItemHead;
 pRet->data.nFrequency+=nFrequency;
 return true;
}
```

```
  //如果没有在修改表中找到,则添加进去

   pTemp=new WORD_CHAIN;//Allocate the word chain node
   if(pTemp==NULL)//Allocate memory failure
    return false;
   memset(pTemp,0,sizeof(WORD_CHAIN));//init it with 0
   pTemp->data.nHandle=nHandle;//store the handle
   pTemp->data.nWordLen=strlen(sWordAdd);
   pTemp->data.sWord=new char[1+pTemp->data.nWordLen];
   strcpy(pTemp->data.sWord,sWordAdd);
   pTemp->data.nFrequency=nFrequency;
   pTemp->next=NULL;

  //插入到修改表中
   if(pRet!=NULL)
   {
    pNext=pRet->next;//Get the next item before the current item
    pRet->next=pTemp;//link the node to the chain
   }
   else
   {
    pNext=m_pModifyTable[nPos].pWordItemHead;
    m_pModifyTable[nPos].pWordItemHead=pTemp;//Set the pAdd as the head node
   }
   pTemp->next=pNext;//Very important!!!! or else it will lose some node
  //把词块数目加一
   m_pModifyTable[nPos].nCount++;//the number increase by one
   return true;
}
```

删除修改过的词条

```
bool CDictionary::DelModified()
{
  PWORD_CHAIN pTemp,pCur;
  if(!m_pModifyTable)
   return true;


  for(int i=0;i<CC_NUM;i++)
  {
     pCur=m_pModifyTable[i].pWordItemHead;

  //删除链表上的节点
   while(pCur!=NULL)
```

```
   {
    pTemp=pCur;
    pCur=pCur->next;
    delete pTemp->data.sWord;
    delete pTemp;
   }
  }
  delete [] m_pModifyTable;
  m_pModifyTable=NULL;
  return true;
}

//采用二分法进行查找

bool CDictionary::FindInOriginalTable(int nInnerCode,char *sWord,int nHandle,int
*nPosRet)
{
    PWORD_ITEM pItems=m_IndexTable[nInnerCode].pWordItemHead;
 int
nStart=0,nEnd=m_IndexTable[nInnerCode].nCount-1,nMid=(nStart+nEnd)/2,nCount
=0,nCmpValue;
 while(nStart<=nEnd)//Binary search
 {
      nCmpValue=strcmp(pItems[nMid].sWord,sWord);

      //如果中间那个正好是要查找的
      if(nCmpValue==0&&(pItems[nMid].nHandle==nHandle||nHandle==-1))
    {
     if(nPosRet)
     {
      if(nHandle==-1)//Not very strict match
      {//Add in 2002-1-28
         nMid-=1;

          //Get the first item which match the current word
           while(nMid>=0&&strcmp(pItems[nMid].sWord,sWord)==0)
                  nMid--;
           if(nMid<0||strcmp(pItems[nMid].sWord,sWord)!=0)
                   nMid++;
      }
      *nPosRet=nMid;
             return true;
     }
        if(nPosRet)
          *nPosRet=nMid;
```

```
        return true;//find it
    }
    else
if(nCmpValue<0||(nCmpValue==0&&pItems[nMid].nHandle<nHandle&&nHandle!=-1))

    {
     nStart=nMid+1;
    }
    else
if(nCmpValue>0||(nCmpValue==0&&pItems[nMid].nHandle>nHandle&&nHandle!=-1))

      {
     nEnd=nMid-1;
    }
    nMid=(nStart+nEnd)/2;
 }
    if(nPosRet)
 {
  //Get the previous position
  *nPosRet=nMid-1;
 }
 return false;
}

//在修改表中查询

bool CDictionary::FindInModifyTable(int nInnerCode,char *sWord,int
nHandle,PWORD_CHAIN *pFindRet)
{
   PWORD_CHAIN pCur,pPre;
   if(m_pModifyTable==NULL)//empty
    return false;
   pCur=m_pModifyTable[nInnerCode].pWordItemHead;
   pPre=NULL;

   //sWord 相等且句柄(nHandle)相等
   while(pCur!=NULL&&(_stricmp(pCur->data.sWord,sWord)<0||(_stricmp(pCur->dat
a.sWord,sWord)==0&&pCur->data.nHandle<nHandle)))
   //sort the link chain as alphabet
   {
       pPre=pCur;
      pCur=pCur->next;
   }
   if(pFindRet)
     *pFindRet=pPre;
```

```
   if(pCur!=NULL &&
_stricmp(pCur->data.sWord,sWord)==0&&(pCur->data.nHandle==nHandle||nHandle
<0))
   {//The node exists, delete the node and return
    return true;
   }
   return false;
}
```

得到词的类型,共三种汉字、分隔符和其他

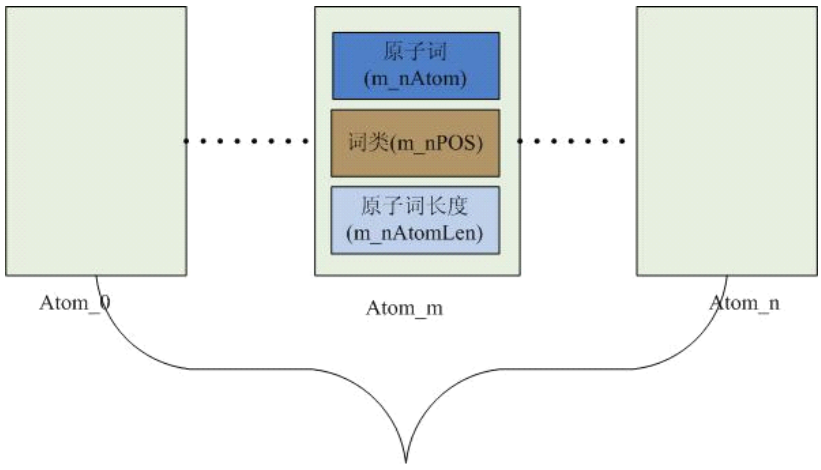
```
int CDictionary::GetWordType(char *sWord)
{
   int nType=charType((unsigned char *)sWord),nLen=strlen(sWord);
   if(nLen>0&&nType==CT_CHINESE&&IsAllChinese((unsigned char *)sWord))
    return WT_CHINESE;//Chinese word
   else if(nLen>0&&nType==CT_DELIMITER)
       return WT_DELIMITER;//Delimiter
   else
    return WT_OTHER;//other invalid
}
```

ICTCLAS 分词系统研究（三）--原子切分 收藏

ICTCLAS 分词的第一步就是原子分词。但在进行原子切分之前，首先要进行断句处理。所谓断句，就是根据分隔符、回车换行符等语句的分隔标志，把源字符串分隔成多个稍微简单一点的短句，再进行分词处理，最后再把各个分词结果合起来，形成最终的分词结果。

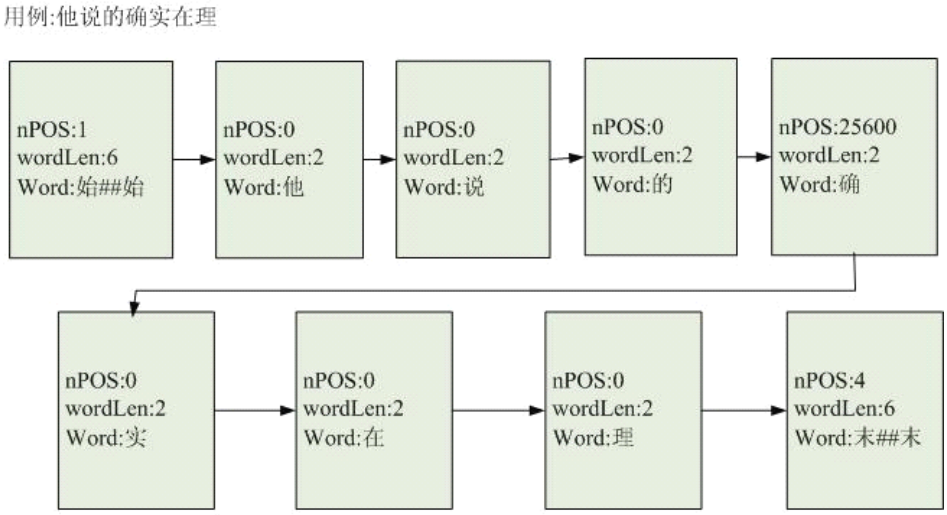
分成短句之后，即可进行原子分词，所谓原子,是指该短句中不可分割的最小语素单位。一个汉字、短句前后的开始结束标识字段、全角标点符号、连在一起的数字字母单字节字符等。最后一种情况可以举例说明，比如：三星 SHX-132 型号的手机 1 元钱，则 SHX-132、1 都是一个原子，其它的每个汉字是一个原子。

按照这种方式，通过简单的汉字分割就形成了原子分词的结果，并对每个原子单位进行词性标注。nPOS＝1 表示是开始标记，nPOS＝4 表示结束标记，nPOS=0 表示未识别词。原子分割后：



原子词数据结构（Atom）　　图一

原子分词后的实例如下图二所示：



进行原子分词后的实例　　图二

ICTCLAS 分词系统研究（四）--初次切分 收藏

经过原子分词后，源字符串成了一个个独立的最小语素单位。下面的初次切分，就是把原子之间所有可能的组合都先找出来。算法是用两个循环来实现，第一层遍历整个原子单位，第二层是当找到一个原子时，不断把后面相邻的原子和该原子组合到一起，访问词典库看它能否构成一个有意义有词组。

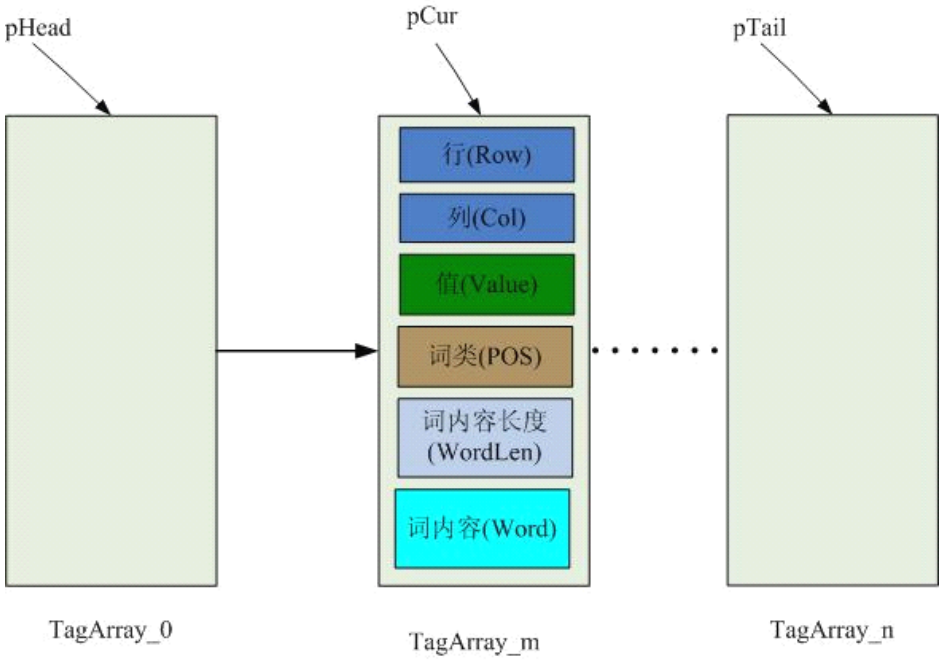
用数学方法可以做如下描述：

有一个原子序列：A(n)(0<=n<m)(其中 m 为原子序列 A 的长度)。当 I=n 时，判断 AnAn+1..Ap 是否为一个词组，其中 n<p<m.

用伪码表示：

