链滴

FFmpeg 官方文档翻译

作者：HaujetZhao

原文链接：https://ld246.com/article/1595480295489

来源网站：链滴

<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg [global_options] {[input_file_options] -i input_url} ... {[output_file_options] output
url} ...

FFmpeg [全局选项]
{[输入文件选项] -i 输入文件路径} ... {[输出文件选项] 输出文件地址} ...
</code></pre>

<p>其中，-i 表示 input，[] 括起
表示可选项，{} 括起来表示一个或多个必输入项，括号外的全部必输入。</p>
<h2 id="描述">描述</h2>
<p>FFmpeg 是一个非常快的音频、视频转换器，他也可以从直播音频、视频
进行抓取。他可以飞快地在各种不同的采样率之间进行转换、调整视频大小、使用高质量多相滤镜。<
p>
<p>FFmpeg 读取一个或多个输入文件（可以是普通文件、管道文件、网络视
流、抓取设备等等）（这些输入文件被 -i 选项指定），然后写到一个或多个输出文件（被一段文本路
指定）。在一行命令中任何不能被识别为选项的内容都会被当成输出路径对待。</p>
<p>每个输入或输出路径可以，原则上，包含许多个不同种类的“流”（视频流、音频流、字幕流、
件流、数据流）。允许包含多少个流，受文件封装格式限制。可以自动或用 -map 选项（参阅“流选
”章节）手动选择哪个文件的哪个流写到哪个输出文件。</p>
<p>在选项中涉及输入文件时，你必须使用它们的索引号（从 0 开始的序号）。例如，第一个输入文
是 0，第二个文件是 1，以此类推。同样地，涉及一个文件内的流时也是使用这样的索引。例如，2:3
注意这里是英文冒号）表示第三个输入文件中的第四个流。（参阅“流选择”章节）。</p>
<p>作为总的原则，选项 总是只作用在它 之后 的那个文
上。因此，顺序很重要，你可以在一行命令中使用同一个选项多次。每次使用这个选项，都只作用在
面那个文件上。有一种例外，就是全局选项（例如啰嗦提示级别）（应当被首先指示出）。</p>
<p>不要混淆输入和输出文件，输入文件是先标出，输出文件是后标出。也不要混淆本该属于不同类
文件的选项。所有选项只对它后面那个文件起作用一次，然后就会被恢复默认值。</p>
<pre><code class="highlight-chroma"><span class="highlight
cl">*设置输出视频的码率为64Kbps：
FFmpeg -i input.av
 -b:v 64k -bufsize 64k output.avi

*强制输出视频帧
为24fps：
FFmpeg -i input.av
 -r 24 output.avi

*强制输入文件帧
为1fps，同时输出视频为24fps：
FFmpeg -r 1 -i inp
t.m2v -r 24 output.avi
</code></pre>
<p>处理的输入文件为原始数据（raw input file）时，可能需要一些格式选项。</p>
<h2 id="详细描述">详细描述</h2>
<p>FFmpeg 对输出文件的转码过程可用下图简述：</p>
<pre><code class="highlight-chroma"><span class="highlight
cl">_______   ______________
| | |
    |
| input | demuxer
 | encoded data | decoder
<span class="highlight-line"><span class="highlight-cl">| file | ---------&gt

```
| packets      | -----+
</span></span><span class="highlight-line"><span class="highlight-cl">|_______|          |__
_________|      |
</span></span><span class="highlight-line"><span class="highlight-cl">
         v
</span></span><span class="highlight-line"><span class="highlight-cl">
      ________
</span></span><span class="highlight-line"><span class="highlight-cl">
     |        |
</span></span><span class="highlight-line"><span class="highlight-cl">
     | decoded |
</span></span><span class="highlight-line"><span class="highlight-cl">
     | frames  |
</span></span><span class="highlight-line"><span class="highlight-cl">
     |________|
</span></span><span class="highlight-line"><span class="highlight-cl"> ________           __
__________       |
</span></span><span class="highlight-line"><span class="highlight-cl">|        |          |
    |      |
</span></span><span class="highlight-line"><span class="highlight-cl">| output | &lt;-----
-- | encoded data | &lt;----+
</span></span><span class="highlight-line"><span class="highlight-cl">| file   |   muxer   |
ackets      |   encoder
</span></span><span class="highlight-line"><span class="highlight-cl">|________|          |__
__________|
</span></span><span class="highlight-line"><span class="highlight-cl">```
</span></span><span class="highlight-line"><span class="highlight-cl">```
</span></span></code></pre>
<p>FFmpeg 先调用包含分流器（demuxer）的 libavformat 库读取输入文件，得到数据包（内含编码后的数据）。当存在多个输入文件时，FFmpeg 将通过跟踪最小的时间戳来尝试同步所有活跃的入流。</p>
<p>编码数据包然后被传递到解码器（除非复制流被选择用于流，详见后面进一步的说明），解码器生的未压缩的帧（原始视频/PCM 音频/...），后者可以进一步被滤镜处理（见下一节）。通过滤镜处后，这些帧被传递到编码器，编码器将其编码，并输出编码后的数据包。最后，这些数据包被传递给合器，以将编码数据写入到输出文件。</p>
<h3 id="滤镜">滤镜</h3>
<p>在使原始数据编码之前，FFmpeg 可以使用 libavfilter 库中的滤镜处理原始音频和视频帧。几个接的滤镜可以形成一个滤镜组（filtergraphs）。 FFmpeg 有两种 filtergraphs：简单和复合。</p>
<h4 id="简单filtergraphs">简单 filtergraphs</h4>
<p>简单 filtergraphs 是那些恰好只有一个输入和输出、并且输入输出都是相同类型的滤镜组。在下中，它们可以被表示为，简单地在解码和编码之间插入一个附加步骤：</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight cl">_________                ______________
</span></span><span class="highlight-line"><span class="highlight-cl">|         |
  |              |
</span></span><span class="highlight-line"><span class="highlight-cl">| decoded |
        | encoded data |
</span></span><span class="highlight-line"><span class="highlight-cl">| frames  |\
     _ | packets      |
</span></span><span class="highlight-line"><span class="highlight-cl">|_________| \
    /||______________|
</span></span><span class="highlight-line"><span class="highlight-cl">           \   ________
_   /
</span></span><span class="highlight-line"><span class="highlight-cl">  simple     _\||
```

```
| /  encoder
</span></span><span class="highlight-line"><span class="highlight-cl">  filtergraph   | filte
ed |/
</span></span><span class="highlight-line"><span class="highlight-cl">                | frames
 |
</span></span><span class="highlight-line"><span class="highlight-cl">                |_______
_|
</span></span><span class="highlight-line"><span class="highlight-cl">```
</span></span><span class="highlight-line"><span class="highlight-cl">```
</span></span></code></pre>
<p>简单 filtergraphs 由对每个流的滤镜选项（-vf 和 -af 分别表示针对视频和音频的滤镜）配置。
个针对视频的简单 FilterGraph 可以看下这个例子：</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">_______        ____________  _______  ________
</span></span><span class="highlight-line"><span class="highlight-cl">|       |     |            |
    |       |     |        |
</span></span><span class="highlight-line"><span class="highlight-cl">| input | ---&gt; |
einterlace | ---&gt; | scale | ---&gt; | output |
</span></span><span class="highlight-line"><span class="highlight-cl">|_______|     |_____
_____|     |_______|     |________|
</span></span></code></pre>
<p>需知晓的是一些滤镜只改变帧的属性而不触及帧的内容。例如在上例中，fps 滤镜只改变帧的数
，不改变帧的内容。又如 setpts 滤镜，仅设置时间戳而保持帧不变。</p>
<h4 id="复合filtergraphs">复合 filtergraphs</h4>
<p>复合 filtergraphs 是那些不能被描述为简单的线性处理链的滤镜组。例如，当滤镜组具有多个输
和（或）输出，或当输入和输出流是不同的类型。它们可以被表示为以下图：</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">_________
</span></span><span class="highlight-line"><span class="highlight-cl">|         |
</span></span><span class="highlight-line"><span class="highlight-cl">| input 0 |\
    _________
</span></span><span class="highlight-line"><span class="highlight-cl">|_________| \
    |         |
</span></span><span class="highlight-line"><span class="highlight-cl">           \   ________
   /| output 0 |
</span></span><span class="highlight-line"><span class="highlight-cl">            \ |         |
|_________|
</span></span><span class="highlight-line"><span class="highlight-cl"> _________     \| co
plex | /
</span></span><span class="highlight-line"><span class="highlight-cl">|         |    |         |/
</span></span><span class="highlight-line"><span class="highlight-cl">| input 1 |----&gt;|
ilter  |\
</span></span><span class="highlight-line"><span class="highlight-cl">|_________|    |
 \   _________
</span></span><span class="highlight-line"><span class="highlight-cl">             /| graph
|  \ |         |
</span></span><span class="highlight-line"><span class="highlight-cl">            / |         |
\| output 1 |
</span></span><span class="highlight-line"><span class="highlight-cl"> _________   /  |____
___|    |_________|
</span></span><span class="highlight-line"><span class="highlight-cl">|         | /
</span></span><span class="highlight-line"><span class="highlight-cl">| input 2 |/
</span></span><span class="highlight-line"><span class="highlight-cl">|_________|
</span></span></code></pre>
```

复合 filtergraphs 可使用 -filter_complex 选项指配。注意，此选项是全局性的，因为复合 Filter raph，就其本质，不能只与单个流、单个文件相关联。
-lavfi 选项相当于 -filter_complex 选项。
一个复合 FilterGraph 的简单的例子是 overlay 滤镜，它具有两个视频输入和一个视频输出，内一个视频叠加在另一个视频上面。其对应功能的音频滤镜是 amix 滤镜。
<h3 id="复制流">复制流</h3>
复制流是通过添加 copy 选项到 -codec 选项完成的。它使 FFmpeg 对指定的流忽略解码和编码骤，所以它只能混合和拆包（例如从一个 mkv 文件拆包提取出视频流，放到一个 mp4 输出文件内打，就仅仅是把视频流复制了，达到了无损转格式）。它对于改变所述容器的格式或修改容器级的元数是有用的。在这种情况下，可以简化为这样：
<pre><code class="highlight-chroma">_______ ______________ ________
input	demuxer	encoded data	muxer	output
file	--------->	packets	------->	file
_______		_____________		________
</code></pre>
由于不存在解码或编码，复制流非常快（几乎等于复制粘贴的速度），没有画质或音质损失。然因为许多因素，某些情况下可能无法使用。应用滤镜显然也是不可能的，因为滤镜仅能作用在未压缩数据上（而拆包后的数据流都还是被编码了的、压缩了的数据）。
<h2 id="流的选择">流的选择</h2>
FFmpeg 提供 -map 选项来手动控制每个输出文件中流的选择。用户可以跳过 -map 选项，让 Fmpeg 执行自动流选择（规则在下面提到）。不管是手动映射（map）还是自动选择，使用 -vn / -an / -sn / -dn 选项分别可以跳过输出文件对视频流、音频流、字幕流、数据流的选择（不过对那些复合 fltergraphs 输出的流不起作用）。
<h3 id="描述-">描述</h3>
接下来的小节叙述了涉及到流的选择的各种规则。后面会有例子帮你理解。
虽然，为了能够精确的反映程序的行为，文档作者们付出了很多努力来完善这个文档，但是 FFmeg 是在被持续开发和完善的，自从写下这个文档，有些代码就可能已经改变了。
<h4 id="自动流选择">自动流选择</h4>
对于某一个输出文件，在映射（map）选项缺失的情况下，FFmpeg 会检查输出文件格式，看些种类的流可以被打包进去，例如视频流、音频流、字幕流。对于每个可以打包进去的种类，FFmpeg 会从所有可用的输入文件中选择一个流。
FFmpeg 会基于以下原则选择流：

对于视频，选择最高分辨率的流
对于音频，选择有最多声道的流
对于字幕，选择第一个被找到的流，但是警告，输出文件的默认字幕编码器可能是基于文本的、可能是基于图像的，只有与输出文件字幕类型相同的字幕流才会被选择。

