### 系统功能性需求分析

本系统是一个基于生活场景帮助人们快速识别文字（OCR）并将识别结果播报（语音合成）出来，有助于不方便使用手机等手持设备时识别内容，并且能根据人们的需求实现语音识别功能。

本系统是Android 客户端的应用程序，方便用户使用手机等安卓手持设备实现识别文字（OCR）等功能。

### 文字识别需求核心技术

1. 文字识别过程主要有：预处理、特征提取、特征匹配、识别后处理。
2. 由于普通拍摄图片会存在干扰，所以要对图片进行预处理，增强图片，降低干扰。
   1. 图片灰度化技术

要对采集到的图片进行灰度化处理，主要是因为灰度图片相对彩色的图片抛出了一部分的数据信息，数据量更小，更容易处理，并且更能显示文字的特点。

RGB图片灰度化技术，通俗点说就是对图像的RGB三个分量进行加权平均得到最终的灰度值。最常见的加权方法如下：

1）Gray=B；Gray=G；Gray=R

2）Gray=max(B+G+R)

3）Gray=(B+G+R)/3

4）Gray= 0.072169B+ 0.715160G+ 0.212671R

5）Gray= 0.11B+ 0.59G+ 0.3R

这三种方法中，第一种为分量法，即用RGB三个分量的某一个分量作为该点的灰度值；第二种方法为最大值法，将彩色图像中的三分量亮度的最大值作为灰度 图的灰度值。第三种方法将彩色图像中的三分量亮度求平均得到一个灰度图；后两种都是属于加权平均法，其中第四种是OpenCV开放库所采用的灰度权值，第 五种为从人体生理学角度所提出的一种权值（人眼对绿色的敏感最高，对蓝色敏感最低）。

#### 2.1 二值技术

图像的二值化处理就是将图像上的点的灰度值为0或255，也就是将整个图像呈现出明显的黑白效果。即将256个亮度等级的灰度图像通过适当的阈值选取而获得仍然可以反映图像整体和局部特征的二值化图像。在数字图像处理中，二值图像占有非常重要的地位，特别是在实用的图像处理中，以二值图像处理实现而构成的系统是很多的，要进行二值图像的处理与分析，首先要把灰度图像二值化，得到二值化图像，这样子有利于在对图像做进一步处理时，图像的集合性质只与像素值为0或255的点的位置有关，不再涉及像素的多级值，使处理变得简单，而且数据的处理和压缩量小。

为了得到理想的二值图像，一般采用封闭、连通的边界定义不交叠的区域。所有灰度大于或等于阈值的像素被判定为属于特定物体，其灰度值为255表示，否则这些像素点被排除在物体区域以外，灰度值为0，表示背景或者例外的物体区域。 如果某特定物体在内部有均匀一致的灰度值，并且其处在一个具有其他等级灰度值的均匀背景下，使用阈值法就可以得到比较的分割效果。

如果物体同背景的差别表现不在灰度值上（比如纹理不同），可以将这个差别特征转换为灰度的差别，然后利用阈值选取技术来分割该图像。动态调节阈值实现图像的二值化可动态观察其分割图像的具体结果。

OpenCV中有两个函数可以实现图片的二值化：

cvThreshold( dst, dst,230 , 255, CV\_THRESH\_BINARY\_INV);

cvAdaptiveThreshold( dst, dst, 255, CV\_ADAPTIVE\_THRESH\_MEAN\_C,CV\_THRESH\_BINARY, 9, -10);

方法（1）是手动指定一个阈值，以此阈值来进行二值化处理。其中的第四个参数决定了该方法的结果：

threshold\_type=CV\_THRESH\_BINARY:

dst(x,y) = max\_value, if src(x,y)>threshold 0, otherwise.

threshold\_type=CV\_THRESH\_BINARY\_INV:

dst(x,y) = 0, if src(x,y)>threshold; dst(x,y) = max\_value, otherwise.

threshold\_type=CV\_THRESH\_TRUNC:

dst(x,y) = threshold, if src(x,y)>threshold; dst(x,y) = src(x,y), otherwise.

threshold\_type=CV\_THRESH\_TOZERO:

dst(x,y) = src(x,y), if (x,y)>threshold ; dst(x,y) = 0, otherwise.

threshold\_type=CV\_THRESH\_TOZERO\_INV:

dst(x,y) = 0, if src(x,y)>threshold ; dst(x,y) = src(x,y), otherwise.

值得一说的是threshold\_type可以使用CV\_THRESH\_OTSU类型，这样该函数就会使用大律法OTSU得到的全局自适应阈值来进行二值化图片，而参数中的threshold不再起作用。比如：cvThreshold( dst, dst,300 , 255, CV\_THRESH\_OTSU | CV\_THRESH\_BINARY\_INV);这种方法对于灰度直方图呈现二峰特征的图片处理起来效果很好。

方法（2）是一个自适应阈值二值化方法，通过设定最后两个参数来调整效果。

#### 2.2 基于统计模型（隐马尔可夫模型）语义识别与纠错

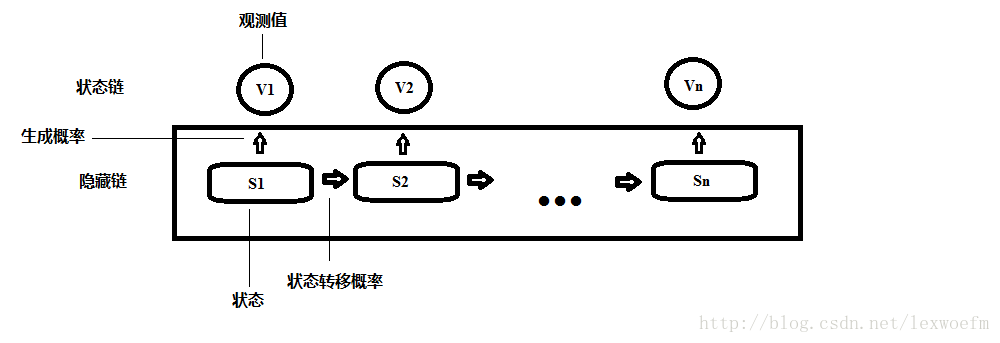
隐马尔可夫模型（Hidden Markov Model，HMM）创立于20世纪70年代。主要用于行为识别，语音识别，文字识别等。它用来描述一个含有隐含未知参数的马尔可夫过程。其难点是从可观察的参数中确定该过程的隐含参数。然后利用这些参数来作进一步的分析，例如模式识别。

是在被建模的系统被认为是一个马尔可夫过程与未观测到的（隐藏的）的状态的统计马尔可夫模型。

#### 2.3 隐马尔可夫原理简述

隐马尔可夫模型由五个部分组成：状态空间S，观测空间O，初始状态概率空间PI，状态概率转移矩阵P以及观测值生成概率矩阵Q。另外，隐马尔可夫模型还包括一条观测链，一条隐藏链。（后面将详述）

