微信语音 SDK(Android 版)上手指南

使用说明

本文档主要介绍微信语音 SDK（Android 版）使用方法，利用 SDK 可以直接完成从录音到网络传输、云端语音识别、结果获取等一系列动作。

本文属于入门级文档，旨在帮助开发者快速学习 Android SDK 的使用并应用到自身开发工作。具体 API 可查询《Android 开发手册》。

获取应用授权码

所有使用本语音识别服务的应用都需要有一个应用授权码。使用该应用授权码可以帮你监控语音识别服务的使用情况。

要获取应用授权码，请执行以下操作：

1. 请访问微信语音开放平台的开发者页面（网址：http://pr.weixin.qq.com/voice/login），并使用你的 QQ 账号登陆；
2. 完善个人或者公司信息；
3. 登记我的应用；

此处需要注意的点，一是开发者需要根据应用的领域选择合适的应用分类（如下图），选择合适的应用分类可以明显提高语音识别准确率。

图 1

另一处需要注意的是应用签名的填写（如下图）：

图 2

可以在 Eclipse 中直接查看，Windows -> Preferences -> Android -> Build，界面如图 3， 其中的 SHA1 fingerprint 就是需要的应用签名值：

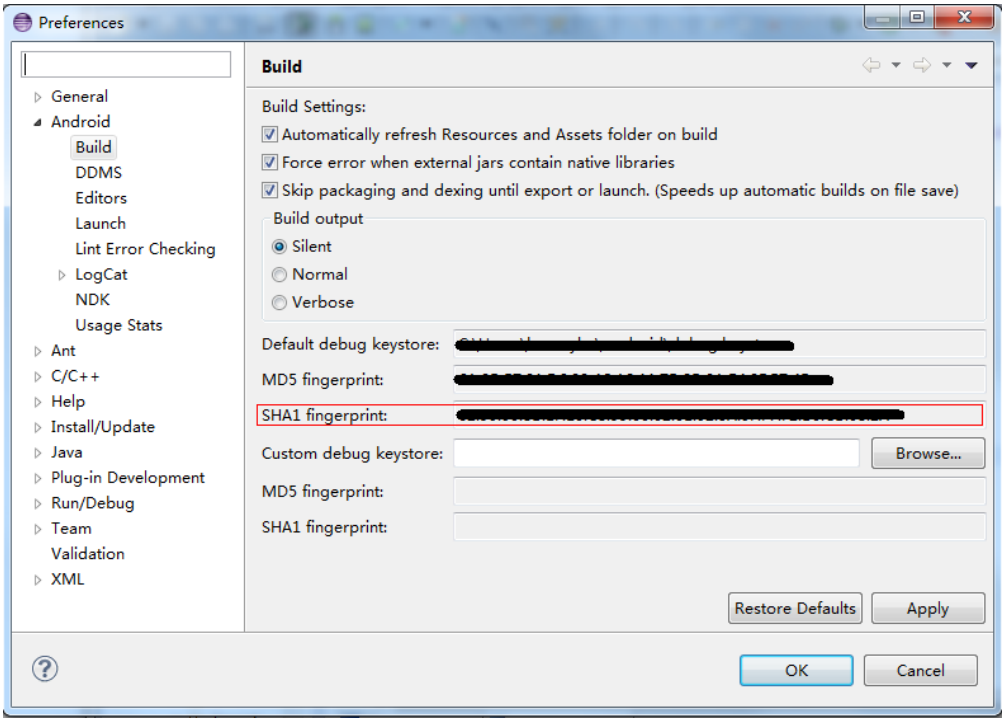

图 3

4. 注册完毕，我们会在 3 个工作日内完成审核工作。待应用审核通过后，用户就可以在“我的应用”界面看到对应的应用授权码。

下载 Android SDK

进入下载页（网址：http://pr.weixin.qq.com/voice/download），下载 Android SDK，压缩包中包括 Demo+SDK+开发文档+测试应用授权码。其中的 Demo 使用 SDK 的各功能 API；SDK 包括.so 和.jar 文件；测试应用授权码的作用是为了让你的应用审核通过之前，使用该应用授权码进行前期的开发工作，为了后续更好的使用，请在审核通过后，尽快用正式的应用授权码替换。