```
for(int I=0;I<m;I++){

  String s=A[I];

   for(int j=I+1;j<m;j++){

       s+=A[j];

       if(s 是一个词组){

          把 s 加入到初次切分的列表中;

          记录该词组的词性;

          记录该词组所在表中的坐标位置及其它信息;

       }

       else

          break;

      }

}
```

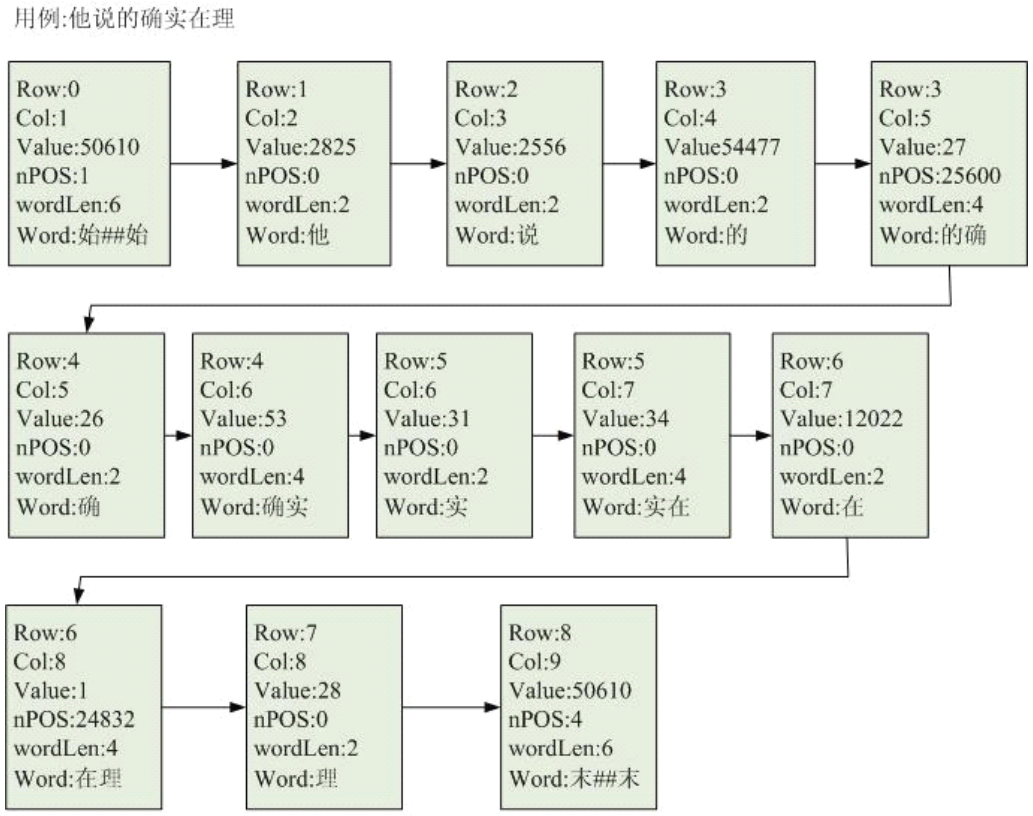
初次切分后的数据结构如下图一所示:



TagArrayChain的数据结构

图一

分词用例”他说的确实在理”经过初次切分后的结果如下图二所示：



进行初步分词后的链表结构(TagArrayChain)实例

图二

用二维表来表示图一中的链表结构如下图二所示:

	0	1	2	3	4	5	6	7	8	9
0		始##始								
1			他							
2				说						
3					的	的确				
4						确	确实			
5							实	实在		
6								在	在理	
7									理	
8										末##末

TagArrayChain实例的二维表表示形式

图三

从上图三可以看出,在二维表中,初次切分后的词组,第一次字相同的在同一行,最后一个字相同的在同一列,原来的原子在对称轴上.

对上述过程进行处理的参考源代码如下:

```
bool CSegment::BiSegment(char *sSentence, double dSmoothingPara, CDictionary
&dictCore, CDictionary &dictBinary, unsigned int nResultCount)
{

 ......

 //在此处完成上图一的处理结果,生成一个链表结构

 m_graphSeg.GenerateWordNet(sSentence,dictCore,true);//Generate words array

 ......
```

在生成图二所示的表结构之后,进一步生成二叉图表.

```
....

//Generate the biword link net

BiGraphGenerate(m_graphSeg.m_segGraph,aBiwordsNet,dSmoothingPara,dictBinary,dic
tCore);
```

....

对该函数进行深入分析:

```
bool CSegment::BiGraphGenerate(CDynamicArray &aWord, CDynamicArray
&aBinaryWordNet,double dSmoothingPara,CDictionary &DictBinary,CDictionary
&DictCore)
{
 ......

 //获取链表的长度
 m_nWordCount=aWord.GetTail(&pTail);//Get tail element and return the words count

 if(m_npWordPosMapTable)
 {//free buffer
  delete [] m_npWordPosMapTable;
  m_npWordPosMapTable=0;
 }

 //分配一个数组,存贮图二中每个结点的词的位置,如下图四所示
 if(m_nWordCount>0)//Word count is greater than 0
  m_npWordPosMapTable=new int[m_nWordCount];//Record the  position of possible
words


//把指针指向当前链表的开头,并计算每个词的位置,然后把它放到数组中

 pCur=aWord.GetHead();
 while(pCur!=NULL)//Set the position map of words
 {
  m_npWordPosMapTable[nWordIndex++]=pCur->row*MAX_SENTENCE_LEN+pCur->col;
  pCur=pCur->next;
 }

 //遍历所有的结点,并计算相临两个词之间的平滑值

 pCur=aWord.GetHead();
 while(pCur!=NULL)//
 {
  if(pCur->nPOS>=0)//It's not an unknown words
   dCurFreqency=pCur->value;
  else//Unknown words
   dCurFreqency=DictCore.GetFrequency(pCur->sWord,2);

