

other-doc-cn-ffmpeg

xdsnet

Published
with GitBook



Table of Contents

1. Introduction
2. 1 命令语法
3. 2 描述/概览
4. 3 详细说明
5. 4 流的选择（指定）
6. 5 选项
7. 技巧/提示（原版已废弃）
8. 6 例子
9. 7 语法
10. 8 表达式计算/求值
11. 9 OpenCL 选项
12. 10 编码选项
13. 11 解码器
14. 12 视频解码
15. 13 音频解码
16. 14 字幕解码
17. 15 编码
18. 16 音频编码器
19. 17 视频编码器
20. 18 字幕编码器
21. 19 比特流滤镜（过滤器）
22. 20 格式选项
23. 21 分离器（解复用）
24. 22 混合器
25. 23 元数据
26. 24 协议
27. 25 设备选项
28. 26 输入设备
29. 27 输出设备
30. 28 重采样(resampler)选项
31. 29 放缩选项
32. 30 滤镜入门
33. 31 graph2dot
34. 32 滤镜链图描述
35. 33 时间线编辑
36. 34 音频滤镜
37. 35 音频源
38. 36 音频槽
39. 37 视频滤镜
40. 38 视频源
41. 39 视频槽
42. 40 多媒体滤镜
43. 41 多媒体源
44. 42 参考
45. 43 开发人员

ffmpeg 翻译文档 (ffmpeg-all 包含重要组件)

目录：

- 1 命令语法
- 2 描述/概览
- 3 详细说明
- 4 流的选择（指定）
- 5 选项
- 技巧/提示（原版已废弃）
- 6 例子
- 7 语法
- 8 表达式计算/求值
- 9 OpenCL 选项
- 10 编码选项
- 11 解码器
- 12 视频解码
- 13 音频解码
- 14 字幕解码
- 15 编码
- 16 音频编码器
- 17 视频编码器
- 18 字幕编码器
- 19 比特流滤镜（过滤器）
- 20 格式选项
- 21 分离器（解复用）
- 22 混合器
- 23 元数据
- 24 协议
- 25 设备选项
- 26 输入设备
- 27 输出设备
- 28 重采样(resampler)选项
- 29 放缩选项
- 30 滤镜入门
- 31 graph2dot
- 32 滤镜链图描述
- 33 时间线编辑
- 34 音频滤镜
- 35 音频源
- 36 音频槽
- 37 视频滤镜
- 38 视频源
- 39 视频槽
- 40 多媒体滤镜
- 41 多媒体源
- 42 参考
- 43 开发人员

1 命令语法

```
ffmpeg [全局选项] {[输入文件选项] -i 输入文件} ... {[输出文件选项] 输出文件} ...
```

即

```
ffmpeg [global_options] {[input_file_options] -i input_file} ... {[output_file_options] output_file} ...
```

2 描述/概览

ffmpeg 是一个非常快的视频/音频转换器，其也可以现场抓取音频/视频源，并在任意采样率、尺寸之间调整视频，以及提供多种高品质的滤镜系统。

ffmpeg 从任意数量/形式的输入文件中进行读取（可以是普通文件，管道，网络流，设备源等等），通过输入文件选项对输入文件进行设定，通过 `-i` 进行标记，并写入到任意数量/形式的输出文件中，任何在命令行中不能被解释为选项的字符串信息（当然也不是被 `-i` 指定为输入文件的信息）都被作为一个输出文件。

原则上每个输入或输出文件都可包含数量不同的数据流（视频/音频/字幕/附件/数据....），具体文件中包含的数量和/或数据类型是文件的容器格式限定的，具体选择那些流从输入文件到输出文件则可能是自动或者依据 `-map` 选项（在[流的选擇](#)章中介绍）来指定。

为了明确指定输入文件，你必须采用从0开始的数字索引法，即第1个输入文件由 `0` 索引，第2个则是 `1`。同样的，在一个文件中指定数据流也是通过同样规则的索引法，即 `2:3` 表示第3个输入文件的第4个数据流。这些内容也可以参考流说明章节。

作为一般规则，选项用于指定紧接着的文件，因此命令中顺序很重要，你可以在命令中多次重复相同的选项，每次都可以应用于紧接着的下一个输入或者输出文件。例外的是全局选项（例如过程信息输出详细程度的选项），这些选项必须首先进行指定，会全局使用。

不要混淆输入和输出文件，要先指定所有的输入文件，然后才是所有的输出文件。也不要混淆选项应用的不同文件，所有的选项仅仅作用于紧接着的输入或者输出文件，除非重复指定选项才能作用于其他需要同样设定的文件。

- 为了设定输出视频码率为64kbit/s：

```
ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
```

- 为了切换帧率到24fps：

```
ffmpeg -i input.avi -r 24 output.avi
```

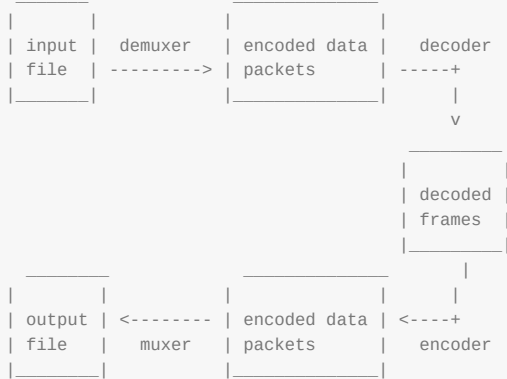
- 为了强制把输入文件帧率设为1fps（仅为了建议raw格式数据），并且把输出文件帧率设置为24fps：

```
ffmpeg -r 1 -i input.m2v -r 24 output.avi
```

注意这里的输入文件必须是raw格式的输入文件。

3 详细说明

ffmpeg 的每个转换过程像下图描述的程序



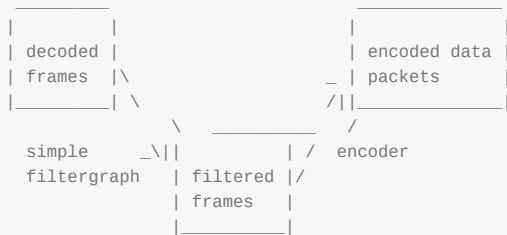
ffmpeg 调用 libavformat 库（含分离器）读取输入文件，分离出各类编码的数据包（流），当有多个输入文件时，ffmpeg 试图跟踪最低时间戳实现任意输入流同步。编码数据包（除非是指定为流式拷贝，相关内容请参考特性描述对[流式拷贝](#)的说明）通过解码器解码出非压缩的数据帧（raw 视频/PCM 格式音频...），这些数据帧可以被滤镜进一步处理（下面会讲到）。经过滤镜处理的数据被重新编码为新的数据包（流），然后经过混合器混合（例如按一定顺序和比例把音频数据包和视频数据包交叉组合），写入到输出文件。

滤镜处理/Filtering

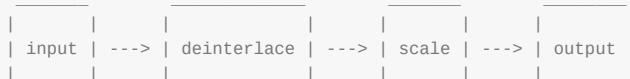
在编码前，ffmpeg 可以对 raw（真实/原）音频和视频使用 libavfilter 库中的滤镜进行处理。多个滤镜可以组成滤镜链图（滤镜链图 filtergraphs）。在 ffmpeg 看来只有 2 种滤镜：简单滤镜，复合滤镜。

简单滤镜

简单滤镜就是只有 1 个输入和输出的滤镜，滤镜两边的数据都是同一类型的，可以理解为在非压缩数据帧到再次编码前简单附加了一步：



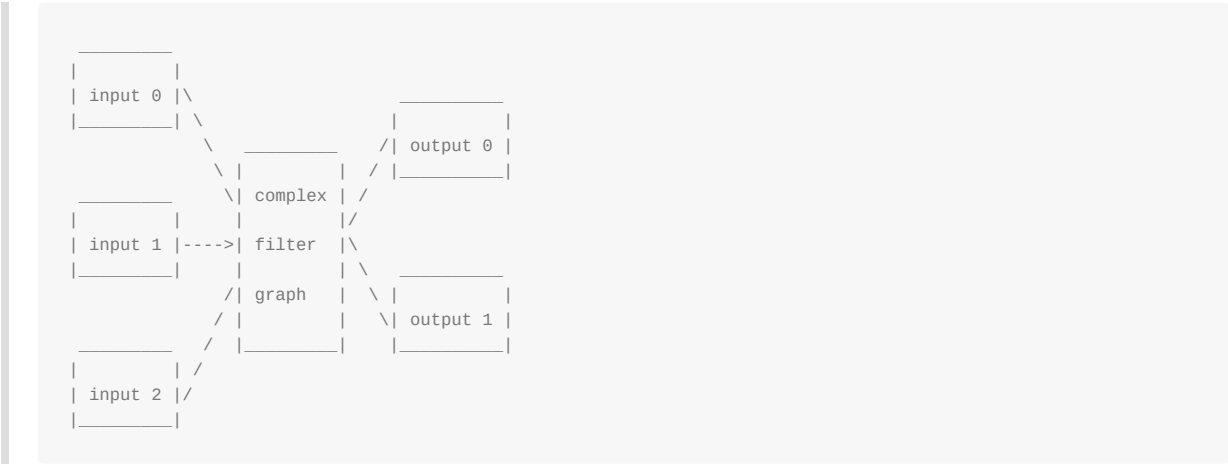
简单滤镜一般用于配置每个流 -filter 选项（-vf 和 -af 分别对应视频和音频）。一个最简单的视频滤镜如下：



注意一些滤镜改变帧属性而不是帧内容。例如前面提到的 fps 滤镜就只是引起帧率的变化，但不处理帧内容，另外一个例子是 setpts 则仅仅设置时间戳，通过滤镜的帧内容完全不变化。

复合滤镜

复合滤镜是那些不能简单描述为一个线性处理过程应用到一个流的情况，例如当过程中有多个输入和/或输出，或者输出流类型不同于输入时，示意图如下：

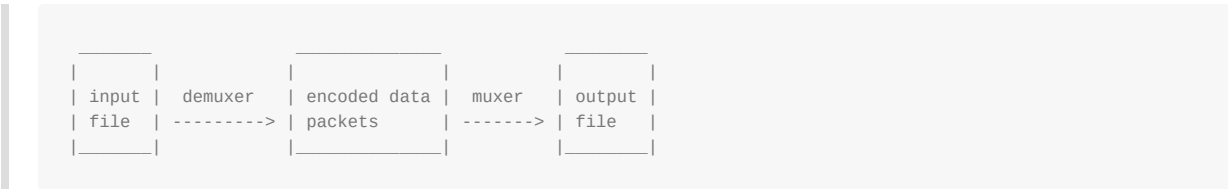


复合滤镜由 `-filter_complex` 选项进行设定。注意这是一个全局选项，因为一个复合滤镜必然是不能只关联到一个单一流或者文件的。`-lavfi` 选项等效于 `-filter_complex`

一个复合滤镜的简单例子就是 `overlay` 滤镜，它从两路输入中，把一个视频叠加到一个输出上。对应的类似音频滤镜是 `amix`。

流拷贝

流拷贝（Stream copy）是一种对指定流数据仅仅进行复制的拷贝（copy）模式。这种情况下 `ffmpeg` 不会对指定流进行解码和编码步骤，而仅仅是分离和混合数据包。这种模式常用于文件包装格式的转换或者修改部分元数据信息，这个过程简单图示如下：



因为这种模式下不存在解码和编码过程，所以也特别快，而且不会造成新的质量损失。然而这也使得这样的模式不能适合很多工作需求，例如这个模式下不能使用大量的滤镜了，因为滤镜仅能对未压缩（编码）的数据进行处理。

4 流的选择（指定）

默认情况下，`ffmpeg` 把输入文件每种类型（视频、音频和字幕）仅仅采用一个流转换输出到输出文件中，就是把最好效果的流进行输出：对于视频就是质量最高的流，对于音频就是包含最多声道的，对于字幕则是第一个字幕轨道，如果有多个同型同率（同样类型，码率相同）则选用索引号最小的流。

当然，你可以禁用默认设置，而采用 `-vn/-an/-sn` 选项进行专门的指定，如果要进行完全的手动控制，则是以 `-map` 选项，它将禁止默认值而选用指定的配置。

5 选项

所有的数值选项，如果没有特殊定义，则需要一个接受一个字符串代表一个数作为输入，这可能跟着一个单位量词首字母，例如 "k", "m" 或 "G"

如果 `i` 是附加到SI单位的首字母，完整的字母将被解释为一个2的幂数单位，这是基于1024而不是1000的，添加 `B` 的SI单位则是再将此值乘以8。例如 `KB`，`MiB`，`G` 和 `B`

对于选项中不带参数的布尔选项，即把相应的值设置为 `true`，它们可以添加 `no` 设置为 `false`，例如 `nofoo` 就相当于 `foo false`。

流说明（限定）符

很多选项是作用于单独的流的，例如码率（bitrate）或者编码（codec），流说明符就是精确的为每个流指定相应的选项。

一个流说明符是一个以冒号分隔的字符串，其中分隔出的部分是附加选项，例如 `-codec:a:1 ac3` 表示编码器是对第2音频流以 `ac3` 编码。

一个流说明符可能匹配多个流，则该选项是所有匹配项的选项，例如 `-b:a 128k` 表示所有的音频流都是128k的码率。

一个空的流说明符匹配所有的流，例如 `-codec copy` 或者 `-codec: copy` 表示所有的流都不进行再次编码（包括视频和音频）

可能的流说明符有：

- `stream_index` : 匹配流的索引，例如 `-threads:1 4` 表示对2号流采用4个线程处理
- `stream_type[:stream_index]` : `stream_type` 有 `v` 表示视频，`a` 表示音频，`s` 表示字幕，`d` 表示数据和 `t` 表示附加/附件等可能，如果 `stream_index` 同时被指定，则匹配该索引对于的该类型的流。例如 `-codec:v:0 h264` 表示第1视频流是h.264编码。
- `p:program_id[:stream_index]` : 如果 `stream_index` 被指定，则表示被 `program_id` 指定的程序仅作用于 `stream_index` 所指流，否则将作用于所有流。
- `#stream_id` 或者 `i:stream_id` : 匹配 `stream_id` 所指流（MPEG-TS中的PID）
- `m:key[:value]` : 匹配在元数据中以标签 `key = value` 值的流，如果 `value` 没有设置，则匹配所有。
- `u` : 匹配不能被配置的流，这时编码器必须被定义且有必要的视频维度或者音频采样率之类的信息。注意，`ffmpeg` 匹配由元数据标识的状态仅对于输入文件有效。

常规选项

这些常规选项也可以用在 `ffmpeg` 项目中其他 `ff*` 工具，例如 `ffplayer`

- `-L` : 显示授权协议
- `-h, -?, -help, --help[arg]` : 显示帮助，一个附加选项可以指定帮助显示的模式，如果没有参数，则是基本选项（没有特别声明）说明被显示，下面是参数定义
 - `long` : 在基本选项说明基础上增加高级选项说明
 - `full` : 输出完整的选项列表，包括编（解）码器，分离器混合器以及滤镜等等的共享和私有选项
 - `decoder=decoder_name` : 输出指定解码器名的详细信息。可以使用 `-decoders` 来获取当前支持的所有解码器名
 - `encoder=encoder_name` : 输出指定编码器名的详细信息。可以使用 `-encoders` 来获取当前支持的所有编码器名
 - `demuxer=demuxer_name` : 输出指定分离器名详细信息。可以使用 `-formats` 来获取当前支持的所有分离器和混合器
 - `muxer=muxer_name` : 输出指定混合器名详细信息。可以使用 `-formats` 来获取当前支持的所有分离器和混合器
 - `filter=filter_name` : 输出指定滤镜名的详细信息。可以使用 `-filters` 来获取当前支持的所有滤镜
- `-version` : 显示版本信息
- `-formats` : 显示所有有效的格式（包括设备）
- `-devices` : 显示有效设备

- `-codecs` : 显示所有已支持的编码（libavcodec中的）格式注意 编码/codec 仅仅是文档中一个用于指示媒体数据流格式的术语。
- `-decoders` : 显示所有有效解码器
- `-encoders` : 显示所有有效的编码器
- `-bsfs` : 显示有效的数据流（bitstream）滤镜
- `-protocols` : 显示支持的协议
- `-filters` : 显示libavfilter中的滤镜
- `-pix_fmts` : 显示有效的像素（pixel）格式
- `-sample_fmts` : 显示有效的实例格式
- `-layouts` : 显示信道名字和信道布局
- `-colors` : 显示注册的颜色名
- `-sources device[,opt1=val1[,opt2=val]...]` : 显示自动识别的输入设备源。一些设备可能需要提供一些系统指派的源名字而不能自动识别。返回的列表不能认为一定是完整的（即有可能还有设备没有列出来）

```
ffmpeg -sources pulse,server=192.168.0.4
```

- `-sinks device[,opt1=val1[,opt2=val]...]` : 显示自动识别的输出设备。一些设备可能需要提供一些系统指派的源名字而不能自动识别。返回的列表不能认为一定是完整的（即有可能还有设备没有列出来）

```
ffmpeg -sinks pulse,server=192.168.0.4
```

- `-loglevel [repeat+]loglevel` 或者 `-v [repeat+]loglevel` : 设置日志层次。如果附加有 `repeat+` 则表示从第一条非压缩行到达到最后消息n次之间的行将被忽略。"repeat" 也可以一直使用，如果没有现有日志层级设置，则采用默认日志层级。如果有多个日志层级参数被获取，使用 "repeat" 不改变当前日志层级。日志层级是一个字符串或数值，有以下可能值：

- `quiet,-8` , 什么都不输出，是无声的
- `panic,0` , 仅显示造成进程失败的致命错误，它当前不能使用
- `fatal,8` 仅仅显示致命错误，这些错误使得处理不能继续
- `error,16` 显示所有的错误，包括可以回收的错误（进程还可以继续的）
- `warning,24` 显示所有警告和错误，任何错误或者意外事件相关信息均被显示
- `info,32` 显示过程中的信息，还包括警告和错误，则是默认值
- `verbose,40` 类似 `info` , 但更冗长
- `debug,48` 显示所有，包括调试信息
- `trace,56`

默认的日志输出是stderr设备，如果在控制台支持颜色，则错误和警告标记的颜色将被显示处理，默认日志的颜色设置可以由环境变量的 `AV_LOG_FORCE_NOCOLOR` 或者 `NO_COLOR` 或者环境变量 `AV_LOG_FORCE_COLOR` 覆盖。环境变量 `NO_COLOR` 不推荐使用，因为其已经不被新版本支持。

- `-report` : 复制所有命令行和控制台输出到当前目录下名为 `program-YYMMDD-HHMMSS.log` 文件中。这常用于报告bug，所以一般会同时设置 `-loglevel verbose`

设置环境变量 `FFREPORT` 可以起到相同的效果。如果值是一个以 `:` 分隔的关键值对，则将影响到报告效果。值中的特殊符号或者分隔符 `:` 必须被转义（参考ffmepg-utils手册中"引用逃逸"("Quoting and escaping")章节）。以下是选项值范围：

- `file` : 设置报告文件名字，`%p` 被扩展为程序名字，`%t` 是时间码，`%%` 表示一个字符 `%`
- `level` : 用数字设定日志信息详略程度（参考 `-loglevel`）

例如：

```
`FFREPORT=file=ffreport.log:level=32 ffmpeg -i input output`
```

会把日志信息输出到环境变量定义的文件中，内容包括简要过程信息，警告和错误。

- `-hide_banner` : 禁止打印输出banner。所有FFmpeg工具使用中常规都会在前面显示一些版权通知、编译选项和库版本等，这个选项可以禁止这部分的显示。
- `cpuflags flags(global)` : 允许设置或者清除cpu标志性和。当前这个选项主要还是测试特性，不要使用，除非你明确需要：

```
ffmpeg -cpuflags -sse+mmx ... ffmpeg -cpuflags mmx ... ffmpeg -cpuflags 0 ...
```

可能的选项参数有：

- x86
 - mmx
 - mmxext
 - sse
 - sse2
 - sse2slow
 - sse3
 - 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 Processors
 - pentium2
 - pentium3
 - pentium4
 - k6
 - athlon
 - athlonxp
 - k8
- `-openc1_bench` : 输出所有有效OpenCL设备的基准测试情况。当前选项仅在编译FFmpeg中打开了 `--enable-openc1` 才有效。

当FFmpeg指定了 `--enable-openc1` 编译后，这个选项还可以通过全局参数 `-openc1_options` 进行设定，参考OpenCL选项，在ffmpeg-utils手册中对于选项的支持情况，这包括在特定的平台设备上支持OpenCL的能力。默认，FFmpeg会运行在首选平台的首选设备上，通过设置全局的OpenCL则可以在选定的OpenCL设备上运行，这样就可以在更快的OpenCL设备上运行（平时节点，需要时才选用性能高但耗电的设备）

这个选项有助于帮助用户了解信息以进行有效配置。它将在每个设备上运行基准测试，并以性能排序所有设备，用户可以在随后调用 `ffmpeg` 时使用 `-openc1_options` 配置合适的OpenCL加速特性。

一般以下面的步骤使用这个参数：

```
ffmpeg -openc1_bench
```

注意输出中第一行的平台ID (`pidx`) 和设备ID (`didx`)，然后在选择平台和设备用于命令行：

```
ffmpeg -openc1_options platform_idx=pidx:device_idx=didx ...
```

- `openc1_options options(global)` :设置OpenCL环境选项，这个选项仅仅在FFmpeg编译选项中打开了 `--enable-openc1` 才有效。

`options`必须是一个由 `:` 分隔的 `key=value` 键值对列表。参考OpenCL选项，在ffmpeg-utils手册中对于选项的支持情况

AV选项

这些选项由特定的库提供（如libavformat，libavdevice以及libavcodec）。为了更多的了解AV选项，使用 `-help` 进行进一步了解。它们可以指定下面2个分类：

- **generic（常规）**：这类选项可以用于设置 容器、设备、编码器、解码器等。通用选项对列在 `AVFormatContext` 中的容器/设备以及 `AVCodecContext` 中的编码有效。
- **private（私有）**：这类仅对特定的容器、设备或者编码有效。私有选项由相应的 容器/设备/编码 指定（确定）。

例如要在一个默认为ID3v2.4为头的MP3文件中写入ID3v2.3头，需要使用`id3v2_version` 私有选项来对MP3混流：

```
ffmpeg -i input.flac -id3v2_version 3 out.mp3
```

所有编码AV选项是针对单独流的，所以必须详细指定。

注意 `-nooption` 语法不能被用于AV选项中的布尔值项目，而必须使用 `-option 0/-option 1`

注意以往使用 `v/a/s` 命名指定每个流的AV选项语法已经不建议使用，它们很快就会失效移除。

主要选项

- `-f fmt (input/output)` :指定输入或者输出文件格式。常规可省略而使用依据扩展名的自动指定，但一些选项需要强制明确设定。
- `-i filename (input)` :指定输入文件
- `-y (global)` :默认自动覆盖输出文件，而不再询问确认。
- `-n (global)` :不覆盖输出文件，如果输出文件已经存在则立即退出
- `c[:stream_specifier] codec (input/output,per-stream)`
- `-codec[:stream_specifier] codec (input/output,per-stream)` 为特定的文件选择编/解码模式，对于输出文件就是编码器，对于输入或者某个流就是解码器。选项参数中 `codec` 是编解码器的名字，或者是 `copy`（仅对输出文件）则意味着流数据直接复制而不再编码。例如：`> ffmpeg -i INPUT -map 0 -c:v libx264 -c:a copy OUTPUT`

是使用libx264编码所有的视频流，然后复制所有的音频流。

再如除了特殊设置外所有的流都由 `c` 匹配指定：`> ffmpeg -i INPUT -map 0 -c copy -c:v:1 libx264 -c:a:137 libvorbis OUTPUT`

这将在输出文件中第2视频流按libx264编码，第138音频流按libvorbis编码，其余都直接复制输出。

- `-t duration (input/output)` :限制输入/输出的时间。如果是在 `-i` 前面，就是限定从输入中读取多少时间的数据；如果是用于限定输出文件，则表示写入多少时间数据后就停止。`duration` 可以是以秒为单位的数值或者 `hh:mm:ss[.xxx]` 格式的时间值。注意 `-to` 和 `-t` 是互斥的，`-t` 有更高优先级。
- `-to position (output)` :只写入 `position` 时间后就停止，`position` 可以是以秒为单位的数值或者 `hh:mm:ss[.xxx]` 格式的时间值。注意 `-to` 和 `-t` 是互斥的，`-t` 有更高优先级。
- `-fs limit_size (output)` :设置输出文件大小限制，单位是字节(bytes)。
- `-ss position (input/output)` :
 - 当在 `-i` 前，表示定位输入文件到 `position` 指定的位置。注意可能一些格式是不支持精确定位的，所以 `ffmpeg` 可能是定位到最接近 `position`（在之前）的可定位点。当有转码发生且 `-accurate_seek` 被设置为启用（默认），则实际定位点到 `position` 间的数据被解码出来但丢弃掉。如果是复制模式或者 `-noaccurate_seek` 被使用，则这之间的数据会被保留。
 - 当用于输出文件时，会解码丢弃 `position` 对应时间码前的输入文件数据。
 - `position` 可以是以秒为单位的数值或者 `hh:mm:ss[.xxx]` 格式的时间值
- `-itsoffset offset (input)` :设置输入文件的时间偏移。`offset` 必须采用时间持续的方式指定，即可以有 `-` 号的时间值（以秒为单位的数值或者 `hh:mm:ss[.xxx]` 格式的时间值）。偏移会附加到输入文件的时间码上，意味着所指定的流会以时间码+偏移量作为最终输出时间码。
- `-timestamp date (output)` :设置在容器中记录时间戳。`date` 必须是一个时间持续描述格式，即 `YYYY-MM-DD|YYYYMMDD|[T|t] |(HHMMSS[m...]))[Z]` 格式。
- `-metadata[:metadata_specifier] key=value (output,per-metadata)` :指定元数据中的键值对。

流或者章的 `metadata_specifier` 可能值是可以参考文档中 `-map_metadata` 部分了解。

简单的覆盖 `-map_metadata` 可以通过一个为空的选项实现，例如：

```
ffmpeg -i in.avi -metadata title="my title" out.flv
```

设置第1声道语言：

```
ffmpeg -i INPUT -metadata:s:a:0 language=eng OUTPUT
```

- `-target type (output)` :指定目标文件类型(vcd,svcd,dvd,dv,dv50)，类型还可以前缀一个 `pal-`、`ntsc-` 或者 `film-` 来设定更具体的标准。所有的格式选项(码率、编码、缓冲尺寸)都会自动设置，而你仅仅只需要设置目标类型：

```
ffmpeg -i myfile.avi -target vcd /tmp/vcd.mpg
```

当然，你也可以指定一些额外的选项，只要你知道这些不会与标准冲突，如：

```
ffmpeg -i myfile.avi -target vcd -bf 2 /tmp/vcd.mpg
```

- `-dframes number (output)` : 设定指定 `number` 数据帧到输出文件, 这是 `-frames:d` 的别名。
- `frames[:stream_specifier] framecount (output,per-stream)` : 在指定计数帧后停止写入数据。
- `-q[:stream_specifier] q (output,per-stream)`
- `-qscale[:stream_specifier] q (output,per-stream)`

使用固定的质量品质(VBR)。用于指定 `q|qscale` 编码依赖。如果 `qscale` 没有跟 `stream_specifier` 则只适用于视频。其中值 `q` 取值在0.01-255,越小质量越好。

- `-filter[:stream_specifier] filtergraph (output,per-stream)` : 创建一个由 `filtergraph` 指定的滤镜, 并应用于指定流。

`filtergraph` 是应用于流的滤镜链图, 它必须有一个输入和输出, 而且流的类型需要相同。在滤镜链图中, 从 `in` 标签指定出输入, 从 `out` 标签出输出。要了解更多语法, 请参考 `ffmpeg-filters` 手册。

参考 `-filter_complex` 选项以了解如何建立多个输入/输出的滤镜链图。

- `-filter_script[:stream_specifier] filename (output, per-stream)` : 这个选项类似于 `-filter`, 只是这里的参数是一个文件名, 它的内容将被读取用于构建滤镜链图。
- `-pre[:stream_specifier] preset_name (output, per-stream)` : 指定预设名字的流 (单个或者多个)。
- `-stats (global)` : 输出编码过程/统计, 这是系统默认值, 如果你想禁止, 则需要采用 `-nostats`。
- `-progress url (global)` : 发送友好的处理过程信息到 `url`。处理过程信息是一种键值对 (`key=value`) 序列信息, 它每秒都输出, 或者在一次编码结束时输出。信息中最后的一个键值对表明了当前处理进度。
- `-stdin` : 允许标准输入作为交互。在默认情况下除非标准输入作为真正的输入。要禁用标准输入交互, 则你需要显式的使用 `-nostdin` 进行设置。禁用标准输入作为交互作用是有用的, 例如FFmpeg是后台进程组, 它需要一些相同的从shell开始的调用 (`ffmpeg ... </dev/null`)。
- `-debug_ts (global)` : 打印时间码信息, 默认是禁止的。这个选项对于测试或者调试是非常有用的特性, 或者用于从一种格式切换到另外的格式 (包括特性) 的时间成本分析, 所以不用于脚本处理中。还可以参考 `-fdebug ts` 选项。
- `-attach filename (output)` : 把一个文件附加到输出文件中。这里只有很少文件类型能被支持, 例如使用Matroska技术为了渲染字幕的字体文件。附件作为一种特殊的流类型, 所以这个选项会添加一个流到文件中, 然后你就可以像操作其他流一样使用每种流选项。在应用本选项时, 附件流须作为最后一个流(例如根据 `-map` 映射流或者自动映射时需要注意)。注意对于 Matroska 你也可以在元数据标签中进行类型设定: `> ffmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv`

