输入法后台设计文档

董浩亮 2011/5/24 最后修订

目录

[输入法后台设计文档 1](#_Toc293996194)

[一、 词库设计需求 1](#_Toc293996195)

[二、 词库相关数据结构及概念 1](#_Toc293996196)

[1、 词语树 1](#_Toc293996197)

[2、 拼音树 2](#_Toc293996198)

[3、 拼音编号 2](#_Toc293996199)

[三、 词库设计格式 2](#_Toc293996200)

[1、 pinyin\_list 3](#_Toc293996201)

[2、 proot.dat 3](#_Toc293996202)

[3、 pnode.dat 3](#_Toc293996203)

[4、 phanzi.dat 3](#_Toc293996204)

[四、 词库相关算法 4](#_Toc293996205)

[1、 词语查找算法 4](#_Toc293996206)

[2、 词语生成算法 5](#_Toc293996207)

[五、 智能组句算法与数据结构 5](#_Toc293996208)

[1、 智能组句算法基本概念 5](#_Toc293996209)

[2、 智能组句中图的生成 6](#_Toc293996210)

[3、 模糊音和不完整拼音 7](#_Toc293996211)

[六、 后台架构设计 7](#_Toc293996212)

# 词库设计需求

词库设计主要要满足一下几个特性：

1. 能够判别某一个拼音是否合法，以及指出拼音序列的后继。（例如，s后面可能的后继为：h,a,e,i,o,u）
2. 在占有少量内存的情况下，能够根据拼音，查找到所有词汇。（例如，shang’hai能够查找到“上海”、“伤害”等词语）。
3. 判断某一拼音序列是否有可能加入一些拼音后组成词汇。（例如，wen’qu虽然不是一个词语，但是wen’qu’xing可能是一个词语“文曲星”，此时应该告诉调用者wen’qu可能在今后是一个词语）。

# 词库相关数据结构及概念

本节介绍词库相关的数据结构，以便读者更好得理解接下来章节出现的一些名词。

## 词语树

词语树是指，以某一拼音为根节点开始，能够往叶子方向画一条路径，路径能够组成词语的一颗树。下图描述了拼音树的基本逻辑关系，以”shang”为例：

shang

ao

hai

…

…

lv

…

hua

…

jiao

ren

zuo

da

由于词语树在实际情况中，会十分庞大，因此仅作部分关键节点以作说明。

如上图所示：词语树记录了’shang’拼音开头可能的词语，例如“shang’ao上凹”、“shang’hai上海、伤害”、“shang’hai’hua上海话”、“shang’hai’jiao’da上海交大”等等。

从根节点开始，到每一个非灰色节点的路径，都是一个可以组成词语的拼音序列（实现了需求2）。根节点到灰色节点，并不能组成词语拼音序列，但是可能在今后输入某拼音后，组成词语（实现了需求3）。在具体实现过程中，灰色节点和非灰色节点可以用一个bit作为区分。

由于每个节点下面的子节点数目可能十分巨大，遍历子节点将消耗大量时间，因此子节点在实际存储时，按照“拼音编号”（概念见下）排序。在实际查找过程中，使用二分查找算法查找子节点，以此降低算法复杂度。

所有400余个词语树会组成一个词语森林，供查询所有词语所用。

## 拼音树

拼音树与词语树类似，记录字母与拼音之间的关系，以字母w为例：

w

a

u

o

e

i

n

i

n

g

g

从根节点开始，到每一个非灰色节点的路径，都是一个拼音。

与词语树不同，每一个节点的子节点最多只有6个，因此在实现上可用长度为6的数组。由于子节点数目不多，也无需对子节点进行排序。由于拼音树占用内存很小，使用频繁，生成方便，在具体实现中，拼音树并不会持久化至文件中，而是在输入法运行初时生成，而后常驻内存。

