0　引　言  
中文分词技术属于自然语言处理技术范畴,对  
于一句话, 人可以通过自己的知识来明白哪些是  
词,哪些不是词,但如何让计算机也能理解? 其处  
理过程就是分词算法。汉语文本中含有许多歧义  
字段,句中某个片段存在两种或两种以上的切分形  
式。  
1　传统中文分词算法  
人们己经提出了许多计算机自动切分词算法,  
这些算法大致可分为两类: 理解式切分法(或称知  
识分词法)和机械匹配法(或形态分词法) 。  
  
111　理解式切分法  
理解式切分法其分词系统由词库、知识库和推  
理机部分组成。词库中存放词条; 知识库中存放形  
式化的语言规则、语法知识以及语言学家在分词过  
程中进行推理判断的常识和经验; 推理机制利用词  
库和知识库提供的大量数据与知识,模拟语言学家  
的逻辑思维过程,实现自动分词。这实际上是一个  
自动分词的专家系统。这种系统开销很大,除了理  
论上的困难外,还存在系统复杂度的问题,实现困  
难。  
112　机械匹配法  
机械匹配算法主要基于字符串匹配的原理。  
在匹配时不进行语法分析,也不进行语义分析,只  
是机械地匹配比较。即它以足够大的词库为依据,  
采用一定的处理策略将文本中的字串与词表中的  
词逐一匹配,若成功,则认定该串为词。常用分词  
  
方法有正向最大匹配法、逆向最大匹配法、最少切  
分法[ 1 - 4 ] 。机械分词法中常用的的最大匹配分词算  
法易于实现,但是存在很多明显的缺陷:第一,长度  
限制,由于最大匹配法必须首先设定一个匹配词长  
的初始值,词长过长,效率就比较低; 词长过短,长  
词就会被切错。第二,效率低,即使可以将字长设  
成相当短, 例如词长为5 , 当我们的大数词长为2  
时,至少有3 次的匹配算法是浪费掉的。第三, 最  
大匹配的并不一定是想要的分词方式,最大匹配法  
基于的理念是找到最大的匹配词,但有的时候除了  
最大匹配词外,也可能只需要这个词的一部分。基  
于以上分析,提出改进的解决方案,使分词算法的  
效率、分词的长度限制甚至歧义处理上得到提高。  
2　改进的中文分词算法  
211　建立词库  
为了使最大匹配法分词算法在效率、分词的长  
度限制甚至歧义处理上得到提高,必须要有一个词  
库,将全文中的词与词库去匹配。这需要对词库进  
行改造,使词库更适合用于匹配与分词。将关系数  
据库的词按字打散,并存放到层次数据库中。以下  
就是一个层次词库示例,如图1 所示。  
  
  
  
    
                 中  
           国^    华  
      人^        人    民  
                   民       族^  
                   共  
                   和  
                   国^  
  
  
  
  
  
图1 　层次词库示例图  
　　灰色的字表示树上面的字串是可以单独组成  
一个词的,例如“感冒”它本身是词库里可以找到的  
词,所有灰色的表示的是终止符。而白色则表示树  
上面的字串是无法单独成词的,例如“感冒解”是不  
存在的词。词库经过这样的改装后,任何一个句子  
都会打散成单字去与树状结构的单字去匹配,词的  
长度变成了树的高度,每一次的匹配变成了树的遍  
历,而这种遍历的效率是线性的。  
树并不是一颗二叉树, 父亲的子节点会有好  
  
