拼音输入法说明文档

Z 输入法, V 1.0

周昊一,张绍文,郑健,张昊,周佳雨

注:本文档是多人合作完成的,因此语言风格上会有一些差异,可能还有一些重复,其中一至四是一部分,五是一部分,六是一部分.组员学号,具体分工等在七中有说明.

一.简介

Z 输入法(暂定名)是一个基于 NLP 课程内容完成的输入法,涉及到许多自然语言处理方面的内容.目前版本是 1.0, 主要涉及了词法分析,语法分析等方面的内容,后面的版本会加入一些统计方面的内容,如词频,语言模型等.

Z 输入法是基于 Android 平台的,开发语言为 Java 和 C++(JNI).开发工具为 eclipse with ADT.通过 GitHub 来进行版本管理和协同工作.

二.音节切分

1. 什么是音节切分

所谓音节切分就是将用户输入的拼音串切分成一组一组的拼音。比如对于输入”mingtian”,我们期望得到的结果是”ming’tian”；对于”ziranyuyan”，期望的结果是”zi’ran’yu’yan”。

之所以需要音节切分，是因为我们的输入法是基于词库而建立的，而词库中的每个词都是用它所对应的拼音作为 key 进行索引。比如说“我们”这个词对应的 key 就是”women”(也可以是 wo’men,依具体实现而定)。

如果用户只是需要输入”我们“这个词，并且他输入的就是”women”这串拼音,那实际上是不需要音节切分的,因为可以直接从词典中插到对应的词。但是假如用户要输入“我们要”这样一个短语，因此他键入了”womenyao”这串拼音，并且假定我们的词库中没有“我们要”这个词，那么我们希望至少能将输出最接近它的东西，即“我们 yao”。如果不进行音节切分，就要先尝试搜索”womenyao”，再搜索”womenya”，再搜索”womeny”,最后直到”women”时,才能找到匹配的词语.而进行音节切分后只需要两次搜索.

并且在涉及到简拼时,也必须需要音节切分的操作.比如用户输入”wom”,这是我们要按照一定规则对其进行预测,或者说补完.最简单的做法就是先进行音节切分,变

成”wo’m”,然后对其中的非终结音节”m”进行扩充.扩充结果是一个集合,包括”women”, ”womeng”, ”wome”等.

2. 音节切分的实现

音节切分的是具体实现是基于一棵拼音树来实现的.树除了根节点外的每一个节点表示一个拼音字母,从根节点(根节点为空)到这个节点的拼音路径即为这个节点对应的拼音音节.

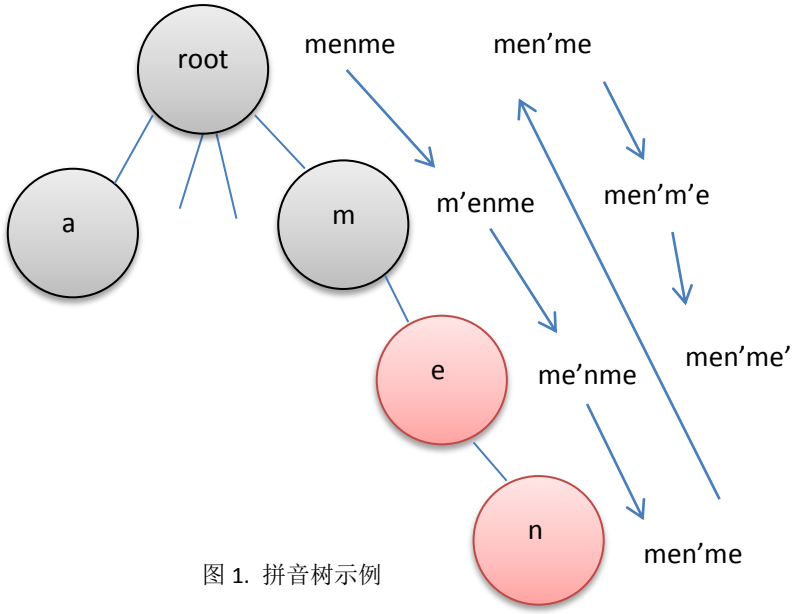

图 1. 拼音树示例

拼音树的构建需要一个指出了所有合法拼音的列表,对于这个列表的中的每一个拼音,在树中为其构建一条路径,并将其最后一个字母所在的节点标记为终结结点(用红色表示) .

