**[对于语音识别测试的学习与思考（一）](http://blog.csdn.net/yimingt/article/details/72731345)**

[作者同类文章](http://blog.csdn.net/u011319920/article/category/6938840)X

**一、 先来说说语音是如何变成文字：**

大家都知道，声音是种波形，处理波形必须使用非压缩的纯波形文件，请大家注意非压缩三个字，我们常用到的mp3等格式的文件都是压缩文件，而非压缩的纯波形文件则是Windows PCM文件，也就是wav文件。wav文件存储的波形示例如下：



开始语音识别前，需要把收尾两端的静音切除降低对后续步骤造成干扰，这个静音切除动作，就一般称为**VAD**，具体含义如下：

**VAD：语音活动检测(Voice Activity Detection)** 目的是从声音信号流里识别和消除长时间的静音期，以达到在不降低业务质量的情况下节省[话路](http://baike.baidu.com/item/%E8%AF%9D%E8%B7%AF/5922604" \t "_blank)资源的作用。

通俗的来解释VAD就是：**判断什么时候有语音什么时候没有语音（也就是静音）。后续的语音信号处理或是语音识别都是在VAD截取出来的有效语音片段上进行的，这样可以极大的减少计算量，同时也能减少噪声误识别等情况。**

结合到实例，大家记得之前有个BUG，就是说话间隔较长（拉长音）会导致crash，我怀疑就是VAD方面处理存在问题。

我们继续讲解语音是如何变成文字的，要对声音进行分析，首先要把声音切开，切成一小段一小段的，每一小段称为一帧，分帧的操作不是简单或者随意的切，这里面一般通过具体的函数来实现，这个…哥也不懂。**而我们只需要关注Rokid有没有切错，切坏就可以了。**

分帧之后，语音就变为了很多个小段，这就可以通过算法变成文字了么？答案是：NO！

分帧后还要进行波形变换，波形变换通常是根据人耳的生理特征，把每一帧波形变为一个多维向量，这个过程叫做声学特征提取，简单理解就是这些向量包含这帧语音的内容信息就可以了，至此声音就成了一个12行（假设声学特征是12维），N列的矩阵，这里N就是总帧数。

到这个时候，才可以将声音通过矩阵变成文本，首先要了解两个概念：

1.       音素：单词的发音有音素组成。汉语一般用全部声母韵母作为音素集，另外汉语还区分声调。英语的音素集我们可以暂不去了解，因为Rokid还未做好melody。我们提过BUG：“我要听Misic”，估计问题就源于音素集

2.       状态：简单理解就是比音素更小的单位即可，通常一个音素划分为3个状态。

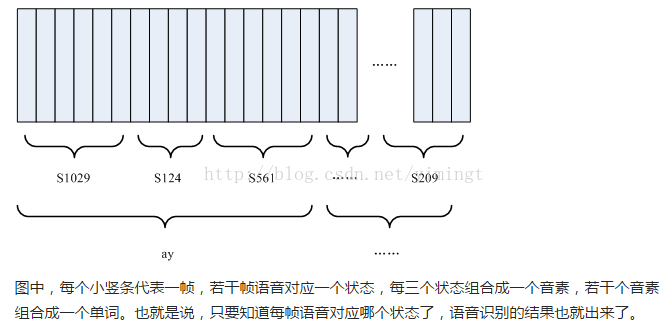
到这里，语音识别是怎么工作的，就可以简单的归纳为：

1.       把帧识别为状态（这是开发的难点，也是BUG最集中的地方）

2.       把状态组合成音素

3.       把音素组合为单词

图例如下：



也就是说，知道每帧语音对应哪个状态，语音识别的结果就知道了，那么怎么判断相对应的状态呢？刚才我说过了，这个是开发的难点，也是BUG最集中的地方。通常的算法是：**通过判断这帧语音对应哪个状态的概率最大，这帧就属于哪个状态。**

举个例子，我们提过bug，对Pebble讲“我想听慢慢”。结果Pebble将慢慢识别为“妈妈”，就说明Rokid的算法中此类（帧）语音对应的状态的概率：“妈妈”>“慢慢”，当然因为两个字发音不同，所以状态识别也存在问题。