下面是隐马尔可夫模型示意图：



因此整个过程就是观测值随状态的转移而生成，而我们所关心的是通过已有的观测值来判断其隐藏的状态，即通过一长串的观测序列推算导致这一结果的可能的状态序列。

#### **2.4 实现方法**

要实现上面所述原理就必须解决三个问题：评估问题（evaluation），解码问题（decoding）和学习问题（learning）

1. **评估问题，即评估当前状态为真实状态的可能性。**

最简单的方法有前向算法和后向算法（当然也可以联合使用这两种算法）。

**前向算法：**从前递归，一层一层计算概率，最后再求总和。

1）t=0（事实上 t 的首项应该为1，但是考虑到编程的方便这里就设首项为0）

https://img-blog.csdn.net/20131117152419484?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，即 alpha(i,t)=PI(i)\*Q(i,t)（伪代码，这只是为了表示方便易懂，与之后的代码可能会有出入）

PS：alpha(i,t)指t时刻状态为Si的概率（下面同义），PI(i)为状态Si的初始概率，Q(i,t)指的是 t 时刻观测值Vt由状态Si生成的概率。

2）t>0 && t<=n

https://img-blog.csdn.net/20131117152517812?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，即alpha(t,i)=Sum[ alpha(j,t-1)\*P(j,i)\*Q(i,t) ]

PS：P(j,i)指由状态 Sj 转移到 Si 的概率

3）

https://img-blog.csdn.net/20131117152605859?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，即将3）所算的所有状态Si的结果再求和。

PS：对应后面的Java类为 AlgorithmFront.java

**后向算法：**与前向算法相反

1）t>=0 && t<n

https://img-blog.csdn.net/20131117152734484?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，即beta(i,t)=Sum[ beta(j,t+1)\*P(i,j)\*Q(j,t+1) ]

PS：beta(i,t)表示t时刻状态为Si的条件下，从t+1时刻到n生成相应观测序列的概率。

2）t=n

https://img-blog.csdn.net/20131117152705906?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

PS：因为下一个时刻就已结束，所以无论是什么状态都是确定的，所以概率都为1。

3）

https://img-blog.csdn.net/20131117152823265?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，与前向算法相似，最后也是将所有结果进行求和。

PS：对应后面的Java类为 AlgorithmBack.java

1. **解码问题，即如何根据观测值，状态转移概率矩阵，生成概率矩阵得到真正的状态序列。（有时候你完全可以根据先验知识给参数设值，这样就无需 学习步骤（Learning） 便可以解码了）**

Viterbi算法：基本原理就是计算概率每一步最高时对应的状态序列

1）初始化

https://img-blog.csdn.net/20131117152900015?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

2）递归

https://img-blog.csdn.net/20131117152929968?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

3）终止

https://img-blog.csdn.net/20131117153130046?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，https://img-blog.csdn.net/20131117153200265?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

PS：https://img-blog.csdn.net/20131117153448859?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast表示 n 时刻沿着X1,X2,...Xn 且在 n 时刻状态Xn=Si 产生相应观测序列的最大概率

https://img-blog.csdn.net/20131117153343156?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast保存着状态序列信息。

4）回溯

根据 https://img-blog.csdn.net/20131117153316984?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast 的结果便可知道相应的状态序列了。

PS：对应的后面的Java类为 HMMDecisionVbImp.java

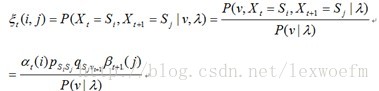
1. **学习问题，即如何通过观测值来获取初始状态概率，状态转移概率矩阵以及生成概率矩阵。**

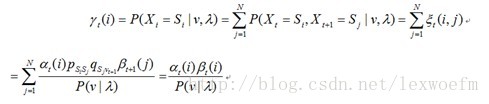
Baum-Welch算法：

Step1: 随机产生一组参数，并代入评估函数（evaluation，例如前向算法），计算结果。

Step2: 利用参数估算初始状态概率，状态转移概率矩阵以及生成概率矩阵

由于：

，即kis(i,j,t)=alpha(i,t)\*P(i,j)\*Q(j,t+1)\*beta(j,t+1)

，即gamma(i,t)=alpha(i,t)\*beta(i,t)

PS：a）伪代码中并没有除以https://img-blog.csdn.net/20131117154033984?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，这主要是为了减少运算量，因为之后计算状态概率矩阵、生成矩阵这项都会被约掉。

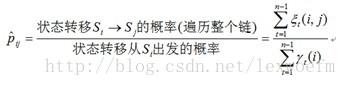
b）kis(i,j,t)即，表示t时刻为状态Si，t+1时刻为状态Sj的概率

c）gamma(i,t)即，表示t时刻状态为Si的概率

d）相应的Java类为 Gammas.java，Ksis.java

所以：

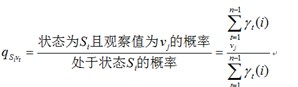
1）估计概率转移概率矩阵



2）估计初始状态概率

https://img-blog.csdn.net/20131117154207078?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，（注意，实际编程实现时这里还需除以之前漏除的https://img-blog.csdn.net/20131117154225453?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGV4d29lZm0=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast）

3）估计概率生成矩阵



Step3: 将刚估计的参数代入 评估函数 进行计算，并与上一次评估的结果做比较，若差异小于某个阈值（thresh，例如 0.05）则接受。否则继续迭代计算。

https://blog.csdn.net/xiaotianlan/article/details/38925197

#### 2.5 HMM 的基本概念

HMM 是马尔可夫链的一种，用于描述随机过程统计特性。 包括双重随机过程，一个是隐含的状态，它的状态不能直接观察 到。另一个是观测向量序列，每一个观测向量是由一个具有相 应概率密度分布的状态序列产生。

HMM 包括 5 个元素: ( 1) 隐含状态的数目 N，状态集为 S = { s1，s2，…，sN } ; ( 2) 观察序列的数目 M，观察序列集为 V = { v1，v2，…，vm } ; ( 3) 状态转移矩阵 A，A = { aij} ，aij = P［qt + 1 = sj | qt = si］， 1≤i，j≤N。其中 qt为在时刻 t 的状态。A 为 N × N 的方阵，行和 列都对应所有的状态，表示状态之间转移的概率; ( 4) 观察序列概率矩阵 B，B = { bj ( k) } ，bj ( k) = P［vt at t qt = sj］，1≤j≤N，1≤k≤M，连续型 HMM 的 B 通过一个连续的 函数得到观察序列与状态的关系，常用的是混合高斯概率密度 函数。 ( 5) 初始状态分布概率 ∏ = { πi} ，πi = P{ s1 = q1 } ，其中 1 ≤ i ≤ N。显然有∑ N i = 1 πi = 1。

HMM 主要可以解决 3 个基本问题:

1) 评估问题即给定一个模型 λ = { A，B，π} 和一组观察序 列 O = O1，O2，…，OT，计算在此模型下产生该观察序列的概率 P( O λ) 。评估问题用前向或后向算法解决。

2) 解码问题，即给定一个模型 λ = { A，B，π} 和一组观察序 列 O = O1，O2，…，OT，求解对应的最大概率的状态序列 Q = q1， q2，…，qT。解码问题用 Viterbi 算法解决。

3) 学习问题，即给定观察序列 O = O1，O2 …，OT，如何调整 模型参数 A，B，π，使得 P( O| λ) 最大。学习问题用 Baum-Welch 算法解决。

#### 2.6 OCR技术难点是什么？

复杂背景、艺术字体、低分辨率、非均匀光照、图像退化、字符形变、多语言混合、文本行复杂版式、检测框字符残缺，等等。

如克服这些难点的？从几个方面入手。一是使用场景，另一方面是从技术上的改进文本检测技术方进行了深度优化，通过设计合理的网络结构来提升各尺度的文字检测／提取能力。引入RNN多层自适应网络和Refinement结构来提升检测完整性和准确性。

##### Android 客户端需求分析

Android客户端应用程序具备多个文字识别的内容，帮助用户在不同场景下快速实现完成文字识别的功能（OCR），并且不方便查看识别结果时将结果语音合成播报，也需要满足人们不方便打字时通过语音识别功能识别人们话语。

图1 详细的分析安卓应用所需要的需求：



图 1

* 1. 系统功能
     1. 通用文字识别

具备调用手机摄像头拍摄任意带有文字的内容，比如一张图片中的文字，纸巾包装上的文字等。

* + 1. 身份证识别

可以识别身份证正反面的内容，识别出姓名，出生年月日，性别等内容。

* + 1. 银行卡识别

可以通过手机摄像头拍摄银行卡识别出银行卡的卡号与发卡行。

* + 1. 车牌识别

可以通过手机摄像头拍摄车牌号（可以是图片），识别出车牌号码。

* + 1. 车型识别

可以通过手机摄像头拍摄常见的车型（可以是图片），系统根据拍摄的图片识别出车型。

* + 1. 语音识别

可以识别出用户的语音，并且显示到手机屏幕上

* + 1. 语音合成

可以将文字识别的结果合成语音，通过手机扬声器播放出来，能设置语速，播讲人等参数。

* 1. 系统非功能性需求分析

整体应用对性能有较高的要求，调取摄像头时要注意防止内存泄漏，也不能出现程序无响应异常，需要站在用户的角度上体验整个应用，关键时刻出现错误，非常影响用户体验。

除此之外，进行应用开发时，心里对整个应用要保持着简洁、美观的要求，一个美观的界面与功能完整但是界面简陋，用户更加倾向于美观简洁的界面，同时，作为开发人人员要具备产品思维，一个好的应用，就就必须注重细节。另外还需要注意的是，Android 手机发展多年，最新版本也发展到 Android 9.0（2019年），我们做出来的应用不仅要注意适配市面上常见的手机设备，还要注意适配Android的版本，这里是非常难的。

## 系统设计

本系统的文字识别引擎与语音合成引擎皆采用百度公司的识别引擎，可以应用于多门语言，比如Java语言，PHP语言，Python语言，C#语言，C++语言，更包括可以应用于Android 与 iOS 平台上。

### 百度文字识别支持系统与硬件

支持的系统和硬件版本

系统：支持 Android 4.0（API Level 15）到Android7.0（API Level 25）系统。需要开发者通过minSdkVersion来保证支持系统的检测。

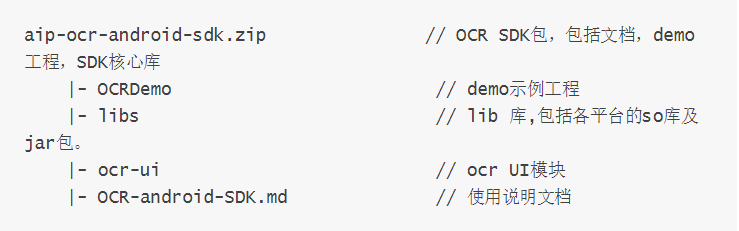
CPU架构：armeabi，arm64-v8a，armeabi-v7a，x86

机型：手机和平板皆可

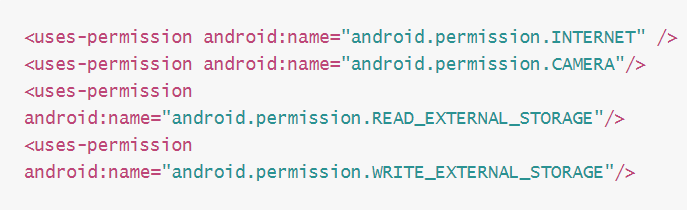
硬件要求：要求设备上有相机模块。

网络：支持WIFI及移动网络，移动网络支持使用NET网关及WAP网关（CMWAP、CTWAP、UNIWAP、3GWAP）。

### 开发包说明



### 程序开始前需要添加的权限



各个权限的用途说明见下表：



### 系统界面设计

Android 界面布局设计，通常两种，一是静态布局文件设计，二是Java代码动态设置。

本应用两者皆有采用，主体不需要变动内容，采用静态布局文件设计，而需要根据不同结果采用不同的内容则采用Java代码动态设置。

### 首页设计

首页作为主要设计页面，可以让用户通过点击Android控件完成数据采集功能。其效果图如图：



图 2

用户打开本应用，在主页面上选择想要实现的功能，系统会调用相应的引擎进行处理，最后返回结果，可以通过语音合成播报识别结果。

#### 文字识别时序图：



系统功能时序图 1

详细流程如下：

1. 启动本应用，系统将进行初始化操作
2. 用户通过点击屏幕相应的功能，开启识别功能。
3. 系统启动手机的摄像头，用户拍摄图片，系统系统已拍摄图片。用户点击确定，调用文字识别引擎，处理图片。
4. 识别完成后将数据返回给首页，调用对话框弹出返回的识别结果，
5. 系统调用语音合成，进行语音播报

### 系统相机获取图片

调用系统的相机模块共两种使用方式，用户可以拍摄图片或者选取手机中存储的图片，下图为调用系统相机的流程图：



图 3 调用系统相机获取图片流程图

### 3.7 Android XML布局部分代码展示：

<**Button**

android:id="@+id/general\_basic\_button"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="通用文字识别"

tools:ignore="HardcodedText" />

<**Button**

android:id="@+id/idcard\_button"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="身份证识别"

tools:ignore="HardcodedText" />

<**Button**

android:id="@+id/bankcard\_button"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="银行卡识别"

tools:ignore="HardcodedText" />

<**Button**

android:id="@+id/license\_plate\_button"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="车牌识别"

tools:ignore="HardcodedText" />

<**LinearLayout**

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

<**Button**

android:id="@+id/car\_model\_button"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="车型识别"

tools:ignore="HardcodedText" />

<**Button**

android:id="@+id/speechrecognition\_button\_start"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="语音识别开始"

tools:ignore="HardcodedText" />

<**Button**

android:id="@+id/speechrecognition\_button\_stop"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="语音识别停止"

tools:ignore="HardcodedText" />

<**RelativeLayout**

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<**ImageView**

android:id="@+id/car\_model\_iv"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:scaleType="centerCrop" />