在相同种类的几个流的参数同样高时，选择索引号最低的那个流。
数据流或附件流不会被自动选择，他们只能被 -map 选项选择。
<h4 id="手动流选择">手动流选择</h4>
当 -map 选项被使用时，除了下面会提到的 filtergraph 输出流，只有用户映射的（user-mappe）流会被包含在那个输出文件。
<h4 id="复合filtergraph">复合 filtergraph</h4>
当有一些复合 filtergraph 输出流带着未标记的 pads 时，这些输出流会被加到第一个输出文件当这个流不被输出文件的格式支持时，就会导致严重错误。当没有 -map 选项时，这些复合 filtergrah 输出流的包含，会导致它们类型的流的自动选择被跳过。当有 -map 选项时，除了被手动映射的流这些复合 filtergraph 输出流也会被额外囊括进去。

<h4 id="流处理">流处理</h4>
流处理是独立于流选择之外的（除了下面会讲到的字幕是例外）。流处理是通过-codec 选项来定的，作用于指定输出文件内的流。尤其是，FFmpeg 是在流选择过程之后才实施流处理过程的，所流处理过程不会影响流选择。如果对某个流没有指定 -codec 选项，FFmpeg 会根据输出文件的混合（muxer）选择默认注册的编码器（encoder）。
对于字幕存在例外。如果一个输出文件的字幕编码器被指定，第一个被找到的字幕流（不管是文、还是图像类型）都会被打包进去。FFmpeg 不检查指定的编码器是否能转换选择的流，也不检查转后的流是否能被输出文件的格式兼容。这条规则是通用的：当用户手动设定一个编码器时，流选择过无法检查被编码过的流可以被混合到输出文件中。如果不能，FFmpeg 会放弃所有输出文件，然后处过程失败。
<h3 id="范例">范例</h3>
接下来的例子会阐释 FFmpeg 流选择方法的行为、难点和限制。
假设有下列三个输入文件：
<pre><code class="highlight-chroma">input file 'A.avi'
  stream 0: video 40x360
  stream 1: audio 2 channels

input file 'B.mp4'
  stream 0: video 920x1080
  stream 1: audio 2 channels
  stream 2: subtitl s (text)
  stream 3: audio 5 1 channels
  stream 4: subtitl s (text)

input file 'C.mkv'
  stream 0: video 280x720
  stream 1: audio 2 channels
  stream 2: subtitl s (image)
</code></pre>
<h4 id="自动流选择-">自动流选择</h4>
<pre><code class="highlight-chroma">FFmpeg -i A.avi -i B.mp4 out1.mkv out2.wav -map 1:a -c:a copy out3.mov
</code></pre>
这里指定了三个输出文件，对于前两个，没有设定 -map 选项，所以 FFmpeg 会为这两个文件动选择流。
out1.mkv 是一个 Mastroska 容器的文件，可以存放视频、音频、字幕流，所以 FFmpeg 会每种类选一个流。

对视频流，会选择 B.mp4 中的 stream 0，也就是有最高分辨率的那个；
对音频流，会选择 B.mp4 中的 stream 3，也就是有最大声道数的那个；
对字幕流，会选择 B.mp4 中的 stream 2，也就是 A.avi 和 B.mp4 中第一个字幕流。

out2.wav 只接受音频流，所以只有 B.mp4 中的 stream 3 会被选择。

对于 out3.mov，因为有了 -map 选项，就不会发生流的自动选择了。-map 1:a 选项会选择第个输入文件 B.mp4 中的所有音频流，不会有任何其它流被打包到这个输出文件。
<h4 id="自动字幕选择">自动字幕选择</h4>
<pre><code class="highlight-chroma">FFmpeg -i C.mkv out1.mkv -c:s dvdsub -an out2.mkv
</code></pre>
尽管 out1.mkv 是一个能够容纳字幕流的 Matroska 容器文件，但这个例子里只有视频和音频流被选择。C.mkv 里的字幕是基于图像的，而 Mastroska 的默认字幕编码器是基于文字的，所以转码失败，因而字幕流不会被选择。不过，out2.mkv 里，我们在命令里手动指定了一个字母编码器，所，除了视频流，字幕流也会被选择加进 out2.mkv。-an 选项禁止了 out2.mkv 的音频流选择。
<h4 id="未写标签的filtergraph-输出">未写标签的 filtergraph 输出</h4>
<pre><code class="highlight-chroma">FFmpeg -i A.avi -i C.mkv -i B.mp4 -filter_complex "overlay" out1.mp4 out2.srt
</code></pre>
这里我们通过-filter_complex 选项设置了一个 filtergraph ，后者包含一个视频滤镜。overlay 镜要求正好两个视频输入，因为没有指定哪两个，所以会默认使用前两个可用的视频，也就是 A.avi 和 C.mkv 。由于这样，本来会选择 B.mp4 里面的流的自动视频流选择过程也就被跳过了。在 B.mp4 的 tream 3 里的音频流由最多的声道，会被自动选择上。然而不会有字幕流会被选择上，因为 mp4 格没有默认注册的字幕编码器，而且用户也没有指定一个字母编码器。
对于第二个输出文件，out2.srt，只接受基于文字的字幕流。所以尽管第一个可用字幕流（那个于 C.mkv 的）存在，但由于它是基于图像的所以会被跳过。被选择的流，B.mp4 中的 stream 2 ，第一个基于文本的字幕流。
<h4 id="写了标签的filtergraph-输出">写了标签的 filtergraph 输出</h4>
（下述命令中，末尾的 / 在终端里表示换一行显示，继续写命令，执行的时候是把那几行命令当行执行的）
<pre><code class="highlight-chroma">FFmpeg -i A.avi -i B.mp4 -i C.mkv -filter_complex "[1:v]hue=s=0[outv];overlay;aresample"

  -map '[outv]' -an out1.mp4 \
out2.mkv \
  -map '[outv]' -ap 1:a:0 out3.mkv
</code></pre>
上述命令会失败，因为带有[outv] 标签的输出板被映射了两次。所有输出文件都不会被处理。
<pre><code class="highlight-chroma">FFmpeg -i A.avi -i B.mp4 -i C.mkv -filter_complex
  "[1:v]hue=s=0[outv];overlay;aresample" \
  -an out1.mp4 \
  out2.mkv \
  -map 1:a:0 out3 mkv
</code></pre>
上述这个命令也会失败，因为带有[outv] 标签的 hue 滤镜的输出，没有被映射到任何一个输出件。
这个命令应该像这样修改：
<pre><code class="highlight-chroma">FFmpeg -i A.avi -i B.mp4 -i C.mkv -filter_complex "[1:v]hue=s=0,split=2[outv1][outv2];overay;aresample" \
-map '[outv1]' -an out1.mp4 \

  out2.mkv \
-map '[outv2]' -m
p 1:a:0 out3.mkv
</code></pre>
B.mp4 中的视频流被送到 hue 滤镜，后者的输出结果被 split 滤镜克隆了一份，两份输出分别映射到第一个和第三个输出文件。
overlay 滤镜，要求两个视频输入，会使用前两个未被使用的视频流，也就是 A.avi 和 C.mkv 中视频流。overlay 滤镜的输出没有被写标签，所以它会被发到第一个输出文件 out1.mp4 （ -map 选对它不起作用）。
第一个未被使用的音频流，也就是 A.avi 的音频流，被送到 aresample 滤镜处理。因为这个滤的输出也是没有写标签的，所以也会被自动映射到第一个输出文件。-an 选项只抑制音频流的自动流择、手动流选择，不会抑制 filtergraphs 送来的音频流。在 out1.mp4 中，这两个滤镜送出的流会排映射的流的前面。
哪些音频、视频、字幕流被映射到 out2.mkv 完全取决于自动流选择。
out3.mkv 包含有 hue 滤镜输出的克隆流和来自 B.mp4 的第一个音频流。
<h2 id="选项">选项</h2>
所有的数字选项，如果没有明确指名，接受由字符串表示的数字作为输入，后可接一个 SI 单位缀，例如：K、M、G。
如果“i”被接到了 SI 单位后缀后，整个后缀会被当成二进制倍数后缀，也就是用 1024 为底的数，而不是 1000 为底的指数。SI 单位后缀后接上“B”，会把其值乘以 8。这就可以用，例如：KB MiB、G、B 作为数字后缀。（很好理解，bit 表示比特，Byte 表示字节，1 字节等于 8 比特）
没有后续参数的选项都是布尔选项，会将该选项设为启用。也可以通过加一个前缀“no”把这选项关闭。例如“-nofoo”会把名字是“foo”的布尔选项关闭。
<h3 id="流标识符-Stream-specifiers-">流标识符（Stream specifiers）</h3>
有些选项可能是应用到具体一个流上的，例如比特率、编码器。流标识符就是用来精确指明一个项是属于哪个流的。
流标识符通常被连接到选项名字的后面，用冒号隔开。例如“-codec:a:1 ac3”包含了“a:1”流识符，匹配第二个音频流（audio 首字母是 a）。因此，对第二个音频流会使用 ac3 作为编码器（coec）。
一个流标识符可以匹配多个流，此时这个选项会应用到所有匹配到的流上。例如“-b:a 128k”匹配所有的音频流。
没有标识符会匹配所有流，例如，-codec copy 或 -codec: copy 会让所有的流进行流复制（而进行重新编码）。
流标识符可用的格式有：

stream_index（流索引号）

匹配这个索引号下的流，例如 -threads:1 会设置第二个流的线程设为 4。如果流索引号是作为加标识符（见下面）的，那它就会选取匹配的所有流下的第索引号个流。流的序号是由 libavformat 测流的顺序决定的，除了当程序 ID 被指定时。这时，序号是基于程序中的顺序决定的。

stream_type[:additional_stream_specifier]（流类型[:额外流标识符]）</l
>

流类型是下列字母之一：v 或 V 表示视频（video），a 为声音（audio），s 为字幕（subtitle，d 为数据（data），t 为附件（attatchments）。“v” 匹配所有的视频流，“V”只会匹配没有加图片、缩略图、封面的视频。如果额外流标识符添加上了，那就只会匹配符合指定类型、并且匹配外流标识符的流。否则，它就只会匹配所有符合指定类型的流

p:program_id[:additional_stream_specifier]（p:程序 ID[:额外流标识符]）</strong

匹配拥有指定 id 的程序内的流。如果额外流标识符添加上了，那就只会匹配在拥有指定 id 的程内的、并且匹配额外流标识符的流。

#stream_id or i:stream_id（流 id 或 i:流 id）

用流 id 匹配流（例如 MPEG-TS 容器内的 PID）。

m:key[:value]（m:项名[:项值]）

匹配元数据（metadata）中拥有指定标签项值的流。如果没有指定项值（value），就只匹配所包含指定标签项的流。

u

匹配哪些拥有可用调整项的流，编码器必须被指定，并且关键信息（例如视频维度或音频采样率必须被呈现。
应当知道，在 FFmpeg 中，通过元数据匹配只对输入文件有效。
<h3 id="通用选项">通用选项</h3>
这些选项在 ff* 工具中通用

-L
显示许可证。

-h, -?, -help, --help [arg]
显示帮助。一个额外参数可以指定显示关于某个指定项的参数。如果没有参数，只会显示基本工具选。
可能的参数值有：
<pre><code class="highlight-chroma">long
在基础工具选项的础上，还显示高级工具选项

full
显示完整的选项列，包括公共选项和编码器、解码器、分流器、混合器、滤镜的私有选项等。

decoder=decoder_name
显示所指定解码器字的解码器详情信息。用 -decoders 选项列出所有的解码器。

encoder=encoder_name
显示所指定编码器字的解码器详情信息。用 -encoders 选项列出所有的编码器。

demuxer=demuxe_name
显示所指定分流器字的分流器详情信息。用 -formats 选项列出所有的分流器和混流器。

muxer=muxer_name
显示所指定混流器
</code></pre>

字的混流器详情信息。用 -formats 选项列出所有的分流器和混流器。

filter=filter_name
显示所指定滤镜名的滤镜详情信息。用 -filter 选项列出所有的滤镜。

bsf=bitstream_filter_name
显示所指定比特流镜名字的滤镜详情信息。用 -bsfs 选项列出所有的比特流滤镜。
</code></pre>

-version
显示版本。

-formats
显示可用格式（包括设备）。

-demuxers
显示可用分流器。

-muxers
显示可用混流器。

-devices
显示可用设备。

-codecs
显示 libavcodec 库里所有已知编码器。
需要提醒下，“codec”（编码器）这个术语，会在这个文档中多次出现，作为严格意义上“媒比特流格式”的别称来使用。