说到这里，那么问题又来了，这个概率，Rokid是怎么来判断获取的呢？有个东东叫做：“声学模型”，具体就不描述了，大家就需要知道这里面存了一大堆参数即可，通过这些参数来判断具体的状态对应的概率。目前这个声学模型是天天的团队负责维护完善。

声学模型有很多种，不一一介绍，大家可以自行百度，目前也不知道Rokid用的哪一种，不过可以这样通俗的理解：

1.       构建一个状态网络，好比一个很大很大的篮子，里面有很多很多的玻璃球，每个玻璃球都不一样。

2.       从状态网络中寻找声音最匹配的路径，好比在这个大篮子里找到特定的几个玻璃球。

我们要测的就是这几个玻璃球Rokid找的对不对，但是这个篮子越大，玻璃球找起来越困难，而Pebble的需求是可以识别任意文本，所以玻璃球就非常非常多，这个篮子也非常非常庞大，而我们发现的问题可能只是冰山一角，所以还需要大家多挖掘。

以上的介绍就是语音如何变成文字的大致过程，**请大家结合流程来补充或者强化自己的测试方法和测试用例。**

**二、 语音识别之VAD**

刚刚给大家介绍过VAD，指的就是判断什么时候有语音什么时候没有语音（静音）。后续的语音信号处理或是语音识别都是在VAD截取出来的有效语音片段上进行的，这样可以极大的减少计算量，同时也能减少噪声误识别等情况。

目前语音识别区分为：进场识别，远场识别，我们的Pebble就属于远场识别，就是必须用到VAD的

在**近场识别**的时候，这个算法可有可无：比如微信或者一些语音输入法中可以用手按着说话按键说话，结束之后松开。即便有，也可以做的比较简单：例如可以用能量、音高、过零率等方式进行判断。因为近场情况下信噪比（SNR）比较高，信号清晰，简单算法也可以做到有效可靠。

**远场识别**情况下不能用手接触设备，必须使用VAD。这时噪声比较大，SNR下降剧烈，这种情况下传统的方法效果不理想。近几年深度神经网络（deep neural network，DNN）火热，基于DNN(包括DNN, CNN, RNN)的语音系统成为标配，因此VAD使用DNN来做也合情合理。但即便是这样，仍然有很大的挑战，简单的如：开关门的声音、鼓掌、跺脚、宠物叫声，**困难的如：电视的声音、多人的交谈声等等**。所以VAD仍然是远场语音识别的一个研究话题。

**So，后续测试的关注点，请多关注其他的声源、多人交谈的识别。**

**三、 语音唤醒(VT)**

可以理解为喊名字，引起听者的注意。比如苹果的“hey Siri”，Google的“OK Google”，以及亚马逊echo系列产品的“Alexa”等。而我们的Pebble就是“若琪”。VT的目的是告诉机器人（或手机），接下来的话是对你讲的哦，认真听好了。VT通常是在VAD检测到人声之后进行工作，他要判断人说的话是不是激活词，如果是激活词，那么后续的语音就进行识别，否则后续的语音不进行处理。

语音识别，不论是远场还是近场，基本都是在云端的，这样可以使用大模型，服务器的高速计算等优势获得好的识别结果。而VT基本是在本地设备上，因此它的要求更高。技术难点具体体现在以下几个方面：

1.       功耗要低。iphone 4s出现Siri，但直到iphone 6s之后才允许不接电源的情况下直接喊“hey Siri”进行语音唤醒，在此之前都需要接入电源的情况下进行语音唤醒。这是因为有6s上有一颗专门进行语音激活的低功耗芯片，当然算法和硬件要进行配合，算法也要进行优化。

2.       计算量要小。计算量小一方面是低功耗的要求，另一方面要是实时率的要求。即1s的语音至少要在1s之内计算完毕，这样才能保障可实用。

3.  唤醒效果。理想状况下的效果是喊它的时候它立马就能相应，不喊它的时候它乖乖的不要插话。但人生不如意十之八九，喊它的时候它不答应这叫做漏报，没喊它的时候它跳出来讲话叫做误报。漏报和误报这俩老兄是此消彼长的关系：比如唤醒词很长有10个字，那么误报肯定小，但同时漏报也就多了，喊了好几次都不理你是不是很气人？！再比如唤醒词很短只有一个字，这个时候漏报是少了，每次喊都很快回应你，但是误报也多了呀，大半夜突然跳出来说几句话那也是相当惊悚的。因此起名字也很关键，通常名字越短，越困难。比如“OK Google”和“Alexa”有四个音节，“hey Siri”有三个音节。因此，起一个朗朗上口，百喊不厌的名字也能从侧面体现出算法水平。