<**TextView**

android:id="@+id/scan\_value\_tv"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_below="@id/car\_model\_iv"

android:background="@color/colorAccent"

android:gravity="center"

android:text="结果扫描中"

android:textColor="@color/colorWhite"

android:textSize="20sp"

android:textStyle="bold"

android:visibility="gone" /

</RelativeLayout>

<**TextView**

android:id="@+id/re\_result\_tv"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

</LinearLayout>

### 3.8 关键代码部分展示：

*// 通用文字识别，跳转调用手机摄像头*

findViewById(R.id.general\_basic\_button).setOnClickListener(new View.OnClickListener() {

@**Override**

**public** **void** onClick(**View** v) {

if (!checkTokenStatus()) {

return;

}

**Intent** intent = new Intent(MainActivity.this, CameraActivity.class);

intent.putExtra(CameraActivity.KEY\_OUTPUT\_FILE\_PATH,

FileUtil.getSaveFile(getApplication()).getAbsolutePath());

intent.putExtra(CameraActivity.KEY\_CONTENT\_TYPE,

CameraActivity.CONTENT\_TYPE\_GENERAL);

startActivityForResult(intent, REQUEST\_CODE\_GENERAL\_BASIC);

}

});

*// 识别成功回调，通用文字识别*

if (requestCode == REQUEST\_CODE\_GENERAL\_BASIC && resultCode == Activity.RESULT\_OK) {

RecognizeService.recGeneralBasic(this, FileUtil.getSaveFile(getApplicationContext()).getAbsolutePath(),

new RecognizeService.ServiceListener() {

@**Override**

**public** **void** onResult(**String** result) {

Log.i(TAG, "onResult: == " + getUniverstalTextJsonBean(result));

infoPopText("识别结果,查看以下内容", getUniverstalTextJsonBean(result));

speak(getUniverstalTextJsonBean(result));

}

});

}

*//弹出对话框*

**private** **void** alertText(**final** **String** title, **final** **String** message) {

this.runOnUiThread(new Runnable() {

@**Override**

**public** **void** run() {

alertDialog.setTitle(title)

.setMessage(message)

.setPositiveButton("确定", new DialogInterface.OnClickListener() {

@**Override**

**public** **void** onClick(**DialogInterface** dialog, **int** which) {

mSpeechSynthesizer.stop();

}

}) .show();

}

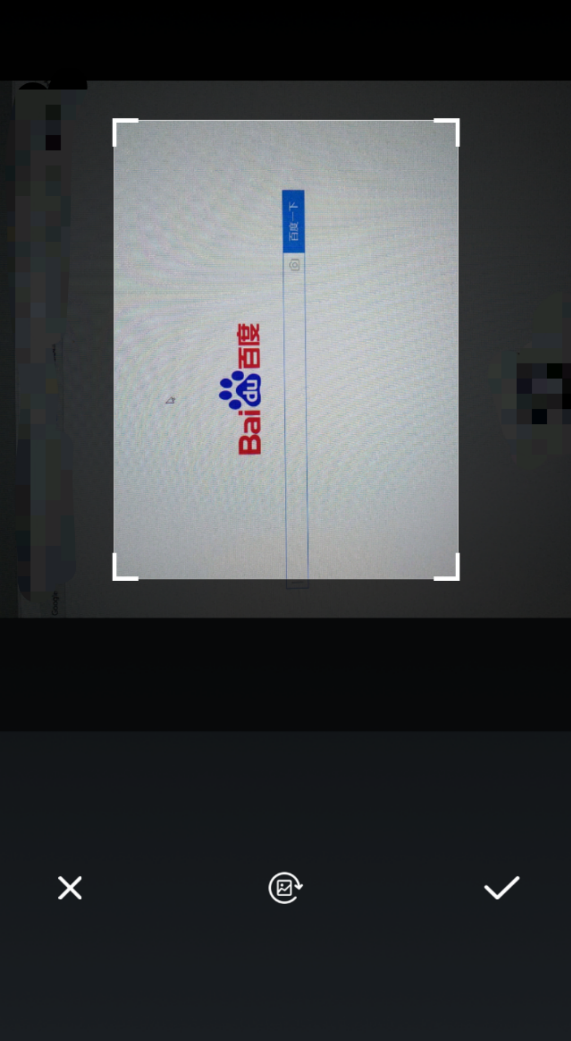
});

}

应用运行截图



图 4



### 3.8 语音合成与语音识别核心技術

语音合成和[语音识别](https://baike.baidu.com/item/%E8%AF%AD%E9%9F%B3%E8%AF%86%E5%88%AB)技术是实现人机语音通信，建立一个有听和讲能力的口语系统所必需的两项关键技术。使电脑具有类似于人一样的说话能力，是当今时代信息产业的重要竞争市场。和语音识别相比，语音合成的技术相对说来要成熟一些，并已开始向产业化方向成功迈进，大规模应用指日可待。

语音合成，又称文语转换（Text to Speech）技术，能将任意文字信息实时转化为标准流畅的语音朗读出来，相当于给机器装上了人工嘴巴。它涉及声学、语言学、[数字信号处理](https://baike.baidu.com/item/%E6%95%B0%E5%AD%97%E4%BF%A1%E5%8F%B7%E5%A4%84%E7%90%86)、计算机科学等多个学科技术，是[中文信息处理](https://baike.baidu.com/item/%E4%B8%AD%E6%96%87%E4%BF%A1%E6%81%AF%E5%A4%84%E7%90%86)领域的一项前沿技术，解决的主要问题就是如何将文字信息转化为可听的声音信息，也即让机器像人一样开口说话。我们所说的“让机器像人一样开口说话”与传统的声音回放设备（系统）有着本质的区别。传统的声音回放设备（系统），如磁带[录音机](https://baike.baidu.com/item/%E5%BD%95%E9%9F%B3%E6%9C%BA)，是通过预先录制声音然后回放来实现“让机器说话”的。这种方式无论是在内容、存储、传输或者方便性、及时性等方面都存在很大的限制。而通过计算机语音合成则可以在任何时候将任意文本转换成具有高自然度的语音，从而真正实现让机器“像人一样开口说话”。

文语转换系统实际上可以看作是一个人工智能系统。为了合成出高质量的语言，除了依赖于各种规则，包括语义学规则、词汇规则、语音学规则外，还必须对文字的内容有很好的理解，这也涉及到自然语言理解的问题。

