# 修改流程

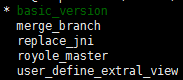
## QWERty UI修改 （royole\_master）

## extralView显示（user\_define\_extral\_view）

## candidateView界面及功能修改（replace\_jni）

## 融合2.3分支并添加修改

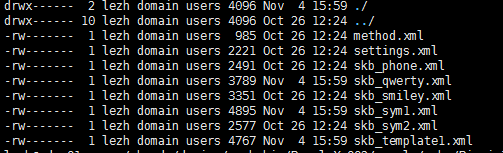
分支如下图



# QWERty UI修改

## XML文件修改。

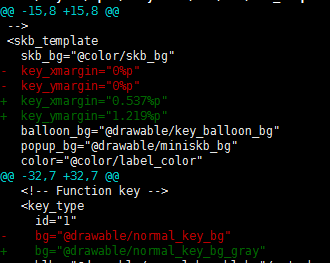
每个skb\_xml文档对应的是一个键盘布局。而其中比较特殊的是qwerty以及template。



qwerty是标准键盘，输入法的主界面。其他的很多skb都是由其切换过来的。

template是一个键盘的模板，很多键的风格以及通用的键位都是在这里面设置的。

**skb\_template1.xml**:



修改了key\_xmargin=”0%p” 以及 bg=”@drawable/normal\_key\_bg\_gray”