**综上所述，我们的激活相关的测试关注点就更清晰了，功耗、响应时长、激活率&误激活率需要大家重点去关注**

**四、回声消除 (Echo Cancelling)**

这是全双工通信中的一个名词，即在播放的同时可以拾音。近场环境下，播放音乐或是语音播报的时候可以按键停止这些，但远场环境下想要通过语音控制，必须得实现回声消除。

回声消除的难点在于它要和智能设备的音响效果之间有一个平衡和折中。如果音响设备重低音比较好，振动就会加剧，这样麦克风的拾音效果就要大打折扣。另外音响离麦克风的距离也会对拾音有影响。Amazon的echo长的很高，音响在最下面，麦克风在顶上的设计也是为了更好的实现回声消除。

**但是我们的Pebble这么一个矮胖子，对算法的要求就很高了，所以我们要对回声消除要更深入的测试。**

**五、最后**

语音识别相关的就先给大家讲这么多，我也在摸索学习中，不足之处欢迎指正，后续我会抽空再给大家科普NLP及其他相关的知识，**但是还请大家在工作之余也能自主的去学习了解一些专业知识，来拓展自己的测试思路**，谢谢！

**一.   了解麦克风阵列，大家了解下即可**

**1.**什么是麦克风阵列技术？

学术上有个概念是“传声器阵列”，主要由一定数目的声学传感器组成，用来对声场的空间特性进行采样并处理的系统。而这篇文章讲到的麦克风阵列是其中一个狭义概念，特指应用于语音处理的按一定规则排列的多个麦克风系统，也可以简单理解为2个以上麦克风组成的录音系统。

麦克风阵列一般来说有线形、环形和球形之分，严谨的应该说成一字、十字、平面、螺旋、球形及无规则阵列等。至于麦克风阵列的阵元数量，也就是麦克风数量，可以从2个到上千个不等。这样说来，麦克风阵列真的好复杂，别担心，复杂的麦克风阵列主要应用于工业和国防领域，消费领域考虑到成本会简化很多。

**2.**为什么需要麦克风阵列？

消费级麦克风阵列的兴起得益于语音交互的市场火热，主要解决远距离语音识别的问题，以保证真实场景下的语音识别率。这涉及了语音交互用户场景的变化，当用户从手机切换到类似Echo智能音箱或者机器人的时候，实际上麦克风面临的环境就完全变了，这就如同两个人窃窃私语和大声嘶喊的区别。

前几年，语音交互应用最为普遍的就是以Siri为代表的智能手机，这个场景一般都是采用单麦克风系统。单麦克风系统可以在低噪声、无混响、距离声源很近的情况下获得符合语音识别需求的声音信号。但是，若声源距离麦克风距离较远，并且真实环境存在大量的噪声、多径反射和混响，导致拾取信号的质量下降，这会严重影响语音识别率。而且，单麦克风接收的信号，是由多个声源和环境噪声叠加的，很难实现各个声源的分离。这样就无法实现声源定位和分离，这很重要，因为还有一类声音的叠加并非噪声，但是在语音识别中也要抑制，就是人声的干扰，语音识别显然不能同时识别两个以上的声音。

显然，当语音交互的场景过渡到以Echo、机器人或者汽车为主要场景的时候，单麦克风的局限就凸显出来。为了解决单麦克风的这些局限性，利用麦克风阵列进行语音处理的方法应时而生。麦克风阵列由一组按一定几何结构（常用线形、环形）摆放的麦克风组成，对采集的不同空间方向的声音信号进行空时处理，实现噪声抑制、混响去除、人声干扰抑制、声源测向、声源跟踪、阵列增益等功能，进而提高语音信号处理质量，以提高真实环境下的语音识别率。

事实上，仅靠麦克风阵列也很难保证语音识别率的指标。麦克风阵列还仅是物理入口，只是完成了物理世界的声音信号处理，得到了语音识别想要的声音，但是语音识别率却是在云端测试得到的结果，因此这两个系统必须匹配在一起才能得到最好的效果。

