# Sample：样本

音频数据最基本的单位，长度常见的有 8 位和 16 位；

# Sample Rate：采样率

即每秒的采样次数，针对帧而言；

**Sample rate = number of samples per second**

Channel：声道数，

1-8个, 单声道在声音处理过程中只有单数据流，而立体声则需要左、右声道的两个数据流。显然，立体声的效果要好，但相应的数据量要比单声道的数据量加倍。

Frame：帧

构成一个完整的声音单元，Frame = Sample \_bits\* channels；

# Frame Rate

**Frame Rate = number of frames per second**

For PCM the sample rate and the frame rate are the same since a frame consists of one sample from each channel. For compressed formats like Ogg Vorbis, mp3 and GSM 06.10, the situation is different. A frame is a block of data as it is output by the encoder. Often, these blocks contain the information for **several sampling intervalls**. For instance, a mp3 frame represents about 24 ms. So the frame rate is about 40 Hz. However, the sample rate of the original is preserved even inside the frames and is correctly restored after decoding.

# Bit Rate 比特率

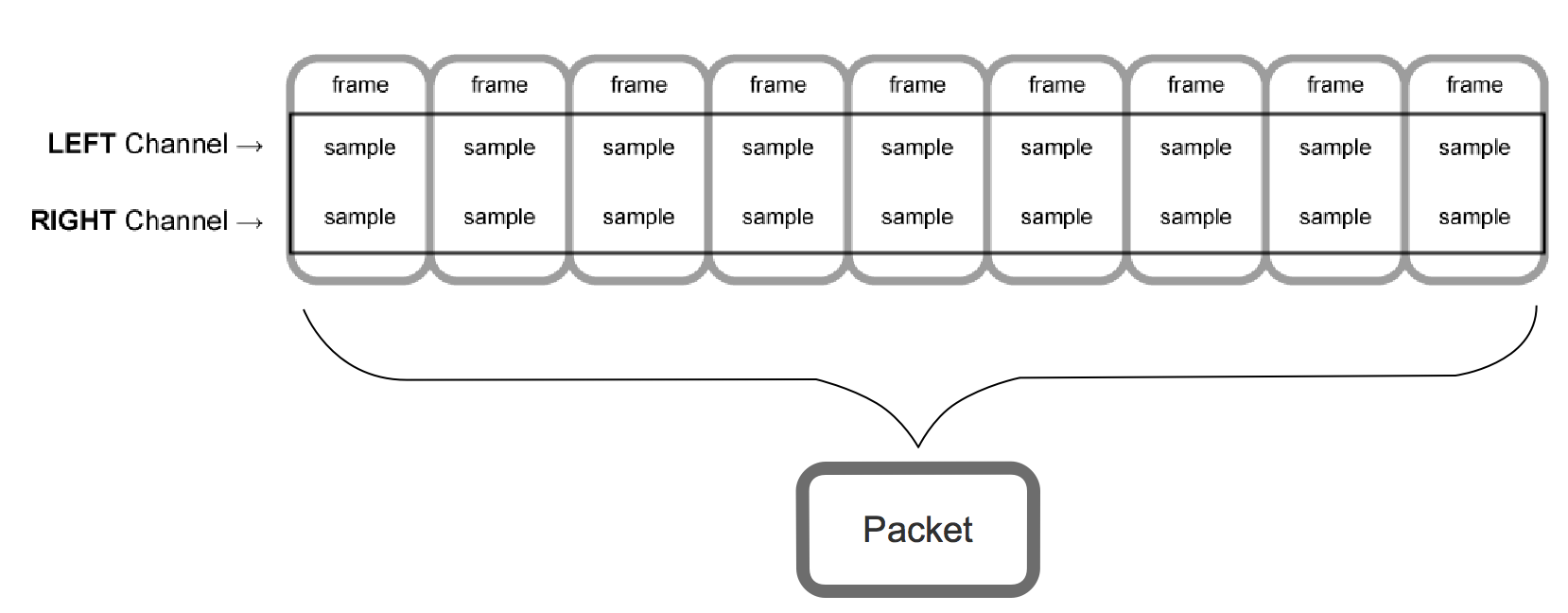
比特率是指每秒传送的比特(bit)数。单位为 bps(Bit Per Second)，比特率越高，传送数据速度越快。声音中的比特率是指将模拟声音信号转换成数字声音信号后，单位时间内的二进制数据量，是间接衡量音频质量的一个指标。

