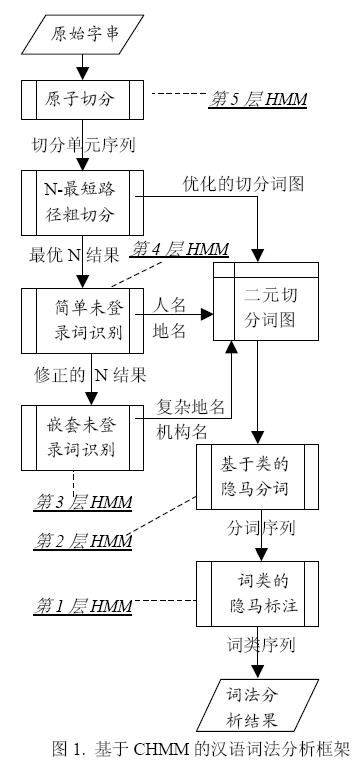
注：我将sinboy的博客中比较重要的内容整理到这个文档中，供以后参考。

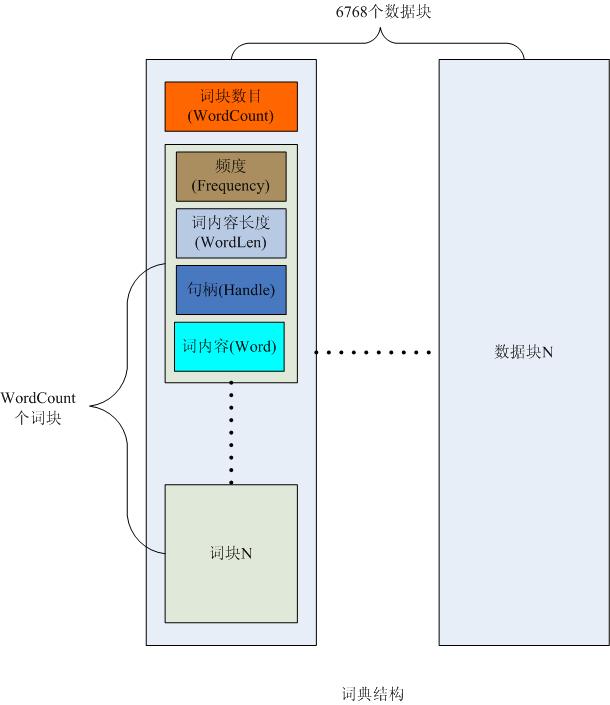
一、框架介绍

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:

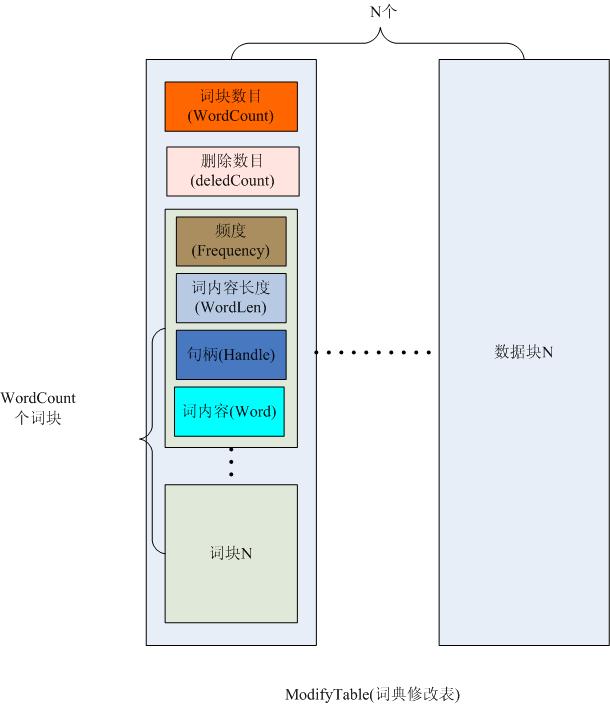


基本思路:先进行原子切分,然后在此基础上进行N-最短路径粗切分,找出前N个最符合的切分结果,生成二元分词表,然后生成分词结果,接着进行词性标注并完成主要分词步骤.