不仅如此，麦克风阵列处理信号的质量还无法定义标准。因为当前的语音识别基本都是深度学习训练的结果，而深度学习有个局限就是严重依赖于输入训练的样本库，若处理后的声音与样本库不匹配则识别效果也不会太好。从这个角度应该非常容易理解，物理世界的信号处理也并非越是纯净越好，而是越接近于训练样本库的特征越好，即便这个样本库的训练信号很差。显然，这是一个非常难于实现的过程，至少要声学处理和深度学习的两个团队配合才能做好这个事情，另外声学信号处理这个层次输出的信号特征对语义理解也非常重要。

**二.   麦克风阵列的关键技术，也是我们测试中需要重点关注的范围**

**1．噪声抑制**：语音识别倒不需要完全去除噪声，相对来说通话系统中需要的技术则是噪声去除。这里说的噪声一般指环境噪声，比如空调噪声，这类噪声通常不具有空间指向性，能量也不是特别大，不会掩盖正常的语音，只是影响了语音的清晰度和可懂度。这种方法不适合强噪声环境下的处理，但是应付日常场景的语音交互足够了。**测试建议：比如旁边有USB风扇或者其他的情况下，语音控制Pebble**

**2．混响消除**：混响在语音识别中是个蛮讨厌的因素，混响去除的效果很大程度影响了语音识别的效果。我们知道，当声源停止发声后，声波在房间内要经过多次反射和吸收，似乎若干个声波混合持续一段时间，这种现象叫做混响。混响会严重影响语音信号处理，比如互相关函数或者波束主瓣，降低测向精度。**测试建议：吵杂环境下，把Pebble放到墙角类似的位置，周围有墙壁可以反射声音，语音控制Pebble**

**3. 回声抵消**：严格来说，这里不应该叫回声，应该叫“自噪声”。回声是混响的延伸概念，这两者的区别就是回声的时延更长。一般来说，超过100毫秒时延的混响，人类能够明显区分出，似乎一个声音同时出现了两次，我们就叫做回声，比如天坛著名的回声壁。实际上，这里所指的是语音交互设备自己发出的声音，比如Echo音箱，当播放歌曲的时候若叫Alexa，这时候麦克风阵列实际上采集了正在播放的音乐和用户所叫的Alexa声音，显然语音识别无法识别这两类声音。回声抵消就是要去掉其中的音乐信息而只保留用户的人声，之所以叫回声抵消，只是延续大家的习惯而已，其实是不恰当的。**测试建议：Pebble播放自由音源，如播放音乐的时候语音控制Pebble**

**4. 声源测向**：这里没有用声源定位，测向和定位是不太一样的，而消费级麦克风阵列做到测向就可以了，没必要在这方面投入太多成本。声源测向的主要作用就是侦测到与之对话人类的声音以便后续的波束形成。声源测向可以基于能量方法，也可以基于谱估计，阵列也常用TDOA技术。声源测向一般在语音唤醒阶段实现，VAD技术其实就可以包含到这个范畴，也是未来功耗降低的关键研究内容。**测试建议：不同方位唤醒Pebble**

**5. 波束形成**：波束形成是通用的信号处理方法，这里是指将一定几何结构排列的麦克风阵列的各麦克风输出信号经过处理（例如加权、时延、求和等）形成空间指向性的方法。波束形成主要是抑制主瓣以外的声音干扰，这里也包括人声。**测试建议：比如几个人围绕Pebble谈话的时候，Pebble应只会识别其中一个人的声音。**

**6．阵列增益**：这个比较容易理解，主要是解决拾音距离的问题，若信号较小，语音识别同样不能保证，通过阵列处理可以适当加大语音信号的能量。**测试建议：远距离激活Pebble或者发送语音指令**

**7. 模型匹配**：这个主要是和语音识别以及语义理解进行匹配，语音交互是一个完整的信号链，从麦克风阵列开始的语音流不可能割裂的存在，必然需要模型匹配在一起。实际上，效果较好的语音交互专用麦克风阵列，通常是两套算法，一套内嵌于硬件实时处理，另外一套服务于云端匹配语音处理。**这个目前Rokid应该是云端处理的，大家了解即可。**