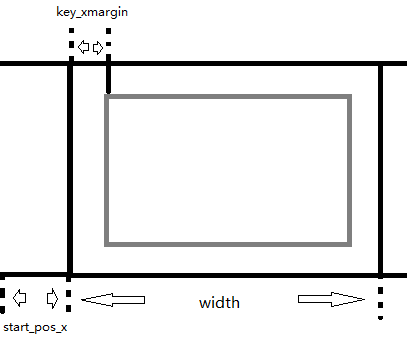
key\_type 定义了不同的按键模板种类。

以id 来区分，bg代表背景的图片。

**skb\_qwerty.xml**:（前一段）



注意到width 、height 。他们使用的都是占键盘百分比来设置大小。而对应关系如下：

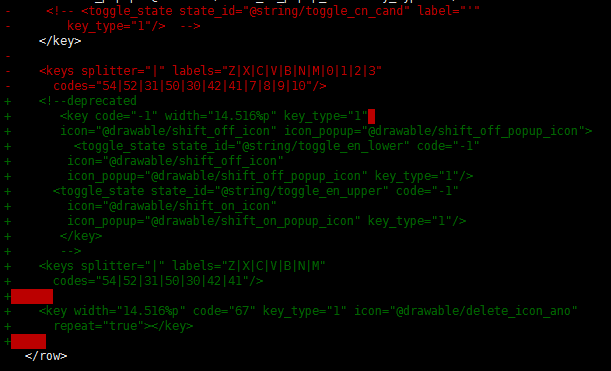


接下来注意到 start\_pos\_x 的值，这个是按照每一行的开头的偏移量。如图所示。

后面需要注意的是：template里面可以设置某个键的 id = ”4” 之类的操作，在qwerty里也可以直接使用id=“4”调用template里的键。

或者直接设置 label （当没有设置code的时候，labels即其输出的值） code icon等值（字面意思可知）。

**skb\_qwerty.xml**:（后一段）

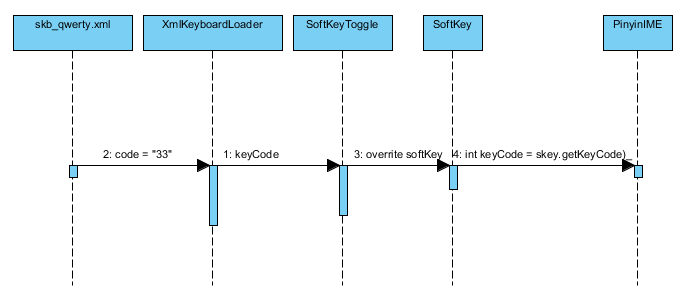


注意标签toggle\_state 用于在不同输入法状态下（如网页的输入，一般的输入等）显示界面不同功能相同的键位。

新添加的虚拟键位处理逻辑代码修改：（在InputModeSwitcher里）

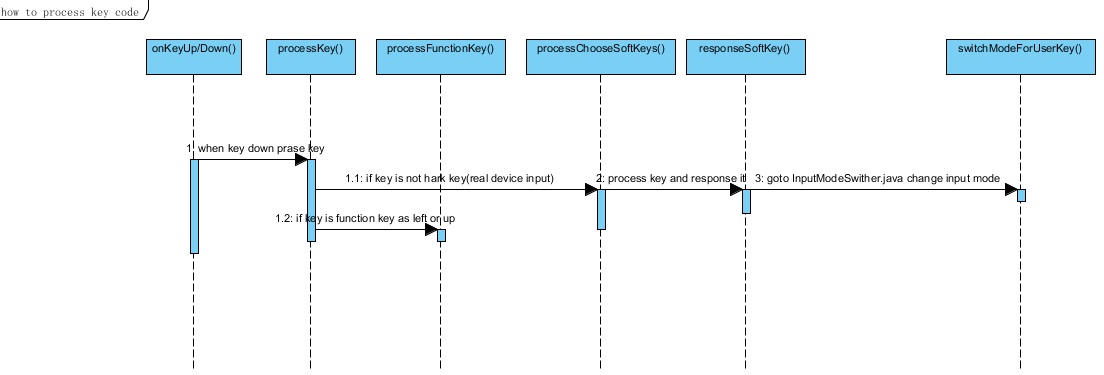


调用关系时序图：

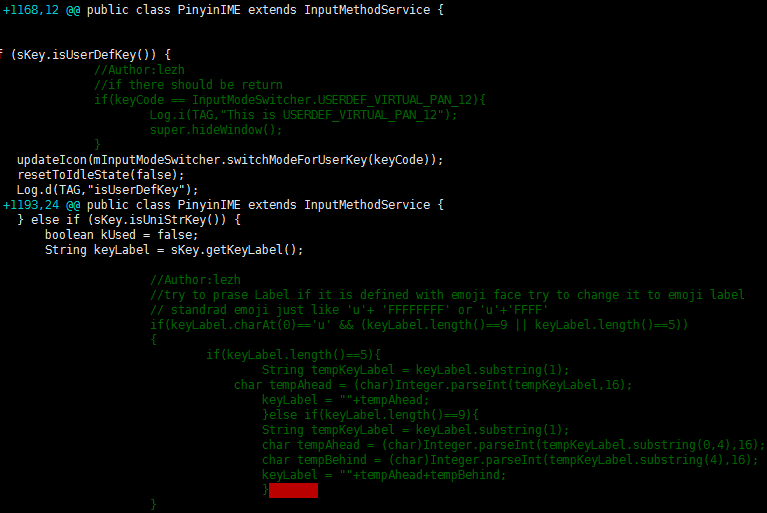


## 逻辑代码修改，添加表情功能。

处理键盘事件时序图（部分）：



修改代码如图所示：

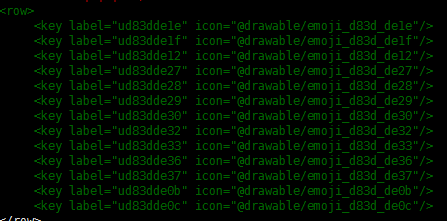
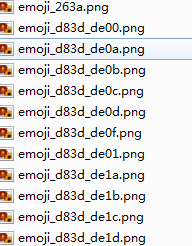


第一部分是响应虚拟按键功能的，点击之后返回。调用InputMethodService里的方法hideWindow()可以缩回，直接可以super调用。如果要查询相关可以看InputMethodService api(<https://developer.android.com/reference/android/inputmethodservice/InputMethodService.html> 但是注意，需要挂VPN) 或者模仿键盘的缩回事件。

第二部分是在responseSoftKeyEvent()方法中处理点击的表情事件。表情使用的是android自带的unicode编码，可以将表情输入到输入框中。

值得特别提出的是，android字库里的emoji表情是由两个java char型变量储存的。

 实际在内存中的编码是 D83D DE0A。只要把两个字符作为String变量输出，在android环境下会自动变成表情。而键盘界面上不能使用emoji表情，只能使用贴图。