(这时要访问到附件流, 则就是访问输出文件中的第3个流)

- `-dump_attachment[:stream_specifier] filename (input,per-stream)` : 从输入文件中解出指定的附件流到文件`filename` :
`> ffmpeg -dump_attachment:t:0 out.ttf -i INPUT`

如果想一次性把所有附件都解出来, 则 `> ffmpeg -dump_attachment:t:"" -i INPUT`

技术说明: 附件流是作为编码扩展数据来工作的, 所以其他流数据也能展开, 而不仅仅是这个附件属性。

- `-noautorotate` : 禁止自动依据文件元数据旋转视频。

视频 (video) 选项

- `-vframes number (output)` : 设置输出文件的帧数, 是 `-frames:v` 的别名。
- `-r[:stream_specifier] fps (input/output,per-stream)` : 设置帧率 (一种Hz值, 缩写或者分数值)。

在作为输入选项时, 会忽略文件中存储的时间戳和时间戳而产生的假设恒定帧率 `fps`, 即强制按设定帧率处理视频产生 (快进/减缓效果)。这不像 `-framerate` 选项是用来让一些输入文件格式如image2或者v412(兼容旧版本的FFmpeg)等, 要注意这一点区别, 而不要造成混淆。

作为输出选项时，会复制或者丢弃输入中个别的帧以满足设定达到 `fps` 要求的帧率。

- `-s[:stream_specifier] size (input/output,per-stream)`：设置帧的尺寸。

当作为输入选项时，是私有选项 `video_size` 的缩写，一些文件没有把帧尺寸进行存储，或者设备对帧尺寸是可以设置的，例如一些采集卡或者raw视频数据。

当作为输出选项是，则相当于 `scale` 滤镜作用在滤镜链图的最后。请使用 `scale` 滤镜插入到开始或者其他地方。

数据的格式是 `wxh`，即 宽度值X高度值，例如 `320x240`，（默认同源尺寸）

- `aspect[:stream_specifier] aspect (output,per-stream)`：指定视频的纵横比（长宽显示比例）。`aspect` 是一个浮点数字符串或者 `num:den` 格式字符串(其值就是num/den)，例如"4:3","16:9","1.3333"以及"1.7777"都是常用参数值。

如果还同时使用了 `-vcodec copy` 选项，它将只影响容器级的长宽比，而不是存储在编码中的帧纵横比。

- `-vn (output)`：禁止输出视频
- `-vcodec codec (output)`：设置视频编码器，这是 `-codec:v` 的一个别名。
- `-pass[:stream_specifier] n (output,per-stream)`：选择当前编码数(1或者2)，它通常用于2次视频编码的场景。第一次编码通常把分析统计数据记录到1个日志文件中（参考 `-passlogfile` 选项），然后在第二次编码时读取分析以精确要求码率。在第一次编码时通常可以禁止音频，并且把输出文件设置为 `null`，在windows和类unix分别是：

```
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y NUL ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y /dev/null
```

- `-passlogfile[:stream_specifier] prefix (output,per-stream)`：设置2次编码模式下日志文件存储文件前导，默认是"ffmpeg2pass"，则完整的文件名就是"PREFIX-N.log"，其中的N是指定的输出流序号（对多流输出情况）
- `-vf filtergraph (output)`：创建一个 `filtergraph` 的滤镜链并作用在流上。它实为 `-filter:v` 的别名，详细参考 `-filter` 选项。

高级视频选项

- `-pix_fmt[:stream_specifier] format (input/output,per-stream)`：设置像素格式。使用 `-pix_fmts` 可以显示所有支持的像素格式。如果设置的像素格式不能被选中（启用），则ffmpeg会输出一个警告和并选择这个编码最好（兼容）的像素格式。如果 `pix_fmt` 前面前导了一个 `+` 字符，ffmpeg会在要求的像素格式不被支持时退出，这也意味着滤镜中的自动转换也会被禁止。如果 `pix_fmt` 是单独的 `+`，则ffmpeg选择和输入（或者滤镜通道）一样的像素格式作为输出，这时自动转换也会被禁止。
- `-sws_flags flags (input/output)`：选择 SwScaler 放缩标志量。
- `-vdt n`：丢弃的门限设置。
- `-rc_override[:stream_specifier] override (output,per-stream)`：在特定时间范围内的间隔覆盖率，`override` 的格式是"int1int1int"。其中前两个数字是开始帧和结束帧，最后一个数字如果为正则量化模式，如果为负则是品质因素。
- `-ilme`：支持交错编码（仅MPEG-2和MPEG-4）。如果你的输入是交错的，而且你想保持交错格式，又想减少质量损失，则选此项。另一种方法是采用 `-deinterlace` 对输入流进行分离，但会引入更多的质量损失。
- `-psnr`：计算压缩帧的 PSNR
- `-vstats`：复制视频编码统计分析到日志文件 `vstats_HHMMSS.log`
- `-vstats_file file`：复制视频编码统计分析到 `file` 所指的日志文件中。
- `-top[:stream_specifier] n (output,per-stream)`：指明视频帧数据描述的起点。`顶部=1/底部=0/自动=-1`（以往CRT电视扫描线模式）
- `-dc precision`：Intra_dc_precision值。
- `-vtag fourcc/tag (output)`：是 `-tag:v` 的别名，强制指定视频标签/fourCC（FourCC全称Four-Character Codes，代表四字符代码 (four character code)，它是一个32位的标示符，其实就是typedef unsigned int FOURCC;是一种独立标示视频数据流格式的四字符代码。）
- `-qphist (global)`：显示 QP 直方图。

- `-vbsf bitstream_filter` : 参考 `-bsf` 以进一步了解。
- `-force_key_frames[:stream_specifier] time[,time...]` (output,per-stream) : (见下)
- `-force_key_frames[:stream_specifier] expr:expr` (output,per-stream) : 强制时间戳位置帧为关键帧, 更确切说是从第一帧起每设置时间都是关键帧 (即强制关键帧率)。

如果参数值是以 `expr`: 前导的, 则字符串 `expr` 为一个表达式用于计算关键帧间隔数。关键帧间隔值必须是一个非零数值。

如果一个时间值是 " chapters [delta]" 则表示文件中从 `delta` 章开始的所有章节点计算以秒为单位的时间, 并把该时间所指帧强制为关键帧。这个选项常用于确保输出文件中所有章标记点或者其他点所指帧都是关键帧 (这样可以方便定位)。例如下面的选项代码就可以使 "第5分钟以及章节 chapters-0.1 开始的所有标记点都成为关键帧" :

```
-force_key_frames 0:05:00,chapters-0.1
```

其中表达式 `expr` 接受如下的内容 :

- `n` : 当前帧序数, 从0开始计数
- `n_forced` : 强制关键帧数
- `prev_forced_n` : 之前强制关键帧数, 如果之前还没有强制关键帧, 则其值为 `NAN`
- `prev_forced_t` : 之前强制关键帧时间, 如果之前还没有强制关键帧则为 `NAN`
- `t` : 当前处理到的帧对应时间。

例如要强制每5秒一个关键帧 :

```
-force_key_frames expr:gte(t,n_forced*5)
```

从13秒后每5秒一个关键帧 :

```
-force_key_frames expr:if(isnan(prev_forced_t),gte(t,13),gte(t,prev_forced_t+5))
```

注意设置太多强制关键帧会损害编码器前瞻算法效率, 采用固定 `GOP` 选项或采用一些近似设置可能更高效。

- `-copyinkf[:stream_specifier]` (output,per-stream) : 流复制时同时复制非关键帧。
- `-hwaccel[:stream_specifier] hwaccel` (input,per-stream) : 使用硬件加速解码匹配的流。允许的 `hwaccel` 值为 :
 - `none` : 没有硬件加速 (默认值)
 - `auto` : 自动选择硬件加速
 - `vda` : 使用Apple的VDA硬件加速
 - `vdpa` : 使用VDPAA (Video Decode and Presentation API for Unix, 类unix下的技术标准) 硬件加速
 - `dxva2` : 使用DXVA2 (DirectX Video Acceleration, windows下的技术标准) 硬件加速。

这个选项可能并不能起效果 (它依赖于硬件设备支持和选择的解码器支持)

注意 : 很多加速方法 (设备) 现在并不比现代CPU快了, 而且额外的 `ffmpeg` 需要拷贝解码的帧 (从GPU内存到系统内存) 完成后续处理 (例如写入文件), 从而造成进一步的性能损失。所以当前这个选项更多的用于测试。

- `-hwaccel_device[:stream_specifier] hwaccel_device` (input,per-stream) : 选择一个设备用于硬件解码加速。这个选项必须同时指定了 `-hwaccel` 才可能生效。它也依赖于指定的设备对于特定编码的解码加速支持性能。
 - `vdpa` : 对应于 `VDPAA`, 在 `x11` (类Unix) 显示/屏幕上的, 如果这个选项值没有选中, 则必须在 `DISPLAY` 环境变量中有设置。
 - `dxva2` : 对应于 `DXVA2`, 这个是显示硬件 (卡) 的设备号, 如果没有指明, 则采用默认设备 (对于多个卡时)。

音频选项

- `-aframes number (output)` : 设置 `number` 音频帧输出, 是 `-frames:a` 的别名
- `-ar[:stream_specifier] freq (input/output,per-stream)`: 设置音频采样率。默认是输出同于输入。对于输入进行设置, 仅仅通道是真实的设备或者raw数据分离出并映射的通道才有效。对于输出则可以强制设置音频量化的采用率。
- `-aq q (output)` : 设置音频品质(编码指定为VBR), 它是 `-q:a` 的别名。
- `-ac[:stream_specifier] channels (input/output,per-stream)` : 设置音频通道数。默认输出会有输入相同的音频通道。对于输入进行设置, 仅仅通道是真实的设备或者raw数据分离出并映射的通道才有效。
- `-an (output)` : 禁止输出音频
- `-acodec codec (input/output)` : 设置音频解码/编码的编/解码器, 是 `-codec:a` 的别名
- `-sample_fmt[:stream_specifier] sample_fmt (output,per-stream)`: 设置音频样例格式。使用 `-sample_fmts` 可以获取所有支持的样例格式。
- `-af filtergraph (output)` : 对音频使用 `filtergraph` 滤镜效果, 其是 `-filter:a` 的别名, 参考 `-filter` 选项。

高级音频选项

- `-atag fourcc/tag (output)` : 强制音频标签/fourcc。这个是 `-tag:a` 的别名。
- `-absf bitstream_filter` : 要深入了解参考 `-bsf`
- `-guess_layout_max channels (input,per-stream)` : 如果音频输入通道的布局不确定, 则尝试猜测选择一个能包括所有指定通道的布局。例如: 通道数是2, 则 `ffmpeg` 可以认为是2个单声道, 或者1个立体声声道而不会认为是6通道或者5.1通道模式。默认值是总是试图猜测一个包含所有通道的布局, 用0来禁用。

字幕选项

- `-scodec codec (input/output)` : 设置字幕解码器, 是 `-codec:s` 的别名。
- `-sn (output)` : 禁止输出字幕
- `-sbsf bitstream_filter` : 深入了解请参考 `-bsf`

高级字幕选项

- `-fix_sub_duration` : 修正字幕持续时间。对每个字幕根据接下来的数据包调整字幕流的时间常数以防止相互覆盖（第一个没有完下一个就出来了）。这对很多字幕解码来说是必须的, 特别是DVB字幕, 因为它在原始数据包中只记录了一个粗略的估计值, 最后还以一个空的字幕帧结束。

这个选项可能失败, 或者出现夸张的持续时间或者合成失败, 这是因为数据中有非单调递增的时间戳。

注意此选项将导致所有数据延迟输出到字幕解码器, 它会增加内存消耗, 并引起大量延迟。

- `-canvas_size size` : 设置字幕渲染区域的尺寸 (位置)

高级选项

- `-map [-]input_file_id[:stream_specifier][,sync_file_id[:stream_specifier]] | [linklabel] (output)` : 设定一个或者多个输入流作为输出流的源。每个输入流是以 `input_file_id` 序号标记的输入文件和 `input_stream_id` 标记的流序号共同作用指明, 它们都以0起始计数。如果设置了 `sync_file_id:stream_specifier`, 则把这个输入流作为同步信号参考。

命令行中的第一个 `-map` 选项指定了输出文件中第一个流的映射规则 (编号为0的流, 0号流), 第二个则指定1号流的, 以此类推。

如果在流限定符前面有一个 `-` 标记则表明创建一个“负”映射, 这意味着禁止该流输出, 及排除该流。

一种替代的形式是在复合滤镜中利用 `[linklabel]` 来进行映射 (参看 `-filter_complex` 选项)。其中的 `linklabel` 必须是输出滤镜链图中已命名的标签。

例子: 映射第一个输入文件的所有流到输出文件:

```
ffmpeg -i INPUT -map 0 output
```

又如，如果在输入文件中有两路音频流，则这些流的标签就是"0:0"和"0:1"，你可以使用 `-map` 来选择某个输出，例如：
`ffmpeg -i INPUT -map 0:1 out.wav`

这 will 只把输入文件中流标签为"0:1"的音频流单独输出到out.wav中。

再如，从文件a.mov中选择序号为2的流（流标签0:2），以及从b.mov中选择序号为6的流(流标签1:6)，然后共同复制输出到out.mov需要如下写: `ffmpeg -i a.mov -i b.mov -c copy -map 0:2 -map 1:6 out.mov`

选择所有的视频和第三个音频流则是: `ffmpeg -i INPUT -map 0:v -map:a:2 OUTPUT`

选择所有的流除了第二音频流外的流进行输出是：`ffmpeg -i INPUT -map 0 -map -0:a:1 OUTPUT`

选择输出英语音频流: `ffmpeg -i INPUT -map 0:m:language:eng OUTPUT`

注意应用了该选项将自动禁用默认的映射。

- `-ignore_unknown`：如果流的类型未知则忽略，而不进行复制。
- `-copy_unknown`：复制类型未知的流。
- `-map_channel [input_file_id.stream_specifier.channel_id|-1][:output_file_id.stream_specifier]` :从输入文件中指定映射一个通道的音频到输出文件指定流。如果 `output_file_id.stream_specifier` 没有设置，则音频通道将映射到输出文件的所有音频流中。

使用 `-1` 插入到 `input_file_id.stream_specifier.chnnel_id` 会映射一个静音通道

例如 `INPUT` 是一个立体声音频文件，你可以分别选择两个音频通道(下面实际上对于输入是交换了2个音频通道顺序进行输出)：`ffmpeg -i INPUT -map_channel 0.0.1 -map_channel 0.0.0 OUTPUT`

如果你想静音第一个通道，而只保留第二通道，则可使用: `ffmpeg -i INPUT -map_channel -1 -map_channel 0.0.1 OUTPUT`

以 `-map_channel` 选项指定的顺序在输出文件中输出音频流通道布局，即第一个 `-map_channel` 对应输出中第一个音频流通道，第二个对应第二个音频流通道，以此类推（只有一个则是单声道，2个是立体声）。联合使用 `-ac` 与 `-map_channel`，而且在输入的 `-map_channel` 与 `-ac` 不匹配（例如只有2个 `-map_channel`，又设置了 `-ac 6`）时将使指定音频流通道提高增益。

你可以详细的对每个输入通道指派输出以分离整个输入文件，例如下面就把有 `INPUT` 文件中的两个音频分别输出到两个输出文件中(`OUTPUT_CH0` 和 `OUTPUT_CH1`)：`ffmpeg -i INPUT -map_channel 0.0.0 OUTPUT_CH0 -map_channel 0.0.1 OUTPUT_CH1`

下面的例子则把一个立体声音频的两个音频通道分离输出到两个相互独立的流（相当于两个单声道了）中（但还是放在同一个输出文件中）：`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`

注意当前一个输出流仅能与一个输入通道连接，既你不能实现利用 `-map_channel` 把多个输入的音频通道整合到不同的流中（从同一个文件或者不同文件）或者是混合它们成为单独的流，例如整合2个单声道形成立体声是不可能的。但是分离一个立体声成为2个独立的单声道是可行的。

如果你需要类似的应用，你需要使用 `amerge` 滤镜，例如你需要整合一个媒体（这里是input.mkv）中的2个单声道成为一个立体声通道(保持视频流不变)，你需要采用下面的命令: `ffmpeg -i input.mkv -filter_complex "[0:1] [0:2] amerge" -c:a pcm_s16le -c:v copy output.mkv`

- `-map_metadata[:metadata_spec_out] infile[:metadata_spec_in] (output,per-metadata)`：在下一个输出文件中从 `infile` 读取输出元数据信息。注意这里的文件索引也是以0开始计数的，而不是文件名。参数 `metadata_spec_in/out` 指定的元数据将被复制，一个元数据描述可以有如下的信息块：

- `g`:全局元数据, 这些元数据将作用于整个文件
- `s[:stream_spec]`:每个流的元数据, `stream_spec` 的介绍在 流指定 章节。如果是描述输入流, 则第一个被匹配的流相关内容被复制, 如果是输出元数据指定, 则所有匹配的流相关信息被复制到该处。
- `c:chapter_index`:每个章节的元数据, `chapter_index` 也是以0开始的章节索引。
- `p:program_index`: 每个节目元数据, `program_index` 是以0开始的节目索引

如果元数据指定被省略, 则默认是全局的。

默认全局元数据会从第一个输入文件每个流每个章节依次复制(流/章节), 这种默认映射会因为显式创建了任意的映射而失效。一个负的文件索引就可以禁用默认的自动复制。

例如从输入文件的第一个流复制一些元数据作为输出的全局元数据 > `ffmpeg -i in.ogg -map_metadata 0:s:0 out.mp3`

与上相反的操作, 例如复制全局元数据给所有的音频流 > `ffmpeg -i in.mkv -map_metadata:s:a 0:g out.mkv`

注意这里简单的 `o` 在这里能正常工作是因为全局元数据是默认访问的。

- `-map_chapters input_file_index (output)`:从输入文件中复制由 `input_file_index` 指定的章节的内容到输出。如果没有匹配的章节, 则复制第一个输入文件至少一章内容(第一章)。使用负数索引则禁用所有的复制。
- `-benchmark (global)`:在编码结束后显示基准信息。则包括CPU使用时间和最大内存消耗, 最大内存消耗是不一定在所有的系统中被支持, 它通常以显示为0表示不支持。
- `-benchmark_all (global)`:在编码过程中持续显示基准信息, 则包括CPU使用时间(音频/视频的编/解码)
- `-timelimit duration (global)`:ffmpeg在编码处理了 `duration` 秒后退出。
- `-dump (global)`:复制每个输入包到标准输出设备
- `-hex (global)`:复制包时也复制荷载信息
- `-re (input)`:以指定帧率读取输入。通常用于模拟一个硬件设备, 例如在直播输入流(这时是读取一个文件)。不应该在实际设备或者在直播输入中使用(因为这将导致数据包的丢弃)。默认 `ffmpeg` 会尽量以最高可能的帧率读取。这个选项可以降低从输入读取的帧率, 这常用于实时输出(例如直播流)。
- `-loop_input`:循环输入流。当前它仅作用于图片流。这个选项主要用于FFserver自动化测试。这个选项现在过时了, 应该使用 `-loop 1`。
- `-loop_output number_of_times`:重复播放 `number_of_times` 次。这是对于GIF类型的动画(0表示持续重复而不停止)。这是一个过时的选项, 用 `-loop` 替代。
- `-vsync parameter`:视频同步方式。为了兼容旧, 常被设置为一个数字值。也可以接受字符串来作为描述参数值, 其中可能的值是:

- `0, passthrough`:每个帧都通过时间戳来同步(从解复用到混合)。
- `1, cfr`:帧将复制或者降速以精准达到所要求的恒定帧速率。
- `2, vfr`:个别帧通过他们的时间戳或者降速以防止2帧具有相同的时间戳
- `drop`:直接丢弃所有的的时间戳, 而是在混合器中基于设定的帧率产生新的时间戳。
- `-1, auto`:根据混合器功能在1或者2中选择, 这是默认值。

注意时间戳可以通过混合器进一步修改。例如 `avoid_negative_ts` 被设置时。

利用 `-map` 你可以选择一个流的时间戳作为凭据, 它可以对任何视频或者音频 不改变或者重新同步持续流到这个凭据。

- `-frame_drop_threshold parameter`:丢帧的阈值, 它指定后面多少帧内可能有丢帧。在帧率计数时1.0是1帧, 默认值是1.1。一个可能的用例是避免在混杂的时间戳或者需要增加精准时间戳的情况下确立丢帧率。
- `-async samples_per_second`:音频同步方式。"拉伸/压缩"音频以匹配时间戳。参数是每秒最大可能的音频改变样本。`-async 1`是一种特殊情况指只有开始时校正, 后续不再校正。

注意时间戳还可以进一步被混合器修改。例如 `avoid_negative_ts` 选项被指定时

已不推荐这个选项, 而是用 `aresample` 音频滤波器代替。

- `-copyts` : 不处理输入的时间戳, 保持它们而不是尝试审核。特别是不会消除启动时间偏移值。

注意根据 `vsync` 同步选项或者特定的混合器处理流程 (例如格式选项 `avoid_negative_ts` 被设置) 输出时间戳会忽略匹配输入时间戳 (即使这个选项被设置)

- `-start_at_zero` : 当使用 `-copyts` ,位移输入时间戳作为开始时间0.这意味着使用该选项, 同时又设置了 `-ss` , 例如 `-ss 50` 则输出中会从50秒开始加入输入文件时间戳。
- `-copytb mode` : 指定当流复制时如何设置编码时间基准。 `mode` 参数是一个整数值, 可以有如下可能:
 - `1` 表示使用分离器时间基准, 从分离器中复制时间戳到编码中。复制可变帧率视频流时需要避免非单调递增的时间戳。
 - `0` 表示使用解码器时间基准, 使用解码器中获取的时间戳作为输出编码基准。
 - `-1` 尝试自动选择, 只要能产生一个正常的输出, 这是默认值。
- `-shortest (output)` : 完成编码时最短输入端。
- `-dts_delta_threshold` : 时间不连续增量阈值。
- `-muxdelay seconds (input)` : 设置最大 解复用-解码 延迟。参数是秒数值。
- `-maxpreload seconds (input)` : 设置初始的 解复用-解码延迟, 参数是秒数值。
- `-streamid output-stream-index:new-value (output)` : 强制把输出文件中序号为 `output-stream-id` 的流命名为 `new-value` 的值。这对应于这样的场景: 在存在了多输出文件时需要把一个流分配给不同的值。例如设置0号流为33号流, 1号流为36号流到一个mpegs格式输出文件中 (这相当于对流建立链接/别名) :

```
ffmpeg -i infile -streamid 0:33 -streamid 1:36 out.ts
```

- `-bsf[:stream_specifier] bitstream_filters (output,per-stream)` : 为每个匹配流设置bit流滤镜。 `bitstream_filters` 是一个逗号分隔的bit流滤镜列表。可以使用 `-bsfs` 来获得当前可用的bit流滤镜。

```
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
```

- `-tag[:stream_specifier] codec_tag (input/output,per-stream)` : 为匹配的流设置标签/fourcc。
- `-timecode hh:mm:ssSEDFf` : 指定时间码, 这里 `SEP` 如果是 `:` 则不减少时间码, 如果是 `;` 或者 `.` 则可减少。

```
ffmpeg -i input.mpg -timecode 01:02:03.04 -r 30000/1001 -s ntsc output.mpg
```

- `-filter_complex filtergraph (global)` : 定义一个复合滤镜, 可以有任意数量的输入/输出。最简单的滤镜链图至少有一个输入和一个输出, 且需要相同类型。参考 `-filter` 以获取更多信息 (更有价值)。 `filtergraph` 用来指定一个滤镜链图。关于 滤镜链图的语法 可以参考 `ffmpeg-filters` 相关章节。

其中输入链标签必须对应于一个输入流。 `filtergraph` 的具体描述可以使用 `file_index:stream_specifier` 语法 (事实上这同于 `-map`)。如果 `stream_specifier` 匹配到了一个多输出流, 则第一个被使用。滤镜链图中一个未命名输入将匹配链接到的输入中第一个未使用且类型匹配的流。

使用 `-map` 来把输出链接到指定位置上。未标记的输出会添加到第一个输出文件。

注意这个选项参数在用于 `-lavfi` 源时不是普通的输入文件。 > `ffmpeg -i video.mkv -i image.png -filter_complex '[0:v][1:v]overlay[out]' -map '[out]' out.mkv`

这里 `[0:v]` 是第一个输入文件的第一个视频流, 它作为滤镜的第一个 (主要的) 输入, 同样, 第二个输入文件的第一个视频流作为滤镜的第二个输入。

假如每个输入文件只有一个视频流, 则我们可以省略流选择标签, 所以上面的内容在这时等价于:

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay[out]' -map '[out]' out.mkv
```

此外, 在滤镜是单输出时我们还可以进一步省略输出标签, 它会自动添加到输出文件, 所以进一步简写为:

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay' out.mkv
```

利用 `lavfi` 生成5秒的 红 color （色块）：

```
ffmpeg -filter_complex 'color=c=red' -t 5 out.mkv
```

- `-lavfi filtergraph (global)`：定义一个复合滤镜，至少有一个输入和/或输出，等效于 `-filter_complex`。
- `-filter_complex_script filename (global)`：这个选项类似于 `-filter_complex`，唯一不同就是它的参数是文件名，会从这个文件中读取复合滤镜的定义。
- `-accurate_seek (input)`：这个选项会启用/禁止输入文件的精确定位（配合 `-ss`），它默认是启用的，即可以精确定位。需要时可以使用 `-noaccurate_seek` 来禁用，例如在复制一些流而转码另一些的场景下。
- `-seek_timestamp (input)`：这个选项配合 `-ss` 参数可以在输入文件上启用或者禁止利用时间戳的定位。默认是禁止的，如果启用，则认为 `-ss` 选项参数是正式的时间戳，而不是由文件开始计算出来的偏移。这一般用于具有不是从0开始时间戳的文件，例如一些传输流（直播下）。
- `-thread_queue_size size (input)`：这个选项设置可以从文件或者设备读取的最大排队数据包数量。对于低延迟高速率的直播流，如果不能及时读取，则出现丢包，所以提高这个值可以避免出现大量丢包现象。
- `-override_ffserver (global)`：对 `ffserver` 的输入进行指定。使用这个选项 `ffmpeg` 可以把任意输入映射给 `ffserver` 并且同时控制很多编码可能。如果没有这个选项，则 `ffmpeg` 仅能根据 `ffserver` 所要求的数据进行传输。

这个选项应用场景是 `ffserver` 需要一些特性，但文件/设备不提供，这时可以利用 `ffmpeg` 作为中间处理环节控制后输出到 `ffserver` 到达所要求。

- `-sdp_file file (global)`：输出 `sdp` 信息到文件 `file`。它会在至少一个输出不是 `rtp` 流时同时输出 `sdp` 信息。
- `-discard (input)`：允许丢弃特定的流或者分离出的流上的部分帧，但不是所有的分离器都支持这个特性。
 - `none`：不丢帧
 - `default`：丢弃无效帧
 - `noref`：丢弃所有非参考帧
 - `bidir`：丢弃所有双向帧
 - `nokey`：丢弃所有非关键帧
 - `all`：丢弃所有帧
- `-xerror (global)`：在出错时停止并退出

作为一个特殊的例外，你可以把一个位图字幕（bitmap subtitle）流作为输入，它将转换作为同于文件最大尺寸的视频（如果没有视频则是720x576分辨率）。注意这仅仅是一个特殊的例外的临时解决方案，如果在 `libavfilter` 中字幕处理方案成熟后这样的处理方案将被移除。

例如需要为一个储存在DVB-T上的MPEG-TS格式硬编码字幕，而且字幕延迟1秒：> `ffmpeg -i input.ts -filter_complex \['#0x2ef' setpts=PTS+1/TB [sub] ; [#0x2d0] [sub] overlay' \ -sn -map '#0x2dc' output.mkv`

(0x2d0, 0x2dc 以及 0x2ef 是MPEG-TS的PIDs，分别指向视频、音频和字幕流，一般作为MPEG-TS中的0:0,0:3和0:7是实际流标签)

预设文件

一个预设文件是选项/值对的序列(option=value)，每行都是一个选项/值对，用于指定一系列的选项，而这些一般很难在命令行中指定（限于命令行的一些限制，例如长度限制）。以 `#` 开始的行是注释，会被忽略。一般 `ffmpeg` 会在目录树中检查 `presets` 子目录以获取预设文件。

有两种类型的预设文件：`ffpreset` 和 `avpreset`。

ffpreset类型预设文件

采用 `ffpreset` 类型预设文件主要包含 `vpre`、`apre`、`spre` 和 `fpre` 选项。其中 `fpre` 选项的参数可以代替预设的名称作为输入预设文件名，以用于任何一种编码格式。对于 `vpre`、`apre` 和 `spre` 选项参数会指定一个预设文件用于当前编码格式以替代

（作为）同类项的预订选项。

选用预设文件传递 `vpre`、`apre` 和 `spre` 的参数 `arg` 有下面一些搜索应用规则：

- 将在目录 `$FFMPEG_DATADIR` (如果设置了)和 `$HOME/.ffmpeg` 目录和配置文件中定义的数据目录（一般是 `PREFIX/share/ffmpeg`），以及 `ffpresets` 所在的执行文件目录下 `ffmpeg` 搜索对应的预定义文件 `arg.ffpreset`，例如参数是 `libvpx-1080p`，则对应于文件 `libvpx-1080p.ffpreset`
- 如果没有该文件，则进一步在前述目录下搜索 `codec_name-arg.ffpreset` 文件，如果找到即应用。例如选择了视频编码器 `-vcodec libvpx` 和 `-vpre 1080p` 则对应的预设文件名是 `libvpx-1080p.ffpreset`

avpreset 类型预设文件

`avpre` 类型预设文件以 `pre` 选项引入。他们工作方式类似于 `ffpreset` 类型预设文件（即也是选项值对序列），但只对于特定编码器选项，因此一些选项值对于不适合的编码器是无效的。根据 `pre` 的参数 `arg` 查找预设文件基于如下规则：