  
多。尤其是第一层,会把词库中所有的首字都取出  
来作为根节点的子节点, 这意味着如果首字有  
4000 个的话,根节点就有4000 个儿子。当然随着  
树层数的增多,节点的儿子数也会减少,毕竟以“感  
冒”开头的词在整个词库也只有四十多个,而以“感  
冒清”开头的词则只有两三个了。这意味着如果设  
计得不合理, 树的匹配遍历过程并不完全是线性  
的。最坏的查找算法是O (N ) (N 代表儿子数) 。  
当然如果建词库时将儿子有序排列,再按照二分查  
找的方法,则我们的复杂度会减到O ( logN ) , 这样  
的复杂度已经可以接受了。但是还有更简单又更  
快的存储方式,那就是哈希表,毕竟在哈希表里查  
找东西时它的效率几乎是线性的,而且实现起来要  
比二分查询简单得多。当然用哈希表要付出存储  
空间变大的代价,但这样的代价来换取速度与简单  
性也是值得的。找到有终结符的字后,必须要将它  
建成一个完整的词。这时必须能从字个往上回溯,  
直到找到根结点。因此我们在每个节点里都保存  
了父节点的指针。  
212　算法设计思想  
有了以上的中文词库后,再来看一下分词的步  
骤:  
( 1 )首先将要分的全文按标点符号打散成一个  
一个的句子。  
( 2 )开始将要处理的句子在树状结构中遍历,  
如果找到匹配的就继续,如果遇到灰色的终止符,  
我们就发现这个词是一个完整的词了,这样就可以  
把这个词作为一个分词了。  
( 3 )从分词后的下一字开始继续做步骤( 2 ) 这  
样的遍历,如此循环往复就将词分完了。  
可以看到,字符匹配效率几乎是线性的。取出  
每一个字去树上找到相应的匹配,每次的匹配代价  
都是O ( 1 ) (如果词库用Hash 表的话) , 这样匹配  
下来的时间复杂度就是字符串本身的长度。对于  
一个长度为n的字符串来说,它的分词复杂度是O  
( n) 。而最大匹配的平均复杂度是O ( n2 ) 。当然  
这里没有考虑歧义包容与分支处理等情况,即使加  
上这些复杂度仍然是有限的。  
当遇到分支时, 会分解成两条路线, 例如当匹  
配到“感冒”的“冒”时,我们会发现一个终止符,代  
表“感冒”是一个完整的字,将它收录到分词中。接  
下来我们会分成两支,一支是继续往下走,匹配树  
的下一层,因为“冒”不是树的叶子,往下走可能会  
碰到更大的匹配词,例如“感冒解毒胶囊”。而另一  
支则从根开始, 直接用“解”去匹配树的第一层节  
  
点,最后发现了“解毒”也是其中的一个词。  
分支虽然可以消除很多的歧义,但是显然它会  
带来副作用:导致分词的复杂度变大。如果一个句  
子很长时,分词的变化也许会呈指数级的增长。很  
多句子虽然会有很多分支,但是这些分支又经常会  
汇聚到一个点,变成一个分支。例如:“感冒解毒胶  
囊可以治感冒”, 在分词的时候可能会出现“感  
冒”,“解毒”,“感冒解毒”,“感冒解毒胶囊”等多  
个分支,但是当我们到达“囊”这个点的时候,所有  
的分支又会汇集到一起,因为接下来要处理的都是  
“可以治感冒”这个字符串。如果有办法在汇聚以  
后只处理一个分支,那么算法的时间复杂度就不会  
象原来想象的那么坏。  
而这刚好利用动态规划法来处理,将所有的子  
问题记录在公有的变量里。当遇到的子问题已经  
被处理过一次了,就直接跳过。这样节约的结果可  
以使算法复杂度得到质的改变,当然由于中文的变  
化多端,无法精确估计使用动态规划法后算法复杂  
度得到了多大的提高。下面是分词的核心算法:  
p rivate void enqueueToken ( int pos, Dictnode  
parentNode )  
{  
if ( pos > = token. term Text ( ) . length ( ) )  
{ / /如果长度超出了就表示分词结束  
return;  
}  
/ / 用动态规划法来判断从当前位置的字符开  
始分词是否已经进行过了  
if ( isTokened ( pos ) && parentNode. getLevel (  
) = = 0  
{  
return;  
}  
if ( parentNode. getLevel ( ) = = 0  
{  
setTokened ( pos ) ;  
}  
String strPrefix = token. termText ( ) . substring  
( pos , pos + 1 ) ;  
DictNode dictNode = getDictTree ( ) . buildOr2  
GetSubNodes ( strPrefix , parentNode ) ;  
if ( null = = dictNode )  
{  
enqueueToken ( pos + 1 , DictNode . Emp ty\_Node  
) ;  
  