  //取得和当前结点列值(col)相同的下个结点
  aWord.GetElement(pCur->col,-1,pCur,&pNextWords);
```

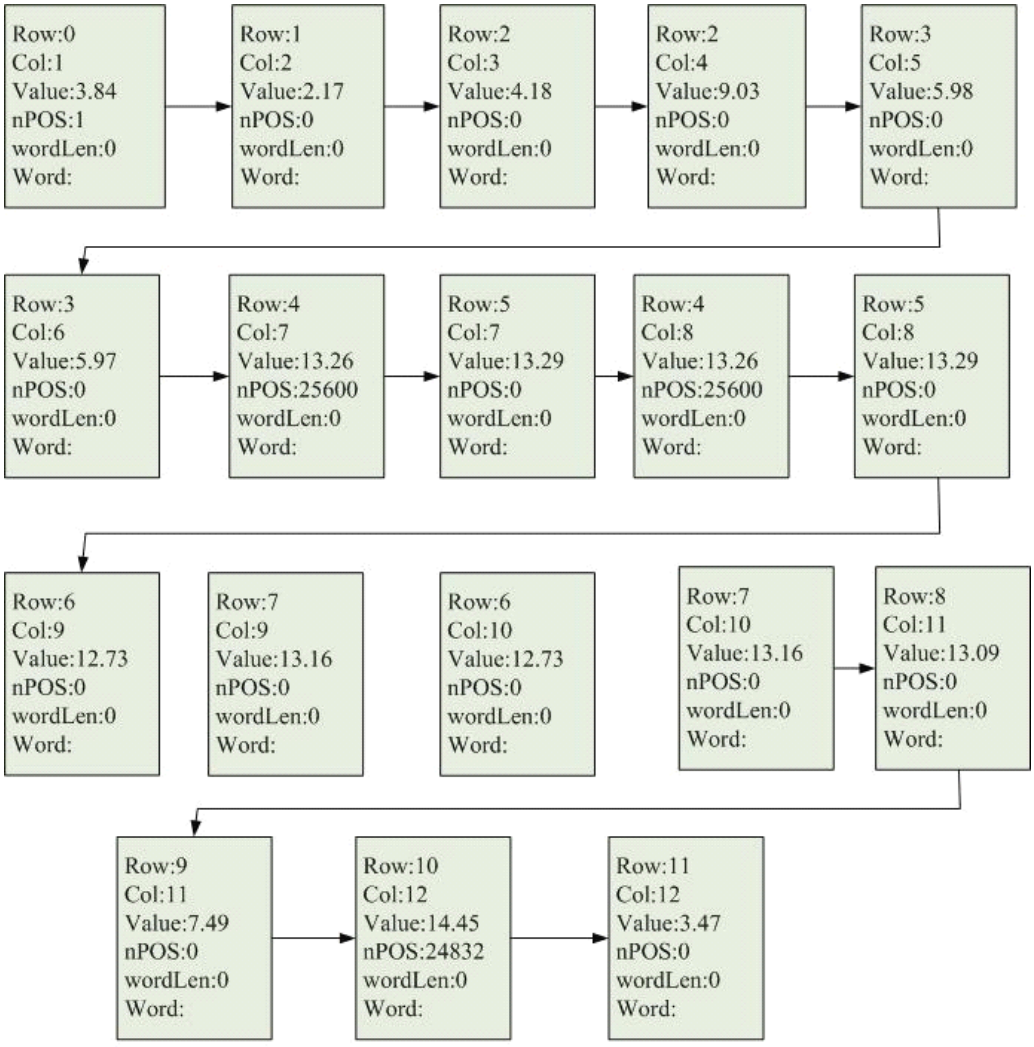
```
  while(pNextWords&&pNextWords->row==pCur->col)//Next words
  {
   //前后两个词用@分隔符连接起来

   strcpy(sTwoWords,pCur->sWord);
   strcat(sTwoWords,WORD_SEGMENTER);
   strcat(sTwoWords,pNextWords->sWord);

   //计算两个连接词的边长
   nTwoWordsFreq=DictBinary.GetFrequency(sTwoWords,3);
   //Two linked Words frequency
   dTemp=(double)1/MAX_FREQUENCE;
   //计算平滑值
   dValue=-log(dSmoothingPara*(1+dCurFreqency)/(MAX_FREQUENCE+80000)+(1-dSmooth
ingPara)*((1-dTemp)*nTwoWordsFreq/(1+dCurFreqency)+dTemp));
   //-log{a*P(Ci-1)+(1-a)P(Ci|Ci-1)} Note 0<a<1
   if(pCur->nPOS<0)//Unknown words: P(Wi|Ci);while known words:1
       dValue+=pCur->value;

   //Get the position index of current word in the position map table
   nCurWordIndex=BinarySearch(pCur->row*MAX_SENTENCE_LEN+pCur->col,m_npWordPosM
apTable,m_nWordCount);
   nNextWordIndex=BinarySearch(pNextWords->row*MAX_SENTENCE_LEN+pNextWords->col,
m_npWordPosMapTable,m_nWordCount);

   //把当前结点在位置表中的位置和下个结点在位置表中的位置及平滑值/词性插入到二
叉链表中
   aBinaryWordNet.SetElement(nCurWordIndex,nNextWordIndex,dValue,pCur->nPOS);
   pNextWords=pNextWords->next;//Get next word
  }
  pCur=pCur->next;
 }
 return true;
}
```

0	1	始##始
1	2002	他
2	4003	说
3	6004	的
4	6005	的确
5	8005	确
6	8006	确实
7	10006	实
8	10007	实在
9	12007	在
10	12008	在理
11	14008	理
12	16009	末##末

PositionMapTable实例(记录词的位置)

图四

最终生成的键表结果如下图五所示：



BinWordNet的链表表示形式

图五

对应的二维图表表示形式如下图六所示：

	1	2	3	4	5	6	7	8	9	10	11	12
0	3.84 始##始@他											
1		2.17 他@说										
2			4.18 说@的	9.03 说@的确								
3					5.98 的@确	5.97 的@确实						
4							13.26 的确@实	13.26 的确@实在				
5							13.29 确@实	13.29 确@实在				
6									12.73 确实@在	12.73 确实@在理		
7									13.16 实@在	13.16 实@在理		
8											13.09 实在@理	
9											7.49 在@理	
10												14.45 在理@末##末
11												3.47 理@末##末

进行初次分词后生成的二叉图表的二维图表示形式

图六

其中小数值代表了相临两个词之间的耦合成度，即构成更大长度词的可能性的机率，值越小说明两个词独立成词的可能性越大。

ICTCLAS 分词系统研究（五）--N 最短路径 收藏

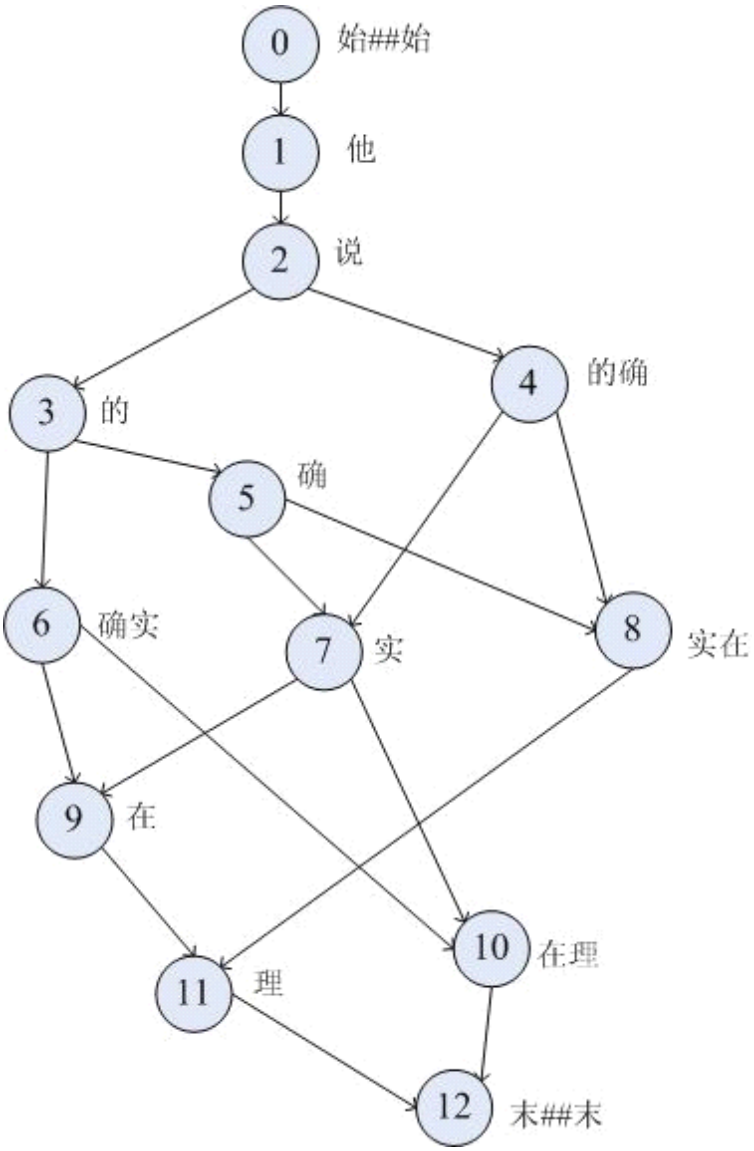
ICTCLAS 和别的分司系统不一样的地方就是于--N 最短路径分词算法。所谓 N 最短路径其实就是最短路径和最大路径的折中，保留前 N 个最优路径。这样做的目的就是对这两种方法取长补短，既能达到一个比较理解的分词不达意效果，又能保证分词不达意速度。在此处，我们中国人的中庸思想被完美体现：）。

在 N－最短路径求解之前，ICTCLAS 首先通过二叉分词图表（邻接表，如下图一所示）表示出了每个词组之间的耦合关系，每一个节点都表示分词图表中的一条边，它的行值代表边的起点（前驱），它的列值代表边的终点（后驱），这一点务必弄清楚。可以通过图一、图二相结合对照来理解。通过计算词组之间的耦合关系，来最终确定初次的分词路径。我们都知道 Dijkstra 算法是求源点到某一点的最短路径，也就是最优的那一条，在此处的 N－最短路径指的是找出前 N 条最优的路径（实际上在 FreeICTCLAS 的源代码当中 N 是等于 1 的，即 nValueKind==1）。按照 Dijkstra 的表示方法把二叉分词图表转化成图二的表示形式，就能比较清楚地看出来，求解的过程实际就是求源点 0 到终于 12 的最短路径，和纯粹的 Dijkstra 算法不同的地方是在此处需要记录每个节点的 N 个前驱，Dijkstra 当中记录一个即可。

	1	2	3	4	5	6	7	8	9	10	11	12
0	3.84 始##始@他											
1		2.17 他@说										
2			4.18 说@的	9.03 说@的确								
3					5.98 的@确	5.97 的@确实						
4							13.26 的确@实	13.26 的确@实在				
5							13.29 确@实	13.29 确@实在				
6									12.73 确实@在	12.73 确实@在理		
7									13.16 实@在	13.16 实@在理		
8											13.09 实在@理	
9											7.49 在@理	
10												14.45 在理@末##末
11												3.47 理@末##末

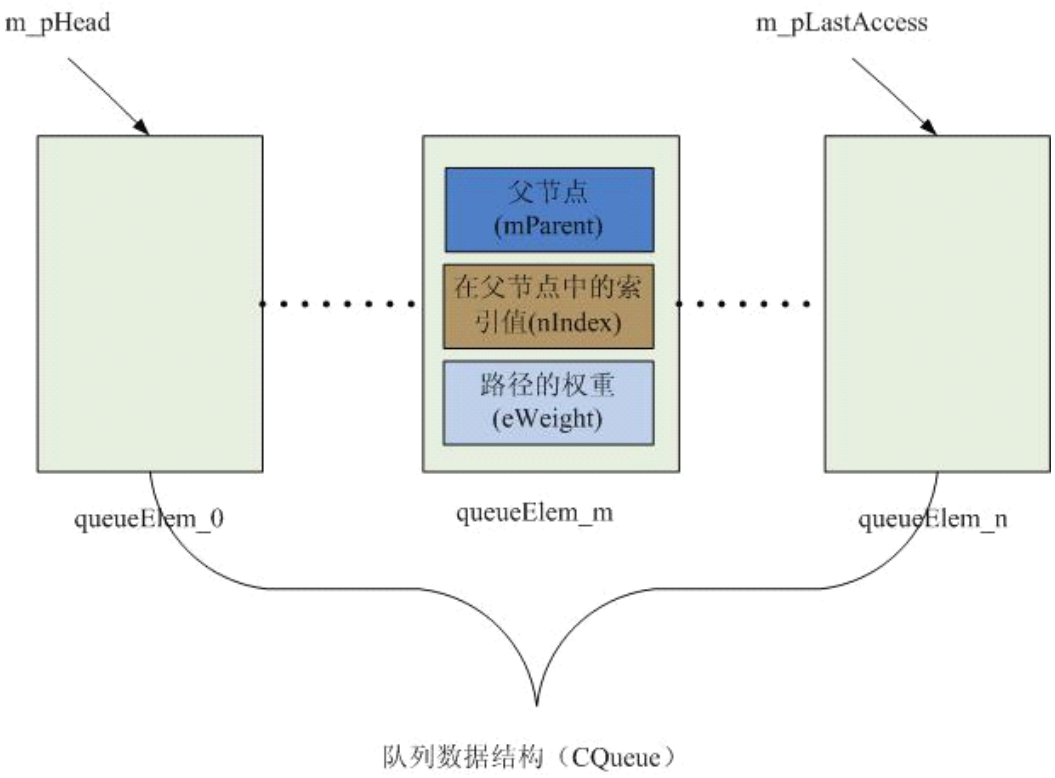
进行初次分词后生成的二叉图表的二维图表示形式

图一



图二

在求解过程中，源程序通过二维数组 m_pParent[i][j]、m_pWeight[m][n]来记录每个节点的 N 个前驱和每个前驱和权重，而求解最短路径权重时借用了一个队列来实现排序，数据结构如下图三所示：



图四

在源程序中，N 最短路径是在 CNShortPath 类里里面实现的。