- 首先搜索 `$AVCONV_DATADIR` 所指目录(如果定义了)，其次搜索 `$HOME/.avconv` 目录，然后搜索执行文件所在目录(通常是 `PREFIX/share/ffmpeg`)，在其下查找 `arg.avpreset` 文件。第一个匹配的文件被应用。
- 如果查找不到，如果还同步还指定了编码（如 `-vcodec libvpx`）再以前面目录顺序，以 `codec_name-arg.avpreset` 再次查找文件。例如对于有选项 `-vcodec libvpx` 和 `-pre 1080p` 将搜索 `libvpx-1080p.avpreset`
- 如果还没有找到，将在当前目录下搜索 `arg.avpreset` 文件。

技巧/提示

- 如果流有非常低的码率，使用低帧率和小GOP尺寸。这对于RealVideo在Linux下面的播放显得不是特别快时特别有用，因为它可以跳过一些帧，例如：

```
ffmpeg -g 3 -r 3 -t 10 -b:v 50k -s qcif -f rv10 /tmp/b.rm
```

- 选项参数 `q` 将打开一个显示编码品质的水平数。值1表示非常高的品质，值31表示最差品质。如果`q=31`经常出现，则表明当前编码码率不足以高品质的压缩保存你的内容，你可能需要提高码率或降低帧率或减小帧尺寸。
- 如果你的计算机不够快，你可以额外设定来加速。你可以使用 `-me zero` 加速运动预测，使用 `-g 0` 则完全禁用运动预测（这对于I帧的JPEG压缩会有益处）
- 如果你需要非常低的音频码率，降低采样率（对于MPEG音频采样22050HZ，对于AC-3采用22050HZ或者11025HZ）
- 需要一个恒定的质量（但编码率可变），使用选项 `qscale n` 进行设定，`n` 取值为1(最好)-31(最差)。

6 例子

视频和音频抓取

如果你指定了输入格式和设备，ffmpeg可以直接抓取视频和音频：

```
ffmpeg -f oss -i /dev/dsp -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

或者采用ALSA音频源(单声道，卡的id是1)替代OSS:

```
ffmpeg -f alsa -ac 1 -i hw:1 -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

注意对于不同的视频采集卡，你必须正确激活视频源和通道，例如Gerd Knorr的 `xawtv`。你还需要设置正确的音频记录层次和混合模式。只有这样你才能采集到想要的视音频。

X11显示的抓取

可以通过ffmpeg直接抓取X11显示内容：

```
ffmpeg -f x11grab -video_size cif -framerate 25 -i :0.0+10, 20 /tmp/out.mpg
```

`0.0` 是X11服务的显示屏幕号(display.screen)，定义于 `DISPLAY` 环境变量。10是水平偏移，20是垂直偏移

视频和音频文件格式转换

任何支持的文件格式或者协议都可以作为ffmpeg输入。例如：

- 你可以使用YUV文件作为输入

```
ffmpeg -i /tmp/test%d.Y /tmp/out.mpg
```

这里可能是这样一些文件

```
/tmp/test0.Y, /tmp/test0.U, /tmp/test1.V, /tmp/test1.Y, /tmp/test1.U, /tmp/test1.V, etc...
```

这里Y还有对应分辨率的2个关联文件U和V。这是一种raw数据文件而没有文件头，它可以被所有的视频解码器生成。你必须利用 `-s` 对它指定一个尺寸而不是让ffmpeg去猜测。

- 你可以把raw YUV420P文件作为输入：

```
ffmpeg -i /tmp/test/yuv /tmp/out.avi
```

test.yuv 是一个包含raw YUV通道数据的文件。每个帧先是Y数据，然后是U和V数据。

- 也可以输出YUV420P类型的文件

```
ffmpeg -i mydivx.avi hugefile.yuv
```

- 可以设置一些输入文件和输出文件


```
ffmpeg -i /tmp/a.wav -s 640x480 -i /tmp/a.yuv /tmp/a.mpg
```

这将转换一个音频和raw的YUV视频到一个MPEG文件中

- 你也可以同时对音频或者视频进行转换

```
ffmpeg -i /tmp/a.wav -ar 22050 /tmp/a.mp2
```

这里把a.wav转换为MPEG音频，同时转换了采样率为22050HZ

- 你也可以利用映射同时编码多个格式作为输入或者输出：

```
ffmpeg -i /tmp/a.wav -map 0:a -b:a 64k /tmp/a.mp2 -map 0:a -b:a 128k /tmp/b.mp2
```

这将同时把a.wav以64k码率输出到a.mp2，以128k码率输出到b.mp2。"-map file:index"指定了对于每个输出是连接到那个输入流的。

- 还可以转换解码VOBs：

```
ffmpeg -i snatch_1.vob -f avi -c:v mpeg4 -b:v 800k -g 300 -bf 2 -c:a libmp3lame -b:a 128k snatch.avi
```

这是一个典型的DVD抓取例子。这里的输入是一个VOB文件，输出是MPEG-4编码视频以及MP3编码音频的AVI文件。注意在这个命令行里使用了B-frames（B帧）是兼容DivX5的，GOP设置为300则意味着有一个内帧是适合29.97fps的输入视频。此外，音频流采用MP3编码需要运行LAME支持，它需要通过在编译是设置 `--enable-libmp3lame`。这种转换设置在多语言DVD抓取转换出所需的语言音频时特别有用。

注意要了解支持那些格式，可以采用 `ffmpeg -formats`

- 可以从一个视频扩展生成图片（序列），或者从一些图片生成视频：

- 导出图片

```
ffmpeg -i foo.avi -r 1 -s WxH -f image2 foo-%03d.jpeg
```

这将每秒依据foo.avi生成一个图片命名为foo-001.jpeg,foo-002.jpeg以此类推,图片尺寸是WxH定义的值。

如果你想只生成有限数量的视频帧，你可以进一步结合 `-vframes` 或者 `-t` 或者 `-ss` 选项实现。

- 从图片生成视频

```
ffmpeg -f image2 -framerate 12 -i foo-%03d.jpeg -s WxH foo.avi
```

这里的语法 `foo-%03d.jpeg` 指明使用3位数字来补充完整文件名，不足3位以0补齐。这类似于C语言的printf函数中的格式，但只接受常规整数作为部分。

当导入一个图片序列时，`-i` 也支持shell的通配符模式(内置的)，这需要同时选择image2的特性选项 `-pattern_type glob`：例如下面就利用了所有匹配 `foo-*.jpeg` 的图片序列创建一个视频：

```
ffmpeg -f image2 -pattern_type glob -framerate 12 -i 'foo-*.jpeg' -s WxH foo.avi
```

- 你可以把很多相同类型的流一起放到一个输出中：

```
ffmpeg -i test1.avi -i test2.avi -map 1:1 -map 1:0 -map 0:1 -map 0:0 -c copy -y test12.nut
```

这里最后输出文件test12.nut包括了4个流，其中流的顺序完全根据前面 `-map` 的指定顺序。

- 强制为固定码率编码(CBR)输出视频：

```
ffmpeg -i myfile.avi -b 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
```

- 使用 `lambda` 工具的4个选项 `lmin` , `lmax` , `mb1min` 以及 `mb1max` 使你能更简单的从 `q` 转换到 `QP2LAMBDA` :

```
ffmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext
```

7 语法

这个章节介绍采用ffmpeg库和工具时的一些语法和格式要求。

引用与转义(Quoting and escaping)

ffmpeg采用如下的引用和转义机制，除非明确规定，以下规则都适用：

- `""` 和 `\` 分别用于（引用和转义）特殊字符。除了它们可能还有其它特殊字符，但这只在特定的语法中有效。
- 一个特殊字符必须有转义前缀 `\`
- 所有的引用字符串都由 `""` 封闭包含。引号 `""` 本身不能被引用，所以你可能需要关闭引用或者转义。
- 前导和尾随的空格字符除非专门由引号引用或者转义，否则都会在解析字符串时移除。

注意在使用命令行或者脚本时，你可能需要2级转义，这取决于你shell环境支持的语法。

声明在 `libavutil/avstring.h` 中的函数 `av_get_token` 被用于任务分析中的引用和转义处理。

ffmpeg源码中的工具 `tools/ffescape` 被用于自动处理引用和转义。

语法例子

- 转义 `Crime d'Amour` 中的 `'` 这个特殊字符:
`Crime d\'Amour`
- 下面的字符串因为是引用，所有其中的 `'` 需要特殊转义处理
`'Crime d\'\'Amour'`
- 包括前导和后随的空格必须要引用模式:
`' this string starts and ends with whitespaces '`
- 转义和引用可以接续在一起：
`' The string "\string\" is a string '`
- 为了包括一个 `\` 需要引用或者转义
`'c:\foo' can be written as c:\foo`

日期

接受如下的语法:

```
[(YYYY-MM-DD|YYYYMMDD)[T|t| ]|((HH:MM:SS[.m...])|(HHMMSS[.m...]))][Z]
now
```

如果值为 `now` 表示当前时间

时间是本地时间，除非 `z` 被附加到最后，它表示采用 `UTC` 时间。如果年-月-日没有指定就以当前的年-月-日。

持续时间

它有两种表示方式：

- `[-][HH:]MM:SS[.m...]`

`HH` 表示小时数，`MM` 表示分钟数（最多2位数字）`ss` 表示秒数（也最多2位数字），`m` 是 `ss` 的小数位值

- `[-]S+[,m...]`

`s` 是秒的数值，`m` 是 `s` 的小数位值。

两种语法前面都可选 `'-'` 号，表示负数持续时间。

持续时间例子

下面均是有效持续时间：

- `'55'` 表示55秒
- `'12:03:45'` 表示12小时3分钟45秒
- `'23.189'` 表示23.189秒

视频尺寸（分辨率）

指定视频源的尺寸大小，它可以是一些表示特定（预设）尺寸的字符串名或者 `widthxheight`（其中`width`和`height`都是数字值）的字符串 下面是一些预设的表示尺寸的字符串名及其对应分辨率：

- `'ntsc'` 720x480
- `'pal'` 720x576
- `'qntsc'` 352x240
- `'qpal'` 352x288
- `'sntsc'` 640x480
- `'spal'` 768x576
- `'film'` 352x240
- `'ntsc-film'` 352x240
- `'sqcif'` 128x96
- `'qcif'` 176x144
- `'cif'` 352x288
- `'4cif'` 704x576
- `'16cif'` 1408x1152
- `'qqvga'` 160x120
- `'qvga'` 320x240
- `'vga'` 640x480
- `'svga'` 800x600
- `'xga'` 1024x768
- `'uxga'` 1600x1200
- `'qxga'` 2048x1536
- `'sxga'` 1280x1024
- `'qsxga'` 2560x2048
- `'hsxga'` 5120x4096
- `'wvga'` 852x480
- `'wxga'` 1366x768
- `'wsxga'` 1600x1024
- `'wuxga'` 1920x1200
- `'woxga'` 2560x1600
- `'wqsxga'` 3200x2048
- `'wquxga'` 3840x2400
- `'whsxga'` 6400x4096
- `'whuxga'` 7680x4800
- `'cga'` 320x200

- 'ega' 640x350
- 'hd480' 852x480
- 'hd720' 1280x720
- 'hd1080' 1920x1080
- '2k' 2048x1080
- '2kflat' 1998x1080
- '2kscope' 2048x858
- '4k' 4096x2160
- '4kflat' 3996x2160
- '4kscope' 4096x1716
- 'nhd' 640x360
- 'hqvga' 240x160
- 'wqvga' 400x240
- 'fwqvga' 432x240
- 'hvga' 480x320
- 'qhd' 960x540

视频帧率

指定视频的帧速率，除了用每秒帧数表示外，还可以用 `frame_rate_num/frame_rate_den` 这样的格式字符串表示，此外还有一些预定义的帧率名字符串。下面就是一些预定义的帧率名及对应的帧率：

- 'ntsc' 30000/1001
- 'pal' 25/1
- 'qpai' 25/1
- 'sntsc' 30000/1001
- 'spal' 25/1
- 'film' 24/1
- 'ntsc-film' 24000/1001

比率

一个比率值可以是一个表达式或者 `numerator.denominator` 一样的含小数值。

注意比率无限值 (1/0) 或者负数值被认为是有效的，这里你需要摒弃以往的一些看法。

未定义的值可以用 "0:0" 字符串表示。

颜色/Color

允许采用下面预定义的颜色名或者一个 `[0x|#]RRGGBB[AA]` 这样序列的16进制数字值，可以通过 `@` 来附加透明度表示，透明度分量 (alpha) 可以是 "0x" 后面跟一个16进制数或者0到1之间的十进制字符串，它代表不透明度值 ('0x00' 或者 '0' 表示完全透明，'0xFF' 或者 '1' 表示完全不透明)，如果没有专门指定透明分量，则默认为 '0xFF'。

'random' 字符串会随机一个颜色。

下面是预定义的颜色名以及对应的颜色值：

- 'AliceBlue' 0xF0F8FF
- 'AntiqueWhite' 0xFAEBD7
- 'Aqua' 0x00FFFF
- 'Aquamarine' 0x7FFFD4
- 'Azure' 0xF0FFFF
- 'Beige' 0xF5F5DC

- 'Bisque' 0xFFE4C4
- 'Black' 0x000000
- 'BlanchedAlmond' 0xFFEBCD
- 'Blue' 0x0000FF
- 'BlueViolet' 0x8A2BE2
- 'Brown' 0xA52A2A
- 'BurlyWood' 0xDEB887
- 'CadetBlue' 0x5F9EA0
- 'Chartreuse' 0x7FFF00
- 'Chocolate' 0xD2691E
- 'Coral' 0xFF7F50
- 'CornflowerBlue' 0x6495ED
- 'Cornsilk' 0xFFFF8DC
- 'Crimson' 0xDC143C
- 'Cyan' 0x00FFFF
- 'DarkBlue' 0x00008B
- 'DarkCyan' 0x008B8B
- 'DarkGoldenRod' 0xB8860B
- 'DarkGray' 0xA9A9A9
- 'DarkGreen' 0x006400
- 'DarkKhaki' 0xBDB76B
- 'DarkMagenta' 0x8B008B
- 'DarkOliveGreen' 0x556B2F
- 'Darkorange' 0xFF8C00
- 'DarkOrchid' 0x9932CC
- 'DarkRed' 0x8B0000
- 'DarkSalmon' 0xE9967A
- 'DarkSeaGreen' 0x8FBC8F
- 'DarkSlateBlue' 0x483D8B
- 'DarkSlateGray' 0x2F4F4F
- 'DarkTurquoise' 0x00CED1
- 'DarkViolet' 0x9400D3
- 'DeepPink' 0xFF1493
- 'DeepSkyBlue' 0x00BFFF
- 'DimGray' 0x696969
- 'DodgerBlue' 0x1E90FF
- 'FireBrick' 0xB22222
- 'FloralWhite' 0xFFFFAF0
- 'ForestGreen' 0x228B22
- 'Fuchsia' 0xFF00FF
- 'Gainsboro' 0xDCDCDC
- 'GhostWhite' 0xF8F8FF
- 'Gold' 0xFFD700
- 'GoldenRod' 0xDAA520
- 'Gray' 0x808080
- 'Green' 0x008000
- 'GreenYellow' 0xADFF2F
- 'HoneyDew' 0xF0FFF0
- 'HotPink' 0xFF69B4
- 'IndianRed' 0xCD5C5C
- 'Indigo' 0x4B0082
- 'Ivory' 0xFFFFF0
- 'Khaki' 0xF0E68C

- 'Lavender' 0xE6E6FA
- 'LavenderBlush' 0xFFFF0F5
- 'LawnGreen' 0x7CFC00
- 'LemonChiffon' 0xFFFFACD
- 'LightBlue' 0xADD8E6
- 'LightCoral' 0xF08080
- 'LightCyan' 0xE0FFFF
- 'LightGoldenRodYellow' 0xFAFAD2
- 'LightGreen' 0x90EE90
- 'LightGrey' 0xD3D3D3
- 'LightPink' 0xFFB6C1
- 'LightSalmon' 0xFFA07A
- 'LightSeaGreen' 0x20B2AA
- 'LightSkyBlue' 0x87CEFA
- 'LightSlateGray' 0x778899
- 'LightSteelBlue' 0xB0C4DE
- 'LightYellow' 0xFFFFE0
- 'Lime' 0x00FF00
- 'LimeGreen' 0x32CD32
- 'Linen' 0xFAF0E6
- 'Magenta' 0xFF00FF
- 'Maroon' 0x800000
- 'MediumAquaMarine' 0x66CDAA
- 'MediumBlue' 0x0000CD
- 'MediumOrchid' 0xBA55D3
- 'MediumPurple' 0x9370D8
- 'MediumSeaGreen' 0x3CB371
- 'MediumSlateBlue' 0x7B68EE
- 'MediumSpringGreen' 0x00FA9A
- 'MediumTurquoise' 0x48D1CC
- 'MediumVioletRed' 0xC71585
- 'MidnightBlue' 0x191970
- 'MintCream' 0xF5FFFA
- 'MistyRose' 0xFFE4E1
- 'Moccasin' 0xFFE4B5
- 'NavajoWhite' 0xFFDEAD
- 'Navy' 0x000080
- 'OldLace' 0xFDF5E6
- 'Olive' 0x808000
- 'OliveDrab' 0x6B8E23
- 'Orange' 0xFFA500
- 'OrangeRed' 0xFF4500
- 'Orchid' 0xDA70D6
- 'PaleGoldenRod' 0xEE8AA
- 'PaleGreen' 0x98FB98
- 'PaleTurquoise' 0xAFEEEE
- 'PaleVioletRed' 0xD87093
- 'PapayaWhip' 0xFFEFD5
- 'PeachPuff' 0xFFDAB9
- 'Peru' 0xCD853F
- 'Pink' 0xFFC0CB
- 'Plum' 0xDDA0DD
- 'PowderBlue' 0xB0E0E6

- 'Purple' 0x800080
- 'Red' 0xFF0000
- 'RosyBrown' 0xBC8F8F
- 'RoyalBlue' 0x4169E1
- 'SaddleBrown' 0x8B4513
- 'Salmon' 0xFA8072
- 'SandyBrown' 0xF4A460
- 'SeaGreen' 0x2E8B57
- 'SeaShell' 0xFFFFEE
- 'Sienna' 0xA0522D
- 'Silver' 0xC0C0C0
- 'SkyBlue' 0x87CEEB
- 'SlateBlue' 0x6A5ACD
- 'SlateGray' 0x708090
- 'Snow' 0xFFFFFA
- 'SpringGreen' 0x00FF7F
- 'SteelBlue' 0x4682B4
- 'Tan' 0xD2B48C
- 'Teal' 0x008080
- 'Thistle' 0xD8BFD8
- 'Tomato' 0xFF6347
- 'Turquoise' 0x40E0D0
- 'Violet' 0xEE82EE
- 'Wheat' 0xF5DEB3
- 'White' 0FFFFFFF
- 'WhiteSmoke' 0xF5F5F5
- 'Yellow' 0xFFFF00
- 'YellowGreen' 0x9ACD32

通道布局

对于多音频通道的流，一个通道布局可以具体描述其配置情况。为了描述通道布局，ffmpeg采用了一些特殊的语法。

除了可以采用ID标识外，也可以采用下表的预定义：

- 'FL' front left 左前
- 'FR' front right 右前
- 'FC' front center 前中
- 'LFE' low frequency 重低音
- 'BL' back left 左后
- 'BR' back right 右后
- 'FLC' front left-of-center 前左中
- 'FRC' front right-of-center 前右中
- 'BC' back center 后中
- 'SL' side left 左侧
- 'SR' side right 右侧
- 'TC' top center 顶中
- 'TFL' top front left 顶左前
- 'TFC' top front center 顶前中
- 'TFR' top front right 顶右前
- 'TBL' top back left 顶左后
- 'TBC' top back center 顶后中
- 'TBR' top back right 顶右后

- 'DL' downmix left 左缩混
- 'DR' downmix right 右缩混
- 'WL' wide left 左边
- 'WR' wide right 右边
- 'SDL' surround direct left 左直通
- 'SDR' surround direct right 右直通
- 'LFE2' low frequency 2 超重低音

标准的通道布局可以采用如下的定义：

- 'mono' FC
- 'stereo' FL+FR
- '2.1' FL+FR+LFE
- '3.0' FL+FR+FC
- '3.0(back)' FL+FR+BC
- '4.0' FL+FR+FC+BC
- 'quad' FL+FR+BL+BR
- 'quad(side)' FL+FR+SL+SR
- '3.1' FL+FR+FC+LFE
- '5.0' FL+FR+FC+BL+BR
- '5.0(side)' FL+FR+FC+SL+SR
- '4.1' FL+FR+FC+LFE+BC
- '5.1' FL+FR+FC+LFE+BL+BR
- '5.1(side)' FL+FR+FC+LFE+SL+SR
- '6.0' FL+FR+FC+BC+SL+SR
- '6.0(front)' FL+FR+FLC+FRC+SL+SR
- 'hexagonal' FL+FR+FC+BL+BR+BC
- '6.1' FL+FR+FC+LFE+BC+SL+SR
- '6.1' FL+FR+FC+LFE+BL+BR+BC
- '6.1(front)' FL+FR+LFE+FLC+FRC+SL+SR
- '7.0' FL+FR+FC+BL+BR+SL+SR
- '7.0(front)' FL+FR+FC+FLC+FRC+SL+SR
- '7.1' FL+FR+FC+LFE+BL+BR+SL+SR
- '7.1(wide)' FL+FR+FC+LFE+BL+BR+FLC+FRC
- '7.1(wide-side)' FL+FR+FC+LFE+FLC+FRC+SL+SR
- 'octagonal' FL+FR+FC+BL+BR+BC+SL+SR
- 'downmix' DL+DR

一个特定的通道布局可以是一组由 + 或者 | 连接起来多个值，其中每个值可以是：

- 前面的标准通道布局名，例如'mono','stereo','4.0','quad','5.0'等等
- 或者单个命名通道如'FL','FR','FC','LFE'等等
- 多个通道数字序号，使用后缀'c'的十进制，对于利用数字表示的默认通道布局，参考 `av-get_default_channel_layout` 说明。
- 通道布局蒙版(mask)，是'0x'起始的16进制，参考在 `libavutil/channel_layout.h` 中声明的 `AV_CH_*` 宏。

从libavutil版本53开始，尾随'c'的十进制数表示通道变成必须（除非采用16进制的蒙版来表示通道）

参考声明于 `libavutil/channel_layout.h` 中的 `av_get_channel_layout` 进行深入了解。

8 表达式计算/求值

在计算表达式时，ffmpeg通过 `libavutil/eval.h` 接口调用内部计算器进行计算。

表达式可以包含一元运算符、运算符、常数和函数

两个表达式 `expr1` 和 `expr2` 可以组合起来成为"`expr1;expr2`"，两个表达式都会被计算，但是新表达式（组合起来的）值为表达式 `expr2` 的值。

表达式支持的二元运算符有: `+` , `-` , `*` , `/` , `^`

一元运算符: `+` , `-`

以及下面的函数：

- `abs(x)`

返回x的绝对值.

- `acos(x)`

计算x反余弦.

- `asin(x)`

计算x的反正弦.

- `atan(x)`

计算x反正切.

- `between(x, min, max)`

判断 $\min \leq x \leq \max$ 是否成立，成立返回1，否则返回0.

- `bitand(x, y)`

- `bitor(x, y)`

返回x和y按位与/或的值

注意计算是作为整数的，转换为整数和转换回浮点数都会损伤精度。这可能造成意外结果(通常 2^{53} 和更大的数)

- `ceil(expr)`

返回大于expr的最接近整数，例如"`ceil(1.5)`" 返回"`2.0`".

- `clip(x, min, max)`

Return the value of x clipped between min and max.

- `cos(x)`

计算x的余弦.

- `cosh(x)`

计算x的双余弦.

- eq(x, y)

如果x==y返回1, 否则为0 otherwise.

- exp(x)

计算指数x, (底数为e,即计算欧拉数 (Euler's number))

- floor(expr)

返回不大于expr的整数, 例如 "floor(-1.5)" 为 "-2.0".

- gauss(x)

计算x的高斯 (Gauss) 函数值,即计算 $(-xx/2) / \sqrt{2\pi}$.

- gcd(x, y)

返回x和y的最大公约数, 如果x和y为0或者任意数小于0则行为未定义。

- gt(x, y)

返回判断x>y的结果, 符合则为1, 否则为0

- gte(x, y)

返回判断x>=y的结果, 符合则为1, 否则为0

- hypot(x, y)

这个和C语言中的函数有相同名字和功能, 相当于计算 $\sqrt{xx + yy}$,是求长为x, 宽为y的斜边长度 (勾股定理)

- if(x, y)

判断x值, 如果x值为非0, 则返回y, 否则返回0

- if(x, y, z)

判断x值, 如果x值为非0, 则返回y, 否则返回z.

- ifnot(x, y)

判断x值, 如果x值为0, 则返回y, 否则返回0

- ifnot(x, y, z)

判断x值, 如果x值为0, 则返回y, 否则返回z.

- isinf(x)

如果x值是正负无穷则返回1.0.否则返回0.0

- isnan(x)

如果x是 NAN 则返回1.0, 否则返回0.0

- `ld(var)`

加载预订的内部变量`var`对应值，其中值是利用`st(var, expr)`存储的

- `log(x)`

计算`x`的自然对数值

- `lt(x, y)`

返回`x<y`判断式值，符合为1，否则为0

- `lte(x, y)`

返回`x<=y`判断式值，符合为1，否则为0

- `max(x, y)`

返回`x`和`y`中的更大的值

- `min(x, y)`

返回`x`和`y`中的更小的值

- `mod(x, y)`

计算`x%y`

- `not(expr)`

如果`expr==0`则返回1，否则返回0

- `pow(x, y)`

计算`"(x)^(y)"`.

- `print(t)`

- `print(t, l)`

以日志层次`l`打印`t`，如果`l`没有定义则采用当前默认日志层次，返回打印内容。

- `random(x)`

返回一个0.0-1.0间的随机数，`x`是一个随机数种子。

- `root(expr, max)`

对于不同的输入计算表达式`expr`的值，直到`max`输入值。即依次取`ld(x)`，`x`的值为0..`max`，把`ld(x)`值作为参数计算`expr`值

表达式`expr`必须是一个连续函数，否则结果不定。

`ld(0)`被用作`expr`表达式的参数，所以表达式可以依据不同的值计算多次。

- `sin(x)`

计算`x`的正弦

- `sinh(x)`

计算x的双曲正弦

- `sqrt(expr)`

计算x的平方根。相当于 " $(\text{expr})^{.5}$ ".

- `squish(x)`

计算 $1/(1 + \exp(4*x))$.

- `st(var, expr)`

对var变量在内部存储一个expr值，供以后使用，var范围为0-9.注意这些变量当前不能在表达式间共享

- `tan(x)`

返回x的正切.

- `tanh(x)`

计算x的双曲正切

- `taylor(expr, x)`

- `taylor(expr, x, id)`

计算泰勒（Taylor）级数值。给出表达式（`ld(id)`）在0阶的导数函数,即`taylor(expr,x)=taylor(expr,x,0)`

如果级数不收敛，则结果是不确定的。

`ld(id)`用来表示expr的导数阶，这意味着对给定的表达式，输入不同的值可以通过`ld(id)`进行多次计算。这里我们假定不是预设的0阶。

注意当你用一个Y值替代默认的0时，相当于计算 `taylor(expr, x-y)`

- `time(0)`

返回当前时间，单位为秒

- `trunc(expr)`

返回expr最接近的（向0）整数，如"`trunc(-1.5)`" 值为 "-1.0".