该设计满足了需求1。

## 拼音编号

在具体实现过程中，为了减少内存消耗，查找方便，避免字符串匹配等原因。存储词语树时，并不直接存储拼音字符串序列，而是为每一个拼音赋予了一个编号。以pinyin\_list文件为准：

|  |
| --- |
| pinyin\_list文件的内容：  'a ' a  'ai ' ai  ……  'e ' e  ……  zuo z uo |

每一行表示一个拼音，每个拼音所在的行号-1就是该拼音的编号。例如：’a的拼音编号是0，’ai的拼音编号是1。

拼音编号存储在拼音树中的每个节点，如果节点为非灰色，则填写该节点的拼音编号，否则填写将拼音编号填写为“-1”。

# 词库设计格式

词库主要分为以下四个文件：

pinyin\_list：存放所有拼音。

proot.dat：记录词语树头节点信息。

pnode.dat：记录词语树具体结构信息。

phanzi.dat：记录拼音序列对应的汉字。

下面介绍具体介绍一下上述四个文件的存放格式：

## pinyin\_list

pinyin\_list是一个以ascii编码方式编码的文本文件。每一行存放一个拼音，一共存放所有的拼音。系统初始化时，将根据此文件在内存中创建拼音树和生成拼音编号。

文件格式如下：

|  |
| --- |
| 拼音0 [空格] 声母0 [空格] 韵母0  拼音1 [空格] 声母1 [空格] 韵母1  …… |

**注**：其中，如果一个拼音不存在声母，则以’记为其声母。

## proot.dat

proot.dat是一个二进制文件，存放的所有词语树的根节点信息。词语森林存放于pnode.dat（见下），根据proot.dat的信息，可以在森林中找到所需拼音开头的词语树。

proot.dat的存放结构如下：

|  |
| --- |
| 拼音0根节点在pnode.dat文件中的偏移（Integer，4字节）  拼音1根节点在pnode.dat文件中的偏移（Integer，4字节）  …… |

**注**：所有数字存放方式均以Big Endian形式，负数以补码方式存储，下同。

## pnode.dat

pnode.dat是一个二进制文件，存放的是词语森林。文件大致结构如下：

|  |
| --- |
| 拼音0开头的词语树（不定长）  拼音1开头的词语树（不定长）  …… |

由于每个词语树不定长，因此需要proot.dat指明每个拼音开头词语树对应的偏移。

每个词语树是由词语树中所有词语的所有节点组成的，词语树结构如下：

|  |
| --- |
| 节点0信息（16个字节）  节点1信息（16个字节）  …… |

**注**：如果一批节点互为兄弟节点，则在文件中连续存放。

每个节点的结构如下：

|  |
| --- |
| pinyin 拼音编号（Integer，2字节）  wordAddress 对应词语在phanzi.dat中的偏移（Integer，4字节）  maxFreqency 对应拼音出现的最大可能次数（Integer， 4字节）  childrenSize 该节点的子节点数目（Integer，2字节）  childrenAddress 该节点首个子节点在pnode.dat的偏移（Integer，4字节） |

**注1**：maxFreqency指的是此拼音对应汉字词语出现的最大次数。每个词语都有一个出现次数，以表现该词语在汉语中的常用性。例如“shang”，其可能对应的汉字有“上”、“尚”等，去出现频率最高的“上”字的频率，记为“shang”的maxFreqency。

**注2**：由于兄弟节点必须连续存放，因此取得childrenAddress，再根据childrenSize即可读出所有子节点信息。

## phanzi.dat

phanzi.dat存放的是拼音序列对应的汉字信息（可能为单字，也可能是词语）。例如pnode.dat中，”shang’hai”一词节点所指向在phanzi.dat中的位置，应当连续存放“上海”、“伤害”。

下面是该文件的具体结构：

|  |
| --- |
| 拼音序列0信息（不定长）  拼音序列1信息（不定长）  …… |

每个拼音序列的具体格式如下：

|  |
| --- |
| 拼音序列的长度l，例如“shang”为1，“shang’hai”2（Integer，2字节）  拼音序列对应词语个数s，例如“shang’hai”为2（Integer，2字节）  词语0信息（2\*l + 4个字节）  词语1信息（2\*l + 4个字节）  ……  词语l-1信息（2\*l + 4个字节） |

每个词语的具体格式如下：

|  |
| --- |
| 对应汉字信息（每个汉字两字节，按Unicode方式编码，共2\*l个字节）  该词语出现的频率（Integer，4字节） |

# 词库相关算法

本节主要描述词库相关算法，分为两部分：第一部分描述根据词库和输入的拼音序列，查找相应词语的算法过程；第二部分描述如何生成词库文件。

## 词语查找算法

以“上海”为例：用户输入拼音”shang’hai”