```
 bool CSegment::BiSegment(char *sSentence, double dSmoothingPara, CDictionar
y &dictCore, CDictionary &dictBinary, unsigned int nResultCount)
{

 ......

 //调用构造函数,生成一个二维链表,如下图一所示。每个链表节点是一个队列，数据结构如下图二
所示
  CNShortPath sp(&aBiwordsNet,nResultCount);

 //最短路径算法实现
  sp.ShortPath();

 //输出最短路径
  sp.Output(nSegRoute,false,&m_nSegmentCount);

  .....

}
```

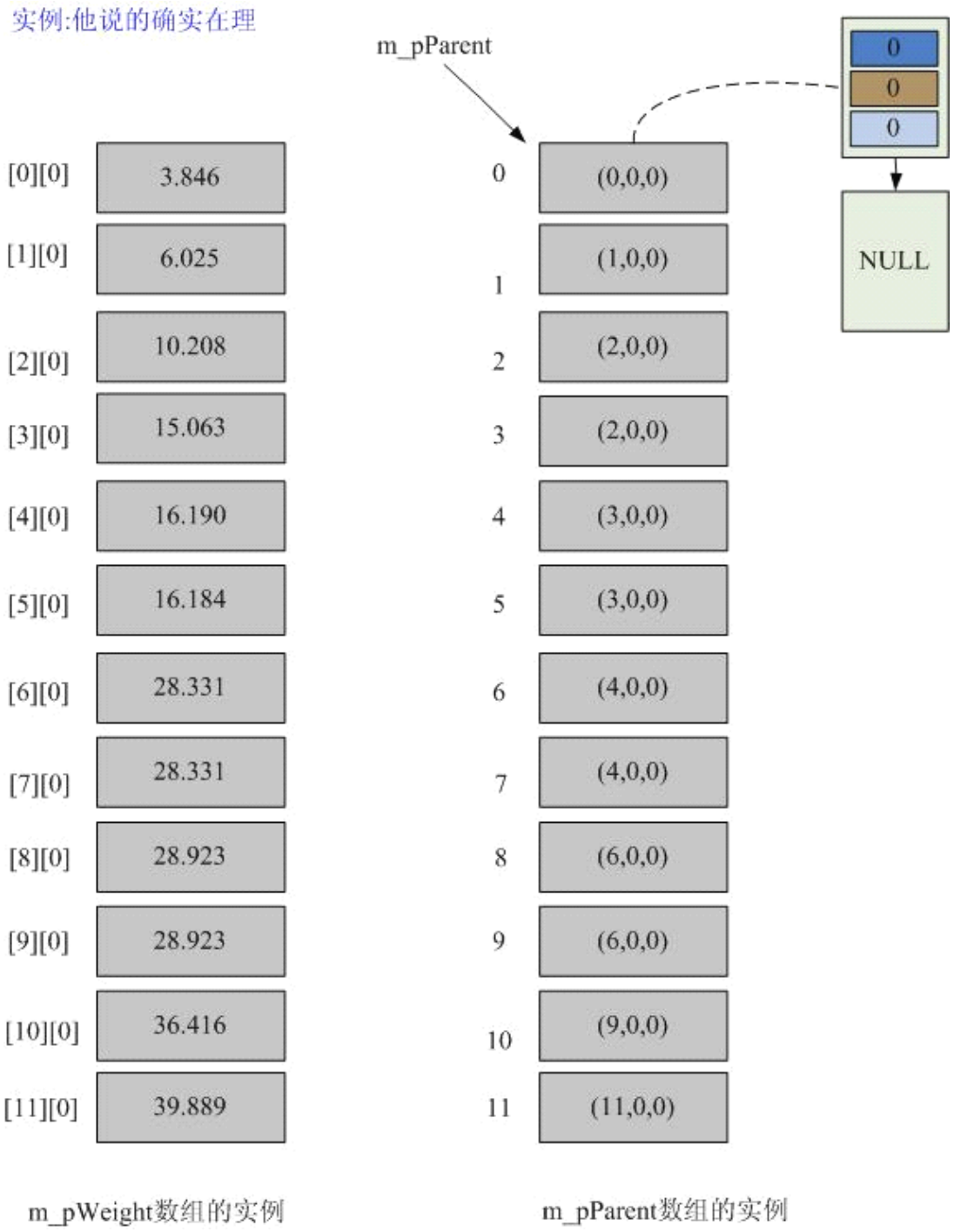
对源代码进行解析，以“他说的确实在理”为实例：

```
//进行 N－最短路径的求解，找出每一个节点的前驱计算前驱的权值（从源点到该前驱节点）
int CNShortPath::ShortPath()
{
    unsigned int nCurNode=1,nPreNode,i,nIndex;
    ELEMENT_TYPE eWeight;
    PARRAY_CHAIN pEdgeList;

    //遍历所示节点,按列优先原则,从 1 开始
    //m_apCost 其实是一个邻接表,或者叫稀疏矩阵,如图一所示，
    //每一个节点代表的是分词路径中的一条边，
    //该节点的行值代表边的起点,该节点的列值代表该边的终点
    for(;nCurNode<m_nVertex;nCurNode++)
    {
      CQueue queWork;

      //得到从 nCurNode 开始的所有结点,列优先原则
      eWeight=m_apCost->GetElement(-1,nCurNode,0,&pEdgeList);//Get all the edg
es

      //遍历列下标等于 nCurNode 的所有结点,即遍历邻接表中所有终点为 nCurNode 的边
      while(pEdgeList!=0 && pEdgeList->col==nCurNode)
      {
        //取得该边的起点
        nPreNode=pEdgeList->row;
        //该条边的权值
        eWeight=pEdgeList->value;//Get the value of edges

        //m_nValueKind 代表的是 N-最短路径的 N,即前 N 条最短分词路径
        //m_pWeight 记录当前节点的最短路径的权值,即从开始点到该点所有边的权值的总和
        //每条边的起点的前驱可能有若干个,在这里只记录权值最小的 m_nValueKind 个
        for(i=0;i<m_nValueKind;i++)
        {
          if(nPreNode>0)//Push the weight and the pre node infomation
          {
            if(m_pWeight[nPreNode-1][i]==INFINITE_VALUE)
              break;

            queWork.Push(nPreNode,i,eWeight+m_pWeight[nPreNode-1][i]);
          }
          else//该条边的起点是 0,即该起点没有父结点,是分词的源点
          {
            queWork.Push(nPreNode,i,eWeight);
            break;
```