- `while(cond, expr)`

当cond不为0时循环执行expr,直至cond为0

有如下一些常量:

- `PI`

单位圆周长与直径比，约3.14

- `E`

`exp(1)`计算值 (Euler's 欧拉数),约2.718

- PHI

黄金分割比, (1+sqrt(5))/2计算值, 约1.618

以及布尔运算, 其中非0值表示"true"(真),以及运算符:

- * 表示 AND 与操作
- + 表示 OR 或操作

例如: 要表示 if (A AND B) then C

```
等效于if(A*B,C)
```

如果你了解C语言代码, 其所有的一元和二元以及定义的常数均可用于表达式。

表达式也支持国际标准的单位前/后缀 (定义), 例如 i 附加在数值后, 表示这个数值是基于1024而不是1000计算幂的, "B"表示"Byte", 并可以附加一个单位前缀或者当地使用, 例如允许 KB, MiB, G 和 B 作为单位后缀。

下面的列表就是当前遵循的国际体系前缀列表, 并给出了对应2的整10次方值:

y

```
10^-24 / 2^-80
```

z

```
10^-21 / 2^-70
```

a

```
10^-18 / 2^-60
```

f

```
10^-15 / 2^-50
```

p

```
10^-12 / 2^-40
```

n

```
10^-9 / 2^-30
```

u

$10^{-6} / 2^{-20}$

m

$10^{-3} / 2^{-10}$

c

10^{-2}

d

10^{-1}

h

10^2

k

$10^3 / 2^{10}$

K

$10^3 / 2^{10}$

M

$10^6 / 2^{20}$

G

$10^9 / 2^{30}$

T

$10^{12} / 2^{40}$

P

$10^{15} / 2^{40}$

E

$10^{18} / 2^{50}$

Z

$10^{21} / 2^{60}$

Y

$10^{24} / 2^{70}$

9 OpenCL选项

当FFmpeg编译时打开了 `--enable-opengl` 配置，则可以在全局使用OpenCL选项。

下面是支持的选项：

- `build_options` :设置编译选项，指定编译的注册核心
参考"OpenCL Specification Version: 1.2 chapter 5.6.4".
- `platform_idx` : 选定指定平台运行OpenCL的编码（格式），这里指定的索引必须是可以由 `ffmpeg -opengl_bench` 或者 `av_opengl_get_device_list()` 查询到的设备列表索引。
- `device_idx` : 指定设备运行的OpenCL编码（格式）。这里指定的索引必须是可以由 `ffmpeg -opengl_bench` 或者 `av_opengl_get_device_list()` 查询到的设备列表索引。

10 编码选项

libavcodec提供一些通用的全局选项设置，可在所有的编码器和解码器起效。另外每个编解码器可以支持所谓的私有化设置，以满足特定的编解码要求。

有时，一个全局选项会影响到特定的编解码器，而对其它编解码产生不良影响或者会不被识别，所以你需要了解这些影响编解码选项的具体意义，了解那些只对特定编码或者解码有效的选项。

这些选项大多可以 `-option value` 的格式在ffmpeg工具中指定，其中 `-option` 是选项名，`value` 是要设置的选项参数值，个别是利用 `AVCodecContext` 选项进行额外配置，还有极个别的使用定义在 `libavutil/opt.h` 中的API在程序过程中配置使用。

下面是这些选项的列表(括号中表示选项有效的状态，可能是decoding-解码时,encoding—编码时，audio-音频，video-视频，subtitles-字幕 以及特定的编码名称，如果mpeg4):

b integer (encoding,audio,video)


设置码率，单位为bits/s。默认200K。

ab integer (encoding,audio)

设置音频码率，单位bits/s。默认128K。

bt integer (encoding,video)

设置视频码率偏离公差 (video bitrate tolerance)，单位bits/s。对于1次编码模式中码率控制公差指愿意偏离目标（码率）的平均码率值，故不能由此



flags flags (decoding/encoding,audio,video,subtitles)

设置常见的标志

可能值有：

`'mv4'`

使用4路监控宏块运动矢量 (mpeg4)。

`'qpel'`

使用1/4像素补偿

`'loop'`

使用循环过滤

`'qscale'`

使用固定放缩qscale

`'gmc'`

使用gmc。

`'mv0'`

一直假定mb中mv=<0,0>。

`'input_preserved'`

`'pass1'`

在第1次编码中使用内部控制段模式。

`'pass2'`



me_method integer (encoding,video)



```

    X1运动估计
    'hex'

    hex运动估计
    'umh'

    umh运动估计
    'iter'

    iter运动估计
```

extradata_size integer

设置扩展数据尺寸

time_base rational number

设置编码的时间基础计量

它是时间的基本单位(秒)的帧时间戳表示。对于固定FPS内容，这个值应该是`1/frame_rate`，每次时间戳都增加1个单位的时间量。

g integer (encoding,video)

设置一组图片的数量，默认是12

ar integer (decoding/encoding,audio)

设置音频采样率（单位Hz）

ac integer (decoding/encoding,audio)

设置音频通道数

cutoff integer (encoding,audio)

设置截止带宽

frame_size integer (encoding,audio)

设置音频帧尺寸

除了最后一帧音频数据，否则每个音频数据帧都包含`frame_size`设定大小的数据。当编码中`CODEC_CAP_VARIABLE_FRAME_SIZE`设置了，则这个值可能

frame_number integer

设置帧数量

delay integer qcomp float (encoding,video)

设置视频量化压缩规模(VBR)。这里用在控制方程的常数。默认rc_eq推荐范围是：0.0-1.0。

qblur float (encoding,video)

设置视频量化尺度模糊(VBR)。

qmin integer (encoding,video)

设置最小视频量化尺度(VBR)。取值范围是-1-69，默认为2

qmax integer (encoding,video)

设置最大视频量化尺度(VBR)。取值范围是-1-1024，默认为31

qdiff integer (encoding,video)

设置量化级之间最大的差异(VBR)。

bf integer (encoding,video)

设置最大B帧间隔

必须是-1 -16间的数。0表示禁止B帧，如果为-1表示依据编码器进行指定，默认值是0

b_qfactor float (encoding,video)

设置P帧和B帧之间的qp因子

rc_strategy integer (encoding,video)

设置码率控制方法

b_strategy integer (encoding,video)

设置I/P/B帧选择策略。

ps integer (encoding,video)

设置RTP播放加载数据量(缓冲)，单位是字节(bytes)

mv_bits integer

header_bits integer
 i_tex_bits integer
 p_tex_bits integer
 i_count integer
 p_count integer
 skip_count integer
 misc_bits integer
 frame_bits integer
 codec_tag integer
 bug flags (decoding,video)

解决不能自动检测/识别编码的错误 (bug)

可能值:

'autodetect'
 'old_msmpeg4'

一些旧lavc处理的msmpeg4v3文件(不能自动检测)
 'xvid_ilace'

Xvid交错错误 (如果强制为fourcc==XVIX则可自动检测)
 'ump4'

(fourcc==UMP4可自动检测)
 'no_padding'

填充错误(自动检测)
 'amv'
 'ac_vlc'

非法vlc错误 (对每个fourcc自动检测)
 'qpel_chroma'
 'std_qpel'

老标准的qpel (对每个fourcc/version自动检测)
 'qpel_chroma2'
 'direct_blocksize'

direct-qpel-blocksize错误 (对每个fourcc/version自动检测)
 'edge'

edge填充bug (对每个fourcc/version自动检测)
 'hpel_chroma'
 'dc_clip'
 'ms'

微软破解解码器上的各种缺陷
 'trunc'

截断帧

lelim integer (encoding,video)

设置亮度单系数消除阈值 (负值也考虑直流系数)。 celim integer (encoding,video)

设置消除色度单系数阈值 (负值也考虑直流系数)

strict integer (decoding/encoding,audio,video)

指定如何遵守标准 (严格程度)

可能的值:

‘very’

严格模式，遵守过时的版本规格或者依软件需求的版本规格进行处理

‘strict’

严格模式，无论如何都严格按照规格处理

‘normal’

‘unofficial’

允许非官方扩展

‘experimental’

允许非标准的实验性质解码/编码器（未完成/未测试/不工作）**注意**实验性质的解码工具可能带来安全风险，不要用这类解码器解码不可信输入

b_qoffset float (encoding,video)

在P帧和B帧间设置QP（帧间偏移）

err_detect flags (decoding,audio,video)

设置错误检测标志

可能值：

‘crccheck’

嵌入CRC验证

‘bitstream’

比特流规范偏差检测

‘buffer’

不当码流长度检测

‘explode’

在出现错误时中止解码

‘ignore_err’

忽略解码错误继续解码。则对于分析视频内容是十分有用的，这时希望无论如何解码都继续工作。

‘careful’

考虑环境支持，一个正确的编码器不能被错误停止

has_b_frames integer block_align integer mpeg_quant integer (encoding,video)

使用MPEG量化代替H.263。

qsquish float (encoding,video)

保持Qmin和Qmax之间量化器。(0 = clip, 1 = use 还可以利用函数进行微调)。

设置实验/量化

rc_qmod_freq integer (encoding,video)

设置实验/量化调制

rc_override_count integer rc_eq string (encoding,video)

速率控制方程组。除了内部标准定义外，可以有以下选择：bits2qp(bits)，qp2bits(qp)。还可以利用下面介绍的常数：iTEx、pTEx、tEx，mv fCode

maxrate integer (encoding,audio,video)

设置最大比特率容差（单位 比特/秒）。要求的缓冲大小被设置。

minrate integer (encoding,audio,video)

设置最小比特率容差（单位 比特/秒）。通常用于CBR编码，否则无意义

bufsize integer (encoding,audio,video)

设置控制缓冲区大小（单位bits）

rc_buf_aggressivity float (encoding,video)

目前无效

i_qfactor float (encoding,video)

设置P帧和I帧间的QP因子

i_qoffset float (encoding,video)

设置P帧和I帧间的QP偏移

rc_init_cplx float (encoding,video)

设置1次编码的初始复杂性

dct integer (encoding,video)

设置DCT（数字转换）算法

可能值：

'auto'

自动选择一个优化质量算法(默认值)

'fastint'

偏重速度

'int'

精准整数


```
'mmx'
'altivec'
'faan'

    浮点AAN DCT
```

lumi_mask float (encoding,video)

```
    压缩高亮
```

tcplx_mask float (encoding,video)

```
    设置临时/时间复杂遮蔽/蒙版
```

scplx_mask float (encoding,video)

```
    设置空间复杂遮蔽/蒙版
```

p_mask float (encoding,video)

```
    设置组间遮蔽
```

dark_mask float (encoding,video)

```
    压缩暗区
```

idct integer (decoding/encoding,video)

```
    选择实施的IDCT

    可能值:

    'auto'
    'int'
    'simple'
    'simplemmx'
    'simpleauto'

    自动应用一个兼容的IDCT
    'arm'
    'altivec'
    'sh4'
    'simplearm'
    'simplearmv5te'
    'simplearmv6'
    'simpleneon'
    'simplealpha'
    'ipp'
    'xvidmmx'
    'faani'

    浮点AAN IDCT
```

slice_count integer

ec flags (decoding,video)

设置错误隐藏策略

可能值:

`'guess_mv'`

运动矢量迭代 (MV)搜索(慢/slow)

`'deblock'`

对损坏的MBs使用强壮的去块滤波

`'favor_inter'`

有利用从前帧预测而不是当前帧预测

bits_per_coded_sample integer

pred integer (encoding,video)

设置预测方法

可能值:

`'left'``'plane'``'median'`

aspect rational number (encoding,video)

设置样本纵横比

debug flags (decoding/encoding,audio,video,subtitles)

输出特定调试信息。

可能值:

`'pict'`

图片相关信息

`'rc'`

码率控制

`'bitstream'``'mb_type'`

宏块 (MB)类型

`'qp'`

每个块的量化参数(QP)

`'mv'`

运动矢量

`'dct_coeff'``'skip'``'startcode'``'pts'``'er'`

错误识别

`'mmco'`

内存管理控制操作(H.264)

`'bugs'``'vis_qp'`

量化参数可视化 (QP),即低的QP显示为绿色

`'vis_mb_type'`

```

    块类型可视化
'buffers'

    图像缓冲区分配
'thread_ops'

    线程操作
'nomc'

    跳跃运动补偿
```

vismv integer (decoding,video)

```

运动矢量可视化 (MVs)。

这个选项是过时的，参考`codecvview`滤镜。

可能值：

'pf'

    p帧前测MVs
'bf'

    B帧前测MVs
'bb'

    B帧后测MVs
```

cmp integer (encoding,video)

```

设置完整图元me压缩功能

可能值：

'sad'

    绝对差异总和，最快(默认)
'sse'

    误差平方和
'satd'

    绝对Hadamard转换差异总和
'dct'

    绝对DCT转换差异总和
'psnr'

    量化误差平方和 (avoid, 低品质)
'bit'

    对特定块需要的比特数
'rd'

    动态失真优化，最慢
'zero'

    0
'vsad'

    绝对垂直差异总和
'vsse'

    垂直差异平方和
'nsse'

    噪声保护的差的平方和
'w53'
```

```
5/3 小波（变换），只用于下雪场景
'w97'

9/7 小波（变换），只用于下雪场景
'dctmax'
'chroma'
```

subcmp integer (encoding,video)

```
设置局部图元me压缩功能

可能值：

'sad'

绝对差异总和，最快(默认)
'sse'

误差平方和
'satd'

绝对Hadamard转换差异总和
'dct'

绝对DCT转换差异总和
'psnr'

量化误差平方和（avoid，低品质）
'bit'

对特定块需要的比特数
'rd'

动态失真优化，最慢
'zero'

0
'vsad'

绝对垂直差异总和
'vsse'

垂直差异平方和
'nsse'

噪声保护的差的平方和
'w53'

5/3 小波（变换），只用于下雪场景
'w97'

9/7 小波（变换），只用于下雪场景
'dctmax'
'chroma'
```

mbcmp integer (encoding,video)

```
设置宏块压缩功能

可能值：

'sad'

绝对差异总和，最快(默认)
'sse'

误差平方和
'satd'

绝对Hadamard转换差异总和
'dct'
```

```

    绝对DCT转换差异总和
    'psnr'

    量化误差平方和 (avoid, 低品质)
    'bit'

    对特定块需要的比特数
    'rd'

    动态失真优化, 最慢
    'zero'

    0
    'vsad'

    绝对垂直差异总和
    'vsse'

    垂直差异平方和
    'nsse'

    噪声保护的差的平方和
    'w53'

    5/3 小波 (变换), 只用于下雪场景
    'w97'

    9/7 小波 (变换), 只用于下雪场景
    'dctmax'
    'chroma'
```

ildctcmp integer (encoding,video)

```

设置隔行dct压缩功能

可能值:

'sad'

    绝对差异总和, 最快(默认)
    'sse'

    误差平方和
    'satd'

    绝对Hadamard转换差异总和
    'dct'

    绝对DCT转换差异总和
    'psnr'

    量化误差平方和 (avoid, 低品质)
    'bit'

    对特定块需要的比特数
    'rd'

    动态失真优化, 最慢
    'zero'

    0
    'vsad'

    绝对垂直差异总和
    'vsse'

    垂直差异平方和
    'nsse'

    噪声保护的差的平方和
    'w53'

    5/3 小波 (变换), 只用于下雪场景
```

```
'w97'

    9/7 小波（变换），只用于下雪场景
'dctmax'
'chroma'
```

dia_size integer (encoding,video)

```
设置运动估计区域类型和尺寸
```

last_pred integer (encoding,video)

```
设置从前帧预测运动量
```

preme integer (encoding,video)

```
设置预运动估计
```

precmp integer (encoding,video)

```
设置预运动估计压缩功能

可能值：

'sad'

    绝对差异总和，最快(默认)
'sse'

    误差平方和
'satd'

    绝对Hadamard转换差异总和
'dct'

    绝对DCT转换差异总和
'psnr'

    量化误差平方和（avoid，低品质）
'bit'

    对特定块需要的比特数
'rd'

    动态失真优化，最慢
'zero'

    0
'vsad'

    绝对垂直差异总和
'vsse'

    垂直差异平方和
'nsse'

    噪声保护的差的平方和
'w53'

    5/3 小波（变换），只用于下雪场景
'w97'

    9/7 小波（变换），只用于下雪场景
'dctmax'
```

‘chroma’

pre_dia_size integer (encoding,video)

设置运动估计预测的区域类型和尺寸

subq integer (encoding,video)

设置子图元运动估计质量

dtg_active_format integer me_range integer (encoding,video)

设置运动矢量极限范围（DivX是1023）。

ibias integer (encoding,video)

设置组内量化偏差

pbias integer (encoding,video)

设置集间量化偏差

color_table_id integer global_quality integer (encoding,audio,video) coder integer (encoding,video)

可能值：

‘vlc’

 可变长编码 / huffman编码

‘ac’

 算术编码

‘raw’

 raw（不进行编码）

‘rle’

 游程长度编码

‘deflate’

 紧缩编码

context integer (encoding,video)

设置上下文模型

slice_flags integer xvmc_acceleration integer mbd integer (encoding,video)

设置宏块选择算法（高质量模式）。

可能值：

```
'simple'

    使用mbcmp, 宏块比较 (默认)
'bits'

    减少数据量
'rd'

    失真率优化
```

stream_codec_tag integer sc_threshold integer (encoding,video)

```
设置场景变化阈值
```

lmin integer (encoding,video)

```
设置最小拉格朗日(lagrange)因子(VBR)。
```

lmax integer (encoding,video)

```
设置最大拉格朗日(lagrange)因子 (VBR)。
```

nr integer (encoding,video)

```
设置降噪
```

rc_init_occupancy integer (encoding,video)

```
设置解码开始前需加载到RC缓冲区的数据量
```

flags2 flags (decoding/encoding,audio,video)

```
可能值:

'fast'

    允许不符合规范的加速技巧。
'sgop'

    失效, 使用mpegvideo私有选项
'noout'

    跳过比特流编码
'ignorecrop'

    忽略sps传来的遮蔽信息
'local_header'

    在全局头而不是每个关键帧放置扩展数据
'chunks'

    帧数据可被分割成多个块
'showall'

    显示第一个关键帧前的所有帧 (一般用于跳跃定位后的播放, 默认是从最近关键帧开始显示, 因为之前的帧不一定能够正确构建)
'skiprd'
```


失效, 使用了mpegvideo私有选项.
`'export_mvs'`

支持它的编码中运动矢量导出给帧间数据(见`AV_FRAME_DATA_MOTION_VECTORS``)。参考`doc/examples/export_mvs.c``

`error integer (encoding,video) qns integer (encoding,video)`

失效, 使用了mpegvideo私有选项

`threads integer (decoding/encoding,video)`

可能值:

`'auto'`

检测使用一个合适的线程数

`me_threshold integer (encoding,video)`

设置运动估计的阈值

`mb_threshold integer (encoding,video)`

设置宏块阈值

`dc integer (encoding,video)`

设置`intra_dc_precision`.

`nssew integer (encoding,video)`

设置`nsse`权重.

`skip_top integer (decoding,video)`

跳过顶部设置多个宏块行

`skip_bottom integer (decoding,video)`

跳过底部设置多个宏块行

`profile integer (encoding,audio,video)`

可能值:

`'unknown'`

```
'aac_main'
'aac_low'
'aac_ssr'
'aac_ltp'
'aac_he'
'aac_he_v2'
'aac_ld'
'aac_eld'
'mpeg2_aac_low'
'mpeg2_aac_he'
'mpeg4_sp'
'mpeg4_core'
'mpeg4_main'
'mpeg4_asp'
'dts'
'dts_es'
'dts_96_24'
'dts_hd_hra'
'dts_hd_ma'
```

level integer (encoding,audio,video)

可能值:

'unknown'

lowres integer (decoding,audio,video)

在 1= 1/2, 2=1/4, 3=1/8 解码。

skip_threshold integer (encoding,video)

设置跳帧阈值

skip_factor integer (encoding,video)

设置跳帧因子

skip_exp integer (encoding,video)

设置跳帧指数。 负值和正值除了归一化原因以外表现相同。正值存在的原因主要是兼容性，所以并不常见

skipcmp integer (encoding,video)

设置跳帧压缩功能

可能值:

'sad'

绝对差异总和, 最快(默认)

'sse'

误差平方和

'satd'

绝对Hadamard转换差异总和

```

'dct'

    绝对DCT转换差异总和
'psnr'

    量化误差平方和 (avoid, 低品质)
'bit'

    对特定块需要的比特数
'rd'

    动态失真优化, 最慢
'zero'

    0
'vsad'

    绝对垂直差异总和
'vsse'

    垂直差异平方和
'nsse'

    噪声保护的差的平方和
'w53'

    5/3 小波 (变换), 只用于下雪场景
'w97'

    9/7 小波 (变换), 只用于下雪场景
'dctmax'
'chroma'

```

border_mask float (encoding,video)

增加接近边界宏块量化。

mblmin integer (encoding,video)

设置最小的宏块的拉格朗日 (lagrange) 因子 (VBR)。

mblmax integer (encoding,video)

设置最大的宏块的拉格朗日 (lagrange) 因子 (VBR)。

mepc integer (encoding,video)

设置运动估计比特率损失补偿 (1.0 = 256)。

skip_loop_filter integer (decoding,video) **skip_idct** integer (decoding,video) **skip_frame** integer (decoding,video)

让解码器丢弃处理由选项值指定的帧类型

skip_loop_filter 跳过循环帧
skip_idct 跳过IDCT/量化(dequantization) 帧
skip_frame 跳过解码

可能值:

'none'

不抛弃帧
 'default'

抛弃无用帧，例如尺寸为0的帧
 'noref'

抛弃所有非参考帧
 'bidir'

抛弃所有双向(预测)帧
 'nokey'

除了关键帧外都抛弃
 'all'

抛弃所有帧

默认值就是'default'。

bidir_refine integer (encoding,video)

细化两个运动矢量用于双向宏块

brd_scale integer (encoding,video)

对动态B帧判定是否下变换

keyint_min integer (encoding,video)

设置IDR帧集的最小间隔

refs integer (encoding,video)

为运动补偿设置参考帧 compensation。

chromaoffset integer (encoding,video)

设置色度中qp对亮度的抵消

trellis integer (encoding,audio,video)

设置比率失真优化

sc_factor integer (encoding,video)

设置一个值(一个补偿因子)乘以`qscale`添加到每一帧的`scene_change_score`

mv0_threshold integer (encoding,video) b_sensitivity integer (encoding,video)

调整`b_frame_strategy`敏感性为1。

compression_level integer (encoding,audio,video) min_prediction_order integer (encoding,audio) max_prediction_order integer (encoding,audio) timecode_frame_start integer (encoding,video)

设置GOP时间码帧开始数, 非去帧格式

request_channels integer (decoding,audio)

设置所需数字音频轨道/通道

bits_per_raw_sample integer channel_layout integer (decoding/encoding,audio)

可能值:

request_channel_layout integer (decoding,audio)

可能值:

rc_max_vbv_use float (encoding,video)
rc_min_vbv_use float (encoding,video)
ticks_per_frame integer (decoding/encoding,audio,video)
color_primaries integer (decoding/encoding,video)
color_trc integer (decoding/encoding,video)
colospace integer (decoding/encoding,video)
color_range integer (decoding/encoding,video)
chroma_sample_location integer (decoding/encoding,video)
log_level_offset integer

设置日志层次

slices integer (encoding,video)

设置划片数, 用于并行编码

thread_type flags (decoding/encoding,video)

选择多线程方式

使用'frame'会导致每个线程解码延迟, 所以如果客户端不提供未来帧状况就不应该使用

可能值:

'slice'

每次解码不超过一个帧的多块数据

划片多线程只用于视频划片编码工作

'frame'

一次解码多个帧

默认值是 'slice+frame'。

audio_service_type integer (encoding,audio)

设置音频服务类型。

可能值：

- 'ma' 主要音频服务
- 'vi' 特效
- 'hi' 视障
- 'di' 听障
- 'co' 对话
- 'em' 评论
- 'vo' 紧急情况
- 'ka' 画外音
- 卡拉OK

request_sample_fmt sample_fmt (decoding,audio)

设置音频解码偏好。默认是none

pkt_timebase rational number sub_charenc encoding (decoding,subtitles)

设置输入的字幕字符编码

field_order field_order (video)

设置/覆盖场序。可能值是：

- 'progressive' 逐行
- 'tt' 隔行，顶场优先编码/显示
- 'bb' 隔行，底场优先编码/显示
- 'tb' 隔行，顶场优先编码，低场优先显示
- 'bt' 隔行，底场优先编码，低场优先显示

skip_alpha integer (decoding,video)

设置为1来禁止处理透明度。不同的值可以类似一个‘灰色(gray)’蒙在画面上。默认值是0。

codec_whitelist list (input)

", " 分隔的允许的解码器列表。默认是都允许

dump_separator string (input)

指定用于在命令行分隔参数、选项域的字符串。例如可以设置一个回车换行作为分隔：

```
ffprobe -dump_separator "
" -i ~/videos/matrixbench_mpeg2.mpg
```

11 解码器

解码器是让FFmpeg能对多媒体流进行解码的配置元素。

默认在编译FFmpeg时所有（内置）有效的解码器都会自动支持。如果解码器需要特别扩展库，则需要手动通过 `--enable-lib` 选项来进行支持。可以在配置编译项目中通过 `--list-decoders` 了解所有有效解码器（包括需要扩展库的）。

也可以通过在配置中采用 `--disable-decoders` 选项单独禁用某个解码器。 `--enable-decoder=DECODER` / `--disable-decoder=DECODER` 分别是启用/禁用 DECODER 解码器。

在 `ff**` 工具中（ffmpeg/ffplayer 等等）选项 `-decoders` 可以显示当前有效的解码器。

12 视频解码器

介绍当前可用的一些视频解码器

rawvideo

用于RAW视频解码。即解码rawvideo流。

rawvideo解码选项

top top_field_first

指定输入视频的呈现字段类型	
-1	
0	步进视频（默认）
1	下场优先（底部优先）
	上场优先（顶部优先）

13 音频解码器

介绍一些有效的音频解码器

ac3

AC-3 音频解码器，该解码器实现在ATSC A/52:2010 和 ETSI TS 102 366部分，以及RealAudio 3（又名DNET）中。

ac3解码器选项

- `-drc_scale value`

动态范围因子。该因子适合应用于从AC-3流中获取的动态值范围。这个值是指数值。有3个显著效果的典型值(范围)：

`drc_scale == 0`

DRC禁用，会产生原始全范围音频

`0 < drc_scale <= 1`

DRC启用，在一部分DRC中应用，音频表现为全范围与全压缩之间。

`drc_scale > 1`

DRC启用。适用于drc_scale不对称场景，表现为响亮的全压缩，轻柔的声音增强。

flac

FLAC音频解码器，它由Xiph实现了对FLAC的完整规格

FLAC解码器选项

- `-use_buggy_lpc`

FLAC编码器用于产生高 `lpc` 值的数据包流（例如在默认设置下），本选项允许正确解码采用老版本 `buggy_lpc` 逻辑编码的FLAC音频。

ffwavesynth

内部波表合成装置。

该解码器根据预定义的序列产生音频波。其使用纯粹内部不公开的数据格式表征特定的序列（集）

libcelt

libcelt解码器再封装

libcelt允许libavcodec解码Xiph CELT超低延迟音频编码。在需要libcelt头和库存在配置（能进一步搜索到库的配置）。在创建ffmpeg工具集时需要显式采用 `--enable-libcelt` 以打开支持

libgsm

libgsm解码器再封装

libgsm允许libavcodec解码GSM全速率语音。需要libgsm头和库存在配置（能进一步搜索到库的配置）。需要在创建ffmpeg时显式设置 `--enable-libgsm` 以启用

该解码器支持普通GSM和微软修订

libilbc

libilbc解码器再封装

libilbc允许libavcodec解码网络低码率编码（iLBC）音频。需要libilbc头和库存在配置。需要在创建ffmpeg时显式设置 `--enable-libilbc` 以启用

libilbc选项

下面的选项被libilbc封装支持

- `enhance`

设置为1，则解码时使音频增强，默认是0（禁用）

libopencore-amrnb

libopencore-amrnb解码器再封装

libopencore-amrnb允许libavcodec解码自适应多速率窄带（Adaptive Multi-Rate Narrowband）音频。需要libopencore-amrnb头和库存在配置才能使用。需要在创建ffmpeg时显式设置 `--enable-libopencore-amrnb` 以启用

现在FFmpeg已经直接支持解码AMR-NB，所以不需要该库了。

libopencore-amrwb

libopencore-amrwb解码器再封装

libopencore-amrwb允许libavcodec解码自适应多速率宽带（Adaptive Multi-Rate Wideband）音频。需要libopencore-amrwb头和库存在配置才能使用。需要在创建ffmpeg时显式设置 `--enable-libopencore-amrwb` 以启用

现在FFmpeg已经直接支持解码AMR-WB，所以不需要该库了。

libopus

libopus解码器再封装

libopus允许libavcodec解码Opus互动音频。需要libopus头和库存在配置才能使用。需要在创建ffmpeg时显式设置 `--enable-libopus` 以启用。

现在FFmpeg已经直接支持解码Opus，所以不需要该库了。

14 字幕解码器

dvdsb

解码用于dvd的bitmap类型字幕解码。该类型也用于vobsub文件和一些Matroska文件。

dvdsb解码选项

- `palette`

指定位图的全局调色板。当存储在VobSub中时，调色板可以依据索引表示颜色。在Matroska文件中，调色板以同于VobSub的格式存储在扩展数据区。在DVD中，存储在IFO文件中，因此在只读取VOB文件时不可用则。

选项的格式是24位（bit）数，用16进制表示字符串（没有前导的0x）例如 0d00ee, ee450d, 101010, eaeaea, 0ce60b, ec14ed, ebff0b, 0d617a, 7b7b7b, d1d1d1, 7b2a0e, 0d950c, 0f007b, cf0dec, cfa80c, 7c127b.

- `ifo_palette`

指定IFO文件中得到全局调色板（实验性质）=

- `forced_subs_only`

只在解码强制字幕。有些字幕在同一轨道中保留了强制字幕和非强制字幕，选用这个选项将只解码强制字幕，即强制字幕标志设为1的字幕。默认是0

libzvbi-teletext

libzvbi允许libavcodec解码DVB的teletext页面和DVB的teletext字幕。需要libzvbi头和库存在配置。在编译是需配置 `--enable-libzvbi` 以启用

libzvbi-teletext选项

- `txt_page`

列出需要解码的teletext页编号。你可以用 * 表示所有页，没有匹配的页都被丢弃。默认是 *

- `txt_chop_top`

取消顶部teletext行。默认是 1.

- `txt_format`

指定解码字幕的格式。这个teletext解码器可解码位图或简单文本，如果页面中有特定的图形和颜色则可能需要设置为"bitmap"，因为简单文本不能包含这些内容。否则可以选用"text"以简化处理。默认是bitmap

- `txt_left`

位图X偏移量, 默认是 0.

- `txt_top`

位图Y偏移量, 默认是 0.