根据“拼音树”将“shang’hai”，转成对应的编号：289,113

在proot.dat中，查找到289号拼音对应的偏移：offset1

在pnode.dat中，跳过offset1个字节，读取289号拼音节点信息，包括了其子节点开头位置信息：offset2，和节点大小size

在pnode.dat中，在offset2位置上读取size\*16个字节

使用二分查找法，查找到113号节点信息，并且读取其wordAddress信息：offset3

在phanzi.dat中，在offset3位置上读取到拼音长度l=2和s

读取s\*(l\*2 + 4)个字节，最终得到两个词语“上海”、“伤害”

## 词语生成算法

词库生成器将读取pinyin\_list.txt文件和一个名为word\_list.txt的文档。其中word\_list.txt记录了所有拼音、词语、词频的数据，是一个文本文件，示例如下：

|  |
| --- |
| 汉字 [空格] 拼音序列 [空格] 出现频率  如：  夷 yi 4  夷门 yi'men 0  夷为平地 yi'wei'ping'di 2 |

词库生成器，将所有的词语和拼音读入内存（经计算消耗约几十兆内存），然后根据上述的数据结构，在内存中生成相应的词语树和拼音树，再按要求进行排序。上述过程并不复杂，这里不再累述。最后将内存中的数据，根据上述文件结构持久化为文件。

在持久化过程中，pnode.dat有一个特殊的规定，即兄弟节点必须连续存储。这一问题的解决方式是，在存储词语树时，应当采用“层序遍历”的顺序进行节点遍历，不能采用一般的遍历顺序。层序遍历使用的是队列结构对树进行遍历，相关细节可以参考网上的算法教程。

# 智能组句算法与数据结构

本节主要介绍描述智能组句算法以及相关数据结构。由于涉及到词库部分的功能特性，因此建议阅读完本文档词库部分后阅读。

## 智能组句算法基本概念

有了词库的相关设计后，可以在一个拼音序列中，很方便得查找到这个拼音序列中，所有可能组成词语的子序列（稍后部分将描述具体过程）。以拼音序列 ce’shi’pin’yin’shu’ru’fa 为例，可以查找到的相关词语子串为：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ce | shi | pin | yin | shu | ru | fa |
| 测，册…… | 是，时…… | 拼，品…… | 阴，颖… | 树、书… | 入，如… | 法，发… |
| 测试，侧室 | | 拼音 | | 输入 | |  |
|  | 视频，食品 | | 引述 | |  |  |
|  |  |  |  | 输入法 | | |

需要说明的是，表格中每个字或词都有一定的出现频率（表格中未列出）。表格中各个词语的排列组合均有可能组成句子。智能组句算法的目的是，根据词语的长度以及词语出现的频率，组合用户最有可能输入的句子，减少用户选字。

上述表格可以很方便地转换成下面的有向无环加权图：

整句图.wmf

上图中每条边都有一个权值，权值设定的原则是，两个字词权值大于一个字，三个字词权值大于两字词。如果词语的字数相同，则选取在汉语中出现频率最高的字或词。所谓智能组句，就是在这个图中寻找到一条从“ce”结点，到“END”结点最长路径。使用一般的最长路径算法即可。

## 智能组句中图的生成

上述的图中所包含的词语数量比较多。下面介绍如何利用词库的设计来生成上面的图。为了简化系统，需要做一些限制。假设最长的句子长度为：MAX\_SETENCE\_LENGTH，最长的词语长度为：MAX\_WORD\_LENGTH。这里以MAX\_SENTENCE\_LENGTH = 7，MAX\_WORD\_LENGTH = 4为例，同样是 ce’shi’pin’yin’shu’ru’fa 描述具体实现过程。

先定义一个二维矩阵searchMatrix[MAX\_SENTENCE\_LENGTH][ MAX\_WORD\_LENGTH]。该矩阵的填写的值是searchMatrix[i][j]的含义是，在拼音序列第i个拼音开始，连续j+1个拼音序列，是否能够组成词语或者接下来能否组成词语（词库需求2），如果能够，则填写词语结点所在pnode.dat中的offse，不能则填写-1。

该矩阵生成算法如下，第一步先将拼音序列中的各个拼音，根据proot.dat的信息，填写第0行的在pnode.dat中的offset。（以下数字仅作示意，与实际并不相符）。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ce | shi | pin | yin | shu | ru | fa |
| 0 | 100 | 200 | 900 | 500 | 400 | 300 | 1000 |
| 1 |  |  |  |  |  |  | -1 |
| 2 |  |  |  |  |  | -1 | -1 |
| 3 |  |  |  |  | -1 | -1 | -1 |