-decoders
显示可用解码器。

-encoders
显示可用编码器。

-bsfs
显示可用比特流滤镜。

-protocols
显示可用协议。

-filters
显示可用的 libavfilter 滤镜。

-pix_fmts
显示可用像素格式。

-sample_fmts
显示可用采样格式。

-layouts
显示各种声道名和标准声道布局。

-colors
显示可以被叫上名字的颜色名称。

-sources device[,opt1=val1[,opt2=val2]...]
显示自动检测到的输入设备源。一些设备可能提供依赖系统的源且无法被自动检测到。不要期待返回列表十分完整。
FFmpeg -sources pulse,server=192.168.0.4

-sinks device[,opt1=val1[,opt2=val2]...]
显示自动检测到的输出设备池。一些设备可能提供依赖系统的池且无法被自动检测到。不要期待返回列表十分完整。
FFmpeg -sinks pulse,server=192.168.0.4

-loglevel [flags+]loglevel | -v [flags+]loglevel
设置日志记录级别和被数据库使用的标记。
可选的标记前缀可包含下列值：
<pre><code class="highlight-chroma"> ‘repeat’
表示重复的日志输不应该被压缩到第一行，并且“最后一个信息重复第n次”这样的行会被省略掉。

  ‘level’
表示日志输出应该每一行消息加上 [level] 前缀。这可以例如要将日志写到一个文件时，被用来做彩色日志的替代方法（日志显示时，不同级别日志在终端里可以有不同颜色的显示，方便区分和查找，但写到文件里，都是一个颜色了）

可以通过添加“+或“-”前缀来单独设置或取消设置一个标记，让这个标记被单独使用。当同时设置日志等级和标记，在最后一个标记和日志等级之间，应当添加一个“+”分隔符。

日志等级是一串包下面数值的字符串或数字：

  ‘quiet, -8’
不显示任何信息，静。

  ‘panic, 0’
只显示会导致处理程崩溃的严重错误。现在这个还没有任何用。

  ‘fatal, 8’
只显示严重错误，些是经过让程序无法继续的错误之后的。

  ‘error, 16’
显示所有错误，包可修复的错误。

  ‘warning, 24’
显示所有警告和错，任何可能错误的、或没有预料到的事件都会被显示。

  ‘info, 32’
显示处理过程中的息消息。只是警告和错误之外的补充。这是默认值。

  ‘verbose, 40’
和info一样，只是了些啰嗦话。

  ‘debug, 48’
显示所有信息，包调试信息。

  ‘trace, 56’
</code></pre>
例如要启用“repeated log”输出，加上 level 前缀，然后设置日志等级为 verbose：
<pre><code class="highlight-chroma">FFmpeg -loglevel repeat+level+verbose -i input output
</code></pre>
另一个例子是启用“repeated log”，但是不影响当前 level 前缀标记或 loglevel 的状态：
<pre><code class="highlight-chroma">FFmpeg [...] -loglevel +repeat
</code></pre>
默认情况下，程序会输出日志到 stderr，如果终端程序支持着色功能，会有颜色标明错误和提醒设置环境变量 AV_LOG_FORCE_NOCOLOR 可以关掉日志着色，设置环境变量 AV_LOG_FORCE_COOR 可以强制开启日志着色。

-report
把所有命令行和日志输出写到当前目录一个名字为 program-YYYYMMDD-HHMMSS.log 的文件中这个文件对于调试很有帮助，后者意味着是 -loglevel 的调试。
把任何值设置到环境变量 FFREPORT 也会有相同效果。如果这个值是一个“:”分隔的“key=vale”序列，这些选项会影响 report；如果选项值包含特殊字符或“:”，应当加上转义符。（详见 FFmeg-utils 手册的“引用和转义符”部分）
下面的选项是被认可使用的：

<pre><code class="highlight-chroma">file
给 report 设置文名；%p 最后会被扩充为程序（program）的名字，%t 最后会被扩充为时间戳（timestamp），%%会被扩充为纯百分号（%）。

level
通过数字值设置日啰嗦级别（见 -loglevel）。

例如，要用 32（info级别的数字值） 日志级别输出 report 到一个叫 ffreport.log 的文件：
FFREPORT=file=ffreport.log:level=32 FFmpeg -i input output

分隔环境变量的错不是严重的，所以不会出现在 report 中。
</code></pre>

-hide_banner
阻止显示 banner
所有 FFmpeg 工具一般都会显示版权提醒、版本选项、数据库版本。这个选项可以组织显示这信息。

-cpuflags flags (global)
允许设置和清除 cpu 标识，这个选项是用于测试的，不懂相关知识可千万别瞎用！
<pre><code class="highlight-chroma">FFmpeg -cpuflags -sse+mmx ...
FFmpeg -cpuflags mmx ...
FFmpeg -cpuflags 0 ...
</code></pre>
这个选项可用的标识有：
<pre><code class="highlight-chroma"> ‘x86’
  ‘mmx’
  ‘mmxext’
  ‘sse’
  ‘sse2’
  ‘sse2slow’
  ‘sse3’
  ‘sse3slow’
  ‘ssse3’
  ‘atom’
  ‘sse4.1’
  ‘sse4.2’
  ‘avx’
  ‘avx2’
  ‘xop’
  ‘fma3’
  ‘fma4’

  '3dnow'
  '3dnowext'
  'bmi1'
  'bmi2'
  'cmov'
  'ARM'
  'armv5te'
  'armv6'
  'armv6t2'
  'vfp'
  'vfpv3'
  'neon'
  'setend'
  'AArch64'
  'armv8'
  'vfp'
  'neon'
  'PowerPC'
  'altivec'
  'Specific Process
rs'
  'pentium2'
  'pentium3'
  'pentium4'
  'k6'
  'k62'
  'athlon'
  'athlonxp'
  'k8'
</code></pre>

<h3 id="AVOptions-音视频选项-">AVOptions（音视频选项）</h3>
这些选项是由 libavformat、libavdevice 和 libavcodec 库直接提供的，若要查看可用的 AVOpt
ons 的列表，使用 -help 选项。它们被分成两类：
<h4 id="Generic-通用-">Generic（通用）</h4>
这些选项可以用于各种容器、编码器、设备。对于容器/设备的通用选项被列在 AVFormatConte
t 选项下，对于编码器的通用选项被列在 AVCodecContext 选项下。
<h4 id="Private-私有-">Private（私有）</h4>
这些选项只能用于特定容器、编码器、设备。私有选项被列在对应的容器、编码器、设备下。</
>
例如要写一个 ID3v2.3 头到一个 mp3 文件，而不是写 ID3v2.4 头，要使用 MP3 混流器 id3v2_
ersion 的私有选项：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i input.flac -id3v2_version 3 out.mp3
</code></pre>
所有的编码器音视频选项都是针对“流”的，所以应当在选项后面加上流标识符：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i multichannel.mxf -map 0:v:0 -map 0:a:0 -map 0:a:0 -c:a:0 ac3 -b:a:0 640k -ac:a:
 2 -c:a:1 aac -b:2 128k out.mp4
</code></pre>
在上述的示例命令中，一个多声道的音频流被两次映射到输出（也就是输出文件被映射了两个音
流，两个音频流的内容一样）。对第一个音频流使用了 ac3 编码器、码率值 640k。对第二个音频流
用了混合成双声道、码率值 128k（这里我们的流标识符使用了数字索引值，2 即代表第三个流）。</

>
<blockquote>
提醒：之前提到的 -nooption 语法不能被用于布尔 AVOptions ，应使用 -option 0 / -option 1 才行。
提醒：有一个没有提到的、老旧的、用于给 AVOptions 指定流的方法是在这些选项前加上 v/a/s ，但是这个方法已经过时，并且即将在后续版本中被删去了，所以别再用了！
</blockquote>
<h3 id="主要选项">主要选项</h3>

-f fmt (input/output)
强制输入文件或输出文件为某个格式。一般输入文件的格式会被自动检测，输出文件的格式会被通过后缀名推测。所以这个选项通常不用。

-i url (input)
输入文件或路径（例如可以是一个网址）

-y (global)
不经过询问，直接覆盖输出文件。

-n (global)
不要覆盖输出文件，如果与某个输出文件同路径同文件名的文件已经存在，直接退出。

-stream_loop number (input)
设定输入的流应循环几次。0 意味着不循环，-1 表示一直循环。

-c[:stream_specifier] codec (input/output,per-stream)
-codec[:stream_specifier] codec (input/output,per-stream)
当用在输出文件之前，选择编码器。当用在输入文件之前，选择解码器。codec 的值是编码器或解码的名字，或者是一个特殊值“copy”（只对输出文件有效），意味着输出流不被经过解码、重编码。
/p>
例如：
<pre><code class="highlight-chroma">FFmpeg -i INPUT -map 0 -c:v libx264 -c:a copy OUTPUT
</code></pre>
会用 libx264 编码器编码所有视频流，但只复制音频流。
对每一个流，只有最后一个匹配的 -c 选项会被应用，例如：
<code>FFmpeg -i INPUT -map 0 -c copy -c:v:1 libx264 -c:a:1 libvorbis OUTPUT</code>
除了第二个视频流会被 libx264 编码、第 138 个音频流会被 libvorbis 编码外，会复制所有流。<
p>

-t duration (input/output)
当作为输入文件选项（用在 -i 之前）时，限制从输入文件读取的数据的时间。
当作为输出文件选项（用在输出路径之前）时，写入指定持续时间的流后，停止写入。
duration 必须是一段时间的长度，详见 FFmpeg-utils(1) 的手册中“Time duration”一节。</
>
-to 和 -t 是相互排它的，-t 有更高优先级。

-to position (input/output)
在某个位置停止读取或写入。position 必须是一个指定的持续时间。详见 FFmpeg-utils(1) 的手册中 Time duration”一节。
-to 和 -t 是相互排它的，-t 有更高优先级。

-fs limit_size (output)
设定文件大小限制，单位是比特。超过指定大小后，不会有数据再被写入。当然，最后总的文件大小能略微超过指定的大小。

-ss position (input/output)
当作为输入选项使用时，找到输入文件中的 position 开始读。应知晓的是，对于大部分文件，很难常精确地找到那个位置，所以 FFmpeg 会寻找最接近指定点的位置。当转码和 -accurate_seek 开启默认都是开启的）时，这个在实际点和预期点之间的小片段就会被扔掉。但当使用流复制或 -noaccurate_seek 时，这一小段就会被保留。
当作为输出选项使用时，会解码所有输入，但达到指定时间戳之前的内容都会被丢弃掉。
position 必须是一段时间的长度，详见 FFmpeg-utils(1) 的手册中“Time duration”一节。

-sseof position (input)
就像是 -ss 选项一样，不过这时的参照点是文件末尾（end of file）。负数表示的位置更早，0 表示文件末尾。

-itsoffset offset (input)
设定输入文件的时间偏移（the input timestamp offset）。
offset 必须是一段时间的长度，详见 FFmpeg-utils(1) 的手册中“Time duration”一节。
偏移被添加到输入文件的时间戳。指定一个正的偏移，表示对应的流会被延迟指定的偏移时长。
可以简记为：input timestamp offset

-itsscale scale (input,per-stream)
重缩放时间戳。scale 应当是一个浮点数。

-timestamp date (output)
设定容器内的录制时间。
date 必须是一个指定日期，详见 FFmpeg-utils(1) 的手册中“Date”一节。

-metadata[:metadata_specifier] key=value (output,per-metadata)
设定元数据的 项/项值 对。
一个可选的元数据标识符可以指定元数据被写在哪条流、章、程序上。详见 -map_metadata 部。
用上 -map_metadata 时，这个选项会覆盖元数据。也可以通过设定一个空数值来删除元数据。/p>
例如，要给输出文件设置一个作品标题（title）：
<pre><code class="highlight-chroma">FFmpeg -i in.avi -metadata title="my title" out.flv
</code></pre>