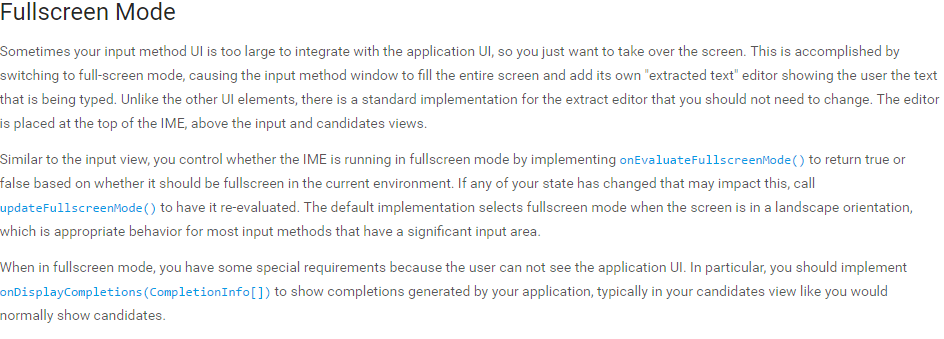
所以贴图命名如左图，在xml里如右图。

最后进行上面代码中的处理，使keyLabel变成emoji表情，最后commitResultText ()的就是表情。

# extracted View 显示

## extracted View相关知识。

1. InputMethodService中对FullscreenMode描述是：

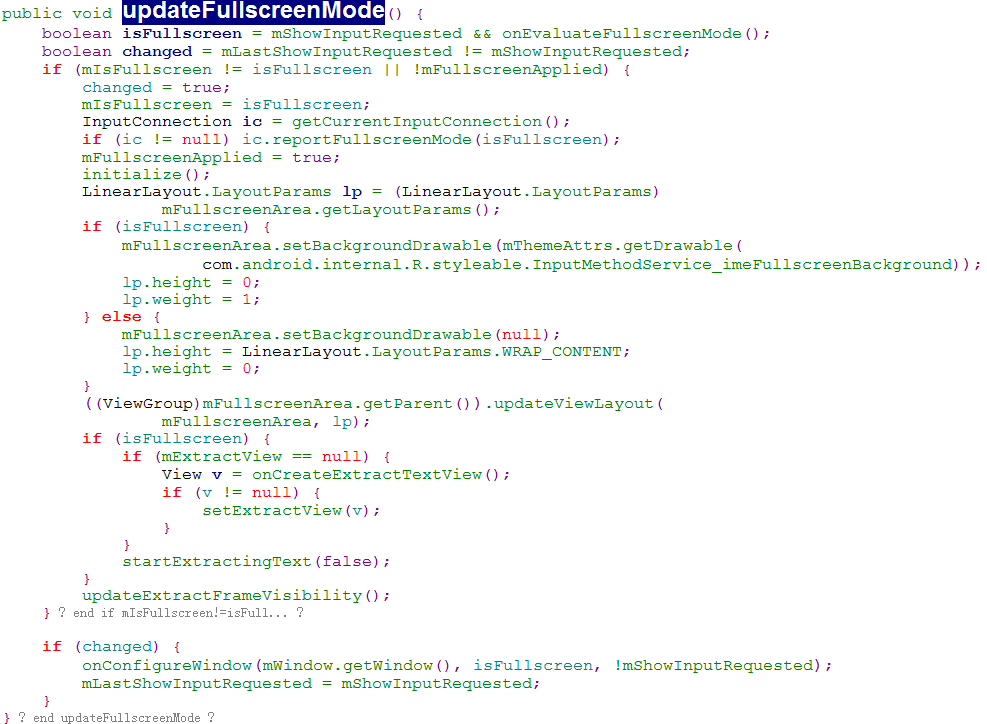


默认的是landscape条件下全屏，最上面填充的是extracted。

而InputMethodService有以下可以重载的函数：

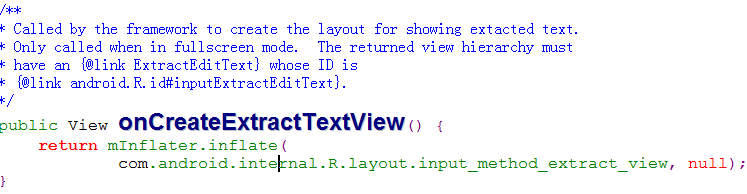
onEvaluateFullscreenMode()的返回值决定是否使用全屏模式。