而查找过程和树的建立过程类似,不断查找对应下一个字母的输入,当遇到一个叶子节点,或者输入已经完结,并且当前节点是一个终结节点,就表明这个音节被切分完毕.如果当输入已经完结时,所在节点不是终结结点,则表明这个音节不完整,需要被扩充.

a) 音节的扩充

如上一节中所提到的,有时会切分出一些不完整音节,这时需要对音节进行扩充,以提供对简拼的支持,这一部分内容会在简拼部分具体讲述.

b) 歧义

进行音节切分的时候是一定会遇到歧义的,比如”xian”,可以切成”xi’an”(西安),也可以切成”xian”(先).而用上述方法,其实是进行最大匹配,因此一定会切成”xian”.但是这样也并不会带来太大的问题,因为即使切成”xian”,候选词中还是能够得到”西安”这个词的,具体见词典部分的说明.

三.词典

词典的功能是很清楚的:对于一个给定的拼音输入,给出一个候选词集合.

对于这个功能,有多种数据结构可以实现,刚开始时曾经考虑过基于哈希树的设计,每个节点表示一个拼音音节,节点内包含从各节点到这里的音节路径对应的候选词.每个节

点的子节点用哈希表的方式进行索引,key 就是对应的音节.这样的结构事实证明不论是构建还是查找都是很快的(在开发机上用 java 编写的代码从开始构建一个 9 万词的树到完成一次查找耗时不到 100ms),但是内存占用较多,9 万词的词库文件是 1.6m,整棵树大概需要 27m 的内存.虽然理论上这样的树结构去掉了一些冗余信息,应该更小才对,但是每个节点用到的各种数据结构和对象的额外开销太大(优化后还有 200 字节左右),是节点本身存储信息(10 字节左右)的十几倍,因此才导致这么大的空间占用.

如果是在 PC 上实现这样一个输入法,其实是不用考虑这些内存占用的(27m 内存对于当前的主流配置来说可以忽略不计).但是无奈我们是在 Android 进行开发,在不同配置的机型下 dalvik 虚拟机限制每个程序的最大堆空间在 40m~50m,如果光一棵树就占去了 27m,再加上一些其他的对象,以及 UI 方面的资源,很轻易的就会超过这个上线.事实上我们在优化之前连 3 万词的词库都无法顺利建成(out of memory),并且及时在优化后,总占用空间也达到了 38m 左右,在建树过程中不停地要进行 gc 操作(因为初始时整个堆空间只分配了大约 10m),使得整个过程非常慢,大约需要 10s 左右,对于用户来说是不能忍受的.

为了解决内存上的问题,最后还是改为使用哈希表来进行词典的表示,但是每个词的 key 不是它所对应的拼音,而是它所对应的拼音所计算出来的 code,是一个长整型.公式很简单,就是把拼音的每一个字母从 a~z 分别当成数字 1~26,把整个拼音串看成是一个 26 进制数,将它转换成 10 进制后得到的就是对应的 code.这一方面是为了节省空间(即使用 char 类型来表示,平均每个拼音串的长度也是超过 7 的,再加上'\0'还占去一位,就超过 long 的 64 比特了,更别提用 string 来标示了).这样还有一个好处,就是在分析词典文件时与查询候选词时,可以用长整型的传递来代替字符串的传递,可以省去构建字符串的时间.

由于候选词的 key 是拼音串对应的 code,因此和拼音的切分是无关的.因此正如切音部分中提到的,对于"xian"这个输入,既能查到"先",又能查到"西安".

最后再提一下,由于在 dalvik 虚拟机下不仅遇到了内存问题,执行效率也遇到了很大的问题,对于一个 9 万行的词典文件,光是这个 9 万次的循环就需要 2s(在循环体是空的情况下).并且开始时还使用了 String 中的 split 方法,事实证明它也非常的慢,并且还很占空间,但是即使重写了一个用来解析词典每一行的方法后,总体执行效率仍然不理想.最后没有办法,只能把词典这部分的代码迁移到 C++上,通过 JNI 来完成 java 中其他部分的代码和词典的通信.