要设定第一个字幕流的语言：
<pre><code class="highlight-chroma">FFmpeg -i INPUT -metadata:s:a:0 language=eng OUTPUT
</code></pre>

-disposition[:stream_specifier] value (output,per-stream)
设定一个流的位移。
这个选项会覆盖从输入流复制的位移。也可以通过设置它为 0 来删除位移。
下面列出的位移可以被识别：
<pre><code class="highlight-chroma">default
dub
original
comment
lyrics
karaoke
forced
hearing_impaired
visual_impaired
clean_effects
attached_pic
captions
descriptions
dependent
metadata
</code></pre>
例如，要设第二个音频流为默认音频流：
<pre><code class="highlight-chroma">FFmpeg -i in.mkv -c copy -disposition:a:1 default out.mkv
</code></pre>
要设置第二个字幕流为默认字幕流，并且移除第一个字幕流的默认位移：
<pre><code class="highlight-chroma">FFmpeg -i in.mkv -c copy -disposition:s:0 0 -disposition:s:1 default out.mkv
</code></pre>
要添加一个内置封面/缩略图：
<pre><code class="highlight-chroma">FFmpeg -i in.mp4 -i IMAGE -map 0 -map 1 -c copy -c:v:1 jpg -disposition:v:1 attached_pic out.mp4
</code></pre>
并非所有混合器支持内置缩略图，而那些支持缩略图的混合器，往往只支持有限的格式，例如 JPG 或 jpg。

-program [title=title:][program_num=program_num:]st=stream[:st=stream...] (output)
创建一个程序，这个程序有指定的标题、程序号，并添加指定的流给这个程序。

-target type (output)
指定目标文件类型 (vcd, svcd, dvd, dv, dv50) 。用 pal-、 ntsc- 或 film- 先指定类型后，就会使用类型相对应的标准。所有的格式选项（码率、编码器、缓存大小）就都会被自动设置好。你只需要输：
<pre><code class="highlight-chroma"><span class="highlight

cl">FFmpeg -i myfile.avi -target vcd /tmp/vcd.mpg
</code></pre>
不过你仍可以添加各种你所知道的附加选项（只要这些选项与设定的类型不冲突就行），就像：<
p>
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i myfile.avi -target vcd -bf 2 /tmp/vcd.mpg
</code></pre>

-dn (input/output)
作为输入选项用时，会阻止所有数据流被施加滤镜、自动映射到任何输出。想只禁用单独一条流的话请参阅 -discard 选项。
作为输出选项用时，会阻止数据录制，如自动选择或自动映射任何数据流。若要完整的手动控制请参阅 -map 选项。

-dframes number (output)
设定输出数据的帧数。这是 -frames:d 的一个老旧别名，以后最好老老实实用 -frames:d 。

-frames[:stream_specifier] framecount (output,per-stream)
达到指定的帧总数后，就停止写输出。

-q[:stream_specifier] q (output,per-stream)
-qscale[:stream_specifier] q (output,per-stream)
使用固定品质缩放（VBR）。q/qscale 的意思是随编码器的不同而改变的。如果 qscale 选项后有跟流标识符，那它只会被用到视频流上，这是为了保持与之前行为的兼容性，并且，当没有使用流识符，同样一个 qscale 数值对于不同的编码器（例如视频编码器和音频编码器）意义是不同的。</p

-filter[:stream_specifier] filtergraph (output,per-stream)
创建一个 filtergraph 并且用这个 filtergraph 给指定的流做滤镜。
filtergraph 是一个用在流上的 filtergraph 的描述，并且必须有相同类型的严格一个输入和一个出流。在 filtergraph 中，输入被与“标签入”关联，输出被与“标签出”关联。更多信关于 filtergrah 的语法信息详见 FFmpeg-filters 手册。
如果你想对多个输入和/或输出创建 filtergraphs 请参阅 -filter_complex 选项部分。

-filter_script[:stream_specifier] filename (output,per-stream)
这个选项和 -filter 很相像，唯一的区别是，这里的参数是一个文件名，所指的文件内包含的 filtergrah 描述会被读取。

-filter_threads nb_threads (global)
规定用多少线程处理滤镜工作流。每个工作流会产生一个为并行处理的、内含这么多个可用线程数的程池。默认值是可用的 CPUs 数量。

-pre[:stream_specifier] preset_name (output,per-stream)
为匹配的流指定预设。

-stats (global)
显示编码的进程/统计数据。默认是开启的，要关掉的话需要使用 -nostats 选项。

-progress url (global)
发送程序可读的进程信息到一个 url（路径、网址）。
进程信息约每秒写一次，编码完成后单独写一次。由每一行一个 "key=value" 的格式组成。key 由字母、数字组成，一系列 key 的最后一个 key 总是“progress”。

-stdin
启用与标准输入的反应。默认开启，除非标准输入被用作输入。要声明关掉，请使用 -nostdin 。</p

关掉与标准输入的响应是很有用的，例如，如果 FFmpeg 是一个后台进程组，使用 FFmpeg 大上可以达到与 < /dev/null 相同的效果，只是需要一个 shell。

-debug_ts (global)
显示时间戳信息。默认是关的。这个选项对于调试和找 bug 很有用，输出格式可能因版本不用而不，所以不要根据这个信息写脚本哦~
还可以参阅 -fdebug ts 选项。

-attach filename (output)
添加附件到输出文件中。少数容器支持这个功能，例如 Matroska 用字体文件来渲染字幕。附件会被入一个特殊类型的流，所以这个选项会添加一个流，然后就可以用通常的对流的选项处理这个流。这选项创建的附件流会在其它流都写入完成后再写入。
提醒下，对于 Matroska 容器，你还需要设置下元数据的 mimetype 项：
FFmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv
（上例假设附件流会是输出文件里的第三个流）

-dump_attachment[:stream_specifier] filename (input,per-stream)
提取匹配的附件流到一个文件。如果 filename 是空的，那写出的文件名会是该元数据项中的值。</p

例如提取一个附件流到一个叫“out.ttf”的文件：
<pre><code class="highlight-chroma">FFmpeg -dump_attachment:t:0 out.ttf -i INPUT
</code></pre>
把所有附件流提取出来，文件名就由元数据中文件流的 filename 项决定：
<pre><code class="highlight-chroma">FFmpeg -dump_attachment:t "" -i INPUT
</code></pre>
技术性提醒：附件是被当成 codec 的额外数据进行处理的，所以这个选项可以从任何流中提取 codec 外的额外数据，而不只是用于提取附件流。

-noautorotate
不基于元数据中的信息自动旋转视频。

<h3 id="视频选项">视频选项</h3>

-vframes number (output)
为输出文件设定视频帧的总数。这是 -frames:v 的一个老旧的别称，以后最好老实用 -frames:v 。 </
>

-r[:stream_specifier] fps (input/output,per-stream)
设置帧速率（单位 Hz，分数或缩写）
作为输入选项时，忽略文件内的时间戳，而生成假设原文件是恒定帧速率（单位 fps）的时间戳
这和用于某些格式（age2、v4l2）的输入文件的 -framerate 选项不同（在一些早期版本的 FFmpeg
中，二者是相同的）。如果不确定，保险起见还是用 -framerate 代替 -r 这个输入选项吧。
作为输出选项时，复制或丢弃输入帧，以达到恒定输出帧速率（单位 fps）。

-s[:stream_specifier] size (input/output,per-stream)
设定帧大小
作为输入选项时，这是一个视频大小私有选项的快捷方式，对于一些帧大小没有被储存在文件中
帧大小可调的输入，例如 raw 视频 或 视频采集卡数据，一些分流器可以使用这个参数。
作为输出选项时，这会插入一个视频放缩滤镜（scale video filter）到 filtergraph 的尾部。如
要在开始或其它地方使用放缩，请直接在需要的地方插入视频放缩滤镜。
大小格式是“宽 x 高（wxh）”，中间是一个小写 x，不是乘法符号哈~。默认数值是源大小。<
p>

-aspect[:stream_specifier] aspect (output,per-stream)
设定视频播放比例为 aspect 数值。
aspect 可以是浮点数，也可以是英文冒号分隔的比例形式“num:den”，num 代表分子（nume
ator），den 代表分母（denominator）。例如：4:3、16:9、1.3333、1.7777 等有效参数。
若同时使用了 -vcodec copy ，只会在容器级别上影响视频比例，如果在编码级别有比例参数的
，对编码级别的比例参数不会有比例影响。

-vn (input/output)
作为输入选项时，阻止文件内的视频流被施加滤镜、自动选择、映射到任何输出。若要禁止单独的流
请参阅 -discard 选项。
作为输出选项时，关闭视频录制（也就是视频流自动选择或映射任何视频流），若要完全手动控
视频流的选择，请参阅 -map 选项。

-vcodec codec (output)
设置视频编码器。这是 -codec:v 选项的别名。

-pass[:stream_specifier] n (output,per-stream)
选择这是第几次压制（1 或 2）。常被用来做 2-pass（二压）视频编码。
第一次压制（1-pass）视频的统计数据会被写到一个 log 文件（参阅 -passlogfile 选项），第
次压制时，会参阅这个 log 文件，以达到更精确的码率控制（复杂的画面给多点码率，简单的画面
给点码率）。
第一次压制的时候，你可以直接取消音频压制，并把输出文件路径指向 NULL （null 路径是一个
殊路径，任何数据输到这个路径都会被销毁为空），省得第一次压制的输出视频占你空间。
以下是对 Windows 和 Unix 的第一次压制示例：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y NUL
FFmpeg -i foo.mo

-c:v libxvid -pass 1 -an -f rawvideo -y /dev/null
</code></pre>
这里除原文外多说一下，2-pass 是要分两步命令进行的，不能直接用 -pass 2 选项，因为你还有 -pass 1 生成的 log。例如，在 Termux 上，你要 2-pass 一个文件码率降到 2000k，应当执行像样两步：
<pre><code class="highlight-chroma">FFmpeg -i input.mp4 -c:v libx264 -pass 1 -an -f rawvideo -y /dev/null
FFmpeg -i input.mp4 -c:v libx264 -pass 2 -b:v 2000k -c:a copy output.mp4
</code></pre>

-passlogfile[:stream_specifier] prefix (output,per-stream)
设置 2-pass 所用 log 文件名的前缀为 prefix，默认的文件名前缀是“FFmpeg2pass”。完整文件名是 PREFIX-N.log，这里的 N 指的是输出流的序数。
这里除了原文外还要多说一点，例如在 FFmpeg Media Encoder 这个 App 中，用这样的参数：/p>
<pre><code class="highlight-chroma">FFmpeg -i /sdcard/input.mp4 -c:v libx264 -pass 1 -an -f rawvideo -y /dev/null
</code></pre>
无法执行，会提示无法打开 stats 文件，应该是没有写入权限，这时我们就可以用-passlogfile 项了：
<pre><code class="highlight-chroma">FFmpeg -i /sdcard/input.mp4 -c:v libx264 -pass 1 -passlogfile /sdcard/FFmpeg/passlog -n -f rawvideo /dev/null
</code></pre>
第一次压制后就会在 /sdcard/FFmpeg/ 目录下生成名为 passlog-0.log 的 stat 文件，然后 2-pass 时的 log 文件路径也选择这个目录：
<pre><code class="highlight-chroma">FFmpeg -i /sdcard/input.mp4 -c:v libx264 -pass 2 -passlogfile /sdcard/FFmpeg/passlog -bv 2000k -c:a copy /sdcard/output.mp4
</code></pre>
或者你也可以试试把 1-pass 出来的视频保留，和 2-pass 出来的视频做对比：
<pre><code class="highlight-chroma">FFmpeg -i /sdcard/input.mp4 -c:v libx264 -pass 1 -passlogfile /sdcard/FFmpeg/passlog -bv 2000k -c:a copy /sdcard/1_pass_output.mp4
FFmpeg -i /sdcard input.mp4 -c:v libx264 -pass 2 -passlogfile /sdcard/FFmpeg/passlog -b:v 2000k -c:a copy /sdcard/2_pass_output.mp4
</code></pre>
压制完成后，看下相同码率下，压制一次和压制两次的画质差异。