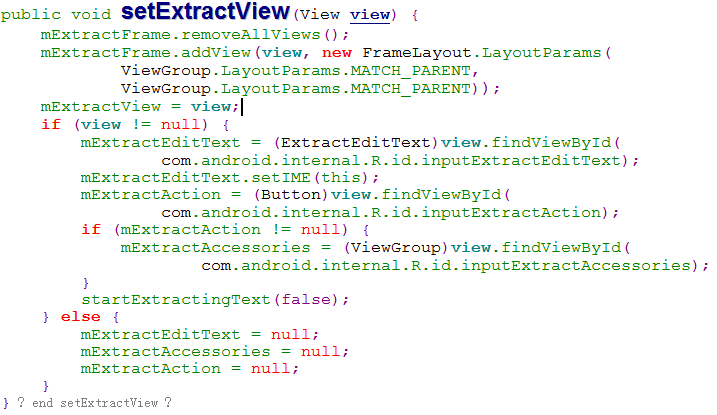
在源码里



现在需求是改动ExtractView的界面布局，所以关注源码中关于布局设置的问题：

onCreateExtractTextView()返回值是View，然后setExtractView()将View设置进去。

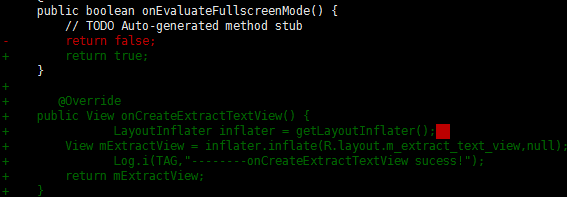




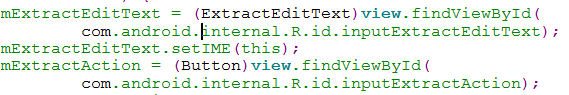
可以看到onCreateExtractTextView()的注释里说明了 返回的view里必须有名为inputExtractEditText的ExtractEditText。

1. 在代码中的尝试性改动。





实际运行时是会报错的，在setExtralView里，对传入的参数view有以下操作：



而自定义的view是没有com.android.internal.R的数据的。所以这里会崩溃。

解决方法有两个：1.反射获得setExtractView中调用的几个变量，在重写函数里重写设置它。

2.直接在framework里改变com.android.internal.R.layout.input\_method\_extract\_view的布局。

而这两个方法都有其不利之处：1.反射的稳定性不能得到保证，在代码其他地方对View有其他改动的都需要使用反射进行修改。实现起来比较麻烦。

2.对famework修改的话，调用IME其他的输入法应用如果对其进行过调整的话，有可能导致程序崩溃。

所以对ExtractView的修改暂告一段落。

# candidateView界面及功能修改

## 数据源的准备。

candidateView需要候选词库，需求是把中文候选词库换成英文候选词库。那么就需要一份英文词库。

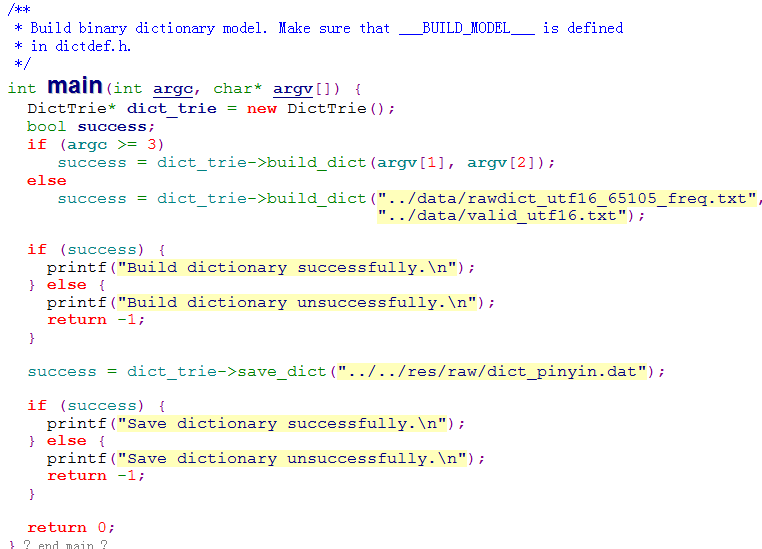
* PinyinIME数据源的生成

PinyinIME的词库的结构比较复杂，首先是字典库的来源：

PinyinIME\jni\command目录下有

PinyinIME\jni\data目录下有

pinyinime\_dictbuilder.cpp：

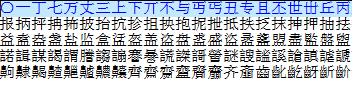


通过上一级的Android.mk设置include $(BUILD\_EXECUTEABLE)生成可执行文件，使用data目录下的两个字典库在android系统里，生成实际使用的dict\_pinyin.dat。

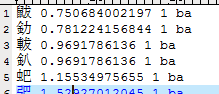
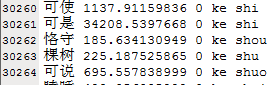
这个字典数据在PinyinIME\res\raw下存在。