然后填写第1行的在pnode.dat中的offset，填写方法是，填写第j列时，根据第0行的offset以及第j+1列所代表的拼音，在pnode.dat查找到相应的结点，然后通过二分查找法查找其是否有名称为j+1列拼音的子节点。如果有则填写子节点的offset，如果没有则填写-1。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ce | shi | pin | yin | shu | ru | fa |
| 0 | 100 | 200 | 900 | 500 | 400 | 300 | 1000 |
| 1 | 1300 | 1500 | 1700 | 1100 | 1800 | -1 | -1 |
| 2 |  |  |  |  |  | -1 | -1 |
| 3 |  |  |  |  | -1 | -1 | -1 |

接下来依次填写第i行offset，一般的在填写m[i][j]单元格的算法是，如果m[i-1][j]=-1则m[i][j]=-1，否则：根据m[i][j]的数值读取pnode.dat中对应结点的所有子节点，然后查找是否有名称为第i+j拼音的子节点，如果有则填写子节点的offset，否则填写-1。

最终生成的矩阵如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ce | shi | pin | yin | shu | ru | fa |
| 0 | 100 | 200 | 900 | 500 | 400 | 300 | 1000 |
| 1 | 1300 | 1500 | 1700 | 1100 | 1800 | -1 | -1 |
| 2 | -1 | -1 | -1 | -1 | 2000 | -1 | -1 |
| 3 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

上述矩阵叫搜索矩阵（searchMatrix），顾名思义，是用来简化词语搜索的矩阵。但是搜索矩阵并不记录拼音序列对应的具体词汇。该工作是由addressMatrix [MAX\_SENTENCE\_LENGTH][ MAX\_WORD\_LENGTH] 所完成的。

addressMatrix矩阵行列的含义与searchMatrix一样。代表的是s[i][j]填写的数值是从第i个拼音开始，以j+1为长度，所组成的词语在phanzi.dat中的offset。填写addressMatrix的方法与searchMatrix一样，具体实现时，与searchMatrix同步填写。

同样为了描述图中的权值信息，还有一个freqencyMatrix [MAX\_SENTENCE\_LENGTH] [ MAX\_WORD\_LENGTH]，含义与填写方式任与searchMatrix相同。

## 模糊音和不完整拼音

为了实现模糊音功能，需要将searchMatrix和addressMatrix扩充一个维度，变成一个三维数组。其中扩充的维度长度未知（具体实现时，用LinkedList实现）。以用户选择了in=ing模糊音为例。searchMatrix应当改成：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ce | shi | pin | yin | shu | ru | fa |
| 0 | 100 | 200 | 900,950 | 500,480 | 400 | 300 | 1000 |
| 1 | 1300 | 1500 | 1700 | 1100 | 1800 | -1 | -1 |
| 2 | -1 | -1 | -1 | -1 | 2000 | -1 | -1 |
| 3 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

上述矩阵由于模糊音的关系，pin和yin的第0行，除了要填写pin和yin本身应填写的offset，还应当填写上ping和ying填写的offset。

接下来几行的填写算法需要进行略微的修改。填写第s[i][j]格时，需要遍历s[i-1]中所有的元素，以及查找其和第i+j个拼音及其拼音模糊音是否可能组合成词语。具体修改只需加入两个for循环，本身并不复杂，不在此累述。

由于维数的增加，可能影响智能组句的性能。为了解决这一个问题，定义一个优先矩阵：firstMatrix[MAX\_SENTENCE\_LENGTH] [ MAX\_WORD\_LENGTH]，当模糊音功能关闭时，其填写的数值与addressMatrix一样。当模糊音开启时，其填写的是addressMatrix中第三个维度对应频率最大的词语的offset。

# 后台架构设计

后台本身架构不复杂，仅涉及少数文件。下面对这些文件作些说明：

PinyinContext：核心算法，负责智能组句，以及维护所有的矩阵。

PinyinRoot、PinyinNode、PinyinHanzi：分别负责读取proot.dat、pnode.dat以及phanzi.dat，包含了读取的算法以及少量Cache。

PinyinTree： 负责拼音树的构建以及使用。

PinyinUtils：一些工具类，比如通过Big Endian方式读取整数等。