- `txt_chop_spaces`

在生成的文本中保留前导和尾随的空格去除空行。该选项常用于字幕需要显示为多行（双行）使得字幕放置的更好看。默认是1。

- `txt_duration`

teletex 页面或者字幕显示的时间，单位是ms，默认值是3000，即30秒

- `txt_transparent`

让生成的teletext位图透明，默认是0，表示有一个不透明（黑）的背景色。

15 编码器

编码器是ffmpeg用来编码多媒体流的配置单元。

当编译生成ffmpeg时，所有内置编码器默认被支持。可以通过手动设置 `--enable-lib` 选项以支持外部（扩展）库。可以在配置选项中利用 `--list-encoders` 了解所有可能的编码器

可以利用 `--disable-encoders` 禁用所有编码器，也可以单独的利用 `--enable-encoder=ENCODER / --disable-encoder=ENCODER` 启用/禁用个别的编码器。

在ffmpeg工具集中利用选项 `-encoders` 可以了解支持的编码器。

16 音频编码器

介绍当前可用的音频编码器

aac

AAC（Advanced Audio Coding）编码器

当前原生（内置）编码器还处于实验阶段，而且只能支持AAC-LC（低复杂度AAC）。要使用这个编码器，必须选择‘experimental’或者‘lower’

因为当前还处于实验期，所以很多意外可能发生。如果需要一个更稳定的AAC编码器，参考 libvo-aacenc ,然而它也有一些负面报告。

aac选项

b

设置码率，单位是bits/s，是自动恒定比特率(CBR)模式的码率

q

设置为可变比特率（VBR）模式。此选项仅用于ffmpeg命令行工具。库接口是`global_quality`

stereo_mode

设置立体声编码模式，可能值有：

‘auto’

在编码时自动判断

‘ms_off’

禁止中端的（即2.0声道，而不是2.1声道）编码，这时默认值

‘ms_force’

强制中端编码，即强制2.1声道 encoding.

aac_coder

设置AAC编码方法，可能值：

‘faac’

FAAC-启发方法.

这个方法是显示方法的简化实现版，它为频带能量比设置阈值，降低所有量化步骤找到合适的量化失真阈值，对低于频带阈值的频带进行编码

这种方法的质量稍微好于下面介绍的两回路搜索法，但很慢

‘anmr’

基于网格的ANMR（平均噪音Average noise to mask ratio)掩比方案

理论上它效果最好，但很慢

‘twoloop’

双环搜索(Two loop searching)法

该方法首先根据波段阈值量化并试图通过添或调整个别量化点减去一个特征值得到一个最佳组合

这种方法和FAAC方法质量相当，是默认值
'fast'

固定量化法

该方法设置所有带定量化，这是最快的方法，但质量最差

ac3和ac3修订版

AC-3音频编码器

这一编码器定义在 ATSC A/52:2010 和ETSI TS 102 366,以及RealAudio 3 (通过dnet)

AC3编码器使用浮点运算，而ac3_fixed编码器仅用定点整数的数学运算。这并不意味着一个人总是更快，只是一个或另一个可能更适合一个特定的系统。浮点编码通常会产生一个给定的比特率，更好的音频质量。ac3_fixed编码器没有任何输出格式的默认编码，所以它必须显式使用选项 `-acodec ac3_fixed` 指定。

AC-3元数据

AC-3元数据选项用于设置音频参数的描述，它们大多数情况下不影响音频编码本身。这些选项不直接影响比特流或影响解码播放，只是提供信息。几个选项会增加音频数据比特输出流，从而影响输出质量。这将在下面的选项列表中注记出来：

下面文档介绍了几个公开文件文档：

- [A/52:2010 Digital Audio Compression \(AC-3\) \(E-AC-3\) Standard](#)
- [A/54 Guide to the Use of the ATSC Digital Television Standard](#)
- [Dolby Metadata Guide](#)
- [Dolby Digital Professional Encoding Guidelines](#)

AC-3元数据控制选项

- `-per_frame_metadata boolean`

允许每个框架的元数据。指定编码器应该检测每帧变化的元数据

0

初始化的元数据用于每帧（不再管变化，默认）

1

每帧都要检测元数据改变

AC-3中置混合水平

- `-center_mixlev level`

AC-3中置混合水平。该值决定编码时根据立体声产生中置音量的标准。它只会写入存在中置通道的输出中。该值为规模因子，有3个有效值：

0.707

应用 -3dB增益

0.595

应用 -4.5dB增益 (默认值)

0.500

应用 -6dB增益

- surround_mixlev level

环绕混合水平。适用于环绕声道增益。它只会写入存在环绕声通道的输出中。该值为规模因子，有3个有效值：

0.707

应用 -3dB增益

0.500

应用 -6dB增益 (默认值)

0.000

静默环绕声道(即没有环绕)

AC-3音频制作信息

音频制作信息描述了可选的混合环境信息，应用中要么都没有，要么同时有两个（即下面两个需要同时设置/或不设置）

- mixing_level number

混合水平. 指定在环境中混合的峰值声压级（SPL-Specifies peak sound pressure level）。有效值是80 - 111或者-1（表示未知）或不指定。默认值为-1，但如果 room_type 不为默认值，则 mixing_level 不能为-1.

- room_type type

空间类型。介绍了混音环境。是按大房间还是按小房间。如果没有指定 mixing_level 则写入默认值

0 notindicated

没有指定（默认）

1 large

大房间

2 small

小房间

其他AC-3元数据选项

- copyright boolean

版权指示。

0 off

不包含 版权(默认iansbaq)

1 on

版权信息

- dialnorm value

对话常态化。表明对于低于平均值的音量保持原样（0dBFS）。 这个参数决定了匹配源的目标音量。值过小会导致相对于源没有变化。有效值为整数，范围为 -31至-1， -31是默认值。 -dsur_mode mode

杜比环绕模式。指定是否使用杜比环绕立体声信号。只对音频流是立体声的输出有效。使用了这个选项并不意味着实际处理会产生杜比环绕。

0 notindicated

未指定（默认）

1 off

不采用

2 on

采用杜比环绕解码

-original boolean

原始流指示器。指音频是原始源而不是副本。

0
off

非原始源

1
on

原始源（默认）

其他扩展比特流信息

这些扩展比特流选项都被定义在A/52:2010标准的附录D中。它分为2个部分（组）。如果组中任意一个参数被指定，则组中所有的值将以默认值写入到流中。如果 `mixing levels` 被设置，则对支持备用比特流语法（`Alternate Bit Stream Syntax`）的解码器将采用这个值以替代 `center_mixlev` 和 `surround_mixlev` 选项定义。

其他扩展比特流信息 第一部分

- `-dmix_mode mode`

优化立体声缩混模式。允许在Lt/Rt (杜比环绕)或者Lo/Ro (常规立体声) 作为优化立体声缩混模式

0 notindicated

未指定 (默认)

1 ltrt

Lt/Rt 缩混优化

2 loro

Lo/Ro 缩混优化

- `-ltrt_cmixlev level`

Lt/Rt模式下中置混合层次。在Lt/Rt模式下解码器输出中置通道的增益

1.414

使用+3dB增益

1.189

使用 +1.5dB增益

1.000

使用 0dB

0.841

使用 -1.5dB

0.707

使用 -3.0dB

0.595

使用 -4.5dB 增益（默认值）

0.500

使用 -6.0dB

0.000

禁用中置通道

- `-ltrt_surmixlev level`

Lt/Rt模式下环绕增益。在Lt/Rt模式下解码器输出环绕通道的增益

0.841

使用 -1.5dB

0.707

使用 -3.0dB

0.595

使用 -4.5dB 增益（默认值）

0.500

使用 -6.0dB

0.000

禁用环绕通道

- `-loro_cmixlev level`

Lo/Ro模式下中置混合层次。在Lo/Ro模式下解码器输出中置通道的增益。

1.414

使用+3dB增益

1.189

使用 +1.5dB增益

1.000

使用 0dB

0.841

使用 -1.5dB

0.707

使用 -3.0dB

0.595

使用 -4.5dB

0.500

使用 -6.0dB 增益(默认值)

0.000

禁用中置通道

- `-loro_surmixlev level`

Lo/Ro模式下中置混合层次。在Lo/Ro模式下解码器输出环绕通道的增益。

0.841

使用 -1.5dB

0.707

使用 -3.0dB

0.595

使用 -4.5dB

0.500

使用 -6.0dB 增益(默认值)

0.000

禁用环绕通道

其他扩展比特流信息 第二部分

- -dsurex_mode mode

Dolby环绕EX模式. 标识是否使用Dolby环绕EX模式（7.1矩阵转5.1）.使用了此选项并不意味着编码器将实际应用Dolby环绕EX模式

0 notindicated

未标识 (default)

1 on

Dolby环绕EX模式关闭

2 off

Dolby环绕EX模式打开

- -dheadphone_mode mode

杜比耳机模式。标识编码为杜比耳机（多通道矩阵合成为2个声道）使用这个选项并不意味着实际应用了杜比耳机模式。

0 notindicated

未标识 (default)

1 on

Dolby 耳机模式关闭

2 off

Dolby 耳机模式打开

- `-ad_conv_type type`

A/D（模数转换）转换类型。标识音频需要HDCD A/D 转换。

0 standard

标准 A/D转换（默认）

1 hdc

HDCD A/D 转换

其他AC-3编码器选项

- `-stereo_rematrixing boolean`

Stereo 再混（Rematrixing）。通过Enables/Disables 来对应立体声输入。它是可选功能，通过选择左/右而当作立体声输出，从而提高输出效果。默认是启用的。因为该编码器会增加程序现象，所以只建议用于测试。

浮点AC-3编码特有选项

这些选项只在浮点AC-3编码时有效，整形AC-3时是不起作用的。

- `-channel_coupling boolean`

Enables/Disables通道的耦合。这是一个可选的音频选项，它从多个通道中获取高频带信息整合输出到一个通道中。这允许更多的比特位用于较低频率音频的同时保持足够的信息重建高频部分。这个选项对浮点AC-3来说主要用于测试或者提高编码速度。

-1 auto

由编码器选择（默认）

0 off

禁用通道耦合

1 on

允许通道耦合

- `-cpl_start_band number`

耦合开始带。设置通道耦合的开始带，从1-15可选。如果设置的值大于通道数，则处理为需耦合最后通道减一。如果auto（自动）被设置，则开始带将根据码率、通道布局、采样率有编码器自动计算。如果通道耦合设置为禁用，则本选项失效。

-1 auto

由编码器 选择(默认)

flac

FLAC(自由低损失音频编码——Free Lossless Audio Codec)编码器

flac选项

下面是FFmpeg中flac编码可用选项

- compression_level

设置压缩级别，如果没有显式设置即采用默认值

- frame_size

设置各个通道的帧大小

- lpc_coeff_precision

设置LPC系数精度，有效值从1到15, 15是默认值

- lpc_type

设置第一阶段LPC算法

‘none’

不采用LPC

‘fixed’

整数LPC

‘levinson’ ‘cholesky’

- lpc_passes

用Cholesky分解LPC的次数

- min_partition_order

最小分区顺序

- max_partition_order

最大分区顺序

- prediction_order_method

‘estimation’ ‘2level’ ‘4level’ ‘8level’ ‘search’

强力搜索

'log'

- `ch_mode`

通道模式

'auto'

自动模式，对每帧自动匹配通道

'indep'

通道独立编码

'left_side' 'right_side' 'mid_side'

- `exact_rice_parameters`

是精确还是近似.如果设置为1表示精确，会减慢编码速度以提高压缩率。

- `multi_dim_quant`

多维量化。如果设置为1，那么第二阶段LPC应用第一阶段结果进行算法调整。这很慢，但可以提高压缩率

libfaac编码

libfaac 是AAC（Advanced Audio Coding）编码器的再封装

要使用它需要libfaac头文件和库存在配置。你还需要在编译ffmpeg时通过 `--enable-libfaac` `--enable-nonfree` 进行配置。

该编码器高质量版本参考[\[aacenc\]](#)

对更多信息，参考libfaac项目介绍<http://www.audiocoding.com/faac.html/>

libfaac相关选项

下面是ffmpeg工具编码时的可用选项

下面的选项适用于libfaac封装，`faac-XXXX` 等效选项列在括号中

- `b (-b)`

设置ABR（平均码率），单位bits/s。如果码率没有特别指定，会自动匹配所选特性（属性配置）。faac比特率单位是kilobits/s。

注意libfaac不支持CBR(Constant Bit Rate——固定码率)，只支持ABR (Average Bit Rate——平均码率)。

如果VBR模式设置为允许，则本选项被忽略

- `ar (-R)`

设置音频采样率，单位Hz

- ac (-c)

设置音频通道数

- cutoff (-C)

设置截至频率。如果没有设置或者设置为0，则自动根据库计算。默认为0

- profile

设置音频特性（属性配置）文件

下面的音频特性文件有效:

‘aac_main’

主要的AAC (Main)

‘aac_low’

低复杂度AAC (LC)

‘aac_ssr’

可扩展采样 (SSR)

‘aac_ltp’

长期预测(LTP—Long Term Prediction)

如果没有指定则表示为‘aac_low’.

- flags +qscale

设置VBR(动态码率Variable Bit Rate)下的品质

- global_quality

设置VBR下的品质， 其为一个数字或者lambda表达式

仅仅在VBR模式，且 flags +qscale 被设置为有效才起作用。它将被 FF_QP2LAMBDA 转换成QP值，并应用于libfaac。QP值的范围为[10-500]，越大品质越好

- q (-q)

允许VBR模式，但设置为负数值，则值作为双精度浮点数

值应用于libface。值的可能范围是[10-500]，数字越大QP值越高。

选项只用于ffmpeg命令行，或者通过 `global_quality` 属性描述问中rs。

libfaac例子

- 使用 `ffmpeg` 把一个音频转换为ABR 128kbps AAC编码格式流放置在M4A（MP4音频）文件中

```
ffmpeg -i input.wav -codec:a libfaac -b:a 128k -output.m4a
```

- 使用 `ffmpeg` 把一个音频转换为VBR AAC编码（采用LTP AAC）格式流放置在M4A（MP4音频）文件中

```
ffmpeg -i input.wav -c:a libfaac -profile:a aac_ltp -q:a 100 output.m4a
```

libfdk aac ###libfdk-aac

libfdk-aac的再封装

该库只用于Fraunhofer FDK AAC 编码格式

要使用，必须有libfdk-aac有头和预配的库，并在编译ffmpeg时用配置选项 `--enable-libfdk-aac` 启用。这个库可能不兼容于GPL，如果你要使用GPL，你必须 `--enable-gpl --enable-nonfree --enable-libfdk-aac`

这个编码器被认为品质高于 内置的[AACenc] 和[libfaac]

VBR编码可以通过 `vbr` 或者 `flags + qscale` 选项启用，它们是实验性质的，只适合于某些参数组合。

libfdk-aac 0.1.3或者更高版本支持7.1声道

更多信息请参考fdk-aac项目<http://sourceforge.net/p/opencore-amr/fdk-aac/>

libfdk-aac选项

下面是可用的一些选项：

- `b`

设置码率，如果未指定，则根据属性特性自动匹配

如果工作于VBR模式，本选项被忽略、

- `ar`

设置采样率 (单位Hz).

- `channels`

设置通道数

- `flags + qscale`

可以调整品质，VBR (Variable Bit Rate)模式。注意 当vbr为正表示VBR模式被隐含启用

- `cutoff`

设置截止频率，如果没有设置或者为0，表示自动根据库计算默认为0

- `profile`

设置音频属性预设文件，可以有：

‘aac_low’

低复杂度AAC(LC)

‘aac_he’

高效率AAC（HE-AAC）

‘aac_he_v2’

高效率AAC版本2（HE-AACv2）

‘aac_ld’

低延迟AAC(LD)

‘aac_eld’

增强低延迟AAC（ELD）

如果没有特别指定则为‘aac_low’.

下面是libfdk_aac私有选项

- afterburner

设置为1表示允许助力，否则为0，它可以增强品质，但要求更多处理能力。

默认为1

- eld_sbr

1表示允许对ELD采样SBR (Spectral Band Replication-频带复制)，否则为 0.

默认为 0

- signaling

设置SBR/PS 指令方式

接受下面的值：

‘default’

选择含蓄信号模式（默认明确为 hierarchical，如果全局头设置为禁止，则隐式表达）

‘implicit’

隐式向后兼容指令信号

'explicit_sbr'

明确为SBR，隐式PS信号

'explicit_hierarchical'

明确为hierarchical信号

默认为'default'.

- latm

设为1表示输出LATM/LOAS封装数据，否则为0

默认为0

- header_period

设置StreamMuxConfig 和 PCE 重复周期 (在帧上)，把LATM/LOAS包含着配置发送缓冲中

必须为16bit的非负整数

默认为 0.

- vbr

设置VBR模式，从1最低品质（但仍足够好），5是最高品质。如果值为0表示禁用VBR，而是采用CBR（固定码率）

当前只有 'aac_low'属性预设支持VBR

VBR模式1-5代表的平均码率:

'1'

32 kbps/channel

'2'

40 kbps/channel

'3'

48-56 kbps/channel

'4'

```
64 kbps/channel
```

```
'5'
```

```
about 80-96 kbps/channel
```

默认0.

libfdk_aac 例子

- 转换为VBR AAC M4A

```
ffmpeg -i input.wav -codec:a libfdk_aac -vbr 3 output.m4a
```

- 转换为CBR 64k AAC，使用高效率AAC属性预设

```
ffmpeg -i input.wav -c:a libfdk_aac -profile:a aac_he -b:a 64k output.m4a
```

libmp3lame

LAME (Lame Ain't an MP3 Encoder) MP3 编码器封装

需要在编译时配置 libmp3lame 头和库，并且显式设置 `-enable-libmp3lame`

参考 libshine 这个整数修正MP3编码器（虽然质量较低）

libmp3lame选项

下面是支持的选项（lame-XXX等效选项列在括号中）：

- b (-b)

设置码率CBR/ABR，单位为bits/s，LAME的码率为kilobits/s\

- q (-V)

设置VBR下的品质。它只用于ffmpeg命令行工具，对于库接口，使用 `global_quality`

- compression_level (-q)

设置算法品质。通过0-9的参数，表示不同的品质，0最高但最慢，9最低但最快

- reservoir

为1（默认）表示允许bit池，否则为0。LAME 也是默认允许但可以被 `--nores` 覆盖

- joint_stereo (-m j)

1表示允许在（每帧）中编码L/R立体声或者mid/side立体声。默认为1 Default value is 1.

- abr (--abr)

为1表示允许ABR，lame `--abr` 设置为一共的码率，这里只是表示采用ABR，码率还是由 `b` 设置

libopencore-amrnb

开放核心自适应多速率窄带(OpenCORE Adaptive Multi-RateNarrowband)编码器

需要相应的头和库进行编译，并利用 `--enable-libopencore-amrnb --enable-version3` 以允许编译配置

是单声道编码器，常用于8k采样率，可以通过设置 `strict` 到 `unofficial` 或者更低来选用更低采样率

libopencore-amrnb选项

- `b`

设置码率，只有下面的码率被支持，设置为其他值将自动用最近的替代

4750 5150 5900 6700 7400 7950 10200 12200

- `dtx`

设置为1表示允许连续传输(产生少量噪音)，默认为0

libshine

Shine整形Mp3编码的封装

Shine是一种整形MP3编码器，它在没有FPU（浮点协处理）的平台上可以更快更好，例如一些armel CPU或者一些电话或者平板上。但是不要期望获得更好的品质（与LAME或者其他产品级编码器比较）。同时，根据项目主页，该编码器可能并不提供给免费bug修正，代码是很久以前写的，已经有5年以上没有更新了。

只支持立体声和单声道，而且是CBR模式。

项目在（最后更新2007年）<http://sourceforge.net/projects/libshine-fxp>，我们的支持更新放置在github上的Savonet/Liquidsoap中，地址是<https://github.com/savonet/shine>

需要头和库支持，并需要配置 `--enable-libshine` 打开编译

参考[libmp3lame]

libshine选项

这个库封装支持如下选项，其对应的 `shineenc-xxxxx` 形式等效选项列在括号中

- `b (-b)`

设置CBR码率，单位bits/s，`shineenc -b` 单位是kilobits/s

libtwolame

双Lame Mp2 编码器封装

编译需要头和库，并且显式打开 `--enable-libtwolame`

libtwolame选项

下面是支持的选项，等效的 `libtwolame-xxx` 选项列在括号中

- `b (-b)`

设置CBR码率单位bits/s，twolame会扩展为以kilobits/s为单位。默认128k

- `q (-V)`

对VBR设置品质等级，从-50至50，常见范围为-10-10.越高品质越好。只适用于ffmpeg命令行，接口需要使用 `global_quality`。

- `mode (--mode)`

设置结果音频模式，允许如下参数：

`'auto'`

基于输入自动适配模式，选项默认值。

`'stereo'`

立体声

`'joint_stereo'`

Joint立体声

`'dual_channel'`

双声道

`'mono'`

单声道

- `psymodel (--psyc-mode)`

为1设置为psychoacoustic（心理声学）模式，接受-1到4的参数，越大效果越好，默认为3

- `energy_levels (--energy)`

为1设置能量扩展模式，否则为0（默认）(disabled).

- `error_protection (--protect)`

为1设置CRC错误保护，否则为0（默认）

- `copyright (--copyright)`

为1设置MPEG音频复制标志，否则为0（默认）

- `original (--original)`

为1设置MPEG音频原音标志，否则为0（默认）

libvo-aacenc

VisualOn AAC编码器

编译时需要头和库文件，以及利用配置选项 `--enable-libvo-aacenc --enable-version3` 打开

它类似于[原生FFmpeg AAC],可以处理多个源

libvo-aacenc选项

VisualOn AAC编码器只支持AAC-LC和最多2个声道，而且是CBR

- b

码率，单位秒

libvo-amrwbenc

VisualOn 自适应多速率宽带编码器

编译时需要头和库文件，以及利用配置选项 `--enable-libvo-amrwbenc --enable-version3` 打开

只支持单声道，通常为16000Hz采样，可以通过设置 `strict` 和 `unofficial` 来覆盖为更低采样

libvo-amrwbenc选项

- b

设置码率，单位bits/s，只允许下列参数，否则自动选取最接近的有效参数

'6600' '8850' '12650' '14250' '15850' '18250' '19850' '23050' '23850'

- dtx

为1允许连续传输 (产生少量噪音)，默认为0

libopus

libopus（Opus交互音频编码）的封装

编译时需要头和库文件，以及利用配置选项 `--enable-libopus` 打开

libopus

更多选项可以通过opus-tools的 `opusenc` 查询，下面仅仅是一些封装中支持的选项（对应的 `opusenc-xxxx` 选项列在括号中）：

- b (bitrate)

设置码率，单位 bits/s, opusenc 中单位为kilobits/s.

- vbr (vbr, hard-cbr, and cvbr)

设置VBR模式，下面为有效参数，其等效于opusenc中对应参数：

‘off (hard-cbr)’

使用CBR码率控制

‘on (vbr)’

使用合适的动态码率（默认）

‘constrained (cvbr)’

使用约束变比特率编码

- compression_level (comp)

设置集编码算法复杂度。有效参数是0-10整数，0最快，但质量最差，10最慢，质量最好，默认为10

- frame_duration (framesize)

设置最大帧尺寸，或者帧对应毫秒时间。有效参数为：2.5, 5, 10, 20, 40, 60,越小的帧延迟越低，但会降低编码率控制质量，尺寸大于20ms在低码率时有较有趣表现，默认20ms

- packet_loss (expect-loss)

设置预期分组丢失率，默认为0

- application (N.A.)

设置预期应用类型，下面为有效参数：

‘voip’

有利于提高语音清晰度

‘audio’

有利于音频输入，默认值

‘lowdelay’

有利于低延迟模式

- cutoff (N.A.)

设置截止屏幕，单位Hz。参数必须是：4000, 6000, 8000, 12000, 或者 20000（分别对应媒体带宽窄带、常规、宽带、超宽带和全频），默认为0，表示禁用cutoff。

libvorbis

libvorbis编码器封装

编译要求头文件和库，还需要专门用 `--enable-libvorbis` 以允许使用

libvorbis选项

下面的选项支持libvorbis封装。等效的 `oggenc-xxx` 选项部分列在括号中。

为了更多的了解 `libvorbis` 和 `oggenc` 选项，请参考<http://xiph.org/vorbis/>和<http://wiki.xiph.org/Vorbis-tools>，以及 `oggenc(1)` 手册

- `b (-b)`

设置ABR模式码率，单位bits/s。oggenc-b单位是kilobits/s.

- `q (-q)`

设置VBR的品质。选项参数是浮点数，范围-1.0至10.0，越大越好，默认‘3.0’.

该选项只用于ffmpeg命令行工具，要在库中使用则需要 `global_quality`

- `cutoff (--advanced-encode-option lowpass_frequency=N)`

设置截止频率，单位Hz，如果为0表示禁用截止频率。 `oggenc` 等效选项单位是kHz。默认值为‘0’ (表示不设置截止频率)。

- `minrate (-m)`

设置最小码率，单位bits/s. oggenc -m 的单位是kilobits/s.选项只在ABR模式起效

- `maxrate (-M)`

设置最大码率，单位bits/s. oggenc -M 的单位是kilobits/s. 选项只在ABR模式起效

- `iblock (--advanced-encode-option impulse_noisetune=N)`

设置负偏压（底噪偏置），选项参数是浮点数，范围-15.0-0.0。指示编码器花费更多资源用于瞬态，这样可以使得获得更好的瞬态响应。

libwavpack

wavpack的通过libwavpack的封装

当前只支持无损32位整数样本模式

编译要求头文件和库，还需要专门用 `--enable-libwavpack` 以允许使用

注意libavcoder原生编码器已支持wavpack编码，而不用使用这个扩展编码器了。相关参考[wavpackenc]

libwavpack选项

wavpack命令行工具相应选项都列在括号中：

- `frame_size (--blocksize)`

默认32768.

- `compression_level`

设置速度与压缩的平衡，允许下面的参数：

‘0 (-f)’

快速模式。

‘1’

常规模式（默认）

‘2 (-h)’

高质量模式

‘3 (-hh)’

非常高质量模式

‘4-8 (-hh -xEXTRAPROC)’

类似‘3’，但允许扩展处理 `enabled`。

‘4’ 类似于 `-x2` ‘8’ 类似于 `-x6`。

wavpack

wavpack无损音频压缩

是libavcodec的原生wavpack编码。这个编码也可以利用 `libwavpack` 完成，但现在看来完全没有必要。

参看[libwavpack]

wavpack选项

等效的wavpack命令行工具列在括号中

wavpack通用选项

下面是wavpack编码的通用选项，下面只介绍了个别特别用于wavpack的选项，其他更多选项参考[编码选项部分]

- `frame_size` (`--blocksize`)

对于这个编码器，参数范围128 至 131072。默认为自动检测（根据采样率和通道数）

为了了解完整的计算公式，可以看 `libavcodec/wavpackenc.c`

- `compression_level` (-f, -h, -hh, and -x)

这个选项同于libwavpack的语法

wavpack私有选项

- `joint_stereo` (-j)

设置是否启用联合立体声， 下列值有效:

'on (1)'

强制mid/side （中置和边）音频编码

'off (0)'

强制left/right音频编码

'auto'

自动检测

- `optimize_mono`

设置是否允许对单声道优化。此选项只对非单声道流有效。 可能值：

'on'

允许

'off'

禁止

17 视频编码器

介绍一些当前有效的视频编码器

libtheora

libtheora的封装

编译需要头和库文件，还需要利用 `--enable-libtheora` 在配置中允许

更多信息参考<http://www.theora.org/>

libtheora选项

下面是映射给libtheora的全局选项，它们对品质和码率产生影响。

- b

对CBR（固定码率编码）设置码率，单位bit/s，在VBR（动态码率编码）模式下本选项被忽略。

- flags

设置是否允许 `qscale`标志（恒定质量模式—VBR模式下）在 `pass1` 和 `pass2``（2次编码方式）

- g

设置GOP尺寸

- global_quality

设置全局质量，在 `lambda`工具集中是一个整数单位的倍数

仅在VBR模式中，同时允许了 `+qscale`。这个值会除以`FF_QP2LAMBDA`转换为QP,范围为[0 - 10]，再乘以6.3得到本地有效libtheora范围[0-63]，越大质量越高

- q

仅作VBR模式下，设置为非负数。作为双精度浮点质量单位值，用于转换计算QP

值范围为 [0-10]，再乘以 6.3 将获得libtheora有效质量参数，范围[0-63]

这个选项仅用于ffmpeg命令行工具，库接口使用 `global_quality`

libtheora例子

- 使用最大恒定质量（VBR）编码：

```
ffmpeg -i INPUT -codec:v libtheora -q:v 10 OUTPUT.ogg
```

- 使用CBR 1000 kbps编码 Theora视频流：

```
ffmpeg -i INPUT -codec:v libtheora -b:v 1000k OUTPUT.ogg
```

libvpx

VP8/VP9格式支持，通过libvpx

编译需要头和库文件，还需要利用 `--enable-libvpx` 在配置中允许

libvpx选项

下面的选项被libvpx封装支持，部分等效的 `vpxenc-xxx` 类型的选项或者值列在括号中。

为了减少文件复制，只有私有的选项和一些需要特别注明（注意）的记录在这里，其他的请参考[10 编码]章节

为了了解更多关于libvpx的选项，可以在命令行中使用 `ffmpeg -h encoder=libvpx` 或者 `ffmpeg -h encoder=libvpx-vp9` 或 `vpxenc --help` 来获取。进一步信息可以在 `libvpx API` 文档中获取。

- `b (target-bitrate)`

设置码率，单位bits/注意FFmpeg中b选项的单位是bits/s，而在vpxenc中目标码率单位是kilobits/s。

- `g (kf-max-dist)`
- `keyint_min (kf-min-dist)`
- `qmin (min-q)`
- `qmax (max-q)`
- `bufsize (buf-sz, buf-optimal-sz)`

设置码率控制缓冲区大小(单位bits)。注意在vpxenc中是指定为多少milliseconds（毫秒），这个封装库通过下面的公式进行转换： $buf-sz = bufsize\ 1000 / bitrate$, $buf-optimal-sz = bufsize\ 1000 / bitrate * 5 / 6$.