语音唤醒就是在设备待机状态下，用户说出特定的语音指令（唤醒词）使设备进入工作状态或完成某一操作。设备为了实现语音唤醒功能，就需要设备实时监听，也就是实时的录音并分析有没有唤醒词来唤醒设备。评价一个语音唤醒系统的标准有三个：唤醒正确率、误报率、功耗。一般情况下，唤醒正确率越高 ，误报率也越高。好的系统就需要唤醒率高，误报率低，功耗低。

**语音唤醒一般经过以下流程：**

**1、输入语音**

**2、语音端点检测（VAD）**

**3、信号处理：降噪处理，压缩音频等**

**4、提取声学特征：声音是模拟信号，声音的时域波形只代表声压随时间变化的关系，不能很好的代表声音的特征，因此，必须将声音波形转换为声学特征向量。**

**5、识别解码：未知语音的模式与已知语音的参考模式逐一进行比较，最佳匹配的参考模式被作为识别结果。**

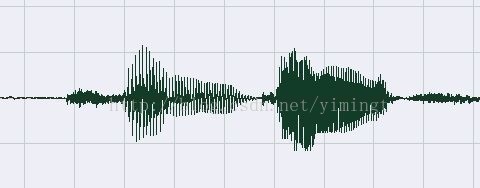
**6、置信度判决：置信度模型可以有效地判断观测数据与语音模型之间的匹配程度 ,因此可以用来对语音识别结果进行假设检验**

**7、唤醒成功，在线解码：唤醒后在线解码之后的语音内容**

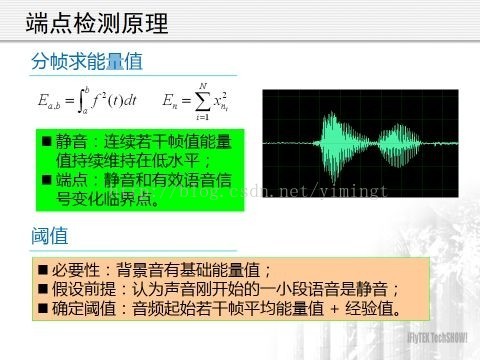
**8、获取识别结果**

**语音端点检测（VAD）**

**音频端点检测**就是从连续的语音流中检测出有效的语音段。它包括两个方面，检测出有效语音的起始点即前端点，检测出有效语音的结束点即后端点。



为了能更清楚说明端点检测的原理，先来分析一段音频。上图是一段只有两个字的简单音频，从图上可以很直观的看出，首尾的静音部分声波的振幅很小，而有效语音部分的振幅比较大，一个信号的振幅从直观上表示了信号能量的大小：静音部分能量值较小，有效语音部分的能量值较大。语音信号是一个以时间为自变量的一维连续函数，计算机处理的语音数据是语音信号按时间排序的采样值序列，这些采样值的大小同样表示了语音信号在采样点处的能量。



采样值中有正值和负值，计算能量值时不需要考虑正负号，从这个意义上看，使用采样值的绝对值来表示能量值是自然而然的想法，由于绝对值符号在数学处理上不方便，所以采样点的能量值通常使用采样值的平方，一段包含N个采样点的语音的能量值可以定义为其中各采样值的平方和。

这样，一段语音的能量值既与其中的采样值大小有关，又与其中包含的采样点数量有关。为了考察语音能量值的变化，需要先将语音信号按照固定时长比如20毫秒进行分割，每个分割单元称为帧，每帧中包含数量相同的采样点，然后计算每帧语音的能量值。

如果音频前面部分连续M0帧的能量值低于一个事先指定的能量值阈值E0，接下来的连续M0帧能量值大于E0，则在语音能量值增大的地方就是语音的前端点。同样的，如果连续的若干帧语音能量值较大，随后的帧能量值变小，并且持续一定的时长，可以认为在能量值减小的地方即是语音的后端点。

现在的问题是，能量值阈值E0怎么取？M0又是多少？理想的静音能量值为0，故上面算法中的E0理想状态下取0。**不幸的是，采集音频的场景中往往有一定强度的背景音，这种单纯的背景音当然算静音，但其能量值显然不为0，因此，实际采集到的音频其背景音通常有一定的基础能量值。**