四.简拼

1. 简拼的完整功能描述

简单的说,简拼就是在不输入完整的拼音的情况下,要求输入法能输出用户所期望的词语.具体说的话,则有以下几个方面:

- 对音节的扩充,比如输入一个”w”,要能扩充成{“w”,”wa”,”wen”,”wo”…}
- 首字母输入,比如输入”zhhrmghg”要求可以得到结果”中华人民共和国”
- 混合输入(全拼占多数),比如”womenyqi”要求能得到结果”我们一起”
- 混合输入(简拼占多数),比如”zhrmo”要求能得到”走火入魔”

这四点至少在我们目前输入法的实现下是不同的四种情况.

2. 目前的实现

第一点实现了部分.方法是通过音节切分,找到不完整的音节,并进行猜测(当前是将所有情况全部试一遍).但是用这种方法基本只能做到一个音节的还原,因为平均每个简拼对应的全拼大概有 10 个,如果要还原两个,搜索空间就是 100,而平均每个全拼对应的字词至少有 20 个,这样的话候选词就有 20*100=2000 个之多,从效率上来说无法接受(当前再只还原一个的情况下在有些时候都已经能明显感觉到卡顿,可能是因为 JNI 在 C++和 Java 交互时的效率问题,毕竟必须要进行一次数据的转换,从 C++的字符串变成 Java 的对象,以后会考虑将这部分也放到 C++中,尽量减少 C++和 Java 的交互).

当然即使是在 PC 机上,这种方法也是不太可取的,因为每多一个还原的音节,搜索空间就上升大约 10 倍.所以以后的版本的改进肯定是要基于一个新的预测词库进行预测,这个词库的生成还需要对现有词库以及大量的语料库进行分析,找到一些高频的组合.

第二点则是已经实现了,这一点在目前的数据结构下很好实现,只要把每个词的首字母简拼也作为 key 再存一遍就可以了.

第三点实在第一点的基础上实现的,就是简单的把所有的组合都搜索一遍,存在的问题也一样,就是搜索空间增长太快,需要和第一点做同一个方向的改进.

第四点目前没有实现,目前估计可以通过对字的 code 进行反查的方式进行实现.即先通过简拼查出所有的可能,如上面的例子中就是先通过”zhrm”查出所有可能的词,再对”zhrmo”进行切分(有两种切法,分别对应前面查出来的不同字数的候选词),再反查到所有候选词中对应”mo”这一位的字的 code(可以通过一个额外的字典表来实现),将所有能匹配的候选词选出.