- `rc_init_occupancy (buf-initial-sz)`

设置解码开始前需要加载到RC的预加载数, 注意vpxenc中指定多少 milliseconds（毫秒），这个封装库按下面公式转换： $rc_init_occupancy * 1000 / bitrate$.

- `undershoot-pct`

设置数据下冲（min）的目标比特率

- `overshoot-pct`

设置数据上冲(max)目标比特率

- `skip_threshold (drop-frame)`
- `qcomp (bias-pct)`
- `maxrate (maxsection-pct)`

设置GOP最大比特率，单位 bits/s，注意vpxenc描述这个为目标码率，这个封装中按如下公式计算： $(maxrate * 100 / bitrate)$.

- `minrate (minsection-pct)`

设置GOP最小比特率，单位 bits/s，注意vpxenc描述这个为目标码率，这个封装中按如下公式计算： $(minrate * 100 / bitrate)$.

- `minrate, maxrate, b end-usage=cbr`

$(minrate == maxrate == bitrate)$.

- `crf (end-usage=cq, cq-level)`

- quality, deadline (deadline)

'best'

使用最优质量期限，不是非常慢，这个选项指定可以有不低于`good`的输出质量（稍微慢一些）。

'good'

使用高质量期限，它在速度、质量，以及CPU使用间进行均衡。

'realtime'

使用实时质量期限

- speed, cpu-used (cpu-used)

设置质量/速度比，高的参数值将加大编码质量成本

- nr (noise-sensitivity)
- static-thresh

设置一个变化阈值，低于它将被编码器跳过

- slices (token-parts)

注意，FFmpeg指定的是切片分区总数，而vpxenc中是标记部分的log2值

- max-intra-rate

设置最大I帧比特率作为目标比特率的百分比，0表示不限

- force_key_frames

VPX_EFLAG_FORCE_KF

- Alternate reference frame related

auto-alt-ref

启用备用参考帧，只在2次编码的pass2起效

arnr-max-frames

设置altref降噪的最大帧数

arnr-type

设置altref降噪参考过滤类型：backward, forward, centered.

arnr-strength

设置altref降噪滤波强度

rc-lookahead, lag-in-frames (lag-in-frames)

设置向前参考帧码率控制

- error-resilient

允许错误弹性

- VP9-specific options

lossless

允许lossless（无损）模式

tile-columns

设置采用的tile columns数，**注意**这里参数是log2(tile_columns)值，例如 8 tile columns要设置 tile-columns 选项值为3。

tile-rows

设置采用的tile rows数， **注意**这里参数是log2(tile_rows)。例如 4 tile rows要设置 tile-rows 选项为2。

frame-parallel

允许并行可译特性

aq-mode

设置自适应量化模式：（0:关闭（默认），1：方差 2：复合，3：循环刷新）。

libwebp

WebP图片编码封装

libwebp是google提高的对于WebP图像格式的编码器，它提供任意有损/无损编码模式。有损图像本质上是对VP8框架的封装。无损图像由google单独编码器支持。

libwebp 像素格式

当前libwebp只支持YUV420的有损图像和RGB无损。两种模式都支持透明通道。因为API限制了进行RGB有损和YUV420无损编码时像素格式会自动转换使用libwebp库中要求的格式（暨无损用RGB，有损用YUV420）。所以这样做无意义，只是提供了接口。

libwebp选项

- -lossless boolean

允许/禁止无损编码，默认为0（禁止）

- -compression_level integer

对于有损，设置质量/速度比，高的值表示获取高质量（同样尺寸）需要更多编码成本（时间）。对于无损，是尺寸/速度比，高的值意味要获取小的尺寸需要更多的成本。更具体的说，就是它控制了额外算法和压缩工具的使用，这些工具的组合使用将影响编码质量/效率。它映射到libwebp选项，有效范围是0-6，默认为4

- -qscale float

对于有损编码，控制品质，范围0-100。对于无损编码，控制资源和时间花费在压缩更多。默认值为75.注意使用livavcodec时它对应于 `global_quality * FF_QP2LAMBDA` .

- -preset type

选取预置选项。提供一些常规可用设置：

none

不采用预置

default

默认预置

picture

数码图片，例如人像拍摄、室内拍摄、

photo

室外图像，自然光 lighting

drawing

手绘或者画线，具有高对比度的细节

icon

小尺寸彩色图像

text

文本之类的

libx264,libx264rgb

x264 H.264/MPEG-4 AVC 编码器封装

编译需要头和库文件，还需要利用 `--enable-libx264` 在配置中允许

libx264提供一些令人印象深刻的特性，包括8x8和4x4自适应空间变化，自适应B帧，CAVLC/CABAC 熵编码，交织（MBAFF），无损模式，物理优化细节保留（自适应量化、psy-RD，psy-trellis）等等

大多数libx264编码器选项均是映射值ffmpeg全局编码选项，仅有少量的是私有的，他们通过libx264中函数 `x264_param_parse`，`x264opts` 和 `x264-params` 提供的单个选项或 `key=value` 序列的多个选项

参考 <http://www.videolan.org/developers/x264.html>以了解更多x264项目内容。

libx264rgb和libx264类似，只是一个编码RGB像素格式，一个是针对YUV像素格式的。

支持的像素格式

x264支持8 到 10 bit的颜色空间。确切的颜色深度在x264配置时设置，在一个特定编译版本的FFmpeg中只支持一种颜色深度，换句话说就是不同位深需要多个版本的ffmpeg x264.

libx264 libx264rgb 选项

下面的选项被libx264（libx264rgb）封装支持，所有的等效 `x264-xxx` 形式的选项和值都列在括号中。

这里只列出了需要特别说明或者私有的选项，其他选项参考[10 编码选项]部分。

为了更多的了解关于libx264的选项，可以使用 `x264 --full-help`（需要x264命令行工具）或者参考 `libx264` 文档。

- b (bitrate)

设置码率，单位bits/s，注意FFmpeg的码率单位是bits/s,而x264中码率单位是kilobits/s.

- bf (bframes)
- g (keyint)
- qmin (qpmin)

最小量化尺度

- qmax (qpmax)

最大量化尺度

- qdiff (qpstep)

量化尺度最大差值

- qblur (qblur)

模糊量化曲线

- qcomp (qcomp)

量化曲线压缩因子

- refs (ref)

每一帧可以使用参考帧数，范围0-16。

- sc_threshold (scenecut)

设置场景变化检测阈值

- trellis (trellis)

执行网格量化以提高效率。默认情况下启用。

- nr (nr)
- me_range (merange)

像素运动最大搜索范围

- me_method (me)

设置运动估计方法。按速度递减顺序可能值：

'dia (dia)' 'epzs (dia)'

半径为1菱形搜索 (fastest)。'epzs'是'dia'的别名

'hex (hex)'

半径为2的正六边形搜索。

'umh (umh)'

多层次六边形搜索。

'esa (esa)'

穷举搜索。

'tesa (tesa)'

Hadamard(阿达玛)穷举搜索(最慢)。

- subq (subme)

亚像素运动估计方法。

- b_strategy (b-adapt)

自适应B帧布局决策算法。仅第一次使用。

- keyint_min (min-keyint)

最小 GOP 尺寸

- coder

设置熵编码器，可能值：

‘ac’

允许CABAC。

‘vlc’

允许CAVLC而且禁止 CABAC。它类似于x264中的`--no-cabac`

- cmp

设置全像素运动估计比较算法，可能值：

‘chroma’

允许chroma

‘sad’

忽略chroma，其等效于 x264中的`--no-chroma-me`

- threads (threads)

编码线程数

- thread_type

设置多线程技术，可能值：

‘slice’

切片多线程，它等效于x264中的`--sliced-threads`

‘frame’

基于帧的多线程

- flags

设置编码标志，它和 `-cgop` 配合可以用来关闭GOP或者打开GOP，类似于x264中的 `--open-gop`

- rc_init_occupancy (vbv-init)

- preset (preset)

设置编码预置

- tune (tune)

设置编码参数整定

- profile (profile)

设置配置文件的限制。

- fastfirstpass

参数为1则当第一次编码（pass1）允许快速设置，参数为0，表示禁止快速设置（等效于x264的 `--slow-firstpass`）

- crf (crf)

设为质量恒定模式（类VBR）

- crf_max (crf-max)

CRF模式下，防止VBV降低质量超越的阈值

- qp (qp)

设定量化率控制方法参数。

- aq-mode (aq-mode)

设置AQ方法，可能值

‘none (0)’

禁止。

‘variance (1)’

方差AQ（复杂蒙版）。

‘autovariance (2)’

自动方差AQ（实验）。

- aq-strength (aq-strength)

设置AQ强度，减少阻塞平面和纹理区域模糊。

- psy

为1表示使用视觉优化。为0则禁用（等效 x264的 `--no-psy`）

- psy-rd (psy-rd)

在psy-rd : psy-trellis中设置视觉优化强度

- rc-lookahead (rc-lookahead)

设置向前预测参考帧数。

- weightb

为1设置帧加权预测，否则为0表示禁止（等效于x264的 `--no-weightb`）

- weightp (weightp)

设置P帧加权预测法，可能值：

'none (0)'

禁止

'simple (1)'

使用加权参考

'smart (2)'

使加权文献和重复

- ssim (ssim)

允许在编码结束后输出SSIM

- intra-refresh (intra-refresh)

为1表示使用周期内刷新代替IDR帧设置

- avcintra-class (class)

配置编码器生成AVC-Intra，有效值50，100，200

- bluray-compatible (bluray-compatible)

配置兼容蓝光标准，是 "bluray-compatible=1 force-cfr=1"的简写

- b-bias (b-bias)

设置B帧如何被影响

- b-pyramid (b-pyramid)

设置保持一些B帧作为参考集的方法，允许值：

'none (none)'

禁用。

'strict (strict)'

严格的分层金字塔

'normal (normal)'

Non-strict (非蓝光兼容)。

● mixed-refs

为1表示每个分区使用一个参考，而不是每个宏块一个参考，否则为0，其等效于x264的 `--no-mixed-refs`

● 8x8dct

为1指采用自适应空间变换矩阵大小 (8x8变换)，否则为0，等效于x264的 `--no-8x8dct`

● fast-pskip

为1表示早期跳过检查。等效于x264的 `--no-fast-pskip`

● aud (aud)

为1启用访问单元分隔设置

● mbtree

为1表示允许使用宏块树，否则（为0）等效于x264的 `--no-mbtree`

● deblock (deblock)

设置环路滤波参数，参数型为alpha:beta

● cplxblur (cplxblur)

QP波动减少（压缩前曲线压缩）

● partitions (partitions)

设置分区规格，参考后面逗号分隔的列表，可能值有：

'p8x8'

8x8 P帧 分区

'p4x4'

4x4 P帧 分区。

‘b8x8’

4x4 B帧分区

‘i8x8’

8x8 I帧分区 .

‘i4x4’

4x4 I帧分区 (‘p4x4’的前提是‘p8x8’也被设置, 允许‘i8x8’ 则需要设置了8x8dct被允许)

‘none (none)’

不考虑分区

‘all (all)’

考虑所有可能分区

● direct-pred (direct)

设置直接MV预测模式, 可能值:

‘none (none)’

禁止MV预测

‘spatial (spatial)’

使空间预测

‘temporal (temporal)’

使时间的预测

‘auto (auto)’

自动识别

● slice-max-size (slice-max-size)

设置每个分片的字节大小限制, 单位字节, 如果不设置但RTP载荷设置了就使用RTP载荷

- stats (stats)

设置多次编码的文件名称

- nal-hrd (nal-hrd)

设置HRD信息信号 (要求vbr-buftype被设置). 可能值:

'none (none)'

禁用HRD信息信号

'vbr (vbr)'

可变比特率

'cbr (cbr)'

固定比特率 (MP4容器不允许).

- x264opts (N.A.)

设置任意的x264选项, 参看x264 --fullhelp 以获取列表

参数是一个由':'分隔的 key=value 序列。对于 filter 和 psy-rd 选项, 也是有":"被','代替作为分隔符。

例如, 要指定使用libx264编码:

```
ffmpeg -i foo.mpg -vcodec libx264 -x264opts keyint=123:min-keyint=20 -an out.mkv
```

- x264-params (N.A.)

使用 : 分隔的 key=value 参数覆盖x264配置,

这个选项类似x264opts, 但其兼容Libav

例如:

```
ffmpeg -i INPUT -c:v libx264 -x264-params level=30:bframes=0:weightp=0:\ cabac=0:ref=1:vbr-maxrate=768:vbr-buftype=2000:analyse=all:me=umh:\ no-fast-pskip=1:subq=6:8x8dct=0:trellis=0 OUTPUT
```

此外编码 ffpresets 还支持一些通用的选项, 可以参考前述[预置]相关文档。

libx265

x265 H.265/HEVC 编码器封装

编译需要头和库文件, 还需要利用 `--enable-libx265` 在配置中允许

libx265选项

- preset

设置x265预置

- tune

设置x265可调参数

- x265-params

使用':'分隔的 `key=value` 列表进行选项设置，参考 `x265 --help` 获取支持的选项

例如采用libx265,并利用-x265-params进行选项设置:

```
ffmpeg -i input -c:v libx265 -x265-params crf=26:psy-rd=1 output.mp4
```

libxvid

Xvid MPEG-4 Part 2 封装

编译需要livxvidcore头和library库文件，还需要利用 `--enable-libxvid` `--enable-gpl` 在配置中允许

当前原生的 `mpeg4` 编码器支持MPEG-4 Part 2格式，所以不一定需要这个库了。

libxvid选项

下面选项是libxvid封装支持的选项，其中部分只列出，而没有文档介绍是因为其同[10 编码选项]中通用选项一致，其它没有列出的通用选项则在库中无效。

- b
- g
- qmin
- qmax
- mpeg_quant
- threads
- bf
- b_qfactor
- b_qoffset
- flags

设置编码标志，可能值:

'mv4'

对宏块使用4个运动检测

'aic'

允许高品质AC预测

'gray'

只编码灰度

'gmc'

全局运动补偿 (GMC)。

'qpel'

1/4像素运动补偿

'cgop'

关闭GOP。

'global_header'

在每个关键帧放置全局头extradata

- trellis
- me_method

设置运动估计方法.按速度降低, 质量增加排列的可能值:

'zero'

不使用运动估计方法 (默认)。

'phods' 'x1' 'log'

启用16x16块和16x16块半像素细化进行菱形区域搜索, 'x1'和'log'是'phods'别名

'epzs'

允许前述所有值, 再加上8x8菱形区域搜索, 8x8半像素细化, 并在色度平面进行运动估计

'full'

允许所有的 16x16和8x8 区域搜索

- mbd

设置宏块选择算法, 依质量提高的可能值:

'simple'

使用宏块比较函数算法 (默认)。

'bits'

允许16x16块半像素和1/4像素细化失真估计

'rd'

允许上述所有可能值，再加上8x8块半像素和1/4像素细化失真估计，并采用方形图案失真估计进行搜索。

- lumi_aq

为1允许lumi遮蔽自适应量化，默认为0 (禁止).

- variance_aq

为1允许方差的自适应量化,默认为0 (禁止).

如果结合lumi_aq,由此产生的质量不会比任何一个单独规定。换句话说，所得到的质量会差于单独使用任何一个选项的效果。

- ssim

设置结构信息（SSIM）显示方法。可能的值：

'off'

禁止SSIM信息

'avg'

在编码后输出平均SSIM。格式为：

Average SSIM: %f

对那些不熟悉C的用户，f表示浮点数或者小数（例如 0.939232）

'frame'

在编码过程中输出每帧SSIM，并且在编码结束后输出平均SSIM，每帧信息格式为：

SSIM: avg: %1.3f min: %1.3f max: %1.3f

对那些不熟悉C的用户，%1.3f表示3位小数的浮点数(例如0.932)。

- ssim_acc

设置SSIM精度。可用的选项参数是在0-4范围的整数，而0给出了最准确的结果和计算速度最快的4。

mpeg2

MPEG-2编码器

mpeg2选项

- seq_disp_ext integer

指定是否写一个 sequence_display_extension到输出

-1 auto

自动检测是否写，是默认值，如果数据被写入不同于默认或指定的值则判断是否写

0 never

从不写

1 always

一直写

png

png图像编码器

png选项

dpi integer

设置像素的物理密度，每英寸点数，没有默认设置

dpm integer

设置像素的物理密度，每米点数，没有默认设置

ProRes

Apple ProRes编码器

FFmpeg包含2种ProRes编码器，prores-aw和prores-ks。它们可以由 `-vcodec` 选项指定

prores-ks私有选项

- profile integer

选择ProRes属性（预置）配置来编码，可能值：

'proxy' 'lt' 'standard' 'hq' '4444'

- quant_mat integer

选择的量化矩阵,可能值：

'auto' 'default' 'proxy' 'lt' 'standard' 'hq'

如果选择auto, 匹配属性的量化矩阵会被选中, 如果没有设置, 则选择最高质量的量化矩阵

- bits_per_mb integer

分配的宏块位, 不同的属性在200-2400间, 最大值为8000

- mbs_per_slice integer

每个切片中宏块数 (1-8), 默认为8, 几乎是所有情况下最好值

- vendor string

重写4字节的供应商ID。例如apl0这个自定义供应商ID会被认为是由苹果编码器产生。

- alpha_bits integer

指定alpha分量的比特数。可能的值是0, 8和16。用0禁用alpha平面编码

速度考虑

在默认操作模式下, 编码器以高质量为目的 (即在不产生超过要求的帧数据限定下, 使输出质量尽可能好)。这种情况下帧内很多小的细节是很难压缩的, 编码器将花更多的时间为每个片寻找合适的量化。

所以设置更高的bits_per_mb限额将提高速度。

要获取最快的编码速度, 则设置qscale参数 (4为推荐值) 和不设置帧数据大小限制。

字幕编码器

dvdsb

这个编码器编码使用者DVD中的位图字幕格式。一般存储字VOBSUB文件中（包括*.idx* *.sub*），它也用于Matroska文件中。

dvdsb选项

- `even_rows_fix`

但设置为1，则让所有的行平顺。它解决了如果最后行是奇数行时可能非法截断的问题。这个选项仅仅在需要的地方添加了一个透明的行，它的开销很低，通常是一个平均字幕字节数。（位图特性上的一种修正）

默认，work-around被禁止

19 比特流滤镜

默认编译时所有的比特流滤镜都被支持，你可以在配置脚本中以 `--list-bsfs` 获取有效的滤镜列表

可以利用 `--disable-bsfs` 禁用所有的比特流滤镜。要指定个别的滤镜可用，则在此基础上 `--enable-bsf=BSF`，或者在默认（没有指定 `--disable-bsfs`）下禁用个别的滤镜 `--disable-bsf=BSF`，这里 `BSF` 是个别滤镜名称。

在ff*工具集中，`-bsfs` 可以列出（编译允许的）支持的比特流滤镜。

在ff*工具集中，`-bsf` 选项可以指定滤镜应用到每个流，具体滤镜由`滤镜名1=选项名1=选项值1/选项名2=选项值2...'给出，例如

```
ffmpeg -i INPUT -c:v copy -bsf:v filter1[=opt1=str1/opt2=str2][,filter2] OUTPUT
```

下面介绍了当前有效的一些比特流滤镜

aac adtstoasc

转换MPEG-2/4 的AAC ADTS 到 MPEG-4 音频的专用配置比特流滤镜。

这个滤镜从MPEG-2/4 的ADTS头创建一个移除了ADTS头的MPEG-4音频专用配置流

这是十分必要的，例如把由raw ADTS转换的AAC音频内容复制到FLV/MOV/MP4文件时就需要进行这样的处理

chomp

移除每个包后面附加的0

dump extra

对过滤包的开始添加extradata

附加参数指定了如何处理，有如下可能值：

- 'a'

对每个包添加extradata,除了flags2 codec context field中local_header被设置的

- 'k'

每个关键包添加extradata

- 'e'

每个包添加extradata

如果没有指定则为 `k`

例如下面ffmpeg强制在编码H.264中采用全局头（禁用单独数据包头），即将全局头添加作为extradata添加修正每个数据包

```
ffmpeg -i INPUT -map 0 -flags:v +global_header -c:v libx264 -bsf:v dump_extra out.ts
```

h264_mp4toannexb

转换H.264编码比特流，从长前导模式为开始码前导模式（定义在ITU-T H.264 的附录B）

它是一些流格式要求的，通常如MPEG-2传输流格式("mpegts")

例如采用ffmpeg重新混编一个H.264的MP4文件，把流转换成 mpegts 格式，可以使用：

```
ffmpeg -i INPUT.mp4 -codec copy -bsf:v h264_mp4toannexb OUTPUT.ts
```

imxdump

修正为可编辑的比特流MOV格式，以用于Final Cut Pro解码。这个滤镜仅仅用于mpeg2video编码，对于需适用于Final Cut Pro 7以下版本的，同时使用 -tag:v 选项

例如，重新混编 30MB/sec 的NTSC IMX 到MOV:

```
ffmpeg -i input.mxf -c copy -bsf:v imxdump -tag:v mx3n output.mov
```

mjpeg2jpeg

转换MJPEG/AVI1 包为全 JPEG/JFIF包

MJPEG是一种视频编码，它每帧基本都是一个JPEG图像，可以直接无损提取，如：

```
ffmpeg -i ../some_mjpeg.avi -c:v copy frames_%d.jpg
```

不幸的是，这些块是不完整的JPEG图像，因为他们缺乏解码所需的DHT段。

<http://www.digitalpreservation.gov/formats/fdd/fdd000063.shtml>:

Avery Lee, writing in the rec.video.desktop newsgroup in 2001, commented that "MJPEG, or at least the MJPEG in A"



这个比特流滤镜可以扩展修复MJPEG流（携带AVI1头ID，但缺失DHT段）为完整的JPEG图像

```
ffmpeg -i mjpeg-movie.avi -c:v copy -bsf:v mjpeg2jpeg frame_%d.jpg
exiftran -i -9 frame*.jpg
ffmpeg -i frame_%d.jpg -c:v copy rotated.avi
```

mjpega dump header

movsub

mp3头解压

mpeg4 unpack bframes

解包DivX包装的B帧

DivX风格的B帧是无效的MPEG-4，且仅用于windows系统的零散解决方案，它使用更多的空间，而能看导致轻微的AV同步问题，需要更多的CPU资源用于解码（除非播放采用一些2,0,2,0帧分组方式），如果直接复制到一个MP4标准封装或MPEG PS/TS流中将引起问题（MPEG-4不能正确解码，因为它不是有效的MPEG-4编码数据）

例如修正一个包含DivX风格B帧数据的AVI文件到MPEG-4流，可以使用：

```
ffmpeg -i INPUT.avi -codec copy -bsf:v mpeg4_unpack_bframes OUTPUT.avi
```

noise

不破坏容器，仅处理内容包。一般用于冻结或测试 错误恢复/隐藏

参数:一个数字字符串，其值通常由输出而被修改，因此值小于等于0被禁止。低于更新频率的字节被修改，1表示每个字节被修改。

```
ffmpeg -i INPUT -c copy -bsf noise[=1] output.mkv
```

表示每个字节都被修改。

remove extra

20 格式选项

`libavformat` 库提供一些常规的全局选项，它们都可被混合器/分离器设置。一些混合器/分离器还支持附加的私有选项，这些都在其组件处介绍。

ffmpeg工具中选项通过特定的 `-option value` 进行设置，或者通过 `AVFormatContext` 选项设置，或者通过 `libavutil/opt.h` 中的 API 设置

下面是一些被支持的选项：

- `avioflags flags (input/output)`

可能值:

`'direct'`

减少缓冲

- `probesize integer (input)`

设置probing（探测）尺寸，单位bytes, 即加载数据的大小来分析得到的数据流信息。较高的值会使检测的更多信息，分散到流，但会增加延迟。必须有一个不小于32的整数。它的默认值是5000000。

- `packetsize integer (output)`

设置包尺寸

- `fflags flags (input/output)`

设置格式标志

可能值:

`'ignidx'`

忽略索引

`'fastseek'`

允许快速定位，但只有个别格式有效

`'genpts'`

常规PTS。

`'nofillin'`

不填补缺失值，则可以精确计算

‘noparse’

禁止AVParsers，它要和 +nofillin联用。

‘igndts’

忽略DTS。

‘discardcorrupt’

丢弃损坏的帧

‘sortdts’

尝试按DTS输出去交织

‘keepside’

不合并侧数据

‘latm’

允许 RTP MP4A-LATM有效载荷

‘nobuffer’

通过可选缓冲减少延迟

‘bitexact’

仅写平台、编译和时间独立的数据。这保证了文件和数据校验和平台之间的可重复性和匹配，它主要用在回归测试

- seek2any integer (input)
为1表示分离器中允许定位到非关键帧，默认为0，表示禁止
- analyzeduration integer (input)
指定多少微秒分析探头输入。较高的值会使检测更准确的信息，但会增加延迟。默认为5000000毫秒= 5秒。
- cryptokey hexadecimal string (input)
设置解密密钥
- indexmem integer (input)

设置最大内存使用时间戳索引（每流）。

- rtbufsize integer (input)

设置用于缓冲实时帧的最大内存。

- fdebug flags (input/output)

打印特定调试信息

可能值:

'ts'

- max_delay integer (input/output)

设置分离或者混合时的最大延迟，单位microseconds.

- fpsprobesize integer (input)

设置用于探测fps的帧数

- audio_preload integer (output)

设置音频包在交错前时间（微秒）量

- chunk_duration integer (output)

设置块时间，单位microseconds

- chunk_size integer (output)

设置块尺寸，单位bytes

- err_detect, f_err_detect flags (input)

设置错误描述标志，f_err_detect已过时，仅用于ffmpeg工具 可能值:

'crccheck'

验证嵌入的CRCs。

'bitstream'

比特流的规范偏差检测。

'buffer'

不当的码流长度检测。

'explode'

在检测到错误时中断

‘careful’

违反规范但并不作为错误

‘compliant’

违反规范都作为错误

‘aggressive’

一个智能编码器不会作为错误的

- max_interleave_delta integer (output)

设置最大缓冲时间交错。时间是用微秒，默认值为1000000（1秒）。

这将确保流交错（混合）的正确。libavformat会在写输出时确保每个流至少有1个数据包。但当一些流十分稀疏（即有很大间隔才需要一个流数据）这将导致过度的缓冲

这里就指定了一个混合在一起的序列中其中流时间的差值不超过设定数，而不管是否已经在数据序列中已经有所有流信息了

如果设置为0，libavformat将缓冲数据包，直到每个流至少有一个数据包，而不管数据包间时间差值

- use_wallclock_as_timestamps integer (input)

使用时钟作为时间戳。

- avoid_negative_ts integer (output)

可能值:

‘make_non_negative’

转移的时间戳，使其非负。也请注意，这仅影响领导负的时间戳，而非非单调负时间戳。

‘make_zero’

转移的时间戳，使其从0开始

‘auto (default)’

目标格式要求时转移时间戳

‘disabled’

禁止转换时间戳

当时间戳转换允许时，所有基于时间戳的流（音频、视频和字幕）都会被转换，以保证相对时间不变（时间基准统一）

- skip_initial_bytes integer (input)

设置为1则读取头和帧前跳过数字节（对解析帧无关），默认为0

- correct_ts_overflow integer (input)

为1则允许单时间溢出，默认为1

- flush_packets integer (output)

为1则每个包都清除底层I/O流。默认为1，可以减少延迟，为0则稍微增加延迟，但可以改善某些情况下的性能

- output_ts_offset offset (output)

设置输出时间偏移

offset 必须是时间持续描述值，参考 `ffmpeg-utils` 中关于时间持续的章节（在ffmpeg-utils(1)手册中）

这个便宜量将在混合时加到时间戳上进行输出

指定一个正偏移意味着相应的流延迟bt（basetime 基准）时间，默认值为0，表示没有偏移

- format_whitelist list (input)

"," 分隔的一个列表用于分离器，默认是所有的都允许

- dump_separator string (input)

在命令行中指定分离器流参数的域，例如用换行和缩进的独立域：

```
ffprobe -dump_separator "
```

```
" -i ~/videos/ matrixbench_mpeg2.mpg
```

格式流指定（说明）

格式流指定允许选择1个或者多个流匹配特定的属性

可能的流指定形式：

- stream_index

有索引匹配流

- stream_type[:stream_index]

联合流类型和流索引指定流，流类型有：'v' 对应视频，'a' 对应音频，'s' 对应字幕，'d' 对应数据和't' 对应附加。如果流索引 stream_index 被设置，则匹配指定类型的流索引，否则匹配所有该类型流

- p:program_id[:stream_index]

如果流索引被设置，其匹配程序id匹配的处理中索引的流，否则匹配该程序中所有流

. stream_id

通过格式指定ID匹配流

声明在 `libavformat/avformat.h` 中的 `avformat_match_stream_specifier()` 用来精确的完成格式流指定（说明）

分离器

分离器是使得ffmpeg能从特定类型文件中读取多媒体流的组件元素。

当编译ffmpeg时，所有支持的分离器都默认被包含，你可以通过编译配置脚本中的 `--list-demuxers` 列出所有支持的分离器。

你也可以通过配置 `--disable-demuxers` 禁用所有的分离器，如果要在此基础上允许单独的分离器可以选用 `--enable-demuxer=DEMUXER` 形式配置，也可以在默认情况下通过 `--disable-demuxer=DEMUXER` 禁用个别分离器。

ff*工具集中 `-formats` 选项可以列出所有编译支持了的分离器（对多媒体容器的支持情况）

下面介绍当前有效的分离器

applehttp

Apple HTTP Live Streaming（苹果http直播流）分离器

这个分离器会从所有变化流中提取出AVStreams，id字段是码率变化（流）索引数。通过在 `AVStreams` 设置丢弃标志（`a` 或者 `v`），调用者可以获取所有变化的流。总的码率变化被放置在元数据关键字段"variant_bitrate"里。

apng

Animated Portable Network Graphics（便携网络图形动画）分离器

这个分离器用于APNG文件。所有的头，从PNG标识到（不包括）第一个作为extradata传输的fcTL块。帧被分割到两个fcTL间的块中，或者是最后一个fcTL到iEND块间。