Demo 介绍

为了更好的理解微信语音 SDK 的使用，下面将通过一个简单的实例来讲解一下 SDK 各个关键 API 接口的使用。

1. 开发工具

Android 开发工具有很多，开发者可以根据自身的喜好来选择。在讲解本示例的时候，我们将使用 Eclipse 来一步步分析。

2. 工程配置

首先新建一个示例工程，按以下步骤进行配置。

i. 引入.so 文件：

如下图，在 libs\armeabi 目录下粘贴 libWXVoice.so 文件。

图 4

ii. 引入 jar 包：

在工程属性->Java Build Path->Libraries 中选择“Add External JARs”，选定 wxvoice.jar，确定后返回，完成后的效果如图 4 所示。

3. AndroidManifest.xml 设置

i. 添加必要的权限支持：

```
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
<uses-permission android:name="android.permission.RECORD_AUDIO"></uses-permission>
```

4. 语音识别 Demo（包括通用识别和语法识别）

关于界面的构建此处不进行介绍， 下面着重介绍语音识别 SDK 各 API 的使用：

i. 初始化

```
VoiceRecognizer.shareInstance().setSilentTime(1000);
VoiceRecognizer.shareInstance().setListener((VoiceRecognizerListener)this);
if (VoiceRecognizer.shareInstance().init(this, screKey) != 0) {
    //初始化失败
}
```

注：此处的 screKey 就是在官网上申请的应用授权码；对于语法识别将 VoiceRecognizer 类替换成 VoiceRecognizerGrammar。

ii. 开始识别

通用识别：

```
VoiceRecognizer.shareInstance().start();
```

语法识别：

```
VoiceRecognizerGrammar.shareInstance().start(text, type)
```

注：text 是语法的具体内容， type 是语法类型，ABNF 语法类型为 0， 自定义词表识别类型为 1。

开始后的各种阶段状态将由 public void onGetVoiceRecordState(VoiceRecordState state)回调获取。

iii. 结束录音接口

结束录音有两种方式， 一种是使用下面的接口：

通用识别：

```
VoiceRecognizer.shareInstance().stop();
```

语法识别：

```
VoiceRecognizerGrammar.shareInstance().stop();
```

另外一点是等待程序的静音检测， 自动结束录音。设置静音检测时长的接口为 setSilentTime(int time)， 上面初始化中已经介绍。 默认静音时长为 1.5 秒。

iv. 取消识别

通用识别：

```
VoiceRecognizer.shareInstance().cancel();
```

语法识别：

```
VoiceRecognizerGrammar.shareInstance().cancel();
```

此接口的如果返回非 0 值， 则表示 cancel 过程结束， 否则其状态将通过 public void onGetVoiceRecordState(VoiceRecordState state)回调获取。

v. 实现 VoiceRecognizerListener，并重写其函数

直接在 Activity 类中实现 VoiceRecognizerListener（或者 VoiceRecognizerGrammar），重写 VoiceRecognizerListener 各函数如下所示：

```
@Override
public void onGetError(int arg0) {
    // TODO Auto-generated method stub
}

@Override
public void onGetResult(VoiceRecognizerResult result) {
    // TODO Auto-generated method stub
```

```
            String res = "";
            if (result != null && result.words != null) {
                int wordSize = result.words.size();
                StringBuilder results = new StringBuilder();
                for (int i = 0; i<wordSize; ++i) {
                    Word word = (Word) result.words.get(i);
                    if (word != null && word.text != null) {
                        results.append("\r\n");
                        results.append(word.text.replace(" ", ""));
                    }
                }
                results.append("\r\n");
                res = results.toString();
            }
        }

        @Override
        public void onGetVoiceRecordState(VoiceRecordState state) {
            // TODO Auto-generated method stub
        }

        @Override
        public void onVolumeChanged(int arg0) {
            // TODO Auto-generated method stub
        }
```

vi. 释放系统资源

通用识别：

VoiceRecognizer.*shareInstance*().destroy();