-vf filtergraph (output)
创建一个由 filtergraph 指定的滤镜组，并用这个滤镜组为流添加滤镜。
这是 -filter:v 的一个别称，详见 -filter 选项。

<h3 id="高级视频选项">高级视频选项</h3>

-pix_fmt[:stream_specifier] format (input/output,per-stream)
设置像素格式。
用 -pix_fmts 可以查看都支持哪些像素格式。

如果所选的像素格式无法使用，FFmpeg 会发出警告，然后使用该输出文件容器支持最佳的像素式。如果此时 pix_fmt 加了个前缀“+”，FFmpeg 会提示错误并退出，并且 filtergraphs 内的自动换也会被关闭。如果 pix_fmt 只是个单纯的“+”，那 FFmpeg 选择与输入（或图形输出）相同的像格式并且禁用自动转换。
除了原文，再额外提下，我们一般用的 mp4 视频的像素格式是 8bit sRGB，就是用三个数值分表示红绿蓝颜色，每个数值的取值范围是 0-255，每个数值占用 8bit，8bit 就是一个字节（Byte）那一个像素就要占用 24bit（3 字节）。
后来就有人觉得，一个像素 3 字节，这样视频传输就太占带宽了，后来也出现了其它的像素格式先后顺序我没去查）。
一个著名的像素格式就是 YUV 格式，YUV 是一个总称，其下又有许多子格式，谷歌的油管视频输用的 webm 格式视频就用到了 yuv420p 像素格式。下面介绍下 yuv420p 格式的意思。
人眼视网膜有三种感光细胞，一种只感受明暗（数量最多），另外两种感受颜色，人眼对于明度比颜色更敏感，产生了 YUV 格式，其中 Y 表示明度（灰阶值），U 和 V 两个颜色向量（色差），这就可以表示所有人眼可见的颜色。
在传输时，对于彩色电视，使用 YUV 信号，对于黑白电视，只需要采用 Y 信号，不管 UV 就行这样，就实现了彩色和黑白电视对同一种信号的兼容。
因为人眼对颜色的敏感度相对较低，我们就可以缩减 UV 占用的空间。
先看一下 YUV444，以同一行四个像素为一组，它们的数据为：
<pre><code class="highlight-chroma"><span class="highlight
cl">Y0 U0 V0 Y1 U1 V1 Y2 U2 V2 Y3 U3 V3
</code></pre>
存储时，我们就要记录：
<pre><code class="highlight-chroma"><span class="highlight
cl">Y0 Y1 Y2 Y3
U0 U1 U2
  U3
V0 V1 V2
  V3
</code></pre>
总共 12 个数值，要占用 12 字节，平均每个像素占 3 字节，四个像素里有 4 个 Y 值，4 个 U，4 个 V 值。
再来看下 YUV422，还是以同一行四个像素为一组,因为两个像素颜色差异很小，只要明度记录确，相邻两个像素用同一个 UV 值，人眼也看不出来，所以它们的数据为：
<pre><code class="highlight-chroma"><span class="highlight
cl">Y0 U0 V0 Y1 U0 V0 Y2 U2 V2 Y3 U2 V2
</code></pre>
存储时，我们就要记录：
<pre><code class="highlight-chroma"><span class="highlight
cl">Y0 Y1 Y2 Y3
U0 U2

V0 V2
</code></pre>
总共 8 个数值，占用 8 字节，平均每个像素占 2 字节，四个像素里有 4 个 Y 值，2 个 U 值，2 V 值。这样理论上对画质有影响，可是我们人眼看不出来！这就省空间了！
接下来看 YUV420，专家们进一步研究表示，每一行相邻两个像素与下一行同位置的两个像素数差异不大，我们还可以进一步丢数据！考虑两行四列 8 个像素为一组，它们的数据为：
<pre><code class="highlight-chroma"><span class="highlight
cl">Y0 U0 V4 Y1 U0 V4 Y2 U2 V6 Y3 U2 V6
Y4 U0 V4 Y5 U
 V4 Y6 U2 V6 Y7 U2 V6
</code></pre>
存储时，我们就要记录：
<pre><code class="highlight-chroma"><span class="highlight

cl">Y0 Y1 Y2 Y3
U0 U2

Y4 Y5 Y6 Y7
V4 V6
</code></pre>
<p>这样的话，对奇数行每四个像素，我们只需要记录 4 个 Y 值、2 个 U 值、0 个 V 值；对偶数行四个像素，我们只需要记录 4 个 Y 值、0 个 U 值、2 个 V 值。平均每四个像素只需记录 6 个数值，用 6 字节，平均每个像素只占 1.5 字节！</p>
<p>YUV 有两种记录方式。</p>
<p>打包（packed）方式就是以相邻几个像素为一组，用一小包数据记录这一组像素的所有 YUV 数。</p>
<p>平面（planar）方式就是把 Y、U、V 分开后，将它们的数值各自存储到一个数组中，就是一个 Y 数组，一个 U 数组，一个 V 数组。就像是把三种不同的数据放到三个不同平面进行存放。</p>
<p>而 YUV420P 中的“P”就表示平面存储方式。</p>
<p>相比较 sRGB 像素格式，视频采用 YUV420P 像素格式来存储视频，就可以在对画质影响较小的提下，将码率折半！</p>

<p>-sws_flags flags (input/output)
设置 SwScaler 标识。</p>

<p>-rc_override[:stream_specifier] override (output,per-stream)
对指定区间覆盖码率控制（Rate control），格式为“int,int,int”（整数，整数，整数），用“/”开的列表。前两个数值是开始帧和结束帧的数值，第三个数值，正值表示量化值（quantizer），负表示质量因子（quality factor）。</p>

<p>-ilme
强制让编码器（只有 MPEG-2 和 MPEG-4 ）支持隔行扫描（interlacing）。如果你的输入文件是隔扫描的，并且你希望以最小的画质损失保持隔行扫描。替代方法是用 -deinterlace 执行“去除隔行扫”，但“去除隔行扫描”会带来质量损失。</p>

<p>-psnr
计算压缩帧的 PSNR（峰值信噪比 Peak signal-to-noise ratio）。</p>

<p>-vstats
将视频编码统计数据（video coding statistics）写入到 vstats_HHMMSS.log 文件中。</p>

<p>-vstats_file file
将视频编码统计数据（video coding statistics）写入到指定的“file”这个文件中。</p>

<p>-vstats_version file
指定使用哪个版本（version）的视频编码统计数据（video coding statistics）。默认值是 2</p>
<pre><code class="highlight-chroma">version = 1 :
<span class="highlight-line"><span class="highlight-cl">frame= %5d q=

2.1f PSNR= %6.2f f_size= %6d s_size= %8.0fkB time= %0.3f br= %7.1fkbits/s avg_br= %7.1fkbts/s

version > 1:
out= %2d st= %2 frame= %5d q= %2.1f PSNR= %6.2f f_size= %6d s_size= %8.0fkB time= %0.3f br= %7.1fkbits s avg_br= %7.1fkbits/s
</code></pre>

-top[:stream_specifier] n (output,per-stream)
top=1/bottom=0/auto=-1 field first （官方文档就这么一行，我也不清楚该咋解读）

-dc precision
Intra_dc_precision. （官方文档就这么一行，我也不清楚该咋解读）

-vtag fourcc/tag (output)
强制视频 tag/fourcc。这是 -tag:v 的别名。

-qphist (global)
显示 QP 直方图（QP histogram）。
-vbsf bitstream_filter
反对用这个，详见 -bsf。

-force_key_frames[:stream_specifier] time[,time...] (output,per-stream)
-force_key_frames[:stream_specifier] expr:expr (output,per-stream)
-force_key_frames[:stream_specifier] source (output,per-stream)
force_key_frames 可以是以下形式的参数：

time[,time...]
如果参数包含时间戳，FFmpeg 会依据编码器时间基准估计出离所给的每个时间戳最近的输出时戳，并在确切计算出的时间戳的位置、或比计算出的时间戳稍微大点的位置的第一帧强制设一个关键。要提醒的是，如果编码器时间基准很粗糙，那么关键帧可能别设在比指定时间靠前一点的那一帧上默认的编码器时间基准是输出帧率的倒数，但也可以通过 -enc_time_base 选项设定。
如果其中一个时间是“chapters[delta]”，那这个时间就会被替换成文件内所有章节开始的时间每连个值之间相差一个 delta，表示方式为以秒为单位的时间。当要确保在输出文件的一个章节标记或其它标识处存在一个 查找点（seek point）时这个选项是很有用的。
例如，要在五分钟处插入一个关键帧，在每一章 0.1 秒前添加一个关键帧：
<pre><code class="highlight-chroma">-force_key_frames 0:05:00,chapters-0.1
</code></pre>

expr:expr
如果参数的前缀加上了”expr:“，那么“expr”就被像表达式一样被解析，并且对于每一帧都会求。当计算结果为非 0 时会强制一个关键帧。
表达式可以包含如下常数：

n

当前总的处理过的帧数，从 0 开始。
n_forced
强制添加的关键帧的数量。
prev_forced_n
之前强制添加的关键帧是整个视频第几帧，当没有关键帧被添加时它的值是“NSN”。
prev_forced_t
之前添加的关键帧的时间，当没有关键帧被添加时它的值是“NSN”。
t
当前处理帧的时间。

例如要每 5 秒添加一个关键帧，你可以这样写：
<pre><code class="highlight-chroma">-force_key_frames expr:gte(t,n_forced*5)
</code></pre>
要从第 13 秒开始，在每前一个关键帧 5 秒后插入一个关键帧：
<pre><code class="highlight-chroma">-force_key_frames expr:if(isnan(prev_forced_t),gte(t,13),gte(t,prev_forced_t+5))
</code></pre>

source
如果参数是“source”，如果当前被编码帧在原视频里就是一个关键帧，FFmpeg 会强制插入个关键帧。

注意，强制太多关键帧对特定编码器的向前算法（lookahead alogorithms）是非常有害的：使 fixed-GOP 选项或相似的选项会更高效。

-copyinkf[:stream_specifier] (output,per-stream)
当做复制流时，也复制在开始找到的非关键帧（copy input non-key frames found at the beginnin）。

-init_hw_device type[=name][:device[,key=value...]]
使用给定的参数，启用一个类型是“type”、名字是“name”的新硬件。如果没有指定的名字那这个硬件会受到一个“type%d”形式的默认名字。
“device”的意义和下面的参数取决于设备类型：

cuda
“device”是 CUDA 设备的序号。

dxva2
“device”是 Direct3D 9 显卡适配器的序号。

vaapi
“device”是 X11 display 的名字或 DRM 渲染节点。如果没有指定，会尝试打开默认的 X11 display ($DISPLAY) 然后打开第一个 DRM 渲染节点（/dev/dri/renderD128）。

vdpau

“device”是 X11 display 的名字。如果没有指明，会尝试打开默认的 X11 display ($DISPLAY) 。<
p>

qsv
“device”在‘MFX_IMPL_*’中选择一个值，可选的值有：
<pre><code class="highlight-chroma"><span class="highlight
cl">auto
sw
hw
auto_any
hw_any
hw2
hw3
hw4
</code></pre>
如果没有指明，会使用“auto_any”。（需注意的是对于 QSV，通过创建平台许可的子设备（
dxva2”或“vaapi”）然后获得出一个 QSV 设备，可能会更容易达到期望的结果）

opencl
“device”像“platform_index.device_index”一样选择平台和设备。

设备集也可以通过 键名-键值对 筛选来找到匹配特定平台或“device string”的唯一的设备
可用作筛选器的“device string”有：
<pre><code class="highlight-chroma"><span class="highlight
cl">platform_profile
platform_version
platform_name
platform_vendor
platform_extensio
s
device_name
device_vendor
driver_version
device_version
device_profile
device_extensions
device_type
</code></pre>
索引号和过滤器必须一起，独一地，选择一个设备。（The indices and filters must together u
iquely select a device.）
例如：
<pre><code class="highlight-chroma"><span class="highlight
cl">-init_hw_device opencl:0.1
选择第一个平台上
第二个设备。