五.关于 Android 平台

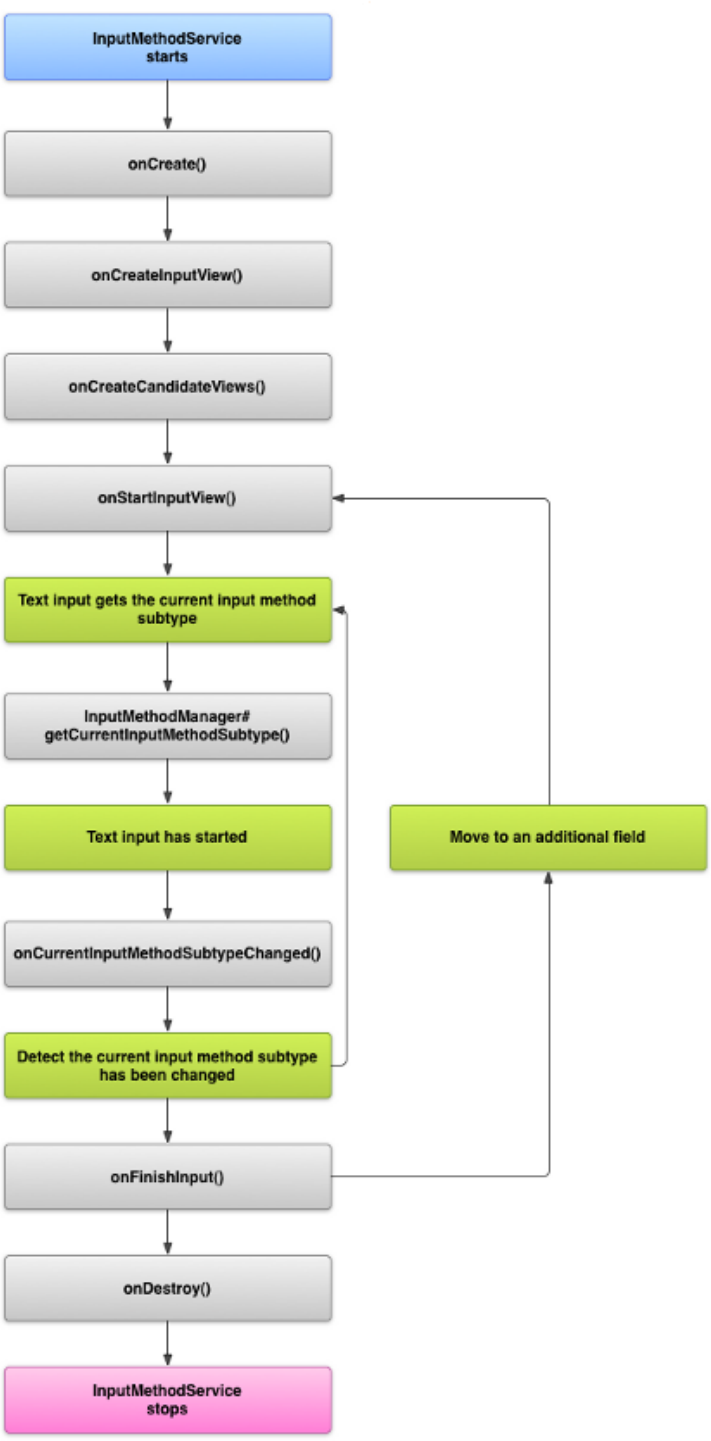

图 2. 输入法的生存周期

在这个生存周期中,我们的输入法内核位于 onStartInputView()到 onFinishInput()这个循环中.前面的三个步骤都是和 UI 初始化有关的.

在初始化时,我们建立好用于拼音切分的树以及词典.当用户每次输入一个字母时,需要对当前的整个输入串进行一次处理.处理流程为

1.切音,将用户的输入切成音节序列.

2.填充(预测),如果切音时发现了不完整的音节

3.查词,将所有的输入情况转换为候选词.这里的所有输入情况是指,对于每一个长度为 n 的音节切分,需要进行 n 次查找,每次的 key 是这个音节切分的一个长度为 k(k=1~n)的子序列.比如对于”di’qiu’ren”这个切分,需要查找”diqiuren”,”diqiu”,”di”,三种情况,余下的音节作为用户的后续输入.

4.排序,将所有的候选词按照优先级进行降序排列.

5.将所有的候选词绘制到 UI 上.

输入法与用户之间的交互是阻塞式的,用户每次输入后需要等待输入法返回结果.但是从用户体验(User Experience)的角度来说每次输入后都要等待输入法是相当不舒服的一种体验,因此我们需要将每次的响应时间缩减到用户可以忽略不计的程度(至少要在 100ms 以下,最好能在 50ms 以下).并且初始化(词典的构建)时间也不能太久,至少要在 1s 以下.这也是我们的输入法必须要采取高效的数据结构的原因,目前我们的初始化时间大概在 500ms 左右(这里说的是从选择输入法到弹出输入法界面的所有时间),用户每次输入的响应时间平均能达到 50ms,偶尔有一些简拼的场景可能会超过 100ms.

为了进一步提升用户体验,其中还采用了异步加载的方式.通过分析可以得知,虽然用户对输入法进行的操作是一个同步操作,输入法内核需要在得出结果前阻塞 UI 线程(实际上它们就是在同一个线程),但是当用户在连续输入时,其实他往往并不关心中间的结果.因此在这里就可以采用异步操作,当一个输入到来后,不阻塞 UI 线程,让用户可以继续输入,当下一个输入到来时,如果上一次的运算还没结束,就取消掉,直到一次运算完成后才将结果加载到 UI 上.通过这样的候选词异步加载,可以减少许多不必要的运算与 UI 操作,使用户明细感觉到流畅度的提升.