### 3.8 一个典型的语音合成系统流程图

如下图所示，一个典型的语音合成系统主要包括前端和后端两个部分。前端部分主要是对输入文本的分析，从输入的文本提取后端建模需要的信息。例如：分词（判断句子中的单词边界），词性标注（名词，动词，形容词等），韵律结构预测（是否韵律短语边界），多音字消岐等等。后端的部分读入前端文本分析结果，并且对语音部分结合文本信息进行建模。在合成过程中，后端会利用输入的文本信息和训练好的声学模型，生成出语音信号，进行输出。

图片包含 屏幕截图

描述已自动生成

图 5

### 3.9 三种现阶段主要的语音合成系统

现阶段的语音合成系统，根据所采用的方法和框架不同，主要可以分为三种：（1）参数语音合成系统。（2）拼接语音合成系统。（3） 基于波形的统计合成系统(WaveNet) 。其中A, B 是现阶段各大公司线上主流的合成系统，C WaveNet 的方法还在研究阶段，是现阶段研究的热门。

1. 参数语音合成系统的特点是，在语音分析阶段，需要根据语音生成的特点，将语音波形(speech waves) 通过声码器转换成频谱，基频，时长等语音或者韵律参数。在建模阶段对语音参数进行建模。并且在语音合成阶段，通过声码器从预测出来的语音参数还原出时域语音信号。参数语音合成系统的优势在于模型大小较小，模型参数调整方便（说话人转换，升降掉），而且合成语音比较稳定。缺点在于合成语音音质由于经过参数化，所以和原始录音相比有一定的损失。
2. 拼接语音合成系统的特点是，不会对原始录音进行参数化，而会将原始录音剪切成一个一个基本单元存储下来。在合成过程中，通过一些算法或者模型计算每个单元的目标代价和连接代价，最后通过Viterbi算法并且通过PSOLA(Pitch Synchronized Overlap-Add)或者WSOLA(Waveform Similarity based Overlap-Add)等信号处理的方法“拼接”出合成语音。因此，拼接语音合成的优势在于，音质好，不受语音单元参数化的音质损失。但是在数据库小的情况下，由于有时挑选不到合适的语音单元，导致合成语音会有Glitch 或者韵律、发音不够稳定。而且需要的存储空间大。
3. WaveNet 波形统计语音合成是Deep Mind 首先提出的一种结构，主要的单元是 Dilated CNN （卷积神经网络）。这种方法的特点是不会对语音信号进行参数化，而是用神经网络直接在时域预测合成语音波形的每一个采样点。优势是音质比参数合成系统好，略差于拼接合成。但是较拼接合成系统更稳定。缺点在于，由于需要预测每一个采样点，需要很大的运算量，合成时间慢。WaveNet 证明了语音信号可以在时域上进行预测，这一点以前没有方法做到。现阶段WaveNet是一个研究热点。

### 3.10 语音识别技术

声学模型、语言模型和解码器可以看作是现代语音识别系统最核心的三个组成部分。虽然最近有一些研究者尝试构建End2end的语音识别系统，但包含声学模型、语言模型和解码器的现代语音识别系统依然是当前最主流和使用最广泛的系统。在这其中，声学模型主要用来构建输入语音和输出声学单元之间的概率映射关系；语言模型用来描述不同字词之间的概率搭配关系，使得识别出的句子更像自然文本；解码器负责结合声学单元概率数值和语言模型在不同搭配上的打分进行筛选，最终得到最可能的识别结果。

随着近几年深度学习的火热，语音识别领域也纷纷投入深度学习的大潮之中。将传统HMM-GMM声学模型替换成HMM-DNN声学模型后，可以获得超过20%的相对提升，在传统N-Gram语言模型基础上叠加NN-LM语言模型也可以获得进一步的提高。在这过程中，声学模型由于更适合采用深度神经网络模型，从而受到研究者更多的关注。本文主要介绍阿里云语音识别技术中采用的声学模型技术和语言模型技术，包括LC-BLSTM声学模型、LFR-DFSMN声学模型和NN-LM语言模型，其中LC-BLSTM是对传统BLSTM模型的一种改进，在保持了高准确率的同时，提供了低延时的特性；而LFR-DFSMN是对RNN声学模型的一种改进，用精巧的模型设计获得更稳定的训练效果和更好的识别准确率；NN-LM语言模型是近年来在传统N-Gram语言模型基础上获得的进一步改进。[6]

#### 3.10.1 Latency-Controlled BLSTM模型

DNN（即fully connected DNN）模型的优点在于通过增加神经网络的层数和节点数，扩展了网络对于复杂数据的抽象和建模能力，但同时DNN模型也存在一些不足，例如DNN中一般采用拼帧来考虑上下文相关信息对于当前语音帧的影响，这并不是反映语音序列之间相关性的最佳方法。自回归神经网络（RNN）在一定程度上解决了这个问题，它通过网络节点的自连接达到利用序列数据间相关性的目的。进一步有研究人员提出一种长短时记忆网络（LSTM-RNN），它可以有效减轻简单RNN容易出现的梯度爆炸和梯度消散问题，而后研究人员又对LSTM进行了扩展，使用双向长短时记忆网络（BLSTM-RNN）进行声学模型建模，以充分考虑上下文信息的影响。

BLSTM模型可以有效地提升语音识别的准确率，相比于DNN模型，相对性能提升可以达到15%-20%。但同时BLSTM模型也存在两个非常重要的问题：

1. 句子级进行更新，模型的收敛速度通常较慢，并且由于存在大量的逐帧计算，无法有效发挥GPU等并行计算工具的计算能力，训练会非常耗时；

2. 由于需要用到整句递归计算每一帧的后验概率，解码延迟和实时率无法得到有效保证，很难应用于实际服务。

对于这两个问题，学术界首先提出Context-Sensitive-Chunk BLSTM（CSC-BLSTM）的方法加以解决，而此后又提出了Latency Controlled BLSTM（LC-BLSTM）这一改进版本，更好、更高效的减轻了这两个问题。我们在此基础上采用LC-BLSTM-DNN混合结构配合多机多卡、16bit量化等训练和优化方法进行声学模型建模，取得了相比于DNN模型约17-24%的相对识别错误率下降。

典型的LSTM节点结构由3个gate组成：input gate、forget gate、output gate和一个cell组成，输入、输出节点以及cell同各个门之间都存在连接；input gate、forget gate同cell之间也存在连接，cell内部还有自连接。这样通过控制不同门的状态，可以实现更好的长短时信息保存和误差传播。

图片包含 文字, 地图

描述已自动生成

LSTM可以像DNN一样逐层堆积成为Deep LSTM，为了更好的利用上下文信息，还可以使用BLSTM逐层堆积构造Deep BLSTM，其结构如下图所示，网络中沿时间轴存在正向和反向两个信息传递过程，每一个时间帧的计算都依赖于前面所有时间帧和后面所有时间帧的计算结果，对于语音信号这种时序序列，该模型充分考虑了上下文对于当前语音帧的影响，能够极大的提高音素状态的分类准确率。