**Bit Rate(bps) = Sample Rate × Audio bit depth × Channel**

**比特率(bps) = 采样率 × 采样位数 × 声道数**

# Packet

是一个或者多个连续frame的集合。对于一个给定的音频数据格式，Packet定义了最小有意义的frame集合，并且是最小的可测量时间单位。



在未压缩的音频中，一个Packet只有一个frame;

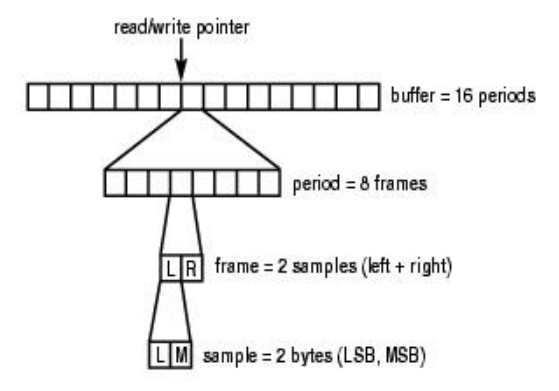
在压缩的音频中，一个Packet是不可在分割的压缩数据块，例如在一个AAC格式的Packet中就包含了1024个采样帧；

Period size周期

，每次硬件中断处理音频数据的帧数，对于音频设备的数据读写，以此为单位；

Buffer size数据缓冲区

这里指 runtime 的 buffer size，而不是结构图 snd\_pcm\_hardware 中定义的 buffer\_bytes\_max；一般来说 buffer\_size = period\_size \* period\_count， period\_count 相当于处理完一个 buffer 数据所需的硬件中断次数。



# 语音的基本概念

语音是一个复杂的现象。我们基本上不知道它是如何产生和被感知的。我们最基础的认识就是**语音是由单词来构成**的，然后每个**单词是由音素**来构成的。但事实与我们的理解大相径庭。语音是一个动态过程，不存在很明显的部分划分。通过音频编辑软件去查看一个语音的录音对于理解语音是一个比较有效的方法。下面就是一个录音在音频编辑器里的显示的例子。

目前关于语音的所有描述说明从某种程度上面讲都是基于概率的（基于频谱？）。这意味着在语音单元或者单词之间并没有确定的边界。语音识别技术没办法到达100%的准确率。这个概念对于软件开发者来说有点不可思议，因为他们所研究的系统通常都是确定性的。另外，对于语音技术来说，它会产生很多和语言相关的特定的问题。

# 语音的构成

在本文中，我们是按照以下方式去理解语音的构成的：

**语音是一个连续的音频流**，它是由**大部分的稳定态**和**部分动态改变**的状态混合构成。

一个单词的发声（波形）实际上取决于很多因素，而不仅仅是音素，例如音素上下文、说话者、语音风格等；

**协同发音**（指的是一个音受前后相邻音的影响而发生变化，从发声机理上看就是人的发声器官在一个音转向另一个音时其特性只能渐变，从而使得后一个音的频谱与其他条件下的频谱产生差异。）的存在使得音素的感知与标准不一样，所以我们**需要根据上下文来辨别音素**。将一个音素划分为几个亚音素单元。如：数字“three”，音素的第一部分与在它之前的音素存在关联，中间部分是稳定的部分，而最后一部分则与下一个音素存在关联，这就是为什么在用HMM模型做语音识别时，选择音素的**三状态HMM模型**。（上下文相关建模方法在建模时考虑了这一影响，从而使模型能更准确地描述语音，只考虑前一音的影响的称为Bi-Phone，考虑前一音和后一音的影响的称为Tri-Phone。）

有时候，音素会被放在上下文中考虑，这样就形成了三元音素或者多元音素。但它与**亚音素**不同，他们在波形中匹配时长度还是和单一音素一样。只是名字上的不同而已，所以我们更倾向于将这样的多元音素称为**senone**。一个senone的上下文依赖比单纯的左右上下文复杂得多，它是一个可以被决策树或者其他方式来定义的复杂函数。（英语的上下文相关建模通常以音素为基元，由于有些音素对其后音素的影响是相似的，因而可以通过音素解码状态的聚类进行模型参数的共享。聚类的结果称为senone。决策树用来实现高效的triphone对senone的对应，通过回答一系列前后音所属类别（元/辅音、清/浊音等等）的问题，最终确定其HMM状态应使用哪个senone。分类回归树CART模型用以进行词到音素的发音标注。）