还有一些细节内容,主要涉及到用户体验,比如说中英文混输的处理,空格键,回车键的处理,以及退格键的行为等,这里就不具体展开了,因为和输入法核心部分关系不大.

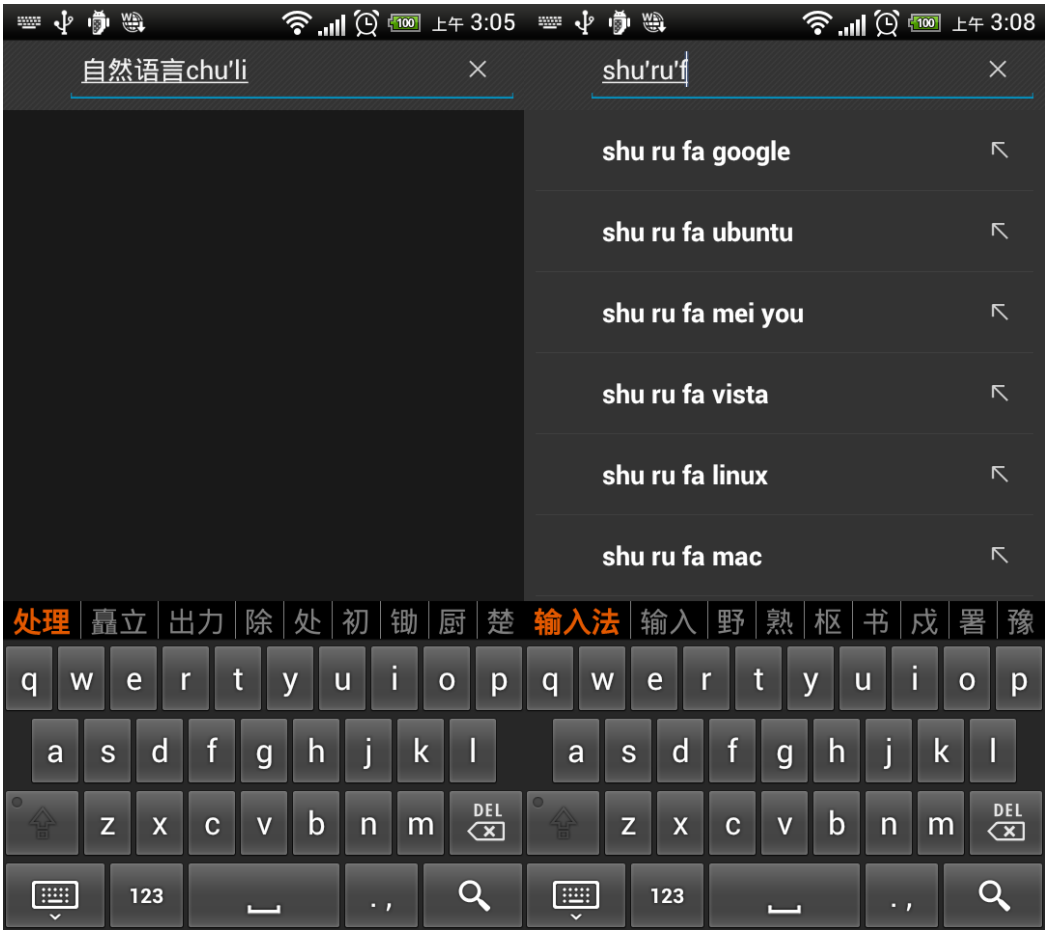

图 3.输入法的实际运行画面

六.词库文件

词库，顾名思义，就是一个包含了大量词组，短句的文字库。它将拼音组合映射到中文词组，按照某种规则，通过输入法的其他模块，最终将符合的词组返回到用户界面，供用户使用。

在设计实现输入法的过程中，词库的设计是必须且重要的一个环节。词库的好坏直接影响着输入法的用户体验。一个好的词库，可以让用户在使用时更加得心应手，加快打字速度，并且拥有一个良好的工作心情。