图片包含 就坐

描述已自动生成

然而由于标准的BLSTM是对整句语音数据进行建模，训练和解码过程存在收敛慢、延迟高、实时率低等问题，针对这些弊端我们采用了Latency Controlled BLSTM进行解决，与标准的BLSTM使用整句语音进行训练和解码不同，Latency Control BLSTM使用类似truncated BPTT的更新方式，并在cell中间状态处理和数据使用上有着自己的特点，如下图所示，训练时每次使用一小段数据进行更新，数据由中心chunk和右向附加chunk构成，其中右向附加chunk只用于cell中间状态的计算，误差只在中心chunk上进行传播。时间轴上正向移动的网络，前一个数据段在中心chunk结束时的cell中间状态被用于下一个数据段的初始状态，时间轴上反向移动的网络，每一个数据段开始时都将cell中间状态置为0。该方法可以很大程度上加快网络的收敛速度，并有助于得到更好的性能。解码阶段的数据处理与训练时基本相同，不同之处在于中心chunk和右向附加chunk的维度可以根据需求进行调节，并不必须与训练采用相同配置。

图片包含 屏幕截图

描述已自动生成

#### 3.10.2 NN-LM语言模型

语言模型，顾名思义，对语言进行建模的模型。语言表达可以看作一串字符序列，不同的字符序列组合代表不同的含义，字符的单位可以是字或者词。语言模型的任务，可以看作是给定字符序列，如何估计该序列的概率，或者说，如何估计该序列的合理性。

P(上海 的 工人 师傅 有 力量)>P(上海 的 工人 食腐 有 力量)

拿这句话做个例子。比如到底应该是“工人师傅有力量”，还是“工人食腐有力量”，哪句话更“合适”。我们容易判断左边这句的概率大一点。于是我们希望通过语言模型的建模，可以给出符合人类预期的概率分配。就像这句，“工人师傅”的概率，大于“工人食腐”的概率。

基于统计词频的传统N元文法模型，通过马尔可夫假设简化了模型结构和计算，通过计数的方式计算，通过查找的方式使用。拥有估计简单、性能稳定、计算快捷的优势，有超过三十年的使用历史。然而其马尔科夫假设强制截断建模长度，使得模型无法对较长的历史建模；基于词频的估计方式也使得模型不够平滑，对于低词频词汇估计不足。随着神经网络（Neural Networks，NNs）的第三次崛起，人们开始尝试通过NN来进行语言模型建模。

图片包含 屏幕截图

描述已自动生成

图 6

一个典型的建模结构是递归神经网络（recurrent neural networks，RNNs），其递归的结构理论上可以对无穷长序列进行建模，弥补了N元文法对于序列长度建模的不足；同时其各层间的全向连接也保证了建模的平滑。此外为了提升模型的性能，研究者们还尝试了通过长短时记忆（Long Short-Term Memory，LSTM）结构来提升基本RNN本身建模能力的不足，进一步提升模型性能。

NN用于大规模语言建模的系统中，需要面对一些问题，例如大词表带来的存储和计算增加。实际线上系统的词表往往比较大，而随着词表的增加，基本RNN结构的存储和计算量都会几何级数爆炸式增长。为此，研究者们进行了一些尝试，压缩词典尺寸成了一个最直接的解决方案，一个经典的方法是词表聚类。该方法可以大幅压缩词表尺寸，但往往也会带来一定的性能衰减。更直接的一个想法是直接过滤掉低频词汇，这样依然会带来一定的性能衰减，据此有一个改进策略，我们发现真正制约速度性能的主要是输出层节点，输入层节点大，借助projection层可以很好解决，于是输入层采用大辞典，而仅对输出层词表进行抑制，这样不仅尽可能地降低了损失，同时过滤掉过低的词频，也有利于模型节点的充分训练，性能往往还会略有提升。

词表的压缩可以提升建模性能，降低计算量和存储量，但仅限于一定的量级，不可以无限制压缩，如何继续降低计算量依然是一个问题。一些方法被提了出来。例如LightRNN，通过类似聚类的方式，利用embedding的思想，把词表映射到一个实值矩阵上，实际输出只需要矩阵的行加矩阵的列，计算量大概也能开个方。和节点数多一起造成计算量大的一个原因就是softmax输出，需要计算所有的节点求个和，然后得到分母。若是这个分母能保持一个常数，实际计算的时候就只算需要的节点，在测试环节就快的多了。于是就有了正则项相关的方法，Variance Regularization，如果训练速度可以接受的话，这种方法在基本不损失模型正确性的情况下可以大幅提升前向计算速度；如果训练的时候也想提速，还可以考虑基于采样，sampling的方法，比如NCE、Importance Sampling、Black Sampling等，本质上就是说，在训练的时候不计算全部节点，只计算正样本（也就是标签为1的节点），以及部分通过某种分布采样的到的负样本，避免高输出造成的计算缓慢。速度上提升还是很明显的。[7]

### 3.11 百度语音使用指南

#### 硬件设备与系统



#### 功能简介

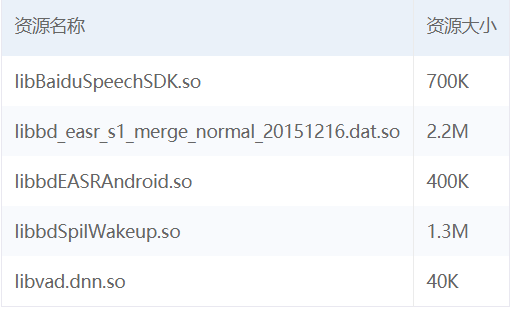
1. 语音识别： 将录音转为文字。目前在线识别支持普通话、英文、粤语和四川话。
2. 在线长语音：任意时长的音频识别
3. 语义理解： 将语音识别出的文字，进行分词及找出意图。

#### SDK库文件



#### NDK so库架构

共计5个架构目录：armeabi，armeabi-v7a，arm64-v8a，x86，x86\_64，每个架构下均有以下5个so库文件。



#### 录音环境

百度语音识别要求安静的环境，真人的正常语速的日常用语，并且不能多个人同时发音。

以下场景讲会导致识别效果变差：

吵杂的环境

有背景音乐，包括扬声器在播放百度合成的语音。

使用非远场语音时，离麦克风较远。

百度识别和合成sdk相互独立，没有类似“相互抵消“的功能。

#### 百度语音识别过程与部分代码展示

1. 初始化
   1. 初始化EventManager类

通过工厂创建语音识别的事件管理器。注意识别事件管理器只能维持一个，请勿同时使用多个实例。即创建一个新的识别事件管理器后，之前的那个置为null，并不再使用。

**EventManager** asr = EventManagerFactory.create(this, "asr");

*// this是Activity或其它Context类*

* 1. 自定义输出事件类

需要实现EventListener的输出事件回调接口。该类需要处理SDK在识别过程中的回调事件。

**EventListener** yourListener = new EventListener() {

@**Override**

**public** **void** onEvent(**String** name, **String** params, **byte** [] data, **int** offset, **int** length) {

if(name.equals(SpeechConstant.CALLBACK\_EVENT\_ASR\_READY)){

*// 引擎就绪，可以说话，一般在收到此事件后通过UI通知用户可以说话了*

}

if(name.equals(SpeechConstant.CALLBACK\_EVENT\_ASR\_FINISH)){

*// 识别结束*

}

}

};