**音素phones**构成**亚单词单元**，也就是**音节syllables**。音节是一个比较稳定的实体，因为当语音变得比较快的时候，音素往往会发生改变，但是音节却不变。音节与节奏语调的轮廓有关。有几种方式去产生音节：基于**形态学**或者基于**语音学**。音节经常在**词汇语音识别**中使用。

**亚单词单元（音节）构成单词**。单词在语音识别中很重要，因为单词约束了音素的组合。假如共有40个音素，然后每个单词平均有7个音素，那么就会存在40^7个单词，但幸运的是就算一个受过优等教育的人也很少使用过20k个单词，这就使识别变得可行。

**单词**和一些**非语言学声音**构成了**话语utterances**，我们把非语言学声音称为**fillers填充物**，例如呼吸，um，uh，咳嗽等，它们在音频中是以停顿做分离的。所以它们更多只是语义上面的概念，不算是一个句子。

## 语音识别过程

语音识别一般的方法是：**录制语音波形，再把波形通过静音silences分割为多个utterances，然后去识别每个utterance所表达的意思**。为了达到这个目的，我们需要**用单词的所有可能组合去匹配这段音频，然后选择匹配度最高的组合**。

# **匹配关键概念**

## 特征：

由于描述一个语音，需要的参数个数非常多，这样对处理速度的要求就很高（而且也没必要处理那么多的信息，我们只需要处理对识别有帮助的就行），所以我们需要做优化，进行降维。我们用**帧frames去分割语音波形**，每帧大概10ms，然后**每帧提取可以代表该帧语音的39个数字**，这39个数字也就是该帧语音的特征，用特征向量来表示。而如何提取特征向量是当下热门的研究课题，但这些提取方法都是**由频谱衍生出来的**。

### Volume（音量）

### ZCR（过零率）

### Pitch（音高）

基本周期（Fundamental Period），有可能以采样点的个数来统计

基本頻率（Fundamental Frequency）

从而Fundamental Frequency代表了一秒内有多少基本周期，从而代表了频率

单纯使用「觀察法」來得到基本周期包含的抽样点个数，并算出音高，並不是太難的事，但是若要電腦自動计算，就需要更深入的研究。

* 旋律辨識（Melody Recognition）：或稱為「哼唱選歌」，也就是如何由使用者的哼唱，找出音樂資料庫中間對應的歌。
* 國語的聲調辨識（Tone Recognition）：辨識使用者講一句話時，每一個字的聲調（一聲、二聲、三聲、四聲等）。
* 語音合成的韻律分析（Prosody Analysis）中的音高分析：如何在合成語音時，使用最自然的音高曲線。
* 語音評分中的音調評分（Intonation Assessment）：如何評估使用者說話的語音，其音高曲線是否標準。
* 語音辨識（Speech Recognition）：我們可以使用語句的音高來提高語音辨識的正確率。

音高追蹤的基本流程如下：

1. 將整段音訊訊號切成音框（Frames），相鄰音框之間可以重疊。
2. 算出每個音框所對應的音高。
3. 排除不穩定的音高值。（可由音量來篩選，或由音高值的範圍來過濾。）
4. 對整段音高進行平滑化，通常是使用「中位數濾波器」（Median Filters）。

我們讓音框重疊的目地，只是希望相鄰音框之間的變化不會太大，使抓出來的音高曲線更具有連續性。但是在實際應用時，音框的重疊也不能太大，否則會造成計算量的過大。在選擇音框的大小時，有下列考量因素：

音框長度至少必須包含 2 個基本週期以上，才能顯示語音的特性。已知人聲的音高範圍大約在 50 Hz 至 1000 Hz 之間，因此對於一個的取樣頻率，我們就可以計算出音框長度的最小值。例如，若取樣頻率 fs = 8000 Hz，那麼當音高 f = 50 Hz（例如男低音的歌聲）時，每個基本週期的點數是 fs/f = 8000/50 = 160，因此音框必須至少是 320 點；若音高是 1000 Hz（例如女高音的歌聲）時，每個基本週期的點數是 8000/1000 = 8，因此音框必須至少是 16 點。