-init_hw_device o
encl:,device_name=Foo9000
选择名字中带有“F
o9000”的设备。

-init_hw_device oencl:1,device_type=gpu,device_extensions=cl_khr_fp16
选择第二个平台上持“cl_khr_fp16”拓展的 GPU 设备。

vulkan
如果“device”是个整数，那就通过取决于系统的设备列表中的索引号选择设备。如果“device”是任何字符串，就选名字中带有这个字符串的第一个设备。

例如：
-init_hw_device vukan:1
选择系统中的第二设备。

-init_hw_device vukan:RADV
选择名字中带有“ADV”的第一个设备。

-init_hw_device type[=name]@source
启动一个类型是“type”的、后者叫“name”的、从已存在的名叫“source”设备中衍生出来的新硬件设备。（Initialise a new hardware device of type type called name, deriving it from the existing device with the ame source.）
</code></pre>

-init_hw_device list
列出所有的被这一版本 FFmpeg 支持的硬件设备种类。

-filter_hw_device name
传递名叫“name”的硬件设备给所有的滤镜和滤镜图形（filter graph）。这可以让设备用“hwupload”滤镜做上传，或让设备用“hwmap”滤镜做映射。当需要硬件设备时，其它滤镜也可以用这个参。要注意的是，通常，当输入不是已经在硬件帧里时，这个选项才会被需要。当输入已经在硬件帧里，滤镜会从“它们从作为输入收到的帧的情景里”获得它们需要的设备（when it is, filters will derive the device they require from the context of the frames they receive as input.）。（这里的翻好难呐，英语的从句套娃一多起来，就难以转换成中文了，翻译难懂的地方，我把原文留下了）
这时一个全局选项，所有滤镜都会收到这个设备。

-hwaccel[:stream_specifier] hwaccel (input,per-stream)
使用硬件加速来解码匹配的流。可用的“hwaccel”值有：

none
不使用硬件加速（默认项）。
auto
自动选择硬件加速方法。
vdpau
使用 VDPAU (Video Decode and Presentation API for Unix) 硬件加速。
dxva2

使用 DXVA2 (DirectX Video Acceleration) 硬件加速。
vaapi
使用 VAAPI (Video Acceleration API) 硬件加速
qsv
对视频转码使用 Intel QuickSync Video 硬件加速。

不像其他值，太合格选项不会启用硬件加速解码（也就是当一个 qsv 解码器被选择时才会自动做），但是会加速转码，而且无需复制帧到系统内存里。
为了能让它生效，解码器和编码器都必须支持 QSV 加速，并且不能使用滤镜。
如果被选择的“hwaccel”不被所使用的编码器支持，这个选项就没有用。
要注意的是，多数加速方法是为了回放而设计的，并且不会比在现代 CPU 上软解快多少。另外，Fmpeg 会经常需要从 GPU 内存中复制被解码的帧到系统内存，导致进一步的性能损失。所以这个选只是主要对测试有用。

-hwaccel_device[:stream_specifier] hwaccel_device (input,per-stream)
选择一个设备来做硬件加速。
只有当 -hwaccel 选项被指派时，这个选项才有使用的道理。它即可以指向“由 -init_hw_device 通过名字创建的已存在的”硬件，也可以指向就像之前调用“-init_hw_device type:hwaccel_device 一样创建的一个新硬件。

-hwaccels
列出所有这一版本 FFmpeg 支持的硬件加速方法。

<h3 id="音频选项">音频选项</h3>

-aframes number (output)
设置输出音频的帧率。这是 -frames:a 的一个老旧的别名，以后最好乖乖用 -frames:a 选项。

-ar[:stream_specifier] freq (input/output,per-stream)
设定音频采样率。对于输出流，默认采样率是对应输入流的采样率。对于输入流，这个选项只有在用音频抓取设备和 raw demuxer 并且被映射到对应的分流器选项时才行的通。

-aq q (output)
设定音频质量 (codec-specific, VBR)。这是 -q:a 的别名。

-ac[:stream_specifier] channels (input/output,per-stream)
设定音频声道数量。对于输出流，默认声道数量是对应输入流的声道数量。对于输入流，这个选项只在用于音频抓取设备和 raw demuxer 并且被映射到对应的分流器选项时才行的通。

-an (input/output)
作为输入选项用时，阻止一切所有音频流被施加滤镜、自动选择、映射到任何一个输出。若要单独只蔽一个流，请参阅 -discard 选项。
作为输出选项是，屏蔽音频录制（也就是自动选择或映射任何音频流）。若要完全手动控制，请阅 -map 选项。

-acodec codec (input/output)
设置音频编码器，这是 -codec:a 的别名。

-sample_fmt[:stream_specifier] sample_fmt (output,per-stream)
设置音频采样格式，使用 -sample_fmts 查看所有支持的音频采样格式。

-af filtergraph (output)
创建由“filtergraph”指定的 filtergraph，并将该 filtergraph 给流加滤镜。
这是 -filter:a 的别名，详见 -filter 选项。

<h3 id="高级音频选项">高级音频选项</h3>

-atag fourcc/tag (output)
强制加上音频 tag/fourcc。这是 -tag:a 的别名。
-absf bitstream_filter
已经不再使用了，详见 -bsf 。
-guess_layout_max channels (input,per-stream)
如果一些输入的声道数是未知的，只有当它最符合指定的数量的声道时尝试猜测。例如，2 就告诉 FFmpeg 去识别为单声道或立体声，而不要去识别是不是 6 声道的 5.1 音频布局。默认总会去猜，使用 0 值关闭猜测。

<h3 id="字幕选项">字幕选项</h3>

-scodec codec (input/output)
指定字幕编码器。这是 -codec:s 的一个别称。

-sn (input/output)
作为输入选项用时，阻止一切所有字幕流被施加滤镜、自动选择、映射到任何一个输出。若要单独只蔽一个流，请参阅 -discard 选项。
作为输出选项是，屏蔽字幕录制（也就是自动选择或映射任何音频流）。若要完全手动控制，请阅 -map 选项。

-sbsf bitstream_filter
已经弃用了，详见 -bsf 。

<h3 id="高级字幕选项">高级字幕选项</h3>

-fix_sub_duration
固定字幕持续时长。对每条字幕，等待同一流中的下一个数据包，并且调整第一个的持续时间，以避重叠。这对于某些字幕编码器是必要的，尤其是 DVB 字幕，因为原数据包里的持续时长只是一个粗的估计，并且结尾只是用一个空字幕帧做的标记。当必要时，没用使用这个选项，可能导致夸张的持时长或由于非一维时间戳（due to non-monotonic timestamps）导致混流失败。
注意这个选项会延迟所有数据的输出，直到下一个数据包被编码：可能会增加内存占用并且增加多延迟。

-canvas_size size
设置用于渲染字幕的画布大小。

<h3 id="高级选项">高级选项</h3>

-map [-]input_file_id[:stream_specifier][?][,sync_file_id[:stream_specifier]] | [linklabel] (output)
对输出文件指定一个或多个流作为输入源。每个输入流都被输入文件索引号“input_file_id”和输入件内的输入流索引号“input_stream_id”规定身份。两种索引都是从 0 开始计数。如果指派了，“input_file_id:input_stream_id”会指定那个输入流被用作展示同步参考（presentation sync reference

第一个 -map 选项会指定 输出流 0 的输入源，第二个 -map 选项会指定 输出流 1 的输入源，以类推。
在流标识符前加一个“-”表示负映射，可以取消已经创建的映射。
在流标识符后加一个“?”表示这个映射可选，也就是当标识的流存在时就映射，不存在就跳过而不是直接失败。如果没有这个“?”，你定义了一个不存在的流标识符的映射，那映射就会失败。</
>
一个可选的 [linklabel] 形式会将一个复合 filtergraphs 的输出映射到输出文件。linklabel 必须应于一个提前定义好的 linklabel。
例如，映射 第一个输出文件的所有流 到 输出：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i INPUT -map 0 output
</code></pre>
例如，如果你第第一个输入文件有两个音频流，这些流被“0:0“0:1”区分，你可以用 -map 选要把哪个流写入输出文件：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i INPUT -map 0:1 out.wav
</code></pre>
这个会把“0:1”表示的这一个音频流写入 out.wav
例如，要选择输入文件 a.mov 中索引号为 2 的流（由“0:2”表示），选择输入文件 b.mov 中引号为 6 的流（由“1:6”表示），写到输出文件 out.mov。
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i a.mov -i b.mov -c copy -map 0:2 -map 1:6 out.mov
</code></pre>
要映射所有视频流和输入文件中第三个音频流到输出文件：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i INPUT -map 0:v -map 0:a:2 OUTPUT
</code></pre>
要映射除了第二个音频流外所有的流到输出文件，使用负映射：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i INPUT -map 0 -map -0:a:1 OUTPUT
</code></pre>
要映射输入文件的音频流和视频流到输出文件，跟接“?”，如果音频流不存在则忽略音频流映：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i INPUT -map 0:v -map 0:a? OUTPUT
</code></pre>
取出一个音频流：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i INPUT -map 0:m:language:eng OUTPUT
</code></pre>
提醒下，用这个选项，会关闭对输出文件的默认流映射。

-ignore_unknown
尝试复制未知类型的流时，忽略输入流，而不是直接失败。

-copy_unknown
尝试复制未知类型的流时，允许复制输入流，而不是直接失败。

-map_channel [input_file_id.stream_specifier.channel_id|-1][?][:output_file_id.stream_specifier]
将一个音频流的指定声道映射到一个输出。如果“output_file_id.stream_specifier”没有指定，那个音频声道会被映射到输出文件所有的音频流中。
不使用“output_file_id.stream_specifier”，而是使用“-1”，会映射一个静音声道。
后加一个“?”表示这个映射可选，也就是当找不到指定的声道时，就跳过，而不是直接失败。
例如，假设 INPUT 是一个立体声文件，你可以用下面的命令将左右声道互换：
<pre><code class="highlight-chroma">FFmpeg -i INPUT -map_channel 0.0.1 -map_channel 0.0.0 OUTPUT
</code></pre>
如果你想静音第一个声道，保持第二个声道：
<pre><code class="highlight-chroma">FFmpeg -i INPUT -map_channel -1 -map_channel 0.0.1 OUTPUT
</code></pre>
使用 "-map_channel" 的顺序决定了该声道在输出流中的声道序号。输出文件的声道布局，是根声道数量猜测的（一个声道就是单声道“mono”，两个声道就是立体声道“stereo”），如果输入件和输出文件的声道布局不一致，将 -ac 和 -map_channel 一起使用时，会使声道增益级别被更新例如用了两次 -map_channel 和一个 -ac 6）。
你可以将输入文件的各个声道分别指定不同的输出文件：
<pre><code class="highlight-chroma">FFmpeg -i INPUT -map_channel 0.0.0 OUTPUT_CH0 -map_channel 0.0.1 OUTPUT_CH1
</code></pre>
下例把输入文件的立体声音频流的两个声道，分开写到同一个输出文件的两个音频流中：
<pre><code class="highlight-chroma">FFmpeg -i stereo.wav -map 0:0 -map 0:0 -map_channel 0.0.0:0.0 -map_channel 0.0.1:0.1 -y out.ogg
</code></pre>
注意，目前，每个输出流只能包含来自同一个文件的多个声道，你无法使用 "-map_channel" 将含于多个音频流的声道们合并到一个输出流里面。例如，无法将两个单声道音频流（分别来自不同的频流）合并成一个立体声音频流。不过将一个立体声音频流分离成两个音频流是可行的。
如果你需要合并不同流的声道到一个流，一个可行的办法是使用“amerge”滤镜。例如，你要把nput.mkv 中的两个单声道音频，合并到一个立体声音频，用下述命令：
<pre><code class="highlight-chroma">FFmpeg -i input.mkv -filter_complex "[0:1] [0:2] amerge" -c:a pcm_s16le -c:v copy output mkv
</code></pre>
从输入文件的第一个音频流映射前两个音频声道，跟上一个“?”，如果它是单声道，就会跳过个音频映射：
<pre><code class="highlight-chroma">FFmpeg -i INPUT -map_channel 0.0.0 -map_channel 0.0.1? OUTPUT
</code></pre>