* 1. 注册自己的输出事件类

asr.registerListener(yourListener);

1. 开始识别/唤醒
   1. 设置识别/唤醒输入参数

*// asr.params(反馈请带上此行日志):{"accept-audio-*

*data":false,"disable-*

*punctuation":false,"accept-audio-volume":true,"pid":1736}*

*//其中{"accept-audio-data":false,"disable-punctuation":false,"accept-audio-volume":true,"pid":1736}为ASR\_START事件的参数*

**String** json ="{\"accept-audio-data\":false,\"disable-punctuation\":false,\"accept-audio-volume\":true,\"pid\":1736}"

* 1. 发送start开始事件

asr.send(SpeechConstant.ASR\_START, json, null, 0, 0);

1. 回调事件
   1. 开始回调事件

回调事件在您实现的EventListener中获取。OnEvent方法

中， name是输出事件名，params该事件的参数，(data,offset, length)三者一起组成额外数据。如回调的音频数据，从

data[offset]开始至data[offset + length] 结束，长度为length。

**public** **void** onEvent(**String** name, **String** params, **byte** [] data, **int** offset, **int** length);

* 1. 控制识别/唤醒

asr.send(SpeechConstant.ASR\_STOP, null, null, 0, 0);

*//发送停止录音事件，提前结束录音等待识别结果*

1. 控制停止识别/唤醒，可以向SDK发送停止事件
   1. 取消本次识别，可以向SDK发送取消事件

asr.send(SpeechConstant.ASR\_CANCE, null, null, 0, 0);

*//取消本次识别，取消后将立即停止不会返回识别结果*

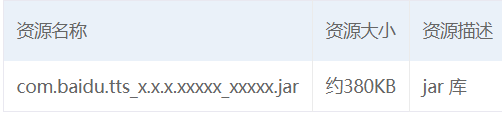
1. 事件管理器退出。
   1. 释放资源

### 百度语音合成使用指南

#### 硬件设备与系统



#### SDK库文件



#### NDK so库架构

共计5个架构目录：armeabi，armeabi-v7a，arm64-v8a，x86，x86\_64，每个架构下均有以下5个so库文件。



#### 功能简介

语音合成分为 在线合成和离线合成。仅有中英文混合这一种语言。目前只有中英文混合这一种语言，优先中文发音。 示例： ” I bought 3 books” 发音 “three”; “ 3 books are bought” 发音 “三”; “我们买了 3 books” 发音“三”

#### 在线和离线判别



没有纯离线模式。

**纯在线模式** ： WIFI 4G 3G 2G 都会尝试连接百度服务器。如果百度服务器失败，那么合成失败。

**离在线混合模式**： WIFI下强制尝试在线优先。其它网络情况可以设置是否为在线优先（如果连接失败，那么切换成离线合成）或者直接离线合成。

* *MIX\_MODE\_DEFAULT*： WIFI下在线优先（连接百度服务器失败或者超时6s，那么切换成离线合成）， 其它网络状况下离线合成。
* *MIX\_MODE\_HIGH\_SPEED\_SYNTHESIZE\_WIFI*： WIFI下在线优先（连接百度服务器失败或者超时1.2s，那么切换成离线合成）， 其它网络状况下离线合成。
* *MIX\_MODE\_HIGH\_SPEED\_NETWORK*： WIFI 4G 3G 2G下在线优先( 如果在线连接百度服务器失败或者超时6s，那么切换成离线合成)， 其它网络状况离线合成。
* *MIX\_MODE\_HIGH\_SPEED\_SYNTHESIZE*： WIFI 4G 3G 2G下在线优先( 如果在线连接百度服务器失败或者超时1.2s，那么切换成离线合成)， 其它网络状况离线合成。

**建议使用场景**： 在小说阅读、导航播报等场景中，若存在网络信号不稳定(频繁断网）的情况，您可以使用百度提供的离在线融合模式。 我们推荐您使用 MIX\_MODE\_HIGH\_SPEED\_NETWORK模式；若您需要在移动网络下不想消耗流量，或是对响应速度有较强需求，请自行根据业务需求选择其他模式。

#### 发音

**在线时支持5种发音** 普通女声 普通男声 特别男声 情感男声<度逍遥> 情感儿童声<度丫丫> 除特别男声外，其它4种发声具体效果可以在<http://ai.baidu.com/tech/speech/tts>上测试

**离线时只支持4种发音** 离线时无特别男声， 其它发音都有离线版本。

**注意：**

1. 在线合成的声音和离线合成的声音会有略微不同。在线合成的效果好。
2. 在极端网络的情况下，可能在线合成与离线合成频繁切换。

**合成效果** 通过对PARAM\_SPEAKER（发音人）、PARAM\_PITCH（音调）、PARAM\_VOLUME（音量）和PARAM\_SPEED（语速）参数的调整，可以获得不同的发声效果，更好满足您业务场景中的播报需求。 且音调越高，声音听起来会显得越年轻。

#### 合成和播放

synthesize 方法直接合成。不播放。可以通过onSynthesizeDataArrived 获取音频数据，自行处理。 speak 方法先合成为音频，之后立即播放。等同调用 synthesize方法，再调用系统播放器。

在SDK内部中有队列，可以不断调用synthesize或者speak方法，将合成的文本添加到队列中。

#### 接口及调用过程及部分代码展示

##### 初始化

##### 获取 SpeechSynthesizer 实例

**SpeechSynthesizer** mSpeechSynthesizer = SpeechSynthesizer.getInstance();

SpeechSynthesizer.getInstance(); 建议每次只使用一个实例。release方法调用后，可以使用第二个。

##### 设置当前的Context

mSpeechSynthesizer.setContext(this); *// this 是Context的之类，如*

*Activity*

注意 setContext只要在SpeechSynthesizer.getInstance();设置一次即可，不必切换Context时重复设置。

##### 设置合成结果的回调

如合成成功后，SDK会调用用户设置的SpeechSynthesizerListener 里的回调方法

mSpeechSynthesizer.setSpeechSynthesizerListener(listener);