音框長度也不能太大，太長的音框無法抓到音訊的特性隨時間而變化的細微現象，同時計算量也會變大。

音框之間的重疊完全是看電腦的運算能力來決定，若重疊多，音框率就會變大，計算量就跟著變大。若重疊少（甚至可以不重疊或跳點），音框率就會變小，計算量也跟著變小。

由一個音框計算出音高的方法很多，可以分為時域和頻域兩大類：

時域（Time Domain）

ACF: Autocorrelation function

AMDF: Average magnitude difference function

SIFT: Simple inverse filter tracking

頻域（Frequency Domain）

Harmonic product spectrum method

Cepstrum method

## 模型：

模型是用来描述一些数学对象的。这些数学对象描述了一些口语的共同属性。在实际应用中，senone的音频模型就是三态高斯混合模型。简单的说，它就是一个最有可能的特征向量。对于模型，有几个问题需要考虑：模型到底多大程度上可以描述实际情况？在模型本身的局限情况下模型能表现得更优吗？自适应模型如何改变条件？

## 匹配算法：

语音识别需要对所有的特征向量和所有的模型做比较匹配，这是一个非常耗时的工作。而在这方面的优化往往是使用一些技巧，在每一点的匹配时，我们通过保留最好的匹配variants，然后通过它在下一帧产生最好的匹配variants。？

## 声学模型acoustic model：

一个声学模型包含每个senone的声学属性，其包括不依赖于上下文的属性（每个音素phone最大可能的特征向量）和依赖于上下文的属性（根据上下文构建的senone）。

## **语音学字典phonetic dictionary**：

字典包含了从单词words到音素phones之间的映射。

字典并不是描述单词words到音素phones之间的映射的唯一方法。可以通过运用机器学习算法去学习得到一些复杂的函数去完成映射功能。

## 语言模型 language model：

语言模型是用来约束单词搜索的。它定义了哪些词能跟在上一个已经识别的词的后面（匹配是一个顺序的处理过程），这样就可以为匹配过程排除一些不可能的单词。大部分的语言模型都是使用n-gram模型，它包含了单词序列的统计。和有限状态模型，它通过有限状态机来定义语音序列。有时候会加入权值。为了达到比较好的识别准确率，语言模型必须能够很好的约束空间搜索，也就是说可以更好的预测下一个词。语言模型是约束词汇包含的单词的，这就出现一个问题，就是名字识别（因为名字可以随便由几个单词组成）。为了处理这种情况，语言模型可以包含更小的块，例如亚单词，甚至音素。但是这种情况，识别准确率将会低于基于单词的语言模型。

特征、模型和搜索算法三部分构成了一个语音识别系统。如果你需要识别不同的语言，那么就需要修改这三个部分。很多语言，都已经存在声学模型，字典，甚至大词汇量语言模型可供下载了。

# 其他用到的概念

## 网格Lattice

是一个代表识别的不同结果的有向图。一般来说，很难去获得一个最好的语音匹配结果。所以Lattices就是一个比较好的格式去存放语音识别的中间结果。

N-best lists和lattices有点像，但是它没有lattices那么密集（也就是保留的结果没有lattices多）。（N-best搜索和多遍搜索：为在搜索中利用各种知识源，通常要进行多遍搜索，第一遍使用代价低的知识源（如声学模型、语言模型和音标词典），产生一个候选列表或词候选网格，在此基础上进行使用代价高的知识源（如4阶或5阶的N-Gram、4阶或更高的上下文相关模型）的第二遍搜索得到最佳路径。）

## 单词混淆网络

是从lattice的边缘得到的一个严格的节点顺序序列。

语音数据库-一个从任务数据库得到的典型的录音集。如果我们开发的是一个对话的系统，那么数据库就是包含了多个用户的对话录音。而对于听写系统，包含的就是朗读的录音。语音数据库是来用训练，调整和测试解码系统的（也就是语音识别系统）。

## 文本数据库

为了训练语言模型而收集的文本。一般是以样本文本的方式来收集形成的。而收集过程存在一个问题就是误把PDFs,webpages,scans等现成文档也当成口语文本的形式放进数据库中。所以，我们就需要把这些文件带进数据库里面的标签和文件头去掉，还有把数字展开为它们的语音形式（例如1展开为英文的one或者汉语的yi），另外还需要把缩写给扩大还原为完整单词。

# 语音的优化

随着语音识别技术的发展，最复杂的难题是如何使搜索（也就是语音解码，可以认为是需要匹配尽可能多的语音变体）更加准确和快速。还有在模型并不完美的前提下如何匹配语音和模型。