-map_metadata[:metadata_spec_out] infile[:metadata_spec_in] (output,per-metadata)

将“infile”指定的元数据写到输出文件。注意，这些是文件索引号（从 0 开始），而不是文件名。选的“metadata_spec_in/out”参数会指定复制哪些元数据。一个元数据标识符可用以下的形式：</>

g
全局（global）元数据，将元数据应用于整个文件。
s[:stream_spec]
流级别的元数据。stream_spec 是一个流标识符章节讲过的流标识符。在输入元数据标识符中，第一匹配的流是复制源。在输出元数据标识符中，所有的匹配流是元数据复制到的目标。
c:chapter_index
章级别的元数据。chapter_index 是从 0 开始数的章节索引号。
p:program_index
程序级别的元数据。program_index 是从 0 开始数的程序索引号。

如果省略了元数据标识符，默认会使用全局标识符。
默认，从第一个输入文件复制全局元数据，流级别和章级别的元数据会随“流”和“章”跟着复过去。可以通过创建任何相关类型的索引，关闭这些默认的映射。
例如将输入文件的第一个流的元数据复制到输出文件的全局元数据：
<pre><code class="highlight-chroma">FFmpeg -i in.ogg -map_metadata 0:s:0 out.mp3
</code></pre>
要做相反的，例如复制全局元数据到音频流元数据：
<pre><code class="highlight-chroma">FFmpeg -i in.mkv -map_metadata:s:a 0:g out.mkv
</code></pre>
提醒下，在这个例子中，只用“0”也会起相同作用。因为全局元数据就是默认项。

-map_chapters input_file_index (output)
从输入文件根据复制 input_file_index 章节到下一个输出文件。如果没有指定映射，那么会从第一个入文件至少复制一个章节。使用负索引号可以阻止所有章节的复制。

-benchmark (global)
在编码结束时显示 Benchmark 信息。显示实际的、系统级的、用户的所用时间和最大内存消耗量。是所有系统都支持显示最大内存消耗量，如果不支持，这个数值会显示为 0。

-benchmark_all (global)
在编码过程中显示 Benchmark 信息。显示实际的、系统级的、用户的所用时间和各种步骤（音视频解码）。

-timelimit duration (global)
当 FFmpeg 使用的 CPU 用户时间超过指定 duration （单位秒）后就退出。

-dump (global)
将每个输出包传到 stderr 。

-hex (global)

当传输输出包时，也传输有效负载（payload）。

-re (input)
用本地帧速率读取输入。主要用于模拟抓取设备、或直播输入流（例如从一个文件读取）。用于真正抓取设备或输入流时别用这一项（会丢包的）。默认情况下 FFmpeg 会用尽可能最快的速度读取输。这一选项能限制读取输入的速率为本地帧速率。对于实时输出很有用（例如直播推流）。

-vsync parameter
视频同步方法。出于兼容原因，可以设置为以前一直使用的数字值。新加入的值将一直是字符串形式

0, passthrough
每一帧将连同它的时间戳，被从分流器传到混流器。
1, cfr
会有帧将会被复制或丢弃，以达到指定的恒定帧速率（constant frame rate）。
2, vfr
帧会连带它们的时间戳被传递、或被丢弃，以防止两帧有相同的时间戳。
drop
和 passthrough 一样，但会销毁所有时间戳，让混流器根据帧速率生成新的时间戳。
-1, auto
取决于混流器，在 1 和 2 之间自动选择。这一项是默认值。

提醒下，在这之后，时间戳可能会被混流器进一步修改。例如，当格式选项 avoid_negative_ts 用时。
使用 -map ，你可以决定从哪个流选用时间戳。你可以保持音频和视频不被修改，并且将剩下的和未被修改的音视频流同步。

-frame_drop_threshold parameter
丢帧阈值，就是在丢帧前标明多少个后视频帧。以视频帧速率为单位，1.0 就是 1 帧。默认值是 -1.1 一个可能的使用场景是在噪声时间戳中防止丢帧或在准确时间戳中增加丢帧精度。（Frame drop threshold, which specifies how much behind video frames can be before they are dropped. In frame rate units, so 1.0 is one frame. The default is -1.1. One possible usecase is to avoid framedrop in case of noisy timestamps or to increase frame drop precision in case of exact timestamps.

-async samples_per_second
音频同步方法。“Stretches/squeezes”（拉伸或挤压）音频流，来匹配时间戳，参数是音频改变后最大每秒采样数。-async 1 是一个特殊的情况，此时只有音频流的开始部分被修正，后面的部分不被改。
提醒下，在这之后，时间戳可能会被混流器进一步修改。例如，当格式选项 avoid_negative_ts 用时。
这个选项已经被弃用，请用音频滤镜替代。

-copyts
不要处理输入时间戳，而是保持它们的数值。尤其是，不要移除“开始时间偏移值”。
注意，取决于 -vsync 选项、或特定混流器处理过程（例如当启用 avoid_negative_ts 时），当个选项开启时，输出时间戳可能与输入时间戳不匹配。

-start_at_zero
当和 copyts 一起用时，移动输入时间戳，使它从 0 开始。
意思是，例如使用“-ss 50”时，会让输出时间戳从 50 秒处开始，不管输入文件开始的时间戳

-copytb mode
指定当复制流时如何设置编码器的时间基准。“mode”是一个整数值，可用以下值：

1
使用分流器时间基准。
从对应的输入分流器复制时间基准到输出编码器。有时（当复制的视频流是可变帧速率时）为了
免单调地增加时间戳会用到这个。

0
使用解码器时间基准。
从对应的输入解码器复制时间基准到输出编码器。

-1
自动选择，以生成合理的输出。

默认值是 -1。

-enc_time_base[:stream_specifier] timebase (output,per-stream)
设定编码器时间基准。“timebase”是一个浮点数值。可以是以下数值：

0
依据媒体类型设置一个默认值。
对于视频使用 1/framerate，对于音频使用 1/samplerate。

-1
尽可能使用输入流时间基准。
如果输入流不可用，会使用默认时间基准。

0
使用提供的数值作为时间基准。
这里可以使用两个整数的比（例如：1:24、1:48000）或者使用一个浮点数（例如：0.04166、2.
833e-5）

默认值是 0

-bitexact (input/output)
让分/混流器、解/编码器启用 bitexact 模式。

-shortest (output)
当最短的输入流被编码完成时就停止。

-dts_delta_threshold
时间戳不连续的阈值。

-dts_error_threshold seconds
时间戳错误阈值。这个阈值用于丢弃受损的时间戳，默认值是 30 小时（这个数值是随机选择的，听来也有点反常）。

-muxdelay seconds (output)
设定最大 分流-解码 延迟。

-muxpreload seconds (output)
设定初始 分流-解码 延迟。

-streamid output-stream-index:new-value (output)
给输出流指定一个新的流 id 值。这个选项应当比该选项所应用的“输出文件名”优先被设置。当存多个输出文件时，流 id 可能被赋予一个不同的值。
例如，对于一个 mpegts 输出文件要设置 流 0 的 PID 为 33，把 流 1 的 PID 设为 36：
<pre><code class="highlight-chroma">FFmpeg -i inurl -streamid 0:33 -streamid 1:36 out.ts
</code></pre>

-bsf[:stream_specifier] bitstream_filters (output,per-stream)
为匹配的流设定比特流滤镜。“bitstream_filters”是英文逗号分隔的一串比特流滤镜。用 -btfs 可查看都有哪些比特流滤镜。
<pre><code class="highlight-chroma">FFmpeg -i h264.mp4 -c:v copy -bsf:v h264_mp4toannexb -an out.h264
FFmpeg -i file.mov -an -vn -bsf:s mov2textsub -c:s copy -f rawvideo sub.txt
</code></pre>

-tag[:stream_specifier] codec_tag (input/output,per-stream)
对匹配的流强制设一个 tag/fourcc 。

-timecode hh:mm:ssSEPff
对“写入”指定时间码。“SEP”对于“non drop timecode”是“:”，对于“drop timecode”是“.”（此处的 SEP 是指分隔符 seperator，drop timecode 我不太懂，可能是有损时间码的意思）。例：
<pre><code class="highlight-chroma">FFmpeg -i input.mpg -timecode 01:02:03.04 -r 30000/1001 -s ntsc output.mpg
</code></pre>

-filter_complex filtergraph (global)
定义一个复合 filtergraph，例如对多个输入和（或）输出文件。对于简单图表（那些只有一个输入和个输出并且都是相同类型的）请参阅 -filter 选项。“filtergraph”是对 filtergraph 的描述，在 FFmpg-filter 手册里“Filtergraph syntax”一节有详细描述。
必须使用[file_index:stream_specifier] 语法（与 -map 用的一样）将输入链接标签与输入流对。如果“stream_specifier ”匹配多个流，只会使用第一个流。一个未贴标签的输入将会连接到第一未使用的、类型匹配的输入流。（An unlabeled input will be connected to the first unused input stream of the matching type.）
输出链接标签与 -map 相关联。未贴标签的输出会被添加到第一个输出文件。
使用这个选项时，不使用普通输入文件，而只使用 lavfi 源 也是可以的。
例如，要将一个图片叠加在一个视频上：
<pre><code class="highlight-chroma">FFmpeg -i video.mkv -i image.png -filter_complex '[0:v][1:v]overlay[out]' -map '[out]' out mkv
</code></pre>
这里 [0:v] 表示第一个视频的第一个视频流，后者被链接到了第一个 overlay filter 的输入。
假设在第一个输入文件里只有一个视频流，我们就可以不写输入标签，此时上面的例子等同于下：
<pre><code class="highlight-chroma">FFmpeg -i video.mkv -i image.png -filter_complex 'overlay[out]' -map '[out]' out.mkv
</code></pre>
之后我们可以省略输出标签，滤镜表的单独输出会被自动添加到输出文件，所以我们可以写成：
<pre><code class="highlight-chroma">FFmpeg -i video.mkv -i image.png -filter_complex 'overlay' out.mkv
</code></pre>
要生成 5 秒的纯红色视频，使用 lavfi “color”源：
<pre><code class="highlight-chroma">FFmpeg -filter_complex 'color=c=red' -t 5 out.mkv
</code></pre>

-filter_complex_threads nb_threads (global)
定义使用多少线程来处理 filter_complex 过程图。和 filter_threads 相似，不过这个只用于 -filter_complex 过程图。默认数值是可用的 CPUs 数量。

-lavfi filtergraph (global)
定义一个复合 filtergraph，也就是有一个或多个输入和（或）输出的 filtergraph。等同于 -filter_complex。

-filter_complex_script filename (global)
与-filter_complex 相像，不过这里的参数是内含复合 filtergraph 描述的文件的文件名。

-accurate_seek (input)
这项可以启用或禁用 -ss 的精确定位。默认是开启的。-noaccurate_seek 可以禁用它。当复制某些、只转码其它流时可能有用。

-seek_timestamp (input)
这项可以启用或禁用通过时间戳精确定位。默认是关闭的。如果启用，-ss 的参数值会被当作实际的有偏移值的时间戳。仅仅对于某些不是从时间戳 0 值处开始的文件有用，例如转码流。

-thread_queue_size size (input)
这一项设定当从文件或设备读取时排队的数据包的最大数量。对于低延迟、高帧率的直播流，如果数包在一定时间内没有被读取，这些数据包就会被丢弃。增加这个值可以避免丢包。

-sdp_file file (global)
将输出流的 sdp 信息写到“file”中。当有至少一个输出文件不是 rtp 流时，该项可以将 sdp 信息记到文件中。（要求至少有一个输出格式是 rtp）

-discard (input)
允许丢弃指定的流或流中的帧。当在分流器发生“从一个流选择性丢帧”时，使用值“all”，任何输流都会被完全丢掉。

none
不要丢帧。
default
默认，也就是不丢帧。
noref
丢掉所有非参考帧（non-reference frames）。
bidir
丢掉所有双向帧（bidirectional frames）。
nokey
除了关键帧，丢掉其它所有帧。
all
丢掉所有帧。

-abort_on flags (global)
在多种情况下停止，下述“flags”值是可用的：
empty_output
没有数据包被传递到混流器，输出是空的。