```
            }
        }//end for
        pEdgeList=pEdgeList->next;

    }//end while

    //Now get the result queue which sort as weight.
    //Set the current node information
    for(i=0;i<m_nValueKind;i++)
    {
        m_pWeight[nCurNode-1][i]=INFINITE_VALUE;
    }
    //memset((void *),(int),sizeof(ELEMENT_TYPE)*);
    //init the weight
    i=0;

    //设置当前节点的 N 个前驱节点的最短路径的权值
    //以"他说的确实在理"为例
    //m_pWeight[0][0]=3.846
    //m_pWeight[1][0]=6.025
    //m_pWeight[2][0]=10.208
    //m_pWeight[3][0]=15.063
    //m_pWeight[4][0]=16.190
    //m_pWeight[5][0]=16.184
    //m_pWeight[6][0]=28.331
    //m_pWeight[7][0]=28.331
    //m_pWeight[8][0]=28.923
    //m_pWeight[9][0]=28.923
    //m_pWeight[10][0]=36.416
    //m_pWeight[11][0]= 39.889
    while(i<m_nValueKind&&queWork.Pop(&nPreNode,&nIndex,&eWeight)!=-1)
    {//Set the current node weight and parent
        if(m_pWeight[nCurNode-1][i]==INFINITE_VALUE)
            m_pWeight[nCurNode-1][i]=eWeight;

        //记录下一个前驱的权值，在 queWork 里面已经做过排序，
        //所以不会有后来的 eWeight 更小的可能
        //我总得把此 if 语句的表达式反过来比较可能会更容易理解一点
        else if(m_pWeight[nCurNode-1][i]<eWeight)//Next queue
        {
            i++;//Go next queue and record next weight
            if(i==m_nValueKind)//Get the last position
                break;
```

```
            m_pWeight[nCurNode-1][i]=eWeight;
        }

        //m_pParent[0][0]=(0,0,0)
        //m_pParent[1][0]=(1,0,0)
        //m_pParent[2][0]=(2,0,0)
        //m_pParent[3][0]=(2,0,0)
        //m_pParent[4][0]=(3,0,0)
        //m_pParent[5][0]=(3,0,0)
        //m_pParent[6][0]=(4,0,0)
        //m_pParent[7][0]=(4,0,0)
        //m_pParent[8][0]=(6,0,0)
        //m_pParent[9][0]=(6,0,0)
        //m_pParent[10][0]=(9,0,0)
        //m_pParent[11][0]=(11,0,0)
        m_pParent[nCurNode-1][i].Push(nPreNode,nIndex);
     }
   }//end for

   return 1;
}
```

经过对每个节点的前驱求解后，得到前驱的最短路径权值和它的父节点，记录如下图四所示：



图四

然后通过队列（其实更象一个栈）来求出二叉分词路径：

```
 //bBest=true: only get one best result and ignore others
 //Added in 2002-1-24
 void CNShortPath::GetPaths(unsigned int nNode,unsigned int nIndex,int **nResult,
bool bBest)
{
    CQueue queResult;
```

```
    unsigned int nCurNode,nCurIndex,nParentNode,nParentIndex,nResultIndex=0;

    if(m_nResultCount>=MAX_SEGMENT_NUM)//Only need 10 result
       return ;
    nResult[m_nResultCount][nResultIndex]=-1;//Init the result

    //先把末节点压栈
    queResult.Push(nNode,nIndex);
    nCurNode=nNode;
    nCurIndex=nIndex;
    bool bFirstGet;
    while(!queResult.IsEmpty())
    {
       while(nCurNode>0)//
       {   //Get its parent and store them in nParentNode,nParentIndex
          //根据 m_pParent 数组中记录的每一个节点的前驱，把相应的前驱也压入栈中，
          //当把 0 节点也压入栈中时，即表示找到一个条完整的最短路径，
          //详情可参考吕震宇的 BLOG：SharpICTCLAS 分词系统简介(4)NShortPath-1
          if(m_pParent[nCurNode-1][nCurIndex].Pop(&nParentNode,&nParentIndex,0,
false,true)!=-1)
          {
            nCurNode=nParentNode;
            nCurIndex=nParentIndex;
          }
          if(nCurNode>0)
             queResult.Push(nCurNode,nCurIndex);
       }
       //当到 0 节点时，也就意为着形成了一条最短路径
       if(nCurNode==0)
       { //Get a path and output
          nResult[m_nResultCount][nResultIndex++]=nCurNode;//Get the first node
          bFirstGet=true;
          nParentNode=nCurNode;

          //输出该条分词怎么，在这里 queResult 并不实际弹出元素，只是下标位移遍历元素
          //遍历元素通过第四个参数 bModify 来控制是否真正删除栈顶元素
          while(queResult.Pop(&nCurNode,&nCurIndex,0,false,bFirstGet)!=-1)
          {
             nResult[m_nResultCount][nResultIndex++]=nCurNode;
             bFirstGet=false;
             nParentNode=nCurNode;
          }
          nResult[m_nResultCount][nResultIndex]=-1;//Set the end
          m_nResultCount+=1;//The number of result add by 1
```

```
            if(m_nResultCount>=MAX_SEGMENT_NUM)//Only need 10 result
                return ;
            nResultIndex=0;
            nResult[m_nResultCount][nResultIndex]=-1;//Init the result

            if(bBest)//Return the best result, ignore others
               return ;
        }

        //首先判断栈顶元素是否有下一个前驱，如果没有则删除栈顶元素直到有下一个前驱的元素出现
        queResult.Pop(&nCurNode,&nCurIndex,0,false,true);//Read the top node
        while(queResult.IsEmpty()==false&&(m_pParent[nCurNode-1][nCurIndex].IsSingle()||m_pParent[nCurNode-1][nCurIndex].IsEmpty(true)))
        {
            queResult.Pop(&nCurNode,&nCurIndex,0);//Get rid of it
            queResult.Pop(&nCurNode,&nCurIndex,0,false,true);//Read the top node
        }

        //如果找到了有下一个前驱的节点，则它的前驱压入栈中，重新循环直到把源点也压入
        if(queResult.IsEmpty()==false&&m_pParent[nCurNode-1][nCurIndex].IsEmpty(true)==false)
        {
            m_pParent[nCurNode-1][nCurIndex].Pop(&nParentNode,&nParentIndex,0,false,false);
            nCurNode=nParentNode;
            nCurIndex=nParentIndex;
            if(nCurNode>0)
               queResult.Push(nCurNode,nCurIndex);
        }
    }
}
```

最终得到最短路么（0，1，2，3，6，9，11，12），里面的数值分别对应研究（四）中图四的下标，到此分词的第一大步就结束了，并形成最终结果：始##始/他/说/的/确实/在/理/末##末

如果想详细 getPaths（）当中的实现原理，推荐大家看吕震宇的 BLOG：

http://www.cnblogs.com/zhenyulu/articles/669795.html

ICTCLAS 分词系统研究（六）--得到初分结果 收藏

仍然以“他说的确实在理”为例，经过 NshortPath 的处理后，我们可以得到 N 条最短二叉分词路径，如下：

初次生成的分词图表：

	1	2	3	4	5	6	7	8	9
0	始##始								
1		他							
2			说						
3				的	的确				
4					确	确实			
5						实	实在		
6							在	在理	
7								理	
8									末##末

初次生成的二叉分词图表：

	1	2	3	4	5	6	7	8	9	10	11	12
0	始##始@他											
1		他@说										
2			说@的	说@的确								
3					的@确	的@确实						
4							的确@实	的确@实在				
5							确@实	确@实在				
6									确实@在	确实@在理		
7									实@在	实@在理		

8											实在@理	
9											在@理	
10												在理@末##末
11												理@末##末

初次生成的二叉分词路径：

序号	二叉分词路径
0	0 1 2 3 6 9 11 12
1	0 1 2 4 7 9 11 12
2	0 1 2 3 5 7 9 11 12

0 1 2 3 6 9 11 12 指的是针对上图二叉分词图表，得出的分词路径的列下标，其实图表中的列对应的是@后面的词，行对应的是@前面的词在分词图表中的位置。得到了二叉分词路径，其实我们就可以得到真正的分词路径，只需要根据分词图表和二叉分词图表之间的对应关系进行一个简单的转换即可。

源代码中是通过这一段代码来实现的：

```
while(i<m_nSegmentCount)
{
    //把二叉分词路径转成分词路径
    BiPath2UniPath(nSegRoute[i]);
    //根据分词路径生成分词结果
    GenerateWord(nSegRoute,i);
    i++;
}
```

初次生成的分词结果：

序号	分词结果
0	他/ 说/ 的/ 确实/ 在/ 理/
1	他/ 说/ 的确/d 实/ 在/ 理/
2	他/ 说/ 的/ 确/ 实/ 在/ 理/

需要注意的是，在 generateWord()函数里对一些特殊情况做一些处理，然后再生成分词结果。主要是对涉及到数字、时间、日期的结果进行合并、拆分，