语法识别：

VoiceRecognizerGrammar.*shareInstance*().destroy();

注：调用此函数， 需要确保识别过程或者 cancel 过程已经结束， 否则会出现问题。

vii. Demo 截图

编译运行，图 5、图 6 和图 7 分别是通用语音识别、自定义词表识别和 ABNF 语法识别显示的界面：

图 5

图 6

图 7

更多内容请参考《Android 开发手册》和 WXVoiceSDKDemo。

5. 语音识别 Demo（有 UI 版）

i. 初始化

```
mVoiceRecognizerDialog = new VoiceRecognizerDialog(this, mType);
mVoiceRecognizerDialog.setSilentTime(1000);
mVoiceRecognizerDialog.setOnRecognizerResultListener(this);
mRecoInitSucc = mVoiceRecognizerDialog.init(MainActivity.screKey);
if (mRecoInitSucc != 0) {
    mResText.setText("语音识别引擎初始化失败");
}
```

注：mType 用于设置识别的类型， -1：通用识别， 0：ABNF 语法识别， 1：自定义词表识别。

ii. 开始识别

通用识别：

```
mVoiceRecognizerDialog.show();
```

语法识别：

```
mVoiceRecognizerDialog.show(mSynWords, mType);
```

iii. 退出弹窗

```
mVoiceRecognizerDialog.onDismiss();
```

iv. 释放系统资源

```
mVoiceRecognizerDialog.onDestroy();
```

v. Demo 截图

图 8

更多内容请参考《Android 开发手册》和 WXVoice。

6. 语音合成 Demo

i. 初始化

```
SpeechSynthesizer.shareInstance().setListener(this);
SpeechSynthesizer.shareInstance().setFormat(0);
SpeechSynthesizer.shareInstance().setVolume(1.0f);
mInitSucc = SpeechSynthesizer.shareInstance().init(this, screKey);
if (mInitSucc != 0) {
        Toast.makeText(this, "初始化失败",
                Toast.LENGTH_SHORT).show();
}
```

注：setFormat 函数用于设置请求的音频类型 0:mp3, 1:wav, 2:amr；setVolume 函数用于设置返回音频音量的大小， 范围在 0 – 2 之间的浮点数值。

ii. 开始合成

```
int ret = SpeechSynthesizer.shareInstance().start(mSynWords);
```

```
if (0 == ret) {
    return;
}
else if (-402 == ret) {
    Toast.makeText(this, "ErrorCode = " + ret +"; 文本不能为空", Toast.LENGTH_LONG).show();
}
else if (-403 == ret) {
    Toast.makeText(this, "ErrorCode = " + ret +"; 字符数超过1024", Toast.LENGTH_LONG).show();
}
else {
    Toast.makeText(this, "ErrorCode = " + ret, Toast.LENGTH_LONG).show();
}
```

iii. 取消合成

SpeechSynthesizer.*shareInstance*().cancel()

此接口的如果返回非0值， 则表示cancel过程结束， 否则其状态将通过public void onGetVoiceRecordState(TextSenderState state)回调获取。

iv. 实现 TextSenderListener， 并重写其函数

直接在 Activity 类中实现 TextSenderListener，重写 TextSenderListener 各函数如下所示：

```
@Override
public void onGetResult(TextSenderResult arg0) {
    SpeechSynthesizerResult result = (SpeechSynthesizerResult)arg0;
    byte [] speech = result.speech;
}

@Override
public void onGetError(int errorCode) {
    // TODO Auto-generated method stub
}

@Override
public void onGetVoiceRecordState(TextSenderState state) {
    // TODO Auto-generated method stub
}
```

v. 释放系统资源

SpeechSynthesizer.*shareInstance*().destroy();