在设计输入法的过程中，通过小组的内部讨论，我们结合我们所需要设计的输入法的需要，我们将词库的词条格式定义如下：

拼音 1^' 拼音 2^'…拼音 n^'　　词组 or 短句　　词频

简单来说，词条的第一部分为拼音部分，每个音节间用符号“’”隔开，第二部分则为拼音部分所对应的中文词组（或者短句），第三部分则是这个词条的词频，

即为一个权重值，值越大，说明该词在文章中出现的概率越高。在词条的不同部分间我们用制表符隔开。下面举个简单的例子：hai'shi'shan'meng 海誓山盟 1。

在各个不同的拼音间用符号“’”隔开，是为了防止出现歧义，例如 xian 与 xi’an，如果没有符号进行标识，那么我们就无法快速识别这两者的区别。尽管有其它解决的方式，但在我们的设计中还是采用了这种解决方式。此外，词库中的词条是按照拼音的排序从 a~z 的顺序排列的，这样有利于后续工作的实现，减少查找的时间开销。

在我们的初始版本中，由于各种条件的限制，我们将各个单词的词频都初始化为 1，因为我们无法获得一个较为准确的词频数据。在今后的版本中，我们会修改词频的初始值。我们的设想是通过对《人民日报》进行数据挖掘与分析，得到各个词出现的词频，并在我们的词库中加入新的单词。

在这次实验中，我们共设计了三个不同大小的词库，分别为 8w，15w 以及 22w 词汇规模的词库，以满足不同的需要。同时，在这些词库中，我们添加了现在流行的网络用语及新兴词汇，同时也添加了不少的唐诗、宋词等传统文学中的名言警句，来丰富我们的词库，期望得到良好的用户体验。

七.测试

不同于英文输入法中文拼音输入法在输入识别时面临着切音，分词，选词等诸多问题，而基于规律或是基于统计的算法都不能穷尽汉语的所有可能，尤其是在本项目中基于词法分析的算法，在面对规则性不强的新词、网络词汇很有可能导致出现在输入之后机器识别出的结果与人的预期结果有着不一致的地方，基于统计规律的由于训练集的有限性也存在着不尽如人意的地方。这使得黑箱测试尤为重要。

在本次试验测试中，采用单元检测的方法，依次检测切音，分词，选词等各个环节。

在切音环节我们随机的从训练集（词库共计有 8 万个词汇）中抽取 1%，对其进行无差别切音检测，比较切音结果与预期是否相同。在比较结果中我们发现形如“xian（西安）”即第二个音无声母，会被识别成“xian（先）”而“先”这个词在日常生活中又是切实存在的，那么我们可以强制要求再输入“西安”时加入切音符“xi’an”，在接下来的工作中则会结合词频，上下文 Ngrammar 概率在“xi’an”识别的基础上加强对于“xian”基于统计规律的多重识别切音，智能识别成“xian（先）”或“xi’an（西安）”；形如“ganga（尴尬）”即的二个音的声母或起始音节被误识别为上一个音的尾音，即会被识别成“gang’a”这是我们所不能接受的，需要结合统计规律来避免这种情况。

在分词环节如键入“dajiazaoshanghao（大家早上好）”我们期望被识别为“dajia|zaoshang|hao”三个词，而由于训练集的有限性，汉语的多义性（歧义性）很有可能被识别成期望外的结果。比如“shuiguoranhoushu（水果然后熟）”我们希望分成

“shuiguo|ranhou|shu”而不是“shui|guoran|hou|shu（水|果然|后熟）”，而这些问题只有通过大量的黑箱测试，找出问题句段，并用这些句段补充训练集才能更好的分词。

选词部分目前是基于词频的统计排序尚未结合上写文，目前检测主要是检测词频的排序以及词频改变是否能及时更新。

八.小组情况

本小组共有 5 个人.

Leader 为周昊一(MF1233055),负责核心代码,以及用户界面的编写.

其余 4 个人为:

张绍文(MF123304),负责字典,词典的搜集,完善,以及后续工作中语言模型的训练.

郑健(MF1233054),同上.

张昊(MG1233046),同上.

周佳雨(MG1233091),负责测试用例的编写,并对程序进行测试.