我们总是假设采集到的音频在起始处有一小段静音，长度一般为几百毫秒，这一小段静音是我们估计阈值E0的基础。对，**总是假设音频起始处的一小段语音是静音**，这一点假设非常重要！！！！在随后的降噪介绍中也要用到这一假设。在估计E0时，选取一定数量的帧比如前100帧语音数据（这些是“静音”），计算其平均能量值，然后加上一个经验值或乘以一个大于1的系数，由此得到E0。这个E0就是我们判断一帧语音是否是静音的基准，大于这个值就是有效语音，小于这个值就是静音。

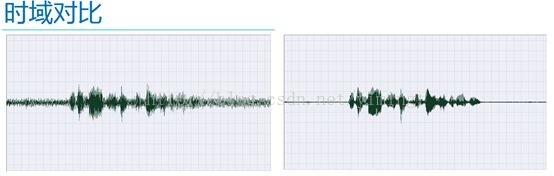
**至于M0，比较容易理解，其大小决定了端点检测的灵敏度，M0越小，端点检测的灵敏度越高，反之越低。**语音应用的场景不同，端点检测的灵敏度也应该被设置为不同的值。例如，在声控遥控器的应用中，由于语音指令一般都是简单的控制指令，中间出现逗号或句号等较长停顿的可能性很小，所以提高端点检测的灵敏度是合理的，M0设置为较小值，对应的音频时长一般为200-400毫秒左右。在大段的语音听写应用中，由于中间会出现逗号或句号等较长时间的停顿，宜将端点检测的灵敏度降低，此时M0值设置为较大值，对应的音频时长一般为1500-3000毫秒。所以M0的值，也就是端点检测的灵敏度，在实际中应该做成可调整的，它的取值要根据语音应用的场景来选择。

以上只是语音端点检测的很简单的一般原理，实际应用中的算法远比上面讲的要复杂。

**信号处理：降噪处理，压缩音频等**

降噪又称噪声抑制（Noise Reduction），前文提到，实际采集到的音频通常会有一定强度的背景音，这些背景音一般是背景噪音，当背景噪音强度较大时，会对语音应用的效果产生明显的影响，比如语音识别率降低，端点检测灵敏度下降（测试时制造多种噪音环境测试）等，因此，在语音的前端处理中，进行噪声抑制是很有必要的。

噪声有很多种，既有频谱稳定的白噪声，又有不稳定的脉冲噪声和起伏噪声，在语音应用中，稳定的背景噪音最为常见，技术也最成熟，效果也最好。本课程只讨论稳定的白噪声，即总是假设背景噪声的频谱是稳定或者是准稳定的。利用上文所说的静音进行降噪处理，此处省略一万字降噪处理方法....下图展示了降噪的效果：



左右两幅图是降噪前后时域中的对比，左面的是含噪语音信号，从图中可以看到噪声还是很明显的。右侧的是降噪后的语音信号，可以看出，背景噪声被大大的抑制了。

在实际应用中，降噪使用的噪声频谱通常不是一成不变的，而是随着降噪过程的进行被持续修正的，即降噪的过程是自适应的。这样做的原因一方面是语音数据前部的静音长度有时不够长，背景噪声数据不足导致得到的噪声频谱往往不够准确，另一方面，背景噪声往往不是绝对稳定的，而是渐变的甚至会突变到另一种稳定的背景噪声。（测试时制造不稳定的背景噪音）

这些原因都要求在降噪的过程中对使用的噪声频谱做及时修正，以得到较好的降噪效果。修正噪声频谱的方法是使用后继音频中的静音，重复噪声频谱提取算法，得到新的噪声频谱，并将之用于修正降噪所用的噪声频谱，所以降噪的过程中仍然要使用端点检测中用到的如何判断静音。噪声频谱修正的方法或者是新旧频谱进行加权平均，或者使用新的噪声频谱完全替换使用中的噪声频谱。

以上介绍的是降噪的非常简单的原理。实际应用中的降噪算法远比上面介绍的要复杂，现实中的噪声源多种多样，其产生的机理和特性也较为复杂，所以噪声抑制在现今仍然是一个较为活跃的研究领域，各种新技术也层出不穷，比如在实际应用中已经使用了多麦克风阵列来进行噪声抑制。