*//listener是SpeechSynthesizerListener 的实现类，需要实现您自己的业务逻辑。SDK合成后会对这个类的方法进行回调。*

##### 设置 App Id和 App Key 及 App Secret

用户在语音官网或者百度云网站上申请语音合成的应用后，会有appId appKey及appSecret

如：

**String** AppId = "8535996";

**String** AppKey = "MxPpf3nF5QX0pnd\*\*\*\*\*\*cB";

**String** AppSecret = "7226e84664474aa09\*\*\*\*\*\*\*\*b2aa434"

mSpeechSynthesizer.setAppId("8535996"*/\*这里只是为了让Demo运行使用的APPID,请替换成自己的id。\*/*);

mSpeechSynthesizer.setApiKey("MxPpf3nF5QX0pnd\*\*\*\*\*\*cB", "7226e84664474aa09\*\*\*\*\*\*\*\*b2aa434"*/\*这里只是为了让Demo正常运行使用APIKey,请替换成自己的APIKey\*/*);

##### 设置合成参数

可以在初始化设置，也可以在合成前设置。 示例：

mSpeechSynthesizer.setParam(SpeechSynthesizer.PARAM\_SPEAKER, "0");

*// 设置发声的人声音，在线生效*

##### 初始化合成引擎

设置合成的参数后，需要调用此方法初始化

mSpeechSynthesizer.initTts(TtsMode.MIX);

*// 初始化离在线混合模式，如果只需要在线合成功能，使用 TtsMode.ONLINE*

##### 合成及播放接口

该接口线程安全，可以重复调用。内部采用排队策略，调用后将自动加入队列，SDK会按照队列的顺序进行合成及播放。 注意需要合成的每个文本text不超过1024的GBK字节，即512个汉字或英文字母数字。超过请自行按照句号问号等标点切分，调用多次合成接口

返回结果不为0，表示出错。错误码请参见“错误码及解决方法”一节

speak方法示例：

**int** speak(**String** text);

**int** speak(**String** text, **String** utteranceId); *// utteranceId在SpeechSynthesizerListener 相关事件方法中回调*

speechSynthesizer.speak("百度一下");

synthesize方法示例：

**int** synthesize(**String** text);

**int** synthesize(**String** text, **String** utteranceId); *// utteranceId在SpeechSynthesizerListener 相关事件方法中回调*

speechSynthesizer.synthesize("百度一下");

调用这两个方法后，SDK会回调SpeechSynthesizerListener中的onSynthesizeDataArrived方法。 音频数据在byte[] audioData参数中，采样率16K 16bits编码 单声道。连续将audioData写入一个文件，即可作为一个可以播放的pcm文件（采样率16K 16bits编码 单声道）。

##### 合成开始

本次合成过程开始时，SDK的回调

void onSynthesizeStart (String utteranceId);

##### 合成过程中的数据回调接口

合成数据过程中的回调接口，返回合成数据和进度，分多次回调。

void onSynthesizeDataArrived(String utteranceId, byte[] audioData, int progress);

* audioData: 合成的部分数据，可以就这部分数据自行播放或者顺序保存到文件。如果保存到文件的话，是一个pcm可以播放的音频文件。 音频数据是16K采样率，16bits编码，单声道。
* progress 大致进度。从0 到 “合成文本的字符数”。

##### 合成结束

本次合成正常结束状态时，SDK的回调

void onSynthesizeFinish (String utteranceId);

##### 播放开始

SDK开始控制播放器播放合成的声音。如果使用speak方法会有此回调，使用synthesize没有。

void onSpeechStart (String utteranceId);

##### 播放过程中的回调

播放数据过程中的回调接口，分多次回调。 如果使用speak方法会有此回调，使用synthesize没有。

void onSpeechProgressChanged(String utteranceId, int progress);

* progress 大致进度。从0 到 “合成文本的字符数”。

##### 播放结束

播放正常结束状态时的回调方法，如果过程中出错，则回调onError，不再回调此接口。

void onSpeechFinish (String utteranceId);

##### 合成和播放过程中出错时的回调

合成和播放过程中出错时回调此接口

onError(String utteranceId，SpeechError error);

SpeechError 类有2个值：

* code：int，错误码。 具体错误码见“错误码及解决方法”一节
* description： String, 具体的错误信息。

##### AndroidManifest.xml 文件

设置权限：

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.MODIFY\_AUDIO\_SETTINGS" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.WRITE\_SETTINGS" />

<uses-permission android:name="android.permission.READ\_PHONE\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.CHANGE\_WIFI\_STATE" />

##### 设置AppId, AppKey, SecretKey

mSpeechSynthesizer.setAppId(appId);

mSpeechSynthesizer.setApiKey(appKey, secretKey);

##### android 6.0 以上版本权限申请

*/\*\**

*\* android 6.0 以上需要动态申请权限*

*\*/*

**private** **void** initPermission() {

**String** permissions[] = {

Manifest.permission.INTERNET,

Manifest.permission.ACCESS\_NETWORK\_STATE,

Manifest.permission.MODIFY\_AUDIO\_SETTINGS,

Manifest.permission.WRITE\_EXTERNAL\_STORAGE,

Manifest.permission.WRITE\_SETTINGS,

Manifest.permission.READ\_PHONE\_STATE,

Manifest.permission.ACCESS\_WIFI\_STATE,

Manifest.permission.CHANGE\_WIFI\_STATE

};

**ArrayList**<**String**> toApplyList = new **ArrayList**<**String**>();

for (**String** perm : permissions) {

if (PackageManager.PERMISSION\_GRANTED != ContextCompat.checkSelfPermission(this, perm)) {

toApplyList.add(perm);

*//进入到这里代表没有权限.*

}

}

**String** tmpList[] = new **String**[toApplyList.size()];

if (!toApplyList.isEmpty()) {

ActivityCompat.requestPermissions(this, toApplyList.toArray(tmpList), 123);

}

}

@**Override**

**public** **void** onRequestPermissionsResult(**int** requestCode, **String**[] permissions, **int**[] grantResults) {

*// 此处为android 6.0以上动态授权的回调，用户自行实现。*

}

## 总结

### 参考文献

1. 吕俊哲，图像二值化算法研究及其实现[J]，科技情报开发与经济，2004，（3）：125-129
2. 苏金明、王永利，MATALAB图形图像[M]，河北：华北电力大学，2008
3. 陈友仁、赵正校，基于隐马尔可夫模型的车牌自动识别技术，上海，2001
4. 张斌,全昌勤,任福继. 语音合成方法和发展综述[J]. 小型微型计算机系统,2016,(01):186-192.
5. 王志超、刘惠义，一种基于隐马尔可夫模型的人脸识别方法[A]，计算机应用与软件，2013 年，第 30 卷第 2 期
6. 52人工智能，语音合成技术，2017年，<https://blog.csdn.net/wja8a45TJ1Xa/article/details/78599509>
7. 鄢志杰, 阿里云语音识别模型端核心技术选讲,2018年，<https://www.jianshu.com/p/7094c0aa37ee>