-xerror (global)
出错时，停止并退出。

-max_muxing_queue_size packets (output,per-stream)
当转码音视频流时，直到有一个数据包（例如流）时，FFmpeg 才会开始向输出写内容。在这个发生前，其它流是在缓存中存储的。这个选项就是对匹配的流设置缓存大小的，单位是数据包。
对大多数情况，这个默认数值是足够高的，所以，只有当你确切是需要动这个选项时，再动这选。
作为一个特例，你可以用一个位图字幕流作为输入：它会被转成和文件里最大的视频同样大小，文件内没有视频时，会被转换成 720x576 这么大。提醒下，这是一个实验性的、临时性的方法。一旦 ibavfilter 有更恰当的字幕支持，这个方法就会被移除掉。
例如，硬码字幕到一个以 MPEG-TS 格式存储的 DVB-T 录制视频上层，字幕延迟 1 秒：
<pre><code class="highlight-chroma">FFmpeg -i input.ts -filter_complex \
'[#0x2ef] setpts=PS+1/TB [sub] ; [#0x2d0] [sub] overlay' \

-sn -map '#0x2dc' output.mkv
</code></pre>
(0x2d0, 0x2dc 和 0x2ef 是当前各个视频、音频和字幕流的 MPEG-TS PID； 0:0, 0:3 和 0:7 也样有用)

<h3 id="预设文件-Preset-files-">预设文件（Preset files）</h3>
预设文件包含一系列“option=value 对”，一行一个，表示一系列用命令行输起来很麻烦的选。一行以”#“开头会被忽略，用于做注释。你可以查看 FFmpeg 源目录的 presets 文件夹查看例子
总共有两种预设文件：ffpreset 和 avpreet 文件。
<h4 id="ffpreset-文件">ffpreset 文件</h4>
ffpreset 文件由 vpre、apre、spre、fpre 选项组成。fpre 选项用预设文件名作为输入（而不是预设名），并且可以被用于所有编码器。至于 vpre、apre、spre 选项，预设文件只用于当前选择的预设选项相同类型的编码器。
传递给 vpre、apre、spre 预设的参数选项依据以下原则决定这个预设被如何用：
首先 FFmpeg 在按以下顺序在以下路径内搜索名为“arg.ffpreset”的文件：
<pre><code class="highlight-chroma">$FFmpeg_DATADIR
 $HOME/.FFmpeg
</code></pre>
数据文件夹（在配置时指定的，通常时是 PREFIX/share/FFmpeg）或 win32 可执行程序内的 ffresets 文件夹
例如，若参数是 libvpx-1080p，那就会找一个叫 libvpx-1080p.ffpreset 的文件。
如果找不到这样一个文件，然后 FFmpeg 会在上述文件夹中查找一个叫 codec_name-arg.ffpreset 的文件（这里 codec_name 表示该预设将作用于的编码器的名字）。例如，如果你选择 -vcodec libpx 编码器，并使用 -vpre 1080p 预设，那就会寻找名叫“libvpx-1080p.ffpreset”的文件。
<h4 id="avpreset-文件">avpreset 文件</h4>
avpreset 文件由 pre 选项定义。它们和 ffpreset 文件的原理类似，不过后者有针对编码器的选。所以，一个“option=value pair”表示无法使用编码器。（好绕口，我把原文留下）
（avpreset files are specified with the pre option. They work similar to ffpreset files, but they nly allow encoder- specific options. Therefore, an option=value pair specifying an encoder cannot be used.）
当 pre 选项被定义上，FFmpeg 会按以下顺序在以下文件夹寻找带有“.avpreset”后缀的文件
<pre><code class="highlight-chroma">$AVCONV_DATADIR
$HOME/.avconv
</code></pre>
数据文件夹（在配置时指定的，通常时是 PREFIX/share/FFmpeg）
首先，FFmpeg 在上述文件夹找到一个叫 “codec_name-arg.avpreset”的文件（这里 codec_ame 表示该预设将作用于的编码器的名字）。例如，你用“-vcodec libvpx”选择了视频编码器，使了“-pre 1080p”，那它就会寻找一个叫“libvpx-1080p.avpreset”的文件。
如果没有找到这个文件，那 FFmpeg 就会在同样的文件夹内找一个叫“arg.avpreset”的文件。/p>
<h2 id="范例-">范例</h2>
<h3 id="音视频抓取">音视频抓取</h3>
如果你指定了输入格式和设备，接着 FFmpeg 就可以直接抓取音频和视频了：
<pre><code class="highlight-chroma">FFmpeg -f oss -i /dev/dsp -f video4linux2 -i /dev/video0 /tmp/out.mpg
</code></pre>
或者不用 OSS，而用 ALSA 音频源 (mono input, card id 1)：
<pre><code class="highlight-chroma"><span class="highlight

cl">FFmpeg -f alsa -ac 1 -i hw:1 -f video4linux2 -i /dev/video0 /tmp/out.mpg
</code></pre>
提醒下，在带着任何 TV 播放器（例如 Gerd Knorr 写的 xawtv）启动 FFmpeg 时，确保激活了确的视频源和声道。你还需要用一个标准混音器正确地设置音频录制 level 。
<h3 id="X11-抓取">X11 抓取</h3>
用 FFmpeg 抓取 X11 的显示内容：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -f x11grab -video_size cif -framerate 25 -i :0.0 /tmp/out.mpg
</code></pre>
0.0 是你的 X11 服务器的显示屏的序号，和 DISPLAY 环境变量相同。
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -f x11grab -video_size cif -framerate 25 -i :0.0+10,20 /tmp/out.mpg
</code></pre>
0.0 是你的 X11 服务器的显示屏的序号，和 DISPLAY 环境变量相同。10 是 x 方向偏移，20 是 y 方向偏移。
<h3 id="视频和音频格式转换">视频和音频格式转换</h3>
任何支持的文件格式和协议都可以当作 FFmpeg 的输入。
例如：

你可以用 YUV 文件作为输入：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i /tmp/test%d.Y /tmp/out.mpg
</code></pre>
它将会使用这些文件：
<pre><code class="highlight-chroma"><span class="highlight
cl">/tmp/test0.Y, /tmp/test0.U, /tmp/test0.V,
/tmp/test1.Y, /tm
/test1.U, /tmp/test1.V, etc...
</code></pre>
Y 文件有 U 和 V 文件两倍的分辨率。它们都是 raw files，没有文件头。可以被各种视频解码器成。你必须用 -s 选项规定图像大小，因为 FFmpeg 不会猜。

你也可以用 raw YUV420P 文件作为输入：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i /tmp/test.yuv /tmp/out.avi
</code></pre>
test.yuv 是一个包含有平面型（Planar）YUV 原始数据的文件。每一帧都包含一个 Y 平面，跟横向和纵向分辨率都折半的 U 和 V 平面。

你也可以输出一个 raw YUV420P 文件：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i mydivx.avi hugefile.yuv
</code></pre>

你也可以设置多个输入文件和输出文件：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i /tmp/a.wav -s 640x480 -i /tmp/a.yuv /tmp/a.mpg
</code></pre>
转换音频文件 a.wav 和 raw YUV 视频文件 a.yuv 成一个 MPEG 文件 a.mpg。

你也可以同时做音视频转换：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i /tmp/a.wav -ar 22050 /tmp/a.mp2
</code></pre>
转换 a.wav 成采样率为 22050 Hz 的 MPEG 音频。

你可以同时转换成多种格式，并指定输入到输出流的映射：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i /tmp/a.wav -map 0:a -b:a 64k /tmp/a.mp2 -map 0:a -b:a 128k /tmp/b.mp2
</code></pre>
转换 a.wav 为 a.mp2 （比特流 64 kbits ）以及 b.mp2 （比特流 128 kbits）。 ’ -map file:in
ex’ 按照输出流定义的顺序，定义了哪个输入流写到哪个输出流。

你也可以转码破解 VOB：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i snatch_1.vob -f avi -c:v mpeg4 -b:v 800k -g 300 -bf 2 -c:a libmp3lame -b:a 128
 snatch.avi
</code></pre>
这是一个典型的 DVD 扒取例子；输入是一个 VOB 文件，输出包含 MPEG-4 编码视频和 MP3
码音频的 AVI 文件。提醒下，这条命令中，我们使用了 B-frames 所以 MPEG-4 流与 DivX5 兼容，
且 GOP 大小是 300，意味着对于 29.97fps 的输入视频每 10 秒一个内帧。进一步，音频流是 MP3
码的所有你应当通过输入 --enable-libmp3lame 启用 LAME 支持。对于 DVD 转码，要想得到期望
音频语言，map 映射是很重要的。
提示：用“FFmpeg -demuxers”查看支持的输入格式。

你也可以从一个视频提取图片，或通过许多图片创建视频：
若要从视频提取音频：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i foo.avi -r 1 -s WxH -f image2 foo-%03d.jpeg
</code></pre>
这会从视频中每秒提取一帧，输出到名字为 foo-001.jpeg, foo-002.jpeg ... 的一系列文件中。
片会被重新缩放为 WxH 的数值。
如果你只想提取出有限数量的帧，你可以在上面命令的基础上使用 -frames:v 或 -t 选项。或者
合 -ss 从某个确切的时间点开始提取。
若要从许多图片创建视频：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -f image2 -framerate 12 -i foo-%03d.jpeg -s WxH foo.avi
</code></pre>
foo-%03d.jpeg 的语法表示使用十进制数构成的三维数（不足位 0 填充）表示序列号。这个和 C
语言 printf 函数的语法相同，只是使用正整数的格式更合适。
当导入一个图片序列时，通过选择 image2-specific 的 -pattern_type glob 选项，-i 本身也能
支持像 shell 一样的通配符匹配（globbing）。
例如，从文件名符合 glob 匹配的 <code> “foo-*.jpeg” </code> 图片创建一个视频：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -f image2 -pattern_type glob -framerate 12 -i 'foo-*.jpeg' -s WxH foo.avi
</code></pre>

你可以在输出中放入许多相同种类的流：
FFmpeg -i test1.avi -i test2.avi -map 1:1 -map 1:0 -map 0:1 -map 0:0 -c copy -y test12.nut

这会让输出文件 test12.nut 以与输入文件相反的顺序包含输入文件的前四个流。

强制恒定比特流 CBR 的视频输出：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i myfile.avi -b 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
</code></pre>

四个选项 lmin, lmax, mblmin 和 mblmax 使用\lambda </s
an> 单位，但也可以使用 QP2LAMBDA 常数简单地用 ’ q’ 单位转换：
<pre><code class="highlight-chroma"><span class="highlight
cl">FFmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext
</code></pre>

<h2 id="更多参考">更多参考</h2>
更多的内容，请参阅 <a href="https://link.ld246.com/forward?goto=http%3A%2F%2FFFm
eg.org%2F" target="_blank" rel="nofollow ugc">http://FFmpeg.org/ 的下列文档：
FFmpeg-all, ffplay, ffprobe, FFmpeg-utils, FFmpeg-scaler, FFmpeg-resampler, FFmpeg-co
ecs, FFmpeg-bitstream-filters, FFmpeg-formats, FFmpeg-devices, FFmpeg-protocols, FFmpeg-
ilters
<h2 id="作者">作者</h2>
原英文文档作者为 FFmpeg 的开发者们。本文由淳帅二代翻译。
若要查看关于作者权的详情信息，请通过在 FFmpeg 的源文件夹输入“git log”命令，或进入 <
href="https://link.ld246.com/forward?goto=http%3A%2F%2Fsource.FFmpeg.org" target="_b
ank" rel="nofollow ugc">http://source.FFmpeg.org 访问在线库，来参阅这个项目（git://so
rce.FFmpeg.org/FFmpeg）的 Git 历史。
源代码树的“MAINTAINERS”文件中列出了所有特定组件的维护者。
该文档的英文原版生成于 2020 年 2 月 16 号。译于 2020 年 2 月 16 号
<h2 id="补充">补充</h2>
如果您帧率模式选择的是 VFR，您可能会遇到视频和音频同步问题。 在这种情况下，您必须使用
“Motion JPEG” 和 “PCM” 编码器。 如果您使用 “Motion JPEG” 和 “PCM” 编码器还有同
问题，则必须降低录制目标的视频分辨率。