- `-ignore_loop` bool

即使文件设置了循环标志也忽略

- `-max_fps` int

每秒最大帧率，0表示不限制

- `-default_fps` int

定义默认帧率（文件中没有特别指定时），0表示最大可能值

asf

Advanced Systems Format（高级系统格式）分离器

这个分离器用于ASF文件和MMS网络流

- `-no_resync_search` bool

通过寻找一定的可选启动代码而不试图重新同步

concat

Virtual concatenation script（虚拟级联脚本）分离器

分离器从一个文本文件中读取一个文件列表和其他指令，然后分离它们，就像文件和指令是被混合在一起的（文件对应特定

类型流，指令也对应特殊流）

时间戳是以第一个文件开始作为0，然后任何以后的文件都以前一个文件播放完成为开始时间。注意这是全局的，即使所有的流并不是相同长度。

文件中必须有相同的流（相同编码、相同时间基准等等）

每个文件的持续时间用来调整下一个文件的时间戳:如果持续时间不正确（因为这是通过比特率计算出的时间，而文件可能是被窃取出来的），这将导致伪影（这里指代后续文件不正确的时间戳），这时时间指令就可以用来覆盖调整存储在文件中的（计算出）的持续时间，而把整个时间戳调整正确。

concat语法

这个脚本文件是extended-ASCII(扩展ASCII)文本文件，每个指令一行。空行和前导空格和由“#”开始的行被忽略。有下面的指令：

- file path

一个文件的路径:如果路径中包含特殊字符和空格，必须通过“\”转义或者由单引号括起来

采用相对于这个文件的相对路径指示包含的文件

- ffmpeg version 1.0

标识脚本类型和版本，也可以设置为安全选项（如果设置为1），默认是-1

为了让ffmpeg自动识别这个格式，这个指令必须是（看起来像）脚本的第一行（没有额外的空间或字节顺序标记）

- duration dur

指示文件持续时间。这个信息可以用于指定文件，以用于有效的帮助处理可能从文件信息中获取（计算）的不可用或不准确持续时间

如果是所有文件的持续时间，则可在整个文件中进行定位

- stream

指定虚拟文件流。随后所有关于流的指令适用于这个指定的流。一些流必须设置为允许识别子文件匹配的流，如果没有指定流则从第一个文件流进行复制

- exact_stream_id id

设置流的id。如果设置了，将作为子文件的标识id来用，这通常用于MPEG-PS(VOB)文件，这里数据流的顺序是不可靠（用）的

concat选项

分离器接受如下选项：

- safe

如果设置为1，将调整不安全的文件路径。所谓安全文件路径指不包括协议规范头，以相对路径指出，且只包括便携字符集（字母、数字、点号、下划线和连字符）且不以点号开始的（即只接受安全路径）

如果设置为0，则任何文件路径都可以被接受

默认值是-1，表示自动探测，如果可能则为0，否则为1

- `auto_convert`

如果为1，则尝试对分组流进行自动转换，默认为1

当前它仅支持在MP4格式中的H.264流添加 `h264_mp4toannexb` 比特流滤镜，这是必要的，尤其有分辨率变化时。

flv

Adobe Flash Video Format 分离器

它用于分离FLV文件和RTMP网络流

- `-flv_metadata bool`

根据元数据的数组内容分配流

libgme

Game Music Emu库是一个汇集视频游戏音乐文件模拟器

参考<http://code.google.com/p/game-music-emu/>获得更多信息

一些文件有很多音轨，这个分离器会把第一个作为默认。`track_index` 选项可以用来指定不同的音轨。音轨索引从0开始。分离器可以从音轨元数据中导出音轨索引

对于巨大的文件，`max_size` 选项可以用来调整

libquvi

从使用quvi项目的因特网服务器播放媒体

这个分离器要求 `format` 选项被指定一个品质，默认是 `best`

参考<http://quvi.sourceforge.net/>了解更多信息

编译时需要 `--enable-libquvi` 以获取支持

gif

Animated GIF（动画GIF）分离器

接受如下选项：

- `min_delay`

设置帧间最小间隔，单位百分之一秒，范围0-6000，默认2

- `max_gif_delay`

设置帧间最大间隔，单位百分之一秒，范围0-65535（大约11分钟），这个值只能被指定

- `default_delay`

设置帧间默认间隔，单位百分之一秒，范围0-6000，默认10

- ignore_loop

GIF文件可以包含一个用于指定循环次数的信息，如果这个选项被设置为1，则文件中的指定将被忽略。设置为0则启用文件中的设定，默认值是1

例如，下面将一个gif文件按循环次数接到mp4文件后，一起输出：

```
ffmpeg -i input.mp4 -ignore_loop 0 -i input.gif -filter_complex overlay=shortest=1 out.mkv
```

注意前面例子中以简写参数形式采用了个滤镜。

image2

图像文件分离器

这个分离器按指定格式规范（模式）读取一个图像文件列表，语法由选项 `pattern_type` 指定。

格式规范可以包含一个后缀扩展名来自动确定包含该图像格式文件。

在列表（序列）中图像文件的尺寸、像素格式和图像格式必须是一样的。

分离器接受下面的选项：

- framerate

设置对应视频流帧率，默认25

- loop

如果为1，则循环设置覆盖输入，否则为0（默认）

- pattern_type

选择格式规范（模式）类型

`pattern_type` 接受下面值：

- none

禁用模式匹配（规则匹配），即只包括指定的图像，如果你不想创建包含多个图像文件的序列和文件名中可能包含特殊字符（会和模式冲突）则采用这个选项参数

- sequence

指定序列模式，通过特定的文件索引指定一个序列文件

序列模式可以包括"%d" 或 "%0Nd"字符串。这将指定所有匹配字符串的文件。如果是"%0Nd"则表示N位数字，如果不足前面用0补足。如果文件名中本身含有“%”，则要用“%%”转义代替

在序列模式中包含了 "%d"或者"%0Nd",则第一个文件是匹配模式的文件，整个序列是start_number到start_number+start_number_range-1，且所有文件成为连续序列

例如匹配设置 "img-%03d.bmp"可匹配img-001.bmp, img-002.bmp, ..., img-010.bmp 等等;而匹配设置 "i%mm%g-%d.jpg" 将匹配 i%mm%g-1.jpg, i%mm%g-2.jpg, ..., i%mm%g-10.jpg 等到

注意该模式不一定包含"%d"或者 "%0Nd",例如转换单个图像img.jpeg 你可以采用:

```
ffmpeg -i img.jpeg img.png
```

- glob

选择一个全局通配符模式

这种模式下类似glob()模板，它仅在libavformat编译支持了globbing时有效。

- glob_sequence (deprecated, will be removed)

混合通配符和序列的模式

如果libavformat编译支持了globbing，且在匹配字符串中包含了至少1个通配符，普通的 `%*?[]{}` 等字符需要加上 `%` 进行转义。这将类似glob()进行通配符匹配，然后再附加序列模板

所有特殊字符 `%*?[]{}` 等字符需要加上 `%` 进行转义，例如对于字符`"%"`需要表示为`"%%"`

例如`foo-%*.jpeg`将匹配所有 `foo-` 且扩展名为`".jpeg"`的文件。`foo-%?%?%?.jpeg`将匹配 `foo-` 前缀，后面接3个字符的序列，扩展名为 `".jpeg"`。

模式类型在favor of glob 以及 sequence中描述

默认值是 `glob_sequence`

- pixel_format

设置像素格式，如果没有指定则猜测一个（从第一个图像获取）

- start_number

设置索引开始计数的数字值，默认是0

- start_number_range

设置找到序列中第一个文件，以此设置全文件时间检测范围。起始时间基准由 `start_number` 指定，默认为5

- ts_from_file

如果为1，则图像文件帧时间码可被编辑。注意不是由时间戳提供：如果没有这个选项，图像将按相同的顺序。默认值为0，如果设置为2，将在纳秒级精确的对图像时间进行修正

- video_size

设置图像尺寸，如果没有指定，将以序列第一个文件进行探测

image2例子

- 从一个文件序列 `img-001.jpeg, img-002.jpeg, ...,` 创建视频，帧率为10:

```
ffmpeg -framerate 10 -i 'img-%03d.jpeg' out.mkv
```

- 类似上例，但开始的数字是100，即索引是从100开始计数:

```
ffmpeg -framerate 10 -start_number 100 -i 'img-%03d.jpeg' out.mkv
```

- 读取`"*.png"`以通配符模式处理，这将包含所有`".png"`结尾的文件:

```
ffmpeg -framerate 10 -pattern_type glob -i "*.png" out.mkv
```

mpegts

MPEG-2传输流分离器

- fix_teletext_pts

依据从第一个程序PCR计算出的时间戳覆盖PTS和DTS，图文信息流部分不丢弃。默认值是1.如果你想保持PTS和DTS不变则设置为0

rawvideo

原始视频（raw video）分离器

这个分离器允许读取原始视频数据。因为没有报头指定数据参数，为了能正确解码用户必须人为设定一些参数。

接受如下的选项：

- framerate

甚至视频帧率，默认25

- pixel_format

设置输入视频的像素格式，默认yuv420p.

- video_size

设置输入视频的分辨率，这个必须进行指定

例如要用ffplay播放input.raw中的原视频，像素格式是rgb24，分辨率320x240，帧率10则相应命令为：

```
ffplay -f rawvideo -pixel_format rgb24 -video_size 320x240 -framerate 10 input.raw
```

sbg

SBaGen 脚本分离器

这个分离器读取SBaGen脚本，<http://uazu.net/sbagen/>可以相关处理细节，一个SBG脚本看起来像：

```
-SE
a: 300-2.5/3 440+4.5/0
b: 300-2.5/0 440+4.5/3
off: -
NOW      == a
+0:07:00 == b
+0:14:00 == a
+0:21:00 == b
+0:30:00 off
```

SBG脚本可以混合绝对和相对时间戳。如果脚本只使用绝对时间戳（包括脚本启动时间）或者只使用相对时间戳，那么它的布局是古代的，转换十分简单。如果脚本是混合时间戳，相对时间将从脚本读取执行剧本开始计算，而一些绝对时间戳指定的内容可能被冻结，这意味着如果脚本是直播播放的形式，实际时间与绝对时间戳影响的声音控制器时间精度会一直，但如果其间用户进行了暂停等操作，则相应的时间戳会方式变化

tedcaptions

用于TED任务的JSON

TED并不提供这样的内容主题，但可以从页面猜测。ffmpeg源代码库中间的文件 `tools/bookmarklets.html`

允许如下选项：

- `start_time`

设置TED任务开始时间。单位milliseconds,默认值15000 (15s).它可以用来同步标题（对于下载视频），因为它包括了15s的输入引导

例子：转换一个主题：

```
ffmpeg -i http://www.ted.com/talks/subtitles/id/1/lang/en talk1-en.srt
```


22 混合器（复用器）

复用器是ffmpeg中负责写入多媒体流到文件中分区的可配置组件。

默认编译时自动允许被支持的混合器。你可以使用 `--list-muxers` 作为参数运行编译配置脚本以了解当前支持的所有混合器。

编译也可以同 `--disable-muxers` 禁用所有的混合器，或者通过 `--enable-muxer=MUXER` / `--disable-muxer=MUXER` 打开/关闭指定的混合器

在ff*工具集中附加 `-formats` 也可以了解到混合器列表。

下面将详细描述有效的混合器直播：

aiff

Audio Interchange File Format (aif) 密码器

aiff选项

接受下面选项：

- `write_id3v2`

如果设为1则允许ID3v2标签，否则0禁止（默认）

- `id3v2_version`

选择id3v2版本

crc值

CRC (Cyclic Redundancy Check)测试格式

这个混合器通过所有输入的音频和视频帧计算（混合）Adler-32 CRC。默认音频会被转换为16bit符号原始音频，视频被解压为原始视频再进行这个计算。

输出会有一个形如 `CRC=0xCRC` 的一行，其中CRC的值是由16进制以0补足的8位数字，它由所有帧解码计算的。

参考[frameccrc] 混合器

crc值计算例子

计算一个crc放置到out.crc:

```
ffmpeg -i INPUT -f crc out.crc
```

计算crc并直接输出到标准输出设备:

```
ffmpeg -i INPUT -f crc -
```

还可以选择对特定音频、视频编码数据计算crc，例如计算输入文件音频转换成PCM 8bit无符号数据格式，视频转换成

MPEG-2 的CRC:

```
ffmpeg -i INPUT -c:a pcm_u8 -c:v mpeg2video -f crc -
```

framecrc

每个数据包的CRC（循环冗余校验）测试格式。

它将对每个数据包做Adler-32 CRC计算并输出。默认音频被转换成16bit符号原始音频，视频被转换成原始视频再进行CRC计算。

输出是针对每个音频/视频数据包都有一行如下格式的信息：

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, 0xCRC
```

其中CRC值是16进制，以0补足的8位数字值。

framecrc例子

对INPUT输入作每数据包CRC计算，输出到out.crc:

```
ffmpeg -i INPUT -f framecrc out.crc
```

直接把计算结果输出到标准输出设备:

```
ffmpeg -i INPUT -f framecrc -
```

通过ffmpeg，还可以选择输出特定音频和视频格式对应的帧CRC值，例如音频转换成PCM8bit无符号编码，视频为mpeg2计算帧CRC校验值：

```
ffmpeg -i INPUT -c:a pcm_u8 -c:v mpeg2video -f framecrc -
```

参看[crc]混合器

framemd5

每个数据包MD5校验值

计算输出每个数据包MD5校验值，默认音频被转换成16bit符号原始音频，视频被转换成原始视频再进行MD5计算

每个数据包计算对应输出一行如下格式数据：

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, MD5
```

其中MD5就是计算出的MD5 哈希值

framemd5例子

计算INPUT输入的帧md5值，其中音频被转换成16bit符号原始音频数据，视频被转换成原始视频数据，输出到out.md5

```
ffmpeg -i INPUT -f framemd5 out.md5
```

直接输出到标准输出设备：

```
ffmpeg -i INPUT -f framemd5 -
```

参考[md5]混合器部分

gif

Animated GIF（动画GIF）混合器

它接受如下选项：

- loop

设置循环次数，-1表示不循环，0表示一直循环（默认值）

- final_delay

强制最后一帧延迟（以厘秒为单位——centiseconds），默认为1，这是一个对于循环gif的特殊设定，它为最后一帧播放到新开始播放设置一个特殊的值，比如你可能希望有一个停顿的感觉。

例如像循环10次，每次重新播放前停顿5秒，则：

```
ffmpeg -i INPUT -loop 10 -final_delay 500 out.gif
```

注意1如果你想提取帧到指定的GIF文件序列，你可能需要 `image2` 混合器

```
ffmpeg -i INPUT -c:v gif -f image2 "out%d.gif"
```

注意2Gif格式有一个非常小的时基：两帧之间的间隔不可小于百分之一秒。

hls

Apple HTTP 直播流混合器，它根据HTTP直播流（HLS）规范进行MPEG-TS分割

它创建一个播放列表文件，包括1个或者多个分段文件，输出文件为指定的播放列表文件。

默认混合器对每段创建一个文件，这些文件有相同的基于播放列表的文件名，段索引数和.tx扩展名

例如，转一个输入文件：

```
ffmpeg -i in.nut out.m3u8
```

这将根据产品播放列表文件out.m3u8产生分段文件:out0.ts out1.ts out2.ts 等等

参考[segment]混合器，它提供了更多可用于HTL分割的常规处理和修正介绍

hls选项

这个混合器支持如下选项

- hls_time seconds

设置段长度，单位秒，默认为2

- hls_list_size size

设置播放列表中字段最大数。如果为0，则包含所有分段。默认为5

- hls_ts_options options_list

设置输出格式选项，使用'-'分割的 `key=value` 参数对，如果包括特殊字符需要被转义处理

- hls_wrap wrap

一种循环机制，设置数量后以0-设定数形成一个环依次循环使用作为输出段号.为0表示不限制，默认为0

选项可避免磁盘被多个段文件填满，并限制写入磁盘的最大文件数

- start_number number

设置播放列表中最先播放的索引号，默认 0.

- hls_allow_cache allowcache

设置客户端是否：可能(1) 或 必须不 (0) 缓冲媒体段

- hls_base_url baseurl

对每个列表中的记录添加一个基本的URL，一般用于采用相对路径描述的列表

注意列表序号必须是每段独特的，不可分割的文件名和序列号，序列号是可循环的，则可能会引起困惑，例如hls_wrap选项设置了

- hls_segment_filename filename

设置段文件名。除非 `hls_flags single_file` 被设置，设置这个文件名可以用于段命名格式化（依据段序数）：

```
ffmpeg in.nut -hls_segment_filename 'file%03d.ts' out.m3u8
```

这个例子中，段文件会输出为: file000.ts, file001.ts, file002.ts, 等等，而不是默认的out0.ts out1.ts out2.ts 等等

- hls_key_info_file key_info_file

使用key_info_file对段进行加密。 `key_info_file` 中的第一行指定一个URI，是写入播放列表的，这个key URL被用于存放播放期访问的加密密钥。第二行指定用于加密过程中的key文件路径。key文件作为一个单一排列的16进制数组以二进制格式数据读入。可选的第三行则指定初始化向量（IV，一个十六进制字符串用于代替部分序列（默认）进行加密）。改变key_info_file将导致段加密采用新的key/IV 以及播放列表中任意条目采用新的 URI/IV

key_info_file 格式:

```
key URI key file path IV (optional)
```

key URIs 例子:

```
http://server/file.key /path/to/file.key file.key
```

key文件路径例子:

```
file.key /path/to/file.key
```

IV例子:

```
0123456789ABCDEF0123456789ABCDEF
```

完整key_info_file 示例:

```
http://server/file.key /path/to/file.key 0123456789ABCDEF0123456789ABCDEF
```

shell脚本例子:

```
#!/bin/sh
BASE_URL=${1:-'.'}
openssl rand 16 > file.key
echo $BASE_URL/file.key > file.keyinfo
echo file.key >> file.keyinfo
echo $(openssl rand -hex 16) >> file.keyinfo
ffmpeg -f lavfi -re -i testsrc -c:v h264 -hls_flags delete_segments \
-hls_key_info_file file.keyinfo out.m3u8
```

- hls_flags single_file

如果这个标记被设置，则会把所有段存储到一个MPEG-TS文件中，且在播放列表中使用字节范围。HLS播放列表在版本4中支持这种方法：

```
ffmpeg -i in.nut -hls_flags single_file out.m3u8
```

这里所有的输出都放置在out.ts中了

- hls_flags delete_segments

在播放的段已经过了持续时间后就删除掉对应的文件。

ico

ICO文件混合器

微软ICON（ICO）文件格式有一些限制需要注意：

- 每个方向不超过256像素
- 仅BMP和PNG图像可以被存储
- 如果是BMP图像，必须有如下像素格式：

```
bmp位深度 ffmpeg像素格式
1bit pal8
4bit pal8
8bit pal8
16bit rgb555le
24bit bgr24
32bit bgra
```

- 如果是BMP图像，必须有 BITMAPINFOHEADER DIB 头

- 如果是PNG图像，必须是rgba像素格式

image2

图像文件混合器

它可以把视频帧重新混合为图像文件

输出文件按模板指定，可以设置成为一个序列数文件。模板中的"%d" 或者 "%0Nd"用于指定序列，其中"%0Nd"表示N位数字，以0补齐。如果文件名中有“%”需要以“%%”转义的形式指定。

如果模板中包含了"%d"或者"%0Nd"则文件名从1计数输出序列

模板可以包含一个后缀用来自动确定图像文件格式

例如模板"img-%03d.bmp"将使得输出为img-001.bmp, img-002.bmp, ...,img-010.bm 等等。而模板"img%%-%d.jpg"则生成img%-1.jpg, img%-2.jpg, ..., img%-10.jpg,等等

image2例子

把输入视频图像帧生成成为img-001.jpeg, img-002.jpeg, ...,

```
ffmpeg -i in.avi -vsync 1 -r 1 -f image2 'img-%03d.jpeg'
```

注意ffmpeg如果没有通过 `-f` 指定输出文件格式，image2混合器将自动被选择，所以前面的等效于

```
ffmpeg -i in.avi -vsync 1 -r 1 'img-%03d.jpeg'
```

如果 `strftime` 选项允许你导出按时间/日期信息命名的文件 "%Y-%m-%d_%H-%M-%S" 模板，在 `strftime()` 的文档中了解相关语法

例如：

```
ffmpeg -f v4l2 -r 1 -i /dev/video0 -f image2 -strftime 1 "%Y-%m-%d_%H-%M-%S.jpg"
```

image2选项

- start_number

设置开始序列的数字，默认为0

- update

如果设置为1，文件名直接作为唯一文件名，而没有模板。即相应的文件被不断改写为新的图像。默认为0

- strftime

如果设置为1，可以让输出文件支持strftime()提供的日期格式，默认为0

这个图像混合器支持.Y.U.V图像文件格式，这种格式将根据每帧输出3个文件，对于每个YUV420P压缩，对于读或者写这种文件格式，只需要指定.Y文件即可，混合器会自动打开需要的.U和.V文件

matroska

Matroska内容混合器

混合输出matroska和webm内容

matroska元数据

混合器需要指定一些必要元数据

- title

设置单个轨道的标题名 language

以Matroska语言字段指定语言

语言可是3个字符（依ISO-639-2 (ISO 639-2/B)）（例如 "fre" 表示法语——French),或者语言混合国家/地区代码，(like "fre-ca" 表示加拿大法语——Canadian French). stereo_mode

设置3D视频两个视图在单个视频轨道播放时的布局规则

允许如下值：

‘mono’

video不是双路的

‘left_right’

两路分别一端，即左眼看左视图，右眼看右视图

‘bottom_top’

上下布局，左眼看下视图，右眼看上视图

‘top_bottom’

与上一个相反，左眼看上，右眼看下

‘checkerboard_rl’

根据序列确认，左眼看第一个

‘checkerboard_lr’

根据序列确认，右眼看第一个

‘row_interleaved_rl’

根据行序列确认，右眼看第一行

‘row_interleaved_lr’

根据行序列确认，左眼看第一行，

‘col_interleaved_rl’

列序列确认，右眼第一列

‘col_interleaved_lr’

列序列确认，左眼第一列

‘anaglyph_cyan_red’

都在一副图中，通过颜色滤镜过滤red-cyan filters

‘right_left’

右眼看左图

‘anaglyph_green_magenta’

混合图，通过green-magenta滤镜看

‘block_lr’

间隔块，左眼先看

‘block_rl’

间隔块，右眼先看

例如，对于3DWebM影片，可以由下面命令建立：

```
ffmpeg -i sample_left_right_clip.mpg -an -c:v libvpx -metadata stereo_mode=left_right -y stereo_clip.webm
```

matroska选项

支持如下选项：

- `reserve_index_space`

默认对于定位索引（可以被Matoska调用）将写到文件的末尾部分，因为一开始不知道需要多少空间放置索引。但这将导致流式播放时定位特别慢（因为不知道定位索引），这个选项将把索引放置到文件的开始。

如果这个选项设置为非0值，混合器将预先在头部放置一个用于写入索引的空间，但如果空间无效则将混合失败。一个较安全的值是大约1小时50KB。

注意这些寻址线索仅当输出文件是可寻址且选项设置了有效值时写入。

md5

MD5检测格式

将计算输出一个MD5值，对于所有的音视频帧。默认音频帧转换为有符号16bit原始音频，视频转换为原始视频来计算。

输出是一个MD5= MD5 格式，其中 MD5 就是计算出的值。

例如：

```
ffmpeg -i INPUT -f md5 out.md5
```

也可以输出到标准输出设备

```
ffmpeg -i INPUT -f md5 -
```

参考[framemd5]混合器

mov,mp4,ismv

MOV/MP4/ISMV (Smooth Streaming——平滑流)混合器

MOV/MP4/ISMV混合器支持零碎文件（指数据的组织形式）。通常MOV/MP4文件把所有的元数据存储在一个位置中（这是不零碎的数据组织形式，通常在末尾，也可以移动到起始以更好的支持随机定位播放，比如使用 `qt-faststart` 工具，并添加 `movflags` 快速启动标志）。这样一个零碎文件包含了很多片段，其中数据包和元数据是存储在一起的。这样零碎数据组织的文件在解码到写中断（普通的MOV/MP4则不能解码了，因为可能缺少元数据）时也能正常解码，而且这种方式要求更少的内存就可以写很大的文件（因为普通形式的MOV/MP4需要收集所有的信息才能最终完成元数据集中存储，则这一过程中这些数据一直需要缓存在内存中，直到编码完成，元数据完成存储），这是一个优势。缺点是这种组织数据的格式不太通用（很多程序不支持）

mov,mp4,ismv选项

零碎形式也支持AVOptions，它可以定义如何切分文件到零碎片段中：

- `-moov_size bytes`

在文件开头设置预留空间用于存储moov原子数据（一些元数据），而不是把这些数据存储在文件尾部。如果预设的空间不够，将导致混合失败

- `-movflags frag_keyframe`

在每个关键帧都开始一个新的碎片

- -frag_duration duration

每duration microseconds时长就创建一个碎片

- -frag_size size

碎片按size字节（这是一个上限）进行划分

- -movflags frag_custom

允许调用者手动切片，通过调用av_write_frame(ctx, NULL) 在当前位置写入一个片段(它仅能与libavformat库集成，在ffmpeg中不支持)

- -min_frag_duration duration

如果少于duration microseconds就不单独创建片段

如果指定了多个条件，当一个条件是，片段被切分出来。例外的是 `-min_frag_duration`，它在任何其它条件满足时都使用来进行判断

此外，输出还可以通过一些其他选项进行调整：

- -movflags empty_moov

写入一个空的moov atom到文件开始,而没有任何样品描述。一般来说，一个mdat/moov在普通MOV/MP4文件开始时写入，只包括了很少的内容，设置了这个选项将没有初始的moov atom，而仅是一个描述了轨道，但没有持续时间的moov atom。

这个选项在ismv文件中隐式设定

- -movflags separate_moof

为每个轨道写独立的moof（电影片段）atom。通常，追踪所有分组是写在一个moof atom中，而通过这个选项，混合器将对每个轨道单独写moof/MDAT，以方便轨道间隔离

这个选项在ismv文件中隐式设定

- -movflags faststart

再次移动index（moov atom）到文件开始位置。这个选项可以与其他选项一起工作，除了碎片化输出模式。默认情况是不允许

- -movflags rtphint

添加RTP打标轨道到输出文件中

- -movflags disable_chpl

禁止Nero章标签(chpl atom)。通常，Nero章标签和QuickTime章标签都被写入到文件中，通过这个选项，可以强制只输出QuickTime标签。Nero章标签可能导致文件在某些程序处理标签时失败，例如 mp3Tag 2.61a 和 iTunes 11.3，可能其他版本也会受到影响

- -movflags omit_tfhd_offset

在tfhd atom（原子数据）中不写入任何绝对 `base_data_offset`。这将避免片段文件/流中的绝对定位绑定

- -movflags default_base_moof

类似 `omit_tfhd_offset`，这个标志避免在 `tfhd atom` 中写绝对 `base_data_offset`，而是用新的 `default-base-is-moof`，这个标志定义在 14496-12:2012。它会使片段在某些情况下更容易被解析（避免通过在前一轨道片段基础上隐式进行追踪计算碎片位置）

mov,mp4,ismv例子

平滑流内容可以通过 IIS 进行发布，例如：



mp3

MP3 混合器通过下面选项写原始的 MP3 流：

- 一个 ID3v2 元数据头会写在开始处（默认），支持版本 2.3 和 2.4，`id3v2_version` 私有选项可以使用 (3 或 4)，如果设置 `id3v2_version` 为 0 表示禁用 ID3v2 头

混合器还支持附加图片（APIC 帧）到 ID3v2 头。这个图片以单一分组视频流的形式提供给混合器。可以有任意数量的这种流，每个都是单独的 APIC 帧。对于 APIC 帧的描述和图片类型要求，以及流元数据标题及内容提交者等参考 <http://id3.org/id3v2.4.0-frames>。

注意 APIC 帧必须写在开始的地方，所以混合器会缓冲音频帧直到所有的图片已经获取完成。因此建议尽快提供图片，以避免过度缓冲。

- Xing/LAME 帧正确放置在 ID3v2 头之后（如果提供）。它也是默认的，但仅仅在输出是可定位情况下写入。`write_xing` 私有选项可以用来禁用它。这些帧中包括的变量信息通常用于解码器，例如音频持续时间或者编码延迟
- 一个遗留的 ID3v1 标签放置在文件的末尾（默认禁止），它可以通过 `write_id3v1` 私有选项来启用，但其意义非常有限，所以不建议采用

一些例子：

- 写一个 mp3，有 ID3v2.3 头和 ID3v1 的末尾标签

```
ffmpeg -i INPUT -id3v2_version 3 -write_id3v1 1 out.mp3
```

- 通过 `map` 附加图片到音频：

```
ffmpeg -i input.mp3 -i cover.png -c copy -map 0 -map 1 -metadata:s:v title="Album cover" -metadata:s:v comment="Cover (Front)" out.mp3
```

- 写入一个“干净”的 MP3，而没有额外特性

```
ffmpeg -i input.wav -write_xing 0 -id3v2_version 0 out.mp3
```

mpegt

MPEG 传输流混合器

这个混合器声明在 ISO 13818-1 和 ETSI EN 300 468 的部分内容中。

对于通用的元数据设置 `service_provider` 和 `service_name`，如果没有特别指明，则默认 `service_provider` 为 "FFmpeg"，`service_name` 为 "Service01"

mpegts 选项

mpegts混合器选项有：

- -mpegts_original_network_id number

设置 `original_network_id` (默认0x0001). 在DVB是一个唯一的网络标识，它用于标识特殊的服务（通过 `Original_Network_ID`和`Transport_Stream_ID`）

- -mpegts_transport_stream_id number

设置 `transport_stream_id` (默认0x0001).在DVB是一个传输的标识

- -mpegts_service_id number

设置 `service_id` (默认0x0001),在DVB作为程序标识 DVB.