注：调用此函数， 需要确保识别过程或者 cancel 过程已经结束， 否则会出现问题。

vi. Demo 截图

图 9

更多内容请参考《Android 开发手册》和 WXVoice。

无 UI 的使用规范

微信语音开放平台免费为你的应用提供语音识别服务，你可以根据自己的风格自由制定 UI，但需在语音采集识别的窗口正确、完整的标注“**Powered by 微信智能**”或“**语音技术由微信智能提供**”的字样。参考如下弹窗：

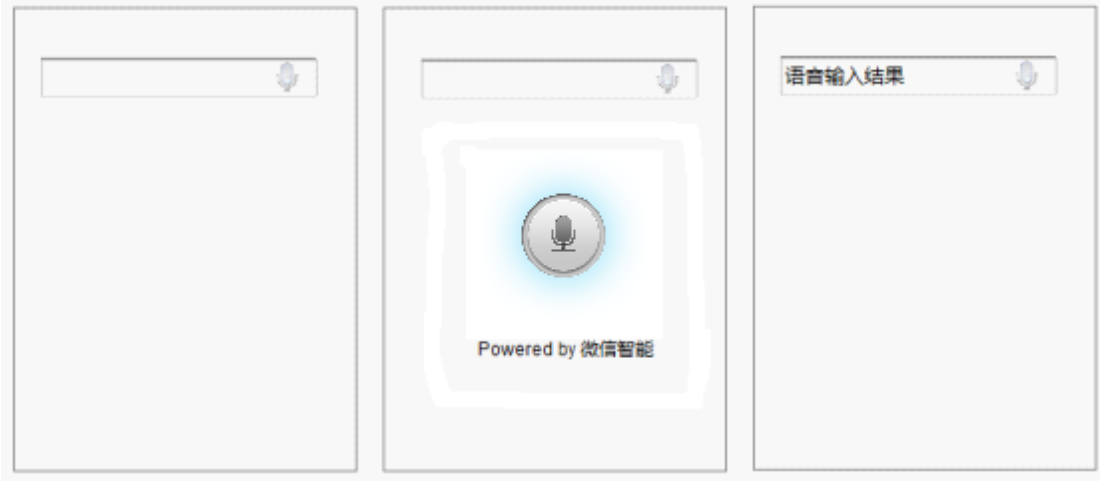

图 10

附录

ABNF 语法手册

语音识别语法用来告知语音识别器（Recognizer）所要识别的语句形式。它包括：

- 用户将要说的词汇（word）
- 将要出现的词汇的模式（pattern）
- 每个词汇的语言

语法文档被编译成识别网络后，将被送往语音识别器。语音识别器提取输入语音的特征信息并在识别网络上进行路径匹配，最终识别出用户说话的内容。因此语法是语音识别系统的输入之一，它是现阶段语音识别得以应用的必要条件。

```
#ABNF 1.0 UTF-8;

language zh-cn;

mode voice;

root $basicCmd;

public $basicCmd =[查询]$allnames 路况;

$allnames = (成乐 | 京昆 | 广深)[高速];
```

例如，开发一个高速路况查询的简单语音识别系统，可以定义如下的语法：

目前，语音识别语法可以采用的是 ABNF 形式，ABNF 形式简练，便于用户理解和书写。

ABNF 文档包括两个部分，文档首部（header）和主体(body)。文档首部定义了文档的各种属性，而文档的主体则具体定义了用户说话的内容和模式。文档首部包括：

- ABNF 文档自标识头
- 语言
- 模式
- 根规则

注意：文档首部必须出现在文档的开头部分，也就是说一旦出现了第一个规则定义，即宣告文档首部的结束，出现在文档主体中的文档首部声明，作为文档主体（规则扩展）看待，通常会引起编译器报错。用户务必注意这一点。

1. ABNF 文档自标识头（Self-Identifying Header，强制）

ABNF 文档必须在其第一行包含一个形式如下的自标识头：

#ABNF *VersionNumber CharEncoding ;*

文档自标识头定义了文档的版本和编码格式。其中：

VersionNumber 指定语法文档版本号，目前版本号必须是1.0。

CharEncoding 指定语法文档字符编码类型，其默认值是 GB2312。

开发语法时，请确保文档自标识头声明的字符编码类型和文件的真实字符编码类型是统一的。

例如，如果声明的文档编码格式是 UTF-8的编码格式，文档的内容必须采用这一格式。因为语法编译系统依靠这一声明的字符编码，把文档转换成统一的 Unicode 格式。因此，如果声明的字符编码和文件真实的字符编码不一致，那么转换会出错。