所有的音频压缩系统都要求有两种对应的算法，一种是运行于源端上的编码算法（encoding），另一种是运行于接收端或用户终端的解码算法（decoding）。

解码后得到的数据和编码之前的原始数据可以是不同的，只要它们听起来或看起来是一样的即可。

音视频编解码算法大多是有损的，因为忍受一些少量信息的丢失，往往可以换来压缩率的大幅提升，音频信号的压缩编码采用了数据编码中的一些技术，如熵编码、波形编码、参数编码、混合编码、感知编码等。

重点介绍感知编码，相对于其他的编码算法，感知编码基于人耳听觉的一些特性（心理声学），去除音频信号中的冗余，从而达到音频压缩的目的。相对于其他的音频编码算法（无损的），在人耳没有感觉到明显失真的条件下，可以达到10倍以上的较大压缩率。

音频压缩的核心是去除冗余。所谓冗余就是语音信号中包含的不能为人耳所感知的信息，比如，人耳能听到的声音频率范围为20-20KHz，无法感知频率低于20Hz的次声波和频率高于20KHz的超声波。再比如，人耳也无法听到一段“不够响”的声音。感知编码就是利用了人类听觉系统的这类特性，达到去除音频冗余信息的目的。

**频率屏蔽** 频率屏蔽在生活中处处可见，比如你在家中坐在沙发上安静的看电视，突然，正在装修的邻居家一阵很刺耳的电钻钻墙的声音传来，这时你所能听到的只有手提电钻发出的很强的噪声，尽管此时电视所发出的声音仍然在刺激着你的耳膜，但你却充耳不闻，也就是说，一段强度很高的声音可以完全屏蔽一段强度较低的声音，这种现象称为频率屏蔽。

**时域屏蔽** 承接前一个例子，不仅在电钻发出声音的时间内人耳听不到电视机的声音，就是在电钻的声音刚停下来的一小段时间内，人耳也听不到电视机的声音，这种现象称为时域屏蔽。产生时域屏蔽的原因是人类的听觉系统是一个增益可调的系统，听强度较大的声音时，增益较低，听强度较小的声音时，增益较高。有时人类甚至借助外部手段来改变听觉系统的增益，比如，捂耳朵以避免强度很大的声音损伤耳膜，而屏住呼吸、侧耳、以手放耳廓后更是听较弱声音时的常见行为。在上例中，强度很大的声音刚消失时，听觉系统需要一小段时间来调高增益，正是在这一小段时间内产生了时域屏蔽。

感知编码的基本规则就是，永远不需要对人耳听不到的信号进行编码，简单来说就是，听不到的信号不需要编码，这句废话恰恰是语音压缩研究的重点之一。废话的另外一种含义就是非常容易理解的正确的话。言归正传，哪些东西听不见呢？功率低于可听度阈值的信号或者说分量，被屏蔽的信号或者说分量，这些人耳都听不见，都是上文提到的“冗余”。

要想很好的理解音频压缩，还需要理解一个更重要的概念：子带。子带（subband）是指这样的一种频率范围，当两个音调的频率位于一个子带内时，人就会把两个音调听成一个（对rokid说话调整自己的说话方式测试等）。更一般的情况是，如果一个复杂信号的频率分布位于一个子带内时，人耳的感觉是该信号等价于一个频率位于该子带中心频率处的简单信号，这是子带的核心内涵。简单说，子带是指一个频率范围，频谱位于这个范围内的信号可以用一个单一频率的分量来代替。码率越低压缩率越高时，子带数量少，同时音质较差。相反的情况也容易理解。

理解了子带编码，音频压缩就很容易理解了，一个信号经过一组三角滤波器（等同于一组子带）后，被精简为数量很少的频率分量。然后考察这些频率分量，能量或者说振幅位于可听度阈值曲线之下的直接无视（删除该分量，因为听不到）。再考察余下的两两相邻的频率分量，如果其中一个被旁边的频率屏蔽，也删除掉。经过以上的处理，一个复杂信号的频谱所含有的频率分量就很简单了，使用很少的数据就可以存储或者传输这些信息。

**综上，我们测试的关注点：唤醒正确率、误报率、功耗，以及测试时环境噪音的制造，以及发指令的方式（结合语音端点检测、信号处理涉及内容），结合流程，结合其他同学的资料发掘测试点。**