```
 //Generate Word according the segmentation route
 bool CSegment::GenerateWord(int **nSegRoute, int nIndex)
{
    unsigned int i=0,k=0;
    int j,nStartVertex,nEndVertex,nPOS;
    char sAtom[WORD_MAXLENGTH],sNumCandidate[100],sCurWord[100];
    ELEMENT_TYPE fValue;
    while(nSegRoute[nIndex][i]!=-1&&nSegRoute[nIndex][i+1]!=-1&&nSegRoute[nIndex][i]<nSegRoute[nIndex][i+1])
    {
       nStartVertex=nSegRoute[nIndex][i];
       j=nStartVertex;//Set the start vertex
       nEndVertex=nSegRoute[nIndex][i+1];//Set the end vertex
       nPOS=0;
       m_graphSeg.m_segGraph.GetElement(nStartVertex,nEndVertex,&fValue,&nPOS);
       sAtom[0]=0;
       while(j<nEndVertex)
       {//Generate the word according the segmentation route
          strcat(sAtom,m_graphSeg.m_sAtom[j]);
          j++;
       }
       m_pWordSeg[nIndex][k].sWord[0]=0;//Init the result ending
       strcpy(sNumCandidate,sAtom);
       //找出连续的数字串
       while(sAtom[0]!=0&&(IsAllNum((unsigned char *)sNumCandidate)||IsAllChineseNum(sNumCandidate)))
       {//Merge all seperate continue num into one number
        //sAtom[0]!=0: add in 2002-5-9
          strcpy(m_pWordSeg[nIndex][k].sWord,sNumCandidate);
          //Save them in the result segmentation
          i++;//Skip to next atom now
          sAtom[0]=0;

          while(j<nSegRoute[nIndex][i+1])
          {//Generate the word according the segmentation route
             strcat(sAtom,m_graphSeg.m_sAtom[j]);
             j++;
          }
          strcat(sNumCandidate,sAtom);
       }
       unsigned int nLen=strlen(m_pWordSeg[nIndex][k].sWord);
       if(nLen==4&&CC_Find("第上成±一＋：·．／",m_pWordSeg[nIndex][k].sWord)||nLen==1&&strchr("+-./",m_pWordSeg[nIndex][k].sWord[0]))
```

```
        {//Only one word
            strcpy(sCurWord,m_pWordSeg[nIndex][k].sWord);//Record current word
            i--;
        }
        else if(m_pWordSeg[nIndex][k].sWord[0]==0)//Have never entering the whil
e loop
        {
            strcpy(m_pWordSeg[nIndex][k].sWord,sAtom);
            //Save them in the result segmentation
            strcpy(sCurWord,sAtom);//Record current word
        }
        else
        {//It is a num

            if(strcmp("——",m_pWordSeg[nIndex][k].sWord)==0||strcmp("—",m_pWor
dSeg[nIndex][k].sWord)==0||m_pWordSeg[nIndex][k].sWord[0]=='-'&&m_pWordSe
g[nIndex][k].sWord[1]==0)//The delimiter "——"
            {
                nPOS=30464;//'w'*256;Set the POS with 'w'
                i--;//Not num, back to previous word
            }
            else
            {//Adding time suffix

                char sInitChar[3];
                unsigned int nCharIndex=0;//Get first char
                sInitChar[nCharIndex]=m_pWordSeg[nIndex][k].sWord[nCharIndex];
                if(sInitChar[nCharIndex]<0)
                {
                    nCharIndex+=1;
                    sInitChar[nCharIndex]=m_pWordSeg[nIndex][k].sWord[nCharIndex];
                }
                nCharIndex+=1;
                sInitChar[nCharIndex]='
```

ICTCLAS 分词系统研究（七）--未登陆词识别 收藏

在研究（六）中，我们经过种种努力，终于得到了梦寐以求的分词结果，我得意的笑得意的笑。。。别急，好戏还在后头呢。我们冷静想一想，前面初分的结果主要都是基于词典库的词条得到的，象人名、地名之类的未登录词（即指该词条不在词典库中）该如何识别呢？

典型的象人名，全国上下、古今中外得有多少人名呀，不可能全部做到词库中，必须依照一定的规则和算法对其进行识别，大家可以张华平、刘群的论文《参考基于角色标注的中国人名自动识别研究》和 DanceFire 的分析文章 http://blog.csdn.net/DanceFire/archive/2007/05/13/1606603.aspx，我 就不多做赘述了。

下面以人名的自动识别为例，做个简单的说明。FreeICTCLAS 源程序中对人名的识别主要有两步：一、对初分结果进行词性标记；二、按照人名识别的十几种模式规则进行套用，从而识别出句子中的人名。

对照源代码进行分析：

```
 bool CUnknowWord::Recognition(PWORD_RESULT pWordSegResult, CDynamicArra
y &graphOptimum,CSegGraph &graphSeg,CDictionary &dictCore)
{
    int nStartPos=0,j=0,nAtomStart,nAtomEnd,nPOSOriginal;
    ELEMENT_TYPE dValue;

    //对初分结果进行词性标记，并记录可能成词的节点位置
    m_roleTag.POSTagging(pWordSegResult,dictCore,m_dict);

    //Tag the segmentation with unknown recognition roles according the core diction
ary and unknown recognition dictionary
    for(int i=0;i<m_roleTag.m_nUnknownIndex;i++)
    {
       //获取未登录词在原子分词链表中的开始下标和结束下标
       while((unsigned int)j<graphSeg.m_nAtomCount&&nStartPos<m_roleTag.m_n
UnknownWords[i][0])
       {
          nStartPos+=graphSeg.m_nAtomLength[j++];
       }
       nAtomStart=j;
       while((unsigned int)j<graphSeg.m_nAtomCount&&nStartPos<m_roleTag.m_n
UnknownWords[i][1])
       {
          nStartPos+=graphSeg.m_nAtomLength[j++];
       }
       nAtomEnd=j;
       if(nAtomStart<nAtomEnd)
       {
```

```
            //如果当前计算出来的值小于原来的值，即该元素实际不存在，则在链表中插入该元素
            graphOptimum.GetElement(nAtomStart,nAtomEnd,&dValue,&nPOSOriginal);

            if(dValue>m_roleTag.m_dWordsPossibility[i])//Set the element with less frequency
                graphOptimum.SetElement(nAtomStart,nAtomEnd,m_roleTag.m_dWordsPossibility[i],m_nPOS,m_sUnknownFlags);
        }
    }
    return true;
}
 //POS tagging with Hidden Markov Model
 bool CSpan::POSTagging(PWORD_RESULT pWordItems,CDictionary &dictCore,CDictionary &dictUnknown)
{
 //pWordItems: Items; nItemCount: the count of items;core dictionary and unknown recognition dictionary
    int i=0,j,nStartPos;
    Reset(false);
    while(i>-1&&pWordItems[i].sWord[0]!=0)
    {
        nStartPos=i;//Start Position
        //首先进行句子的分断，依据为找到一个 unknonwDict 中不存在的词为止
        //然后找出每一个词所有可能的词性及其对应的词频
        i=GetFrom(pWordItems,nStartPos,dictCore,dictUnknown);
        //计算每一个词与前面一个词的所有的词性之间的耦合度，找出值最小的那个
        //然后找出每一个词最佳的词性
        GetBestPOS();
        switch(m_tagType)
        {
        case TT_NORMAL://normal POS tagging
            j=1;
            while(m_nBestTag[j]!=-1&&j<m_nCurLength)
            {//Store the best POS tagging
                pWordItems[j+nStartPos-1].nHandle=m_nBestTag[j];
                //Let 。be 0
                if(pWordItems[j+nStartPos-1].dValue>0&&dictCore.IsExist(pWordItems[j+nStartPos-1].sWord,-1))//Exist and update its frequncy as a POS value
                    pWordItems[j+nStartPos-1].dValue=dictCore.GetFrequency(pWordItems[j+nStartPos-1].sWord,m_nBestTag[j]);
                j+=1;
            }
            break;
        case TT_PERSON://Person recognition
```

```
            //按照人名的十几种模式，进行匹配，记录所有构成人名的词的位置坐标
            PersonRecognize(dictUnknown);
            break;
        case TT_PLACE://Place name recognition
        case TT_TRANS_PERSON://Transliteration Person
            PlaceRecognize(dictCore,dictUnknown);
            break;
        default:
            break;
        }
        Reset();
    }

    //print all pos info

    for(int m=0;m_nTags[m][0]>=0;m++){
        for(int n=0;m_nTags[m][n]>=0;n++){
            int pos=m_nTags[m][n];
            double value=m_dFrequency[m][n];
            TRACE ("%s %d %5d %f ", "word:",m, pos,value);
        }
    }
    return true;
}
```

在 FreeICTCLAS 中，对初分结果进行词性标记及后续处理时用到了一个循环，即把初分结果按照一定的条件进行分隔，进行多次处理。这个条件就是当初分结果中的词在 unknownDict 没有对应的词性时从此处断开，我个人认为没有太大必要，在 ictclas4j 的处理中我舍弃了这个循环，直接对所有初次结果进行词性标记，减少代码的复杂度。

```
 bool CSpan::PersonRecognize(CDictionary &personDict)
{
  char sPOS[MAX_WORDS_PER_SENTENCE]="z",sPersonName[100];
                    //0    1    2    3    4   5
  char sPatterns[][5]={ "BBCD","BBC","BBE","BBZ","BCD","BEE","BE","BG",
                      "BXD","BZ", "CDCD","CD","EE", "FB", "Y","XD",""};
                 //BBCD       BBC       BBE     BBZ    BCD        BEE      BE        BG
  double dFactor[]={0.003606,0.000021,0.001314,0.000315,0.656624, 0.000021,0.
146116,0.009136,
                 // BXD      BZ   CDCD     CD      EE      FB       Y        XD
                  0.000042,0.038971,0,0.090367,0.000273,0.009157,0.034324,0.0097
35,0
  };
  //About parameter:
/*
```