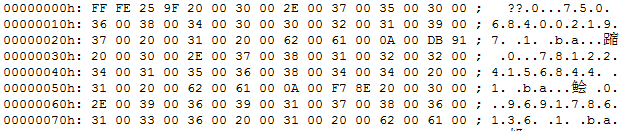
* PinyinIME原始数据的结构

valid\_utf16.txt：



rawdict\_utf16\_65105\_freq.txt：



先看上图，rawdict\_utf16中每个字或词组都有以下四个信息：

1。输出的词 （词必须是valid\_utf16中出现的，否则编译可执行文件会报错）

2。优先级 （一串字符表示的浮点型数据，数字越大，越优先显示）

3。是否有词组（猜测的功能，如果有词组并可以显示predictItem则为0）

4。输入的字符 （实际输入的字符串，而在生成字典时，如果输入字符长度短与输出的词，那么无法显示在candidateView里面。所以英文输入法无法套用中文输入法的字典生成方法）

接下来可以看到rawdict\_utf16\_65105\_freq实际编码的时候开头是FFFE，这是unicode little-endian编码格式开头。（稍后补充编码的知识）

字符编码使用的utf16格式，但是要注意的是每一行的结尾是使用的0A00(换行符)结束。

extral info:unicode & utf

unicode:

* 首先出现的是**ascll**编码，最开始在美国使用，前0x20都是控制码，用于换行，蜂鸣等功能。后面一直到0X7F都是大小写字符，数字等等。
* 接下来有其他国家开始使用，那么从0X80-0XFF都被拿来使用，称作**扩展字符集**。
* 当中文需要被编码的时候，大于0X7F的两个词合起来表示一个中文，也是全角。低于0X7F的还是表示正常的英文字符，一般是半角。这个时候的编码方式称作**GB2312**。
* 当中文中需要加入更多奇奇怪怪的字符的时候，规定只要第一位字符大于0X7F就是一个中文字符，这种编码方式叫做**GBK**，兼容了GB2312.
* 这个时候全世界的编码都乱套了，于是出现了**unicode(ucs)**，规定用2个字节来表示所有字符，包括英文字符。但是发现unicode由过于浪费英文传输时的空间了。那么为了解决这个问题就出现了**UTF**（UCS Transfer Format）。
* UTF编码的方法是1.单字节的字符，字节的第一位设0，对于英文UTF-8还是只占一个字节。与ASCLL码是一样的。2.n个字节的字符，第一个字节前n位设1，第n+1位设0，后面字节前两位设为10，这n个字节其余位填充unicode 码，高位用0补足。中文的格式是：1110xxxx 10xxxxxx 10xxxxxx
* FFFE和FEFF是unicode的little-endian和big-endian。如果有一个字符0XABCD在存储到内存时，little-endian 存储顺序是 CD AB（前面是低位）;big-endian则相反。

（unicode码表可以在<http://www.fileformat.info/info/unicode/> 查询）

* 适用与英文的候选词设计

英文输入法候选词需要的信息只有1.词名 2.优先级

在字符规模小的时候加载在内存里比较合适，但当字符很多的时候直接在硬盘里随机读写可能是更好的策略，所以在设计的时候，选择后者。

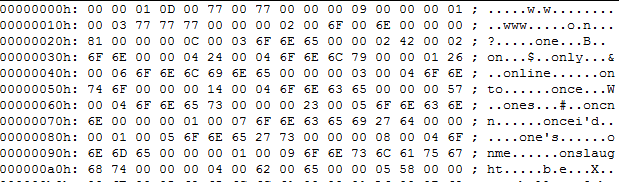
而考虑到每次搜索的时候需要从头搜到尾的效率太低，英文输入使用前两位字符作为索引进行搜索加速，当搜索到索引所在条目时，继续搜索匹配的字符，不匹配时跳过相应硬盘位置。所以设计的时候需要将候选词分成三个层次，

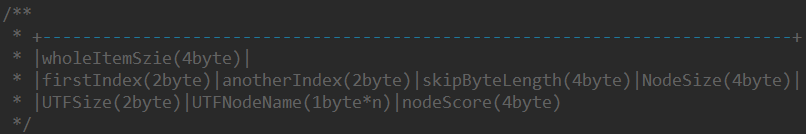
最低层次的原子结点保存1.词名2.优先级信息

条目的层次保存1.索引信息2.储存的原子结点信息

最高级的字典层次保存1.条目数量2.条目信息

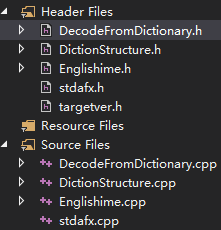
* 具体生成字典库的代码见TestProject工程 ./TestProject (java工程)
* 抓取词典的目标是txt格式英文小说，抓取规则是按照空格，冒号等分隔符分隔的整串英文字符（英文字符加上单引号）。
* 生成结果示例：