一般来说系统需要通过一个测试数据库来验证准确性，也就是是否达到了我们的预定目标。

## 表征系统的性能参数：

单词错误率：我们有一个N个单词长度的原始文本和识别出来的文本。（对单词串进行识别难免有词的插入，替换和删除的误识）I代表被插入的单词个数，D代表被删除的单词个数，S代表被替换的单词个数，那么单词错误率就定义为：WER=(I+D+S)/N

单词错误率一般通过百分百来表示。

准确度。它和单词错误率大部分是相似的，但是它不计算插入单词的个数，它定义为：Accuracy=(N–D–S)/N

对于大部分任务来说，准确度事实上是一个比较差的度量方法，因为插入的情况对于识别结果的影响也是很重要的。但对于一些任务而言，准确度也是一个合理的评价解码器性能的参数。

速度：假设音频文件是2个小时，而解码花费了6个小时，那么计算出来的速度就是3xRT。（3倍速）

ROC曲线：对于一个检测任务，检测会出现误报和命中两种情况。ROC曲线就是用来评价检测性能的。ROC曲线就是描述误报和命中的数目比例的。而且可以通过ROC曲线取寻找一个最优点，在这个点误报最小，而命中最大，也就是接近100%的命中率。

还有其他的方法来衡量识别性能，虽然这里没有提及，但对于很多的实际应用来说还是比较重要的。你的第一个工作应该是建立这样一个评价体系，然后系统地应用到开发过程中。第二个工作就是收集一个测试数据库来测试你的系统性能。

# 傅里叶变换

假设采样频率为Fs，信号频率F，采样点数为N。那么FFT之后结果就是一个为N点的复数。假设FFT之后某点n用复数a+bi表示，那么这个复数的模就是An=根号a\*a+b\*b，相位就是Pn=atan2(b,a)。

**每一个点**就对应着**一个频率点**。这个点的**模值**，就是该频率值下的**幅度**特性。

## **幅度关系**

假设原始信号的峰值为A，那么FFT的结果的每个点（除了第一个点直流分量之外）的模值就是A的N/2倍。而第一个点就是直流分量，它的模值就是直流分量的N倍。

## 相位关系

而每个点的相位就是在该频率下的信号的相位。

## 频率关系

某点n所表示的频率为：

第一个点n=1 时，表示直流分量（即0Hz）

n=N+1时，即最后一个点N的再下一个点（实际上这个点是不存在的，这里是假设的第N+1个点，也可以看做是将第一个点分做两半分，另一半移到最后）则表示采样频率Fs

中间被N-1个点平均分成N等份，每个点的频率依次增加。

## 频率的分辨率

由上面的公式可以看出，对Fn进行N个抽样点所能分辨到频率为为**Fs/N**

如果采样频率Fs为1024Hz，采样点数为1024点，则可以分辨到1Hz。1024Hz的采样率采样1024点，刚好是1秒，也就是说，采样1秒时间的信号并做FFT，则结果可以分析到1Hz，如果采样2秒时间的信号并做FFT，则结果可以分析到0.5Hz。

如果要提高频率分辨力，则必须增加采样点数，也即采样时间。频率分辨率和采样时间是倒数关系。

// todo

/\*

\* 使用这个类可以很强轻松地将音频数据在Android系统上播放出来，下面贴出我自己写的源码： AudioTrack audio = new

\* AudioTrack( AudioManager.STREAM\_MUSIC, // 指定在流的类型 32000, // 设置音频数据的采样率

\* 32k，如果是44.1k就是44100 AudioFormat.CHANNEL\_OUT\_STEREO, //

\* 设置输出声道为双声道立体声，而CHANNEL\_OUT\_MONO类型是单声道 AudioFormat.ENCODING\_PCM\_16BIT, //

\* 设置音频数据块是8位还是16位，这里设置为16位。好像现在绝大多数的音频都是16位的了 AudioTrack.MODE\_STREAM //

\* 设置模式类型，在这里设置为流类型，另外一种MODE\_STATIC貌似没有什么效果 ); audio.play(); //

\* 启动音频设备，下面就可以真正开始音频数据的播放了 // 打开mp3文件，读取数据，解码等操作省略 ... byte[] buffer = new

\* buffer[4096]; int count; while(true) { //

\* 最关键的是将解码后的数据，从缓冲区写入到AudioTrack对象中 audio.write(buffer, 0, 4096); if(文件结束)

\* break; } // 最后别忘了关闭并释放资源 audio.stop(); audio.release();

\*/