/ /当前字无法组成词,向后移一位,继续跑  
return;  
}  
if ( dictNode . hasEnd ( ) )  
{  
dictNode . inUseCount ( ) ;  
Token curToken = new Token ( dictNode . get2  
TokenValue ( ) ,  
token . startOffset ( ) + pos + 1 - dictNode .  
getLevel ( ) , token . startOffset ( ) + pos + 1 ) ;  
tokenQueue . offer ( curToken ) ;  
enqueueToken ( pos + 1 , dictNode . Emp ty \_  
Node ) ;  
/ /截完上一个词,开始截下一个词  
}  
enqueueToken ( pos + 1 , dictNode ) ;  
/ /继续往下,看能不能组成更大的词  
}  
它是一个递归的过程,初始时我们调用的参数  
里pos为0 ,这样它就会一级一级递归下去并将所  
有可能的分词放入到tokenQueue里。  
213　中文分词的实验结果  
选择一篇2000 字的文章, 然后根据两种算法  
编写的分词器对它进行处理,中文分词实验结果如  
表1 所示。  
表1　中文分词算法耗时表  
Analyzer　　　　分词算法　　　耗时  
Lucene  
基于正向  
最大匹配法  
80ms  
ChineseAnalyzer 改进的分词算法63ms  
　　由此可见,改进算法的分词速度超过在网上找  
到的基于正向最大匹配法的Lucene分词器。  
3　结束语  
改进的分词算法实现的关键在建立词库,有许  
多的细节需要注意。①首先是词库的保存格式,最  
简单的方法是利用java 的serialization 的功能,把整  
个内存中的树状结构直接序列化成磁盘的文本文  
件,且读写的效率也会相当的高。(下转第37 页)  
  
对象,可以用SQLCA 与一个数据库连接。在创建  
事务对象前,应考虑它的使用范围,可根据需要声  
明为全局对象或实例对象[ 3 ] 。该例给出了要在同  
一个数据窗口中同时连接数据源为grades的数据  
库和SQL server数据库的程序代码:  
Transaction Tr\_ t1  
Transaction Tr\_ t2  
/ /建立事务对象  
Tr\_ t1 = Create Transaction  
Tr\_ t2 = Create Transaction  
/ /给事务对象的属性赋值  
Tr\_ t1. DBMS = "ODBC"  
Tr\_ t1. AutoCommit = False  
Tr\_ t1. DBParm = " ConnectString = ’DSN = grades’ "  
Tr\_ t2. DBMS = "MSS Microsoft SQL Server 7. x"  
Tr\_ t2. Database = " student"  
Tr\_ t2. ServerName = " 192. 168. 31. 15 "  
Tr\_ t2. LogId = "DBA"  
Tr\_ t2. AutoCommit = False  
Tr\_ t2. DBParm = " "  
/ /建立数据库连接  
  
Connect Using Tr\_ t1;  
Connect Using Tr\_ t2;  
. . . . . .  
如果需要访问多于两个数据库时,可以仿照以  
上方法,设计出同时访问多个不同数据库的程序。  
4　结束语  
综上所述, 介绍了多个数据库之间自动连接,  
限于篇幅, 对其他一些次要的内容不再讨论。程  
序在PB8. 0 下调试通过,并在实际项目的设计和使  
用中提高了软件开发效率和设计质量。  
参考文献  
[ 1 ]　陈明, 杨劲松. POWERBU ILDER 8. 0 高级编程技术  
[M ]. 北京: 希望电子出版社, 2002  
[ 2 ] 　王蓉. POWERBU ILDER 7. 0 应用开发技术详解  
[M ]. 北京: 电子工业出版社, 2000  
[ 3 ] 　网冠科技. POWERBU ILDER 7. 0 时尚编程百例  
[M ]. 北京: 机械工业出版社, 2001