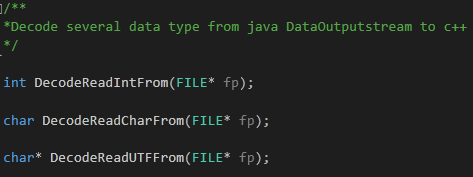


可继续优化的点：1.在字典中按优先度排序会提升平均时间开销。2.可以精简数据结构。

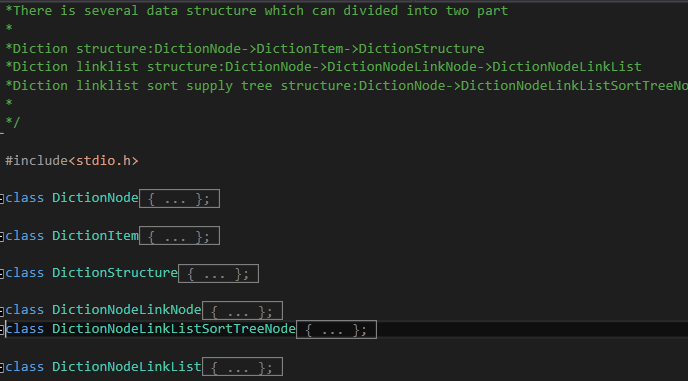
## 2．功能性代码的设计。

接下来在jni层以下的功能性代码需要重写，以适应新的字典。功能性代码在vs中进行编写，调试。 结构如图：

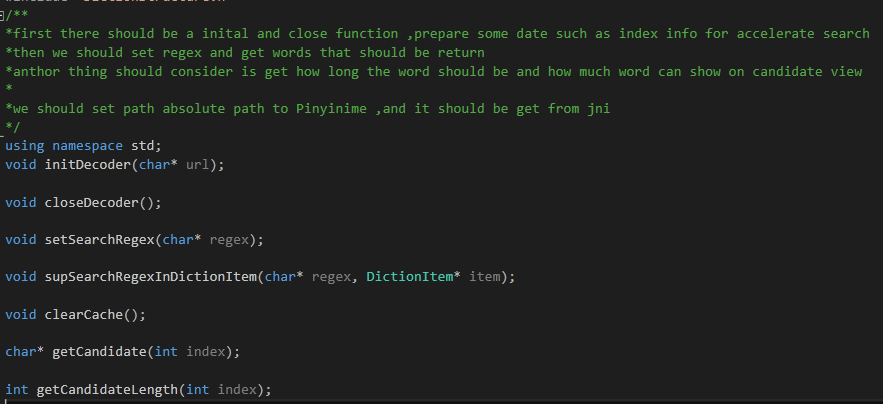
* DecodeFromDictionary（负责读取字典库里的信息，将java生成的字典读取成c++的数据，解析int,char,utf等数据类型）



* DictionStructure（负责构造字典数据结构以及排序字典数据结构，排序用堆排序构造结点以及树，最后进行搜索）



* Englishime（负责总的处理数据结构并提供接口）



一些代码中需要注意的事项记录在 ./ c++问题.txt 里。

## 3．java与jni层替换。

将功能性代码完成之后，需要将功能性代码嫁接到原有代码上。

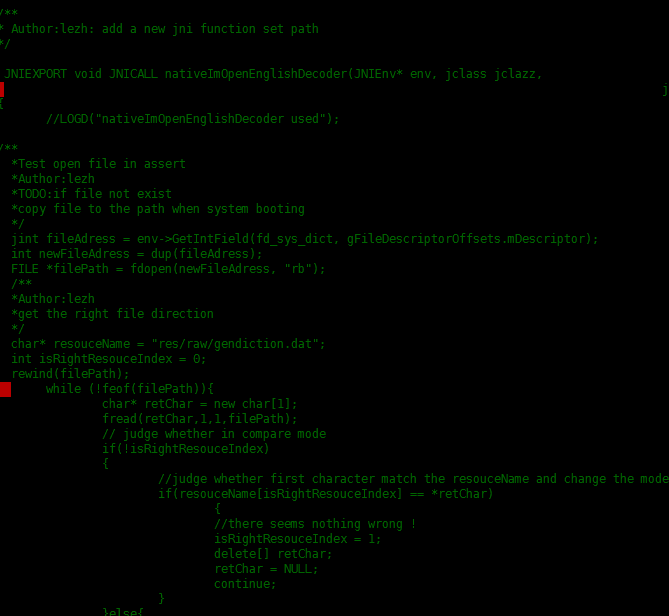
改动的时候一些问题记录在 ./ 功能代码融合在CPP代码过程中出现问题.docx 里有记录。