讯飞语音识别系统的语法编译子系统可以支持 ISO-8859-1、GB2312、GBK、UTF-8、UTF-16LE、UTF-16BE 等多种格式的文本编码方式。

推荐中国大陆用户使用 GB2312编码类型，并在语法中预先声明它。

```
#ABNF 1.0 GB2312;
```

2. 语言声明（Language，可选）

在进行中文识别时，推荐使用的语言声明为 zh-cn。ABNF 语言声明具有以下形式：

```
language zh-cn;
```

3. 模式（Mode，强制）

目前，语法引擎仅支持 voice 模式。

```
mode voice;
```

整个语法主体的规则扩展展开是一棵或多棵语法树，在应用中推荐为语法指定一个根规则，根规则可以看作是整个语法树的根，同时根规则也定义了外部引用该语法的默认引用规则。外部引用的根规则必须定义成 public 的。注意一个语法文档能且只能定义一个根规则：

ABNF 根规则声明具有如下形式：

```
root $basicCmd;
```

语法文档中可以使用注释。ABNF 形式的注释如下：

```
// C++/Java-style single-line comment
/* C/C++/Java-style comment */
/** Java-style documentation comment */
```

语音识别语法是通过规则定义（Rule Definition）和规则引用（Rule Reference）来组成语法主体（Body）的。规则引用的各种组合通称为规则扩展（Rule Expansion）。规则扩展是一个正则表达式。一个规则定义用“=”把规则名称和规则内容联系起来，规则名称具有“$+字符串的形式”，而规则内容就是所谓的规则扩展。规则扩展的最基本的结构是顺序、选择和循环。

```
[scope] $ruleName = ruleExpansion;
```

在高速公路路况查询的语法中：

```
public $basicCmd = [查询] $allnames 路况;

$allnames =

( 成乐 | 京昆 | 广深 )[高速];
```

从而造成编译错误。一个规则定义可以连续引用多个规则名，记号名以及它们的各种组合：包括顺序结构、选择结构、重复结构、可选结构。

7. 顺序结构

一个规则定义可以连续引用多个规则名，记号名以及它们的各种组合。序列相当于程序设计中的顺序结构。例如 ABNF 形式：

这是一个测试用例 // token 的序列
$action $object //规则引用的序列
(查询 $allnames 路况) //用括号来封装

8. 选择结构

在规则扩展中选择结构表示说话时只可能覆盖其中的一条路径,它相当于程序设计语言中的选择结构。选择结构的在 ABNF 中用“|”来分隔多个选择分支：

```
$allnames = 成乐 | 京昆 | 广深 ;
```

重复结构用来在语法中表示需要重复说出的内容,它特别适合表示诸如数字串识别等有一些简单词语反复出现构成的语法结构。

ABNF 形式的重复结构可以由在被引用语法结构（规则、记号或它们的任意组合）后加上重复标签<min - max>来设定。min 表示最小重复次数，max 表示最大重复次数。

ABNF 形式的重复结构表示的一个例子如下：

```
$rulename = $digit<2-8>;

$digit = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0;
```

10. 可选结构：

在高速公路路况查询的语法中，查询一词可出现，也可不出现，用[]表示其可选性：

```
public $basicCmd = [查询] $allnames 路况;
```

在语法开发的过程中，用户还必须注意下面这些问题：

- 不要出现规则的递归定义，及一个规则的定义直接或者间接地引用自己。实用语法不需要递归结构。
- 开发的过程中考虑准确性和性能的统一。例如：稍微长一些的词语有利于提高识别率；能够合并的部分尽可能合并已减少语法网络的大小，这样可以有效地提高识别系统的性能。