```
BBCD 343 0.003606
BBC 2 0.000021
BBE 125 0.001314
BBZ 30 0.000315
BCD 62460 0.656624
BEE 0 0.000000
BE 13899 0.146116
BG 869 0.009136
BXD 4 0.000042
BZ 3707 0.038971
CD 8596 0.090367
EE 26 0.000273
FB 871 0.009157
Y 3265 0.034324
XD 926 0.009735
*/
  //The person recognition patterns set
  //BBCD:姓+姓+名 1+名 2;
  //BBE: 姓+姓+单名;
  //BBZ: 姓+姓+双名成词;
  //BCD: 姓+名 1+名 2;
  //BE:  姓+单名;
  //BEE: 姓+单名+单名;韩磊磊
  //BG:  姓+后缀
  //BXD: 姓+姓双名首字成词+双名末字
  //BZ:  姓+双名成词;
  //B:    姓
  //CD:  名 1+名 2;
  //EE:  单名+单名;
  //FB:  前缀+姓
  //XD:  姓双名首字成词+双名末字
  //Y:   姓单名成词
  int nPatternLen[]={4,3,3,3,3,3,2,2,3,2,4,2,2,2,1,2,0};

  for(int i=1;m_nBestTag[i]>-1;i++)//Convert to string from POS
    sPOS[i]=m_nBestTag[i]+'A';
  sPOS[i]=0;
  int j=1,k,nPos;//Find the proper pattern from the first POS
  int nLittleFreqCount;//Counter for the person name role with little frequecy
  bool bMatched=false;
  while(j<i)
  {
    bMatched=false;
    for(k=0;!bMatched&&nPatternLen[k]>0;k++)
```

```
  {
    //如果从找到了和模式库中匹配的字符串，且该串前面和后面的字符都不是一个圆点，则认为是一个可能的人名组合
    if(strncmp(sPatterns[k],sPOS+j,nPatternLen[k])==0&&strcmp(m_sWords[j-1],"·")!=0&&strcmp(m_sWords[j+nPatternLen[k]],"·")!=0)
    {//Find the proper pattern k

      //如果前缀＋姓成词并且后面一个名或后缀，则该规则失效
      if(strcmp(sPatterns[k],"FB")==0&&(sPOS[j+2]=='E'||sPOS[j+2]=='C'||sPOS[j+2]=='G'))
      {//Rule 1 for exclusion:前缀+姓+名 1(名 2): 规则(前缀+姓)失效；
        continue;
      }
/*      if((strcmp(sPatterns[k],"BEE")==0||strcmp(sPatterns[k],"EE")==0)&&strcmp(m_sWords[j+nPatternLen[k]-1],m_sWords[j+nPatternLen[k]-2])!=0)
      {//Rule 2 for exclusion:姓+单名+单名:单名+单名 若 EE 对应的字不同，规则失效.如：韩磊磊
        continue;
      }

      if(strcmp(sPatterns[k],"B")==0&&m_nBestTag[j+1]!=12)
      {//Rule 3 for exclusion: 若姓后不是后缀，规则失效.如：江主席、刘大娘
        continue;
      }
*/      //Get the possible name
      nPos=j;//Record the person position in the tag sequence
      sPersonName[0]=0;
      nLittleFreqCount=0;//Record the number of role with little frequency
      while(nPos<j+nPatternLen[k])
      {//Get the possible person name
       //
        if(m_nBestTag[nPos]<4&&personDict.GetFrequency(m_sWords[nPos],m_nBestTag[nPos])<LITTLE_FREQUENCY)
          nLittleFreqCount++;//The counter increase
        strcat(sPersonName,m_sWords[nPos]);
        nPos+=1;
      }
/*
      if(IsAllForeign(sPersonName)&&personDict.GetFrequency(m_sWords[j],1)<LITTLE_FREQUENCY)
      {//Exclusion foreign name
       //Rule 2 for exclusion:若均为外国人名用字 规则(名 1+名 2)失效
        j+=nPatternLen[k]-1;
        continue;
```

```
            }
*/          if(strcmp(sPatterns[k],"CDCD")==0)
            {//Rule for exclusion
             //规则(名 1+名 2+名 1+名 2)本身是排除规则:女高音歌唱家迪里拜尔演唱
              //Rule 3 for exclusion:含外国人名用字 规则适用
             //否则，排除规则失效:黑妞白妞姐俩拔了头筹。
                if(GetForeignCharCount(sPersonName)>0)
                   j+=nPatternLen[k]-1;
                continue;
            }
/*          if(strcmp(sPatterns[k],"CD")==0&&IsAllForeign(sPersonName))
            {//
                j+=nPatternLen[k]-1;
                continue;
            }
            if(nLittleFreqCount==nPatternLen[k]||nLittleFreqCount==3)
            //马哈蒂尔;小扎耶德与他的中国阿姨胡彩玲受华黎明大使之邀，
            //The all roles appear with two lower frequecy,we will ignore them
                continue;
*/          m_nUnknownWords[m_nUnknownIndex][0]=m_nWordPosition[j];
            m_nUnknownWords[m_nUnknownIndex][1]=m_nWordPosition[j+nPatternL
en[k]];
            m_dWordsPossibility[m_nUnknownIndex]=-log(dFactor[k])+ComputePossib
ility(j,nPatternLen[k],personDict);
            //Mutiply the factor
            m_nUnknownIndex+=1;
            j+=nPatternLen[k];
            bMatched=true;
        }
    }
    if(!bMatched)//Not matched, add j by 1
       j+=1;
 }
 return true;
}
```

举例说明，比如例句“张华平说的确实在理”：

初次生成的分词结果：

序号	分词结果
0	张/ 华/ 平/ 说/ 的/ 确实/ 在/ 理/

做为人名的“张华平”还并没有被识别出来，需要按照人名的模式进行标记，结果为 BCDAAAAA，BCD 符合人名模式：姓＋名 1＋名 2，因此我们可以把前面个初分结果进行合并，实际上是在原来的初次结果中插入一个代表“张华平”这样组合的节点，如下图（1,4）节点所示：

经过人名、地名识别后的分词图表：

	1	2	3	4	5	6	7	8	9	10	11
0	始##始										
1		张		未##人							
2			华	未##人							
3				平							
4					说						
5						的					
6								确实			
7											
8									在		
9										理	
10											末##末

ICTCLAS 分词系统研究（八）--生成最终分词结果 收藏

经过人名、地名等未登陆词的识别之后，再次生成二叉分词图表，求取 N－最短路径。为何再次执行这样的循环，是因为在得到初分结果后又增加了新的节点（比如：人名或地名）到结果链表中，需要再次求取最短路径：

经过优化后的二叉分词图表：

	1	2	3	4	5	6	7	8	9	10	11
0	始##始@张	始##始@未##人									
1			张@华	张@未##人							
2						未##人@说					
3					华@平						
4						未##人@说					
5						平@说					
6							说@的				
7								的@确实			
8									确实@在		
9										在@理	
10											理@末##末

经过优化后的二叉分词路径：

序号	二叉分词路径
0	0 2 6 7 8 9 10 11

至此，我们得到了最终的分词路径，正确的把人名识别出来，但在这个结果只有一部分词正确标注了词性，主要是未登陆词，即源码中以"未＃＃X"表示的，其它的分词并未成功的进行记性标记。所以需要再次调用记性标记这一次过程，把剩余的词的词性成功标注出来。

经过优化后的分词结果：

序号	分词结果
0	张华平/nr 说/v 的/uj 确实/ad 在/p 理/n

ICTCLAS 分词系统研究（九）--对最终结果做优化调整 收藏

在研究（八）中，我们得到了最终的分词结果了，好兴奋呀。不过，还有临门一脚不能忘了，对一些特殊情况做处理。主要是对叠词（相邻的两个字或词一样）及个别词性进行合并处理。

比如，以"一片片的白云很好看"，他的最终分词结果是：

经过优化后的分词结果：

序号	分词结果
0	一/m 片/q 片/q 的/uj 白云/n 很/d 好看/a

很显然，"一片片"应该为一个整体，没有拆分的必要，看源代码的调整过程：

```
//Adjust the result with some rules
bool CResult::Adjust(PWORD_RESULT pItem,PWORD_RESULT pItemRet)
{
    int i=0,j=0;
    unsigned int nLen;
    char sSurName[10],sSurName2[10],sGivenName[10];
    bool bProcessed=false;//Have been processed
    while(pItem[i].sWord[0]!=0)
    {
        nLen=strlen(pItem[i].sWord);
        bProcessed=false;

        //Rule1: adjust person name
        if(pItem[i].nHandle==28274&&ChineseNameSplit(pItem[i].sWord,sSurName,s
SurName2,sGivenName,m_uPerson.m_dict)&&strcmp(pItem[i].sWord,"叶利钦")!=0)//'
nr'
        {//Divide name into surname and given name

            if(sSurName[0])
            {
                strcpy(pItemRet[j].sWord,sSurName);
                pItemRet[j++].nHandle=28274;
            }
            if(sSurName2[0])
            {
                strcpy(pItemRet[j].sWord,sSurName2);
                pItemRet[j++].nHandle=28274;
            }
            if(sGivenName[0])
            {
                strcpy(pItemRet[j].sWord,sGivenName);
                pItemRet[j++].nHandle=28274;
```