而在代码中改动最大的几个地方几种在：

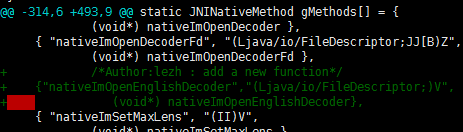
* jni层字典文件从apk中拷贝到/data目录下（中间遇到raw文件读取出来的问题）

在系统启动时就调用了 nativeImOpenDecoderFd() 方法，所以在模仿这个方法里拷贝出来文件。

自定义方法 nativeImOpenEnglishDecoder()



接下来还要进行注册：



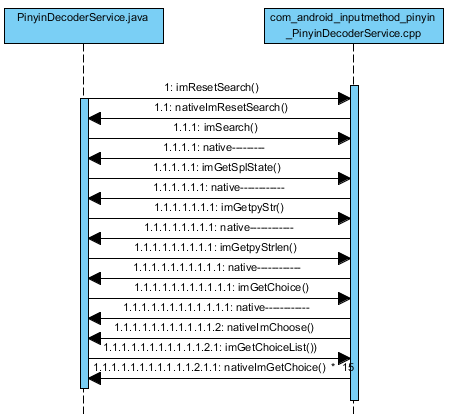
主要代码在 com\_android\_inputmethod\_pinyin\_PinyinDecoderService.cpp 的 nativeImOpenEnglishDecoder 函数里。

遇到的问题在 功能代码融合在CPP代码过程中出现问题.docx 中有记录。

* jni层返回查询的逻辑的修改

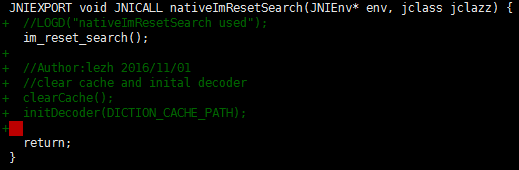
首先要理清楚的是函数的调用关系，调用关系分成两个层面，以及两种事件。

1. 中文输入法点击时的候选词逻辑：



比较**重要**并修改的逻辑：

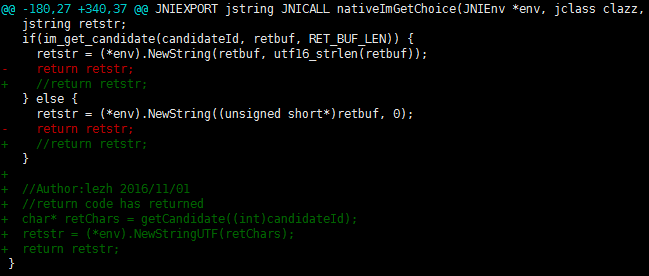
nativeImResetSearch()：重设候选词服务，清空所有之前留下来的数据，并准备下一次。



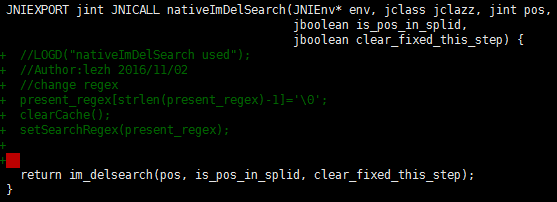
nativeImSearch()：注意传入了jbyteArray代表需要搜索的字符串值。在这里转型成我们需要的char\* 传入我们的功能性代码中。