二、词典结构（详细分析见sinboy的博客）



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:



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:(鼢)鼠

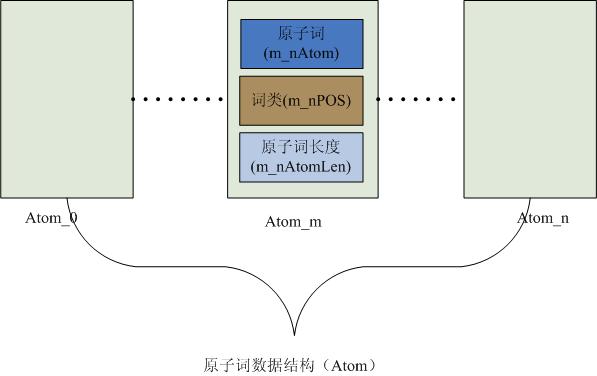
 块6761   
    count:2   
     wordLen:4 frequency:0 handle:28160 word:(鼬)鼠皮   
     wordLen:2 frequency:0 handle:28160 word:(鼬)獾

三、原子切分

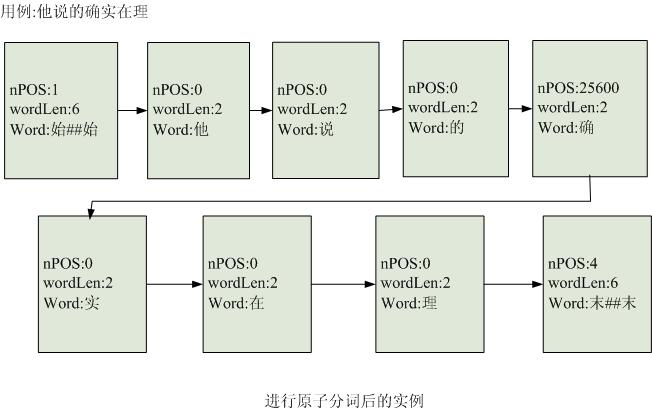
ICTCLAS分词的第一步就是原子分词。但在进行原子切分之前，首先要进行断句处理。所谓断句，就是根据分隔符、回车换行符等语句的分隔标志，把源字符串分隔成多个稍微简单一点的短句，再进行分词处理，最后再把各个分词结果合起来，形成最终的分词结果。

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。原子分割后的数据结构如下图一所示:



                                                                            图一



四、初次切分

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;

}

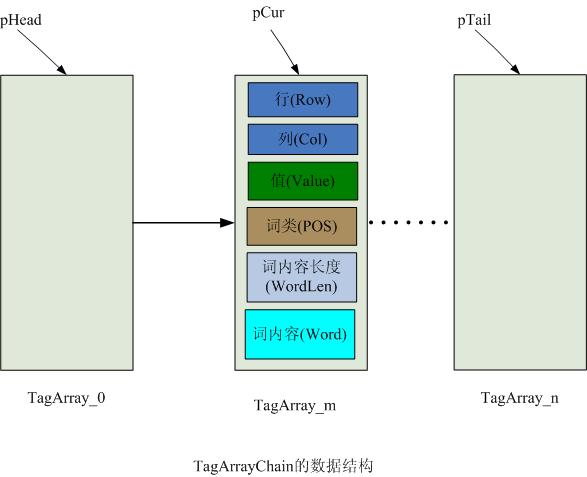
else

   break;

}

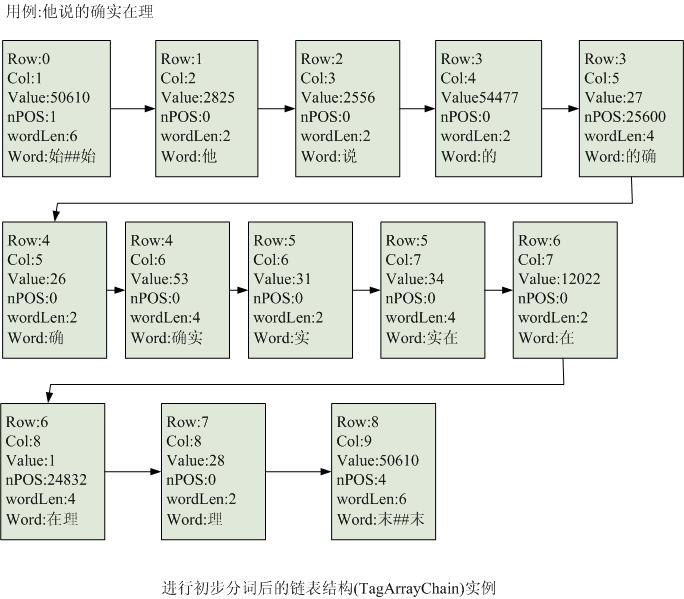
}

初次切分后的数据结构如下图一所示:



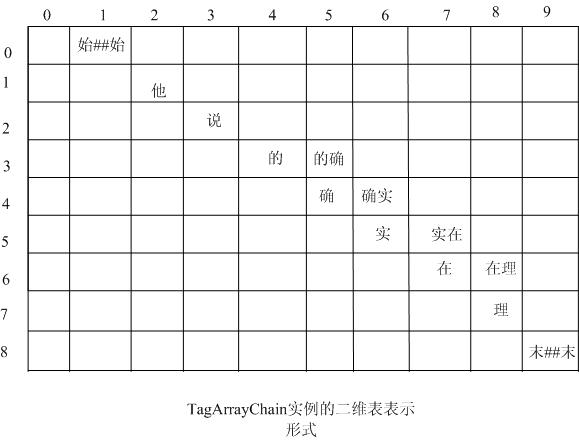
图一

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:



                                 图二

用二维表来表示图一中的链表结构如下图二所示:



                                     图三

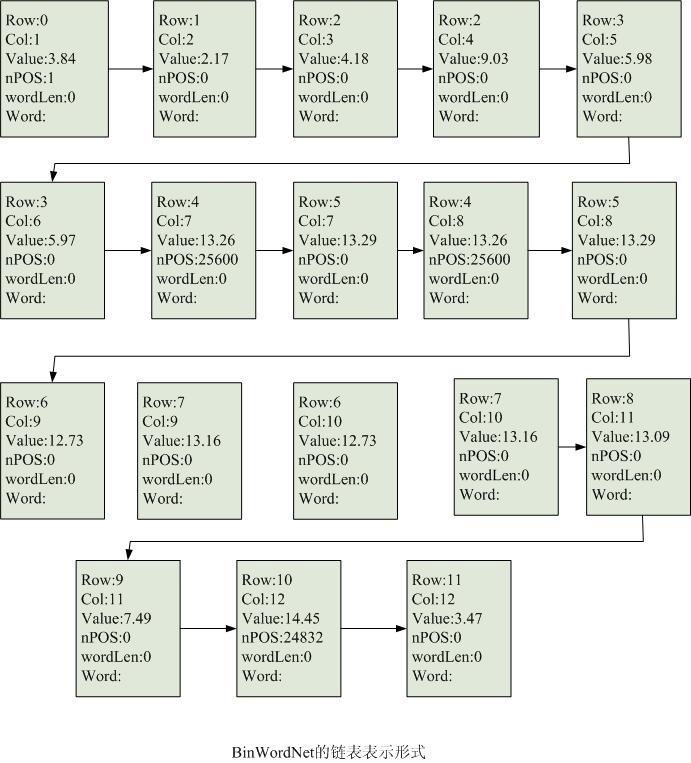
从上图三可以看出,在二维表中,初次切分后的词组,第一次字相同的在同一行,最后一个字相同的在同一列,原来的原子在对称轴上.

。。。。。。。。中间的代码分析为c++的代码，详细见sinboy博客



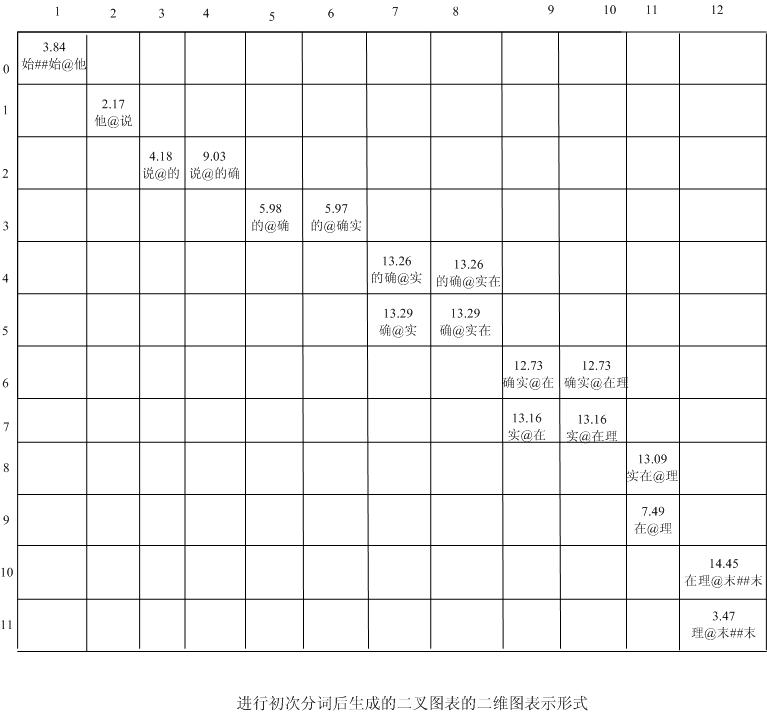
      图四

最终生成的键表结果如下图五所示:



                                             图五

对应的二维图表表示形式如下图六所示:



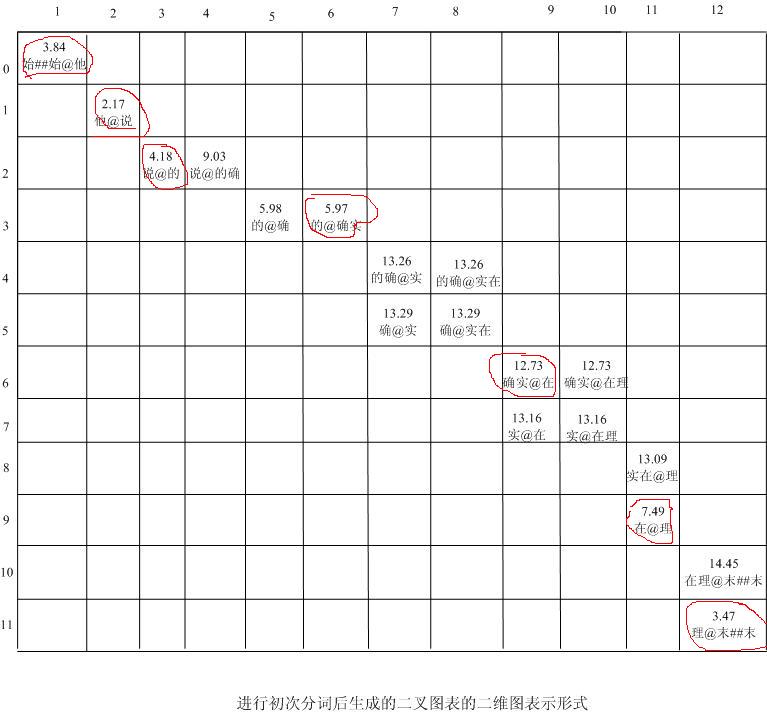
                                                                                 图六

 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。

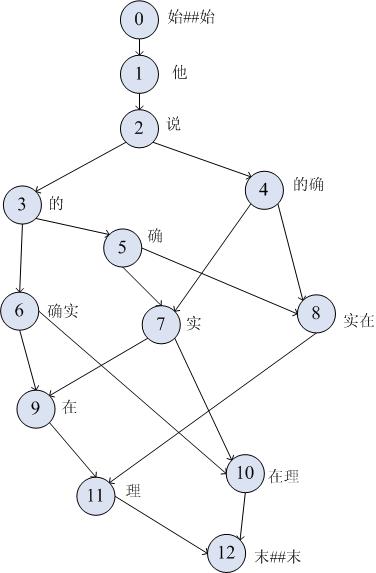
五、N-最短路径算法

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。

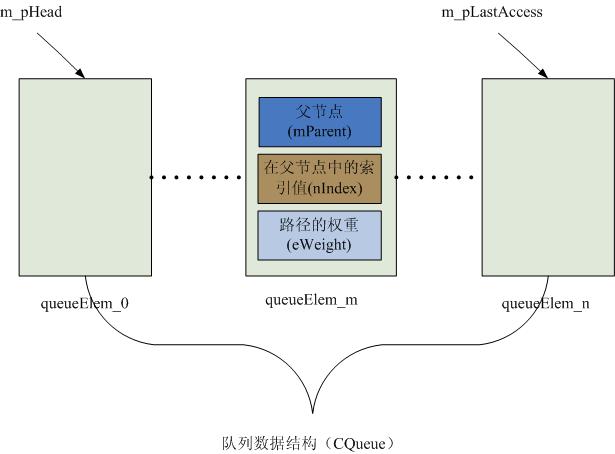


              图一



           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

六、词性标注

把连在一起的汉字按词义断开只是完成分词系统的第一步，下面还有很重要的一步是对断开后的每一个词进行词性标注。有了词性标注，才能有助于我们进一步的做语法分析等更高层面的中文理解。

源程序里面反复用到一个很重要的属性叫句柄（Handle），它应该是记录词的词性信息。

七、未登录词识别

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。

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。

八、生成最终分词

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** |

九、对最终结果进行优化处理

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** |