```
        }
        bProcessed=true;
    }
    //Rule2 for overlap words ABB 一段段、一片片
    else if(pItem[i].nHandle==27904&&strlen(pItem[i+1].sWord)==2&&strcmp(pI
tem[i+1].sWord,pItem[i+2].sWord)==0)
    {//(pItem[i+1].nHandle/256=='q'||pItem[i+1].nHandle/256=='a')&&
        strcpy(pItemRet[j].sWord,pItem[i].sWord);
        strcat(pItemRet[j].sWord,pItem[i+1].sWord);
        strcat(pItemRet[j].sWord,pItem[i+2].sWord);
        pItemRet[j].nHandle=27904;
        j+=1;
        i+=2;
        bProcessed=true;
    }
    //Rule3 for overlap words AA
    else if(nLen==2&&strcmp(pItem[i].sWord,pItem[i+1].sWord)==0)
    {
        strcpy(pItemRet[j].sWord,pItem[i].sWord);
        strcat(pItemRet[j].sWord,pItem[i+1].sWord);
         //24832=='a'*256
        pItemRet[j].nHandle=24832;//a
        if(pItem[i].nHandle/256=='v'||pItem[i+1].nHandle/256=='v')//30208='v'8
256
        {
            pItemRet[j].nHandle=30208;
        }
        if(pItem[i].nHandle/256=='n'||pItem[i+1].nHandle/256=='n')//30208='v'8
256
        {
            pItemRet[j].nHandle='n'*256;
        }
        i+=1;
        if(strlen(pItem[i+1].sWord)==2)
        {//AAB:洗/洗/脸、蒙蒙亮
            if((pItemRet[j].nHandle==30208&&pItem[i+1].nHandle/256=='n')||
              (pItemRet[j].nHandle==24832&&pItem[i+1].nHandle/256=='a')
              )
            {
               strcat(pItemRet[j].sWord,pItem[i+1].sWord);
               i+=1;
            }
        }
        j+=1;
```

```
            bProcessed=true;
        }

        //Rule 4: AAB 洗/洗澡
        else if(nLen==2&&strncmp(pItem[i].sWord,pItem[i+1].sWord,2)==0&&strlen
(pItem[i+1].sWord)==4&&(pItem[i].nHandle/256=='v'||pItem[i].nHandle==24832))
//v,a
        {
            strcpy(pItemRet[j].sWord,pItem[i].sWord);
            strcat(pItemRet[j].sWord,pItem[i+1].sWord);
             //24832=='a'*256
            pItemRet[j].nHandle=24832;//'a'
            if(pItem[i].nHandle/256=='v'||pItem[i+1].nHandle/256=='v')//30208='v'8
256
            {
                pItemRet[j].nHandle=30208;
            }

            i+=1;
            j+=1;
            bProcessed=true;
        }
        else if(pItem[i].nHandle/256=='u'&&pItem[i].nHandle%256)//uj,ud,uv,uz,ul,u
g->u
            pItem[i].nHandle='u'*256;
        else if(nLen==2&&strncmp(pItem[i].sWord,pItem[i+1].sWord,2)==0&&strlen
(pItem[i+1].sWord)==4&&strncmp(pItem[i+1].sWord+2,pItem[i+2].sWord,2)==0)
        {//AABB 朴朴素素 枝枝叶叶
                strcpy(pItemRet[j].sWord,pItem[i].sWord);
                strcat(pItemRet[j].sWord,pItem[i+1].sWord);
                strcat(pItemRet[j].sWord,pItem[i+2].sWord);
                pItemRet[j].nHandle=pItem[i+1].nHandle;
                i+=2;
                j+=1;
                bProcessed=true;
        }
        else if(pItem[i].nHandle==28275)//PostFix
        {
            if(m_uPlace.m_dict.IsExist(pItem[i+1].sWord,4))
            {
                strcpy(pItemRet[j].sWord,pItem[i].sWord);
                strcat(pItemRet[j].sWord,pItem[i+1].sWord);
                pItemRet[j].nHandle=28275;
                i+=1;
```

```
            j+=1;
            bProcessed=true;
        }
        else if(strlen(pItem[i+1].sWord)==2&&CC_Find("队",pItem[i+1].sWord))
        {
            strcpy(pItemRet[j].sWord,pItem[i].sWord);
            strcat(pItemRet[j].sWord,pItem[i+1].sWord);
            pItemRet[j].nHandle=28276;
            i+=1;
            j+=1;
            bProcessed=true;
        }
        else if(strlen(pItem[i+1].sWord)==2&&CC_Find("语文字杯",pItem[i+1].sWor
d))
        {
            strcpy(pItemRet[j].sWord,pItem[i].sWord);
            strcat(pItemRet[j].sWord,pItem[i+1].sWord);
            pItemRet[j].nHandle=28282;
            i+=1;
            j+=1;
            bProcessed=true;
        }
        else if(strlen(pItem[i+1].sWord)==2&&CC_Find("裔",pItem[i+1].sWord))
        {
            strcpy(pItemRet[j].sWord,pItem[i].sWord);
            strcat(pItemRet[j].sWord,pItem[i+1].sWord);
            pItemRet[j].nHandle=28160;
            i+=1;
            j+=1;
            bProcessed=true;
        }
    }
    else if(pItem[i].nHandle==30208||pItem[i].nHandle==28160)//v
    {
        if(strlen(pItem[i+1].sWord)==2&&CC_Find("员",pItem[i+1].sWord))
        {
            strcpy(pItemRet[j].sWord,pItem[i].sWord);
            strcat(pItemRet[j].sWord,pItem[i+1].sWord);
            pItemRet[j].nHandle=28160;
            i+=1;
            j+=1;
            bProcessed=true;
        }
    }
```

```
        else if(pItem[i].nHandle==28280)
        {//www/nx ./w sina/nx; ＥＩＭ/nx -６０１/m
            strcpy(pItemRet[j].sWord,pItem[i].sWord);
            pItemRet[j].nHandle=28280;
            while(pItem[i+1].nHandle==28280||strstr(".. ",pItem[i+1].sWord)||(pItem
[i+1].nHandle==27904&&IsAllNum((unsigned char *)pItem[i+1].sWord)))
            {
                strcat(pItemRet[j].sWord,pItem[i+1].sWord);
                i+=1;
            }
            j+=1;
            bProcessed=true;
        }

        if(!bProcessed)
        {//If not processed,that's mean: not need to adjust;
         //just copy to the final result
            strcpy(pItemRet[j].sWord,pItem[i].sWord);
            pItemRet[j++].nHandle=pItem[i].nHandle;
        }
        i++;
    }
    pItemRet[j].sWord[0]=0;//Set ending
    return true;
}
```

调整合并后的最终分词结果为：

一片片/m 的/u 白云/n 很/d 好看/a

至此，我们终于完成了整个分词的流程，得到了正确的切分结果^_^

ICTCLAS 分词系统研究（十）--后记 收藏

FreeICTCLAS 中文分词系统从 2006 年 3 月就开始接触，之后通过研读相关论文和源代码，写了一系列的学习笔记，给很多同样的中文分词爱好者提供了一个可参考的文档资料。但因为工作及其它原因（嘿嘿，说白就了就是比较偷懒，没有坚持下去），把该项目做成一个 java 版的原始想法一度中断。之后，也曾多次尝试重新拾起，完成我的一个心愿，但复杂的工作都让我半途而费。

4 月份的时候，一个爱好才 MSN 上问我相关问题，又激起了我的原始想法，同时看到吕震宇老师只用了半个月的时间就完成了 C#版本的工作，并且写了完成的系列文章，又大大刺激我的神经。想想我的系列文章只到半道，ictclas4j 的程序也是半拉子工程，真是惭愧之极。于是下定决定，一定要把这个项目完成，给自己也给关心 ictclas 的朋友一个交待。

经过三个星期的不懈努力，到现在为止，基本上完整的实现了原 FreeICTCLAS 所实现的功能。原 VC＋＋实现的程序比我想象的更复杂，中间涉及大量的临时性的数据结构和大量的全局变量，搞的我非常头大，好几次都有放弃的想法。不过谢天谢地，我这次终于坚持下来了。从 java 程序的角度从出，我对原来的数据结构做了大量调整和优化，去掉了很多不必要的中间变量。经过优化后，在整个分词过后中只用到两个对象：Atom、SegNode，原子和分词结点，整个分词过程就是对 SegNode 的不断调整和改进，最终得到分词结果。

因为到现在为止，我只是做了一些简单的测试，可能还有很多的 BUG 在里面（对标点符号的处理就是一问题），并且分词的速度还远远达不到我的要求，程序还有很多改进的地方，所以暂时源代码还不会放上来，但我想最迟一个星期之内，我可以上传到论坛供大家测试。同时，我已在 Google Code 上申请了 ictclas4j 的开源项目，期望有兴趣的朋友加入进来共同改进。

再次感谢张华平、刘群老师，你们的杰作 ICTCLAS 分词系统给国内的自然语言研究提供了一个很好的入门工具。也非常感谢吕震宇、DanceFire 精辟入理的相关分析文章！

相关参考：

ICTCLAS 分词系统论坛组：http://groups.google.com/group/ictclas

ICTCLAS for java 研究，sinboy 的 BLOG：
http://blog.csdn.net/sinboy/category/207165.aspx

ICTCLAS for C#研究，吕震宇的 BLOG：
http://www.cnblogs.com/zhenyulu/category/85598.html

DanceFire 的专栏：http://blog.csdn.net/DanceFire/category/294373.aspx

ICTCLAS 的老家：http://www.i3s.ac.cn/index.htm

ICTCLA4J 开源项目：http://code.google.com/p/ictclas4j/