nativeImGetChoice()：传入一个值，代表返回的词组的优先度排序。返回一个jstring类型值。



nativeImDelSearch()：当在界面上按删除时调用这个函数，需要重新进行查询。

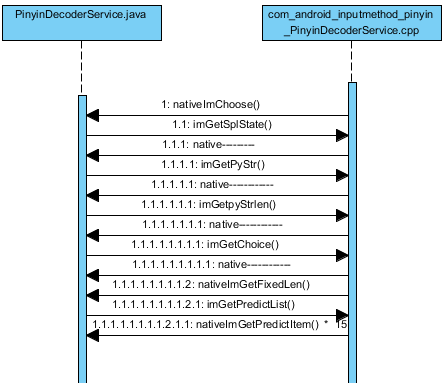


nativeImGetSplState():返回当前活跃的字串（即老版ComposingView上显示的字串）

nativeImGetPystr():返回当前输入的拼音（即完整的输入的字串）

nativeImGetStrlen():返回当前输入的拼音字串长度

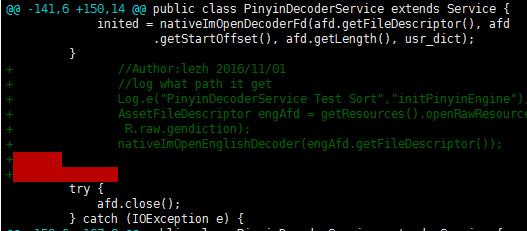
1. 选择一个中文后的联想词逻辑：



后期如果想添加联想词的话就按照这个逻辑进行修改。

1. java代码里的修改

在PinyinDecoderService.java 的 initPinyinEngine()里



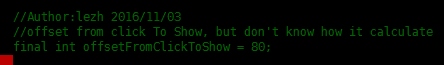
* java层获取返回值的CandidateView的调整

主要问题出在：

1. 点击位置有偏移量，点击事件触发不准

解决办法：

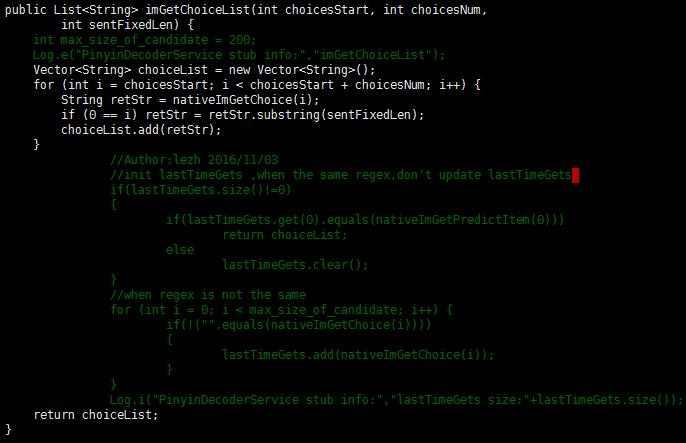
手动测量偏移量，将偏移量计算进去，如下图





1. 返回值储存逻辑复杂， 返回空指针，不易于按照逻辑获取到返回值

解决办法是自己创建一个列表保存上次得到的所有返回词信息，同时限制返回词的个数，避免占用太多内存。



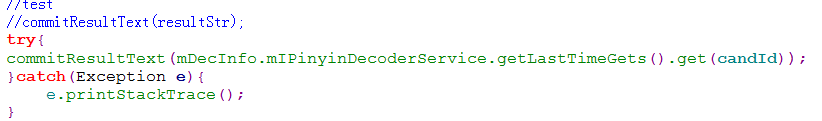
在PinyinIME里：



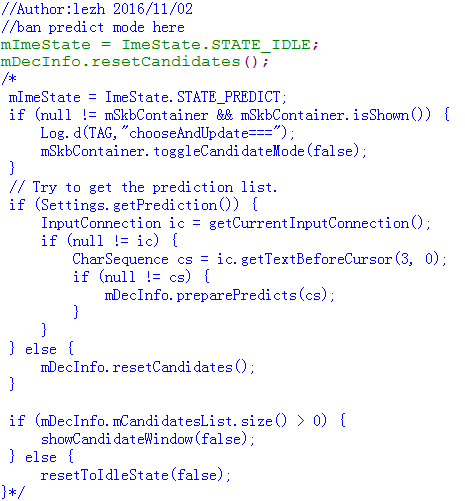
* java层解决联想词逻辑的修改

在PinyinIME.java的chooseAndUpdate()

这里作为返回候选词的位置



这里注释掉转换成联想词模式的代码，并将当前状态还原到STATE\_IDLE。



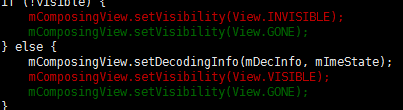
# 融合2.3分支并添加修改

## 融合

直接使用git merge得到融合的代码，没有产生冲突。

## 修改界面到UI的需求。

PinyinIME 的 updateComposingText()中修改



使composing view消失。