- -mpegts_service_type number

设置程序 `service_type` (默认digital_tv), 参考下面预设值

- -mpegts_pmt_start_pid number

对PMT设置第一个PID (默认 0x1000,最大0x1f00).

- -mpegts_start_pid number

对数据包设置第一个PID(默认0x0100,最大0x0f00).

- -mpegts_m2ts_mode number

如果设置为1则允许 `m2ts` 模式，默认为-1，表示禁止 value is -1 which disables m2ts mode.

- -muxrate number

设置内容为混合码率（默认VBR）

- -pcr_period numer

覆盖默认的PCR重传时间（默认20ms），如果 `muxrate` 被设置将会被忽略

- -pes_payload_size number

以单位字节设置最小PES播放加载包大小

- -mpegts_flags flags

设置一个标志(后面介绍).

- -mpegts_copyts number

如果设置为1则保留原始时间戳。默认为-1，将从0开始更新时间戳

- -tables_version number

设置PAT, PMT 和SDT版本 (默认0,范围0-31)。这个选项允许更新流结构，以便用户可以检测到更改。比如在打开 `AVFormatContext`（API使用时）或重启FFMPEG来周期性改变 `tables_version` 时：

```

ffmpeg -i source1.ts -codec copy -f mpegts -tables_version 0 udp://1.1.1.1:1111
ffmpeg -i source2.ts -codec copy -f mpegts -tables_version 1 udp://1.1.1.1:1111
... ffmpeg -i source3.ts -codec copy -f mpegts -tables_version 31 udp://1.1.1.1:1111
ffmpeg -i source1.ts -codec copy -f mpegts -tables_version 0 udp://1.1.1.1:1111
ffmpeg -i source2.ts -codec copy -f mpegts -tables_version 1 udp://1.1.1.1:1111
...

```

选项 `mpegts_service_type` 接受如下值:

- `hex_value`

一个16进制值, 范围0x01到0xff, 定义在 ETSI 300 468.

- `digital_tv`

数字TV服务

- `digital_radio`

数字广播服务

- `teletext`

图文电视服务

- `advanced_codec_digital_radio`

高级编码数字广播服务

- `mpeg2_digital_hdtv`

MPEG2数字HDTV服务

- `advanced_codec_digital_sdtv`

高级编码数字SDTV服务

- `advanced_codec_digital_hdtv`

高级编码数字HDTV服务

选项 `mpegts_flags` 可以设置如下标志:

- `resend_headers`

写下一个包前反弹PAT/PMT

- `latm`

对AAC编码使用LATM打包

mpegts例子

```

ffmpeg -i file.mpg -c copy \
-mpegts_original_network_id 0x1122 \
-mpegts_transport_stream_id 0x3344 \
-mpegts_service_id 0x5566 \

```

```
-mpegts_pmt_start_pid 0x1500 \
-mpegts_start_pid 0x150 \
-metadata service_provider="Some provider" \
-metadata service_name="Some Channel" \
-y out.ts
```

null

Null混合器

这个混合器将不产生任何输出文件，通常用于测试和基准检测

例如要检测一个解码器，你可以使用：

```
ffmpeg -benchmark -i INPUT -f null out.null
```

注意前面的命令行并不读写out.null，仅仅是因为ffmpeg语法要求必须有个输出

等效的，你可以采用：

```
ffmpeg -benchmark -i INPUT -f null -
```

nut

- -syncpoints flags

利用nut改变同步点：

- default:默认采用低开销的定位模式。没有不使用同步点的，但可减少开销，只是流是不可定位的。
- none:一般不建议采用这个选项，因为它导致文件是损坏敏感的（稍微破坏就不能正常解码了），且不可定位。一般同步点开销是很小以至于可以忽略的。注意 `-write_index 0` 可用于禁止所有增长的数据表，允许重复使用有效的内存，而没有这些缺点。
- timestamped:时间戳字段扩展来与时钟同步。

`none` 和 `timestamped` 还处于试验阶段

- -write_index bool

在最后写索引，这是写索引的默认值

```
ffmpeg -i INPUT -f_strict experimental -syncpoints none - | processor
```

ogg

Ogg内容混合器

- -page_duration duration

首选页面持续时间（其实是定位点间隔），单位microseconds。混合器将尝试按设定时间创建页面。这允许用户在定位和容器粒度开销间进行平衡。默认1秒。如果设为0，将填充所有字段，使索引数据很大。在大多数情况下，设为1将使得每个页面1个数据包，且可以有一个很小的定位粒度，但将产生额外的容器开销（文件变大）

- `-serial_offset value`

用于设置流序号的一些值。设置来不同且足够大，可以保证产生的ogg文件可以安全的被锁住

segment, stream_segment, ssegment

基本流分段

混合器将输出流到指定的文件（根据最接近的持续时间分段）。输出文件名模板可以采用类似与[image2]的方式，或者使用 `strftime` 模板（如果 `strftime` 选项被允许）

`stream_segment` 是用于流式输出格式的混合器变种，例如不需要全局头，并要求诸如MPEG传输流分段输出的情况。`ssegment` 是 `stream_segment` 的别名。

每个片段都开始于所选流的关键帧，这是通过 `reference_stream` 选项设置的

注意如果你想精确分割视频文件，你需要准确输入按关键帧整数倍对应的预期分割器，或者指定混合器按新片段必须是关键帧开始。

分段混合器对于固定帧率的视频有更好的工作表现

或者它可以生成一个创建段的列表，这需要通过 `segment_list` 选项设置，列表的类型由 `segment_list_type` 选项指定。在段列表输入一个文件名被默认为相应段文件的基本名称。

参看[hls]混合器，其提供更多关于 HLS 分段的特定实现

segment, stream_segment, ssegment选项

segment混合器支持如下选项：

- `reference_stream specifier`

由字符串指定参考流，如果设置为 `auto` 将自动选择参考流。否则必须指定一个流（参看 流说明符 章节）作为参考流。默认为 `auto`

- `segment_format format`

覆盖内容自身格式。默认根据文件扩展名检测（猜测）

- `segment_format_options options_list`

使用“:”分隔的 `key=value` 列表作为选项参数以一次定义多个选项，其中值如果包含“:”等特殊符号需进行转义

- `segment_list name`

指定生成文件的名字列表。如果不指定将没有列表文件生成。

- `segment_list_flags flags`

设置影响生成段序列的标志

可以有下面的标志：

- `'cache'`

允许缓存（只能用于M3U8列表文件）。

- 'live'

允许直播友好文件生成

- segment_list_type type

选择列格式Select the listing format.

`flat` 使用简单的 flat列表单元

`hls` 使用类似m3u8的结构

- segment_list_size size

当列表文件包含了指定个数段后更新文件,如果为0则列表文件会包含所有的段, 默认为0

- segment_list_entry_prefix prefix

对每条记录添加一个前导修饰。常用于生成绝对路径。默认没有前导添加

下面的值被允许:

- 'flat'

按 `flat` 列表生成段, 每行一个段

- 'csv, ext'

按列表生成段, 每行一段, 每行按如下格式(逗号进行分割), :

`segment_filename,segment_start_time,segment_end_time`

`segment_filename` 是输出文件名字, 混合器根据提供的模板产生输出文件名(参考 RFC4180)

`segment_start_time` 和 `segment_end_time` 指定段开始和结束时间, 单位秒

文件列表如果以 ".csv" 或 ".ext"作为扩展名, 将自动匹配这个列表格式

'ext'是对不喜欢 'csv'的替代

- 'ffconcat'

分析ffconcat文件生成段。 file for the created segments. The resulting file can be read using the FFmpeg concat demuxer.

列表文件以 ".ffcat"或 ".ffconcat"作为扩展名时会自动选择这个格式

- 'm3u8'

分析M3U8的文件, 版本3, 符合<http://tools.ietf.org/id/draft-pantos-http-live-streaming>

如果列表文件有 ".m3u8"扩展名将自动选择这个格式

如果不指定就从文件扩展名中进行猜测

- segment_time time

设置段持续时间, 这个值必须指定, 默认为2, 参考 `segment_times` 选项.

注意划分可能不太精确，除非强制到流中关键帧间隔时间。参考下面的例子

- `segment_atclocktime 1|0`

如果设置为"1"，将从00:00开始计时，利用 `segment_time` 为间隔划分出多个段

例如如果 `segment_time` 设置为"900"，这个选项将在12:00、12:15、12:30等时间点创建文件

默认为"0"。

- `segment_time_delta delta`

指定一个时间作为段开始时间，其表示为一个时间规范，默认为"0"。

当 `delta` 被指定，关键帧将开始一个新的段以使PTS满足如下关系：

$$PTS \geq start_time - time_delta$$

这个选项通常用来划分视频内容，其总是在GOP边界划分，它在指定点前找到一个关键帧来划分

它可以结合ffmpeg的 `force_key_frames` 选项，通过 `force_key_frames` 可以强制指定一个时间点的是关键帧而不是自动计算。因为四舍五入的原因关键帧时间点可能不是很精确，而可能在设置的时间点之前。对于恒定帧率的视频，在实际值和依 `force_key_frames` 设定值间最坏有 $1/(2*frame_rate)$ 的差值

- `segment_times times`

指定一个划分点的列表。列表是逗号分隔的升序列表，每个是持续时间。也可以参考 `segment_time` 选项

- `segment_frames frames`

指定划分视频帧的序号列表。列表以逗号分隔的升序列表

这个选项指定一个新段开始于参考流关键帧和序列（从0开始），下个值则需要表明下一个段切分点

- `segment_wrap limit`

环形索引限制

- `segment_start_number number`

设置片段开始序号，默认为0

- `strftime 1|0`

定义是否使用 `strftime` 功能来产生新段。如果设置了，输出段名需要依模板由 `strftime` 生成，默认为0。

- `break_non_keyframes 1|0`

如果设置为允许，将允许段在非关键帧点切分。这将改善一下关键帧间隔不一致的播放，但会产生很多奇怪的问题。默认为0

- `reset_timestamps 1|0`

在每个段都重新开始时间戳。所以每个段都有接近于0的时间戳。这有利于片段的播放，但很多混合器/编码器不支持，默认为0

- `initial_offset offset`

指定时间戳抵消适用于输出包的时间戳。参数必须是一个时间规范,默认为 0.

segment, stream_segment, ssegment例子

- 重新混合输入的in.mkv生成out-000.nut, out-001.nut,列表, 并且把生成文件的列表写入out.list:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.list out%03d.nut
```

- 按输出格式、选项分拆输入:

```
ffmpeg -i in.mkv -f segment -segment_time 10 -segment_format_options movflags=+faststart out%03d.mp4
```

- 按指定时间点分 (由 segment_times 进行指定) 拆输入文件

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 out%03d.nut
```

- 使用 force_key_frames 选项强制关键点进行切分, 还指定了 segment_time_delta 来处理计算机进程。时间点不够精确

```
ffmpeg -i in.mkv -force_key_frames 1,2,3,5,8,13,21 -codec:v mpeg4 -codec:a pcm_s16le -map 0 \
-f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 -segment_time_delta 0.05 out%03d.nut
```

为了强制关机帧, 必须进行转码

- 按帧号进行分段, 由 segment_frames 选项指定了若干帧号:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.csv -segment_frames 100,200,300,500,800
out%03d.nut
```

- 转换in.mkv 为TS段, 并且采用了libx264 和 libfaac 编码器:

```
ffmpeg -i in.mkv -map 0 -codec:v libx264 -codec:a libfaac -f ssegment -segment_list out.list out%03d.ts
```

- 对输入分段, 创建了M3U8直播列表 (可以作为HLS直播源):

```
ffmpeg -re -i in.mkv -codec copy -map 0 -f segment -segment_list playlist.m3u8 \
-segment_list_flags +live -segment_time 10 out%03d.mkv
```

smoothstreaming

平滑流混合器生成一组文件 (清单、块), 适用于传统web服务器

- window_size

指定清单中保留的片段数。默认是0, 表示保留所有的

- extra_window_size

从磁盘移除前, 保留清单外片段数, 默认5

- lookahead_count

指定先行片段数, 默认2

- min_frag_duration

指定最小片段持续时间 (单位microseconds), 默认5000000.

- `remove_at_exit`

指定完成后是否移除所有片段，默认0，表示不移除

tee

tee混合器可以用于同时把相同数据写入多个文件，或者任何其他类型的混合器。例如使用它可以同时把视频发布到网络上以及保存到磁盘上。

它不同于在命令行指定多个输出，因为利用tee混合器，音频和视频数据只被编码了一次，而编码是一个非常昂贵的行为。它是很有效的，当利用libavformat的API直接可以把相同的数据包用于多个混合器输出（多种封装格式或者场景）

多个输出文件由'|'分隔，如果参数中包含任意前导或尾随的空格，任何特殊字符都必须经过转义(参考 `ffmpeg-utils(1)`手册中的 "Quoting and escaping" 章节)。

混合器的选项可以由被“:”分隔的 `key=value` 列表进行指定。如果这种形式下选项参数值包含特殊字符，例如“:”则必须被转义。注意这个第二层次的转义

下列选项被要求:

- `f`

指定格式名，通常用于不能由输出名后缀推测格式的情况

- `bsfs[/spec]`

指定一个比特流滤镜应用到指定的输出

它可以为每个流指定一个比特流滤镜，通过"/"添加一个流选择（说明符），有些流必须由说明符进行指定(格式规范见流说明符)。如果流说明符没有指定，则比特流滤镜适用于所有输出流。

可以同时指定多个比特流滤镜，用","分隔。

- `select`

选择一些流，它们可以映射到一些输出，通过流说明符进行指定。如果没有指定，则默认会选择所有输入流

tee例子

- 同时编码到WebM文件和UDP协议上的MPEG-TS流(流需要明确的被映射):

```
ffmpeg -i ... -c:v libx264 -c:a mp2 -f tee -map 0:v -map 0:a
```

```
"archive-20121107.mkv|[f=mpegts]udp://10.0.1.255:1234/"
```

- 使用ffmpeg编码输入，有3个不同的目标。`dump_extra` 比特流滤镜被用来为所有输出的视频关键帧添加额外的信息，其作为MPEG-TS格式的要求。对out.aac附加的选项是为了让它只包含音频。

```
ffmpeg -i ... -map 0 -flags +global_header -c:v libx264 -c:a aac -strict experimental
```

```
-f tee "[bsfs/v=dump_extra]out.ts|[movflags=+faststart]out.mp4|[select=a]out.aac"
```

- 下面，将只选择一个音频流给音频输出。注意第二层引号必须经过转义，":"作为特殊字符被用于标识选项

```
ffmpeg -i ... -map 0 -flags +global_header -c:v libx264 -c:a aac -strict experimental
```

```
-f tee "[bsfs/v=dump_extra]out.ts|[movflags=+faststart]out.mp4|[select='\a:1\']out.aac"
```

注意一些编码器会根据输出格式的不同要求不同的选项，在tee混合器下自动检测可能会失效。主要有 global_header 的例子

webm_dash_manifest

WebM DASH 清单混合器。

这个混合器实现了按WebM DASH清单规范生成DASH清单XML文件。它还支持生成DASH直播流

更多参考：

- WebM DASH Specification: <https://sites.google.com/a/webmproject.org/wiki/adaptive-streaming/webm-dash-specification>
- ISO DASH Specification: http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip

webm_dash_manifest选项

支持如下选项：

- adaptation_sets

这个选项参数有如下语法："id=x,streams=a,b,c id=y,streams=d,e" 这里的x, y都是唯一合适设置的标识符，a,b,c,d和e是相应的音频和视频流的指代。任何合适的数字可以被用于这个选项。

- live

如果为1表示创建一个直播流DASH，默认为0

- chunk_start_index

第一个块的索引号，默认为0，它将作为清单中'SegmentTemplate'元素的'startNumber'属性值

- chunk_duration_ms

每个块的持续时间，单位milliseconds,默认1000，将作为清单中'SegmentTemplate'元素的'duration'属性值

- utc_timing_url

URL将指示从何处获取UTC时间戳（ISO格式的），它作为清单中'UTCTiming'元素的'value'属性值，默认：None.

- time_shift_buffer_depth

最小时间（单位秒）的移动缓冲区，为保障可用的任意值，作为清单中'MPD'元素的'timeShiftBufferDepth'属性值，默认：60.

- minimum_update_period

清单最小更新时间（单位秒），清单中'MPD'元素的'minimumUpdatePeriod'属性值，默认：0.

webm_dash_manifest例子

```
ffmpeg -f webm_dash_manifest -i video1.webm \
-f webm_dash_manifest -i video2.webm \
-f webm_dash_manifest -i audio1.webm \
-f webm_dash_manifest -i audio2.webm \
-map 0 -map 1 -map 2 -map 3 \
-c copy \
-f webm_dash_manifest \
-adaptation_sets "id=0,streams=0,1 id=1,streams=2,3" \
manifest.xml
```

webm_chunk

WebM直播块混合器

这个混合器输出WebM头和块分离文件，通过DASH它可以被支持WebM直播流的客户端处理。

webm_chunk选项

支持如下选项：

- chunk_start_index

第一个块的序号，默认0

- header

文件名将写入初始化数据的头

- audio_chunk_duration

每个音频块时间，单位milliseconds (默认5000).

webm_chunk例子

```
ffmpeg -f v4l2 -i /dev/video0 \
-f alsa -i hw:0 \
-map 0:0 \
-c:v libvpx-vp9 \
-s 640x360 -keyint_min 30 -g 30 \
-f webm_chunk \
-header webm_live_video_360.hdr \
-chunk_start_index 1 \
webm_live_video_360_%d.chk \
-map 1:0 \
-c:a libvorbis \
-b:a 128k \
-f webm_chunk \
-header webm_live_audio_128.hdr \
-chunk_start_index 1 \
-audio_chunk_duration 1000 \
webm_live_audio_128_%d.chk
```

23 元数据（metadata）

FFmpeg能够提取媒体文件元数据，并转储到一个简单的utf-8编码的类INI文本文件中,然后在分离器/混合器中再次使用

转储的文件格式为：

1. 文件包含一个头，以及一些元数据标签，元数据放置在各自子节的行中
2. 文件头有一个‘FFMETADATA’字符串，紧接着版本号（目前为1）
3. 元数据标签以‘key=value’形式给出
4. 头紧接着是全局元数据
5. 在全局元数据后可能有分部的元数据（每个流/每个章）
6. 分节元数据从分节名，由('[,]')括起的大写字符串（STREAM 或者 CHAPTER），直至下一节或者文件结束
7. 在一章的开始部分可能有一个可选的时基用于开始/结束值(start/end)，其形如 `TIMEBASE=num/den`，这里 `num` 和 `den` 是整数。如果没有设置，则开始/结束 时间以milliseconds为单位

下一章（节）的元数据描述包含了开始结束时间的（形如‘START=num’, ‘END=num’）则时间值（这里的 `num`）必须是正整数

8. 空行（无效）,开始字符是";"或者"#"的行被忽略
9. 如果元数据标签或者值中包含特殊字符('=', ';', '#', '\和 回车/换行)，必须由'\'进行转义
10. 注意空格在元数据中（例如‘foo = bar’）会被认为是标签的一部分（前面的标签关键字是‘foo’——注意有一个空格的，值是‘bar’——也有一个空格的）

一个ffmetadata文件大致像：

```
;FFMETADATA1
title=bike\shed
;this is a comment
artist=FFmpeg troll team
[CHAPTER]
TIMEBASE=1/1000
START=0
#chapter ends at 0:01:00
END=60000
title=chapter \#1
[STREAM]
title=multi\
line
```

通过使用ffmetadata，混合器和分离器可以从输入的ffmetadata文件中导出元数据，也可以编辑ffmetadata文件以转换输出到输出文件中

利用ffmetadata导出元数据：

```
ffmpeg -i INPUT -f ffmetadata FFMETADATAFILE
```

从FFMETADATAFILE 文件中加载元数据信息输出到输出文件中：

```
ffmpeg -i INPUT -i FFMETADATAFILE -map_metadata 1 -codec copy OUTPUT
```

24 协议

FFmpeg 协议配置元素,用于访问资源时要求特定的协议。

默认编译时会自动支持所有可用协议。你可以在编译脚本中添加 "-list-protocols"选项来了解有哪些协议被支持。

你也可以在编译时通过 "-disable-protocols"禁止所有的协议支持，然后通过 "-enable-protocol=PROTOCOL"来启用个别协议，或者在默认基础上通过 "-disable-protocol=PROTOCOL"关闭个别协议支持。

在ff*工具集中，"-protocols"选项可以了解编译支持了的协议

当前有效协议介绍如下。

bluray

读取BluRay（蓝光）播放列表，其接受如下选项：

- angle

BluRay 角度/方向

- chapter

开始章(1...N)

- playlist

在(BDMV/PLAYLIST/?????.mpls) 读取播放列表

例如：

从加载映射目录/mnt/bluray 读取很长的蓝光播放列表

```
bluray:/mnt/bluray
```

从加载映射目录/mnt/bluray 读取蓝光播放列表，从章2（chapter 2）开始，读取 playlist 4 的 angle 2

```
-playlist 4 -angle 2 -chapter 2 bluray:/mnt/bluray
```

cache

对输入流的缓冲封装

缓存输入流到临时文件，以提供对在线流的搜寻/定位支持。

```
cache:URL
```

concat

物理连接协议

从许多资源（资源序列，把它们当作一种独特的资源）中读取和搜索定位

这个协议按如下语法接受URL：

```
concat:URL1|URL2|...|URLN
```

这里的URL1、URL2...URLN都是需要连接起来的资源url，每个可能是一个不同的协议

例如，需要利用 `ffplay` 播放一个序列的资源(包含split1.mpeg,split2.mpeg,split3.mpeg):

```
ffplay concat:split1.mpeg\|split2.mpeg\|split3.mpeg
```

注意在大多数 `shell` 中可能你需要对"`|`"进行转义

crypto

AES加密(AES-encrypted)流读取协议，其接受如下选项：

- `key`：设置AES解密密钥，是用16进制表示的二进制数据块
- `iv`:设置AES解密初始化向量二进制块，也是用16进制表示。

允许如下的URL格式：

- `crypto:URL`
- `crypto+URL`

data

在URI中的行内数据，参考http://en.wikipedia.org/wiki/Data_URI_scheme

例如可以利用ffmpeg 转换行内的GIF文件数据

```
ffmpeg -i
"data:image/gif;base64,R0lGODdhCAAIAMIEAAAAAAAAA//8AAP/AP//////////ywAAAAACAAIAADF0gEDLojDgdGi
JdJqUX02iB4E8Q9jUMkADs=" smiley.png
```

file

文件访问协议，实现对文件的读写。

文件协议的URL格式为：

```
file:filename
```

这里的 `filename` 是文件的路径

如果在URL中没有前导协议头则会作为一个文件URL。根据编译的版本，URL可以是Windows下的包含盘符的文件URL（通常在类Unix系统上不支持）

例如利用 `ffmpeg` 读取 `input.mpeg` 文件：


```
ffmpeg -i file:input.mpeg output.mpeg
```

这个协议支持下面的选项：

- truncate

如果设置为1，则如果文件存在，则截断文件的写入（进行覆盖），如果设置为0则文件存在时不会被删除（覆盖掉），默认为1

- blocksize

设置可选的I/O最大块尺寸，单位bytes。默认值是 `INT_MAX`，它让请求的块没有尺寸限制。合理设置这个值可以提高用户请求反应时间，这对低速媒体上的文件读取有益。

ftp

FTP（File Transfer Protocol——文件传输）协议

通过FTP协议进行远程读写

请求语法是：

```
ftp://[user[:password]@]server[:port]/path/to/remote/resource.mpeg
```

这个协议接受如下选项：

- timeout

以毫秒（microseconds）为单位，设置socket I/O操作超时时间。默认值为 `-1` 表示没有指定

- ftp-anonymous-password

当作为 `anonymous` 用户登录时的密码，通常为一个 `e-mail` 地址（`anonymous@FTP-address`）

- ftp-write-seekable

控制持续编码时连接的可搜索性。如果设置为1表示可以搜索，如果设置为0则不可搜索，默认为0

注意协议可以用于输出，但通常不建议这样使用，除非特殊任务要求（测试时、定制的服务器配置等等）。不同的FTP服务提供不同的持续定位处理。ff*工具对它们的支持是不完整的。

hls

把Apple的HTTP直播分段流作为一个统一的进行读取。描述分段的 `m3u8` 播放列表文件可以是远程HTTP资源或者本地文件（通过标准文件协议）。通过在 `hls` 后附加 `+proto` 的方式嵌套指定 `hls` 的URI，这里 `proto` 可以是 `file` 或者 `http`

```
hls+http://host/path/to/remote/resource.m3u8
hls+file://path/to/local/resource.m3u8
```

使用这个协议将自动采用 `hls` 分离器（如果不是请报告问）。为了使用 `hls` 分离器，可以把URL简单的指向 `m3u8` 文件。

http

HTTP (Hyper Text Transfer Protocol)协议

协议支持如下选项：

- seekable

控制连接是否可搜索，如果设置为1表示资源支持搜索，否则为0。如果设置为-1则尝试自动检测是否支持搜索。默认为-1

- chunked_post

如果设置为1则启用分块传输编码，默认为1

- content_type

设置一个特定内容类型的消息。

- headers

设置定制的 HTTP 头，它可以覆盖掉默认头。这些值必须是可在头中编码的字符串。

- multiple_requests

为1表示使用持久连接，默认为0

- post_data

设置定制的 HTTP 数据

- user-agent

- user_agent

覆盖掉用户代理（客户端类型）头。如果没有指定，则采用在 `libavformat` 编译中的字符串 ("Lavf/")

- timeout

以毫秒（microseconds）为单位设置I/O操作超时时间。默认为-1，表示没有指定。

- mime_type

导出MIME类型

- icy

如果设置为1将从服务器请求ICY (SHOUTcast)元数据。如果服务器支持，将通过 `icy_metadata_headers` 和 `icy_metadata_packet` 传递具体的请求，默认值为1。

- icy_metadata_headers

如果服务器支持ICY元数据，它将包含由换行符分隔（即每行1个）的ICY-specific HTTP 应答头

- icy_metadata_packet

如果服务器支持ICY元数据，且 `icy` 被设置为1，这将包含服务器发送的最后非空元数据包，它用于定期处理程序执行期间中间流元数据更新

- cookies

设置未来请求的cookie。格式为 `Set-Cookie HTTP` 响应一样，多个cookie值可以由换行符隔开。

- offset

设置初始字节偏移量

- end_offset

尝试偏移限定，单位字节

- method

当用在客户端选项时，它设置请求的HTTP方法。

当用在服务器端选项时，它设定（预计）从客户机获要用的方法。如果预期和收到的HTTP方法都不能与客户端匹配将是一个很糟糕的问题。当前对是否设置本方式不检测，以后会自动检测。

- listen

如果设置为1，将使用实验性质的HTTP服务。它可以用于在输出中指定数据（例如）——对输出文件，或者通过 `HTTP POST` 读取输入端数据

```
# 服务器段（发送）：
ffmpeg -i somefile.ogg -c copy -listen 1 -f ogg http://server:port

# 客户端（接收）：
ffmpeg -i http://server:port -c copy somefile.ogg

# 客户端可以像下面这样工作
wget http://server:port -O somefile.ogg

# 服务器端接收

# 客户端（发送）：
ffmpeg -i somefile.ogg -chunked_post 0 -c copy -f ogg http://server:port

# 客户端也可以联用在`wget`中
wget --post-file=somefile.ogg http://server:port
```

HTTP Cookie

一些HTTP请求将被拒绝，除非cookie值按要求进行传递。`cookies` 选项允许对cookie进行指定。至少cookie需要指定一个路径和域。HTTP请求将自动匹配域和路径，并把包含的cookie值放置HTTP Cookie头中。多个cookie可以由换行分隔。

下面的请求语法就是播放一个流时指定了cookie：

```
ffplay -cookies "nlqptid=nltid=tsn; path=/; domain=somedomain.com;" http://somedomain.com/somestream.m3u8
```

Icecast

Icecast (stream to Icecast servers) 协议

协议支持如下的选项：

- ice_genre

设置流样式

- ice_name
设置流名字
- ice_description
设置流说明
- ice_url
设置流web URL
- ice_public
为1表示流可以被公开，否则为0表示不能公开，默认为0（不公开）
- user_agent
覆盖用户代理头。如果没有指定则 "Lavf/" 被使用
- password
设置Icecast挂载password.
- content_type
设置内容类型。它对audio/mpeg 必须设置为不同
- legacy_icecast
这将使得支持 Icecast的版本< 2.4.0,不过源方法将不再支持HTTP PUT方法。

协议URL指定方法：

```
icecast://[username[:password]@]server:port/mountpoint
```

mmst

基于TCP的MMS (Microsoft Media Server)协议.

mmsh

基于HTTP的MMS (Microsoft Media Server)协议

请求语法:

```
mmsh://server[:port][/app][/playpath]
```

md5

MD5输出协议

计算要写数据的MD5,它可以写入文件或者标准输出设备上（如果没有指定输出文件），可以用来测试混合器的输出而无需真

实写入文件。

下面是一些例子：

- 把编码成avi文件对应的md5写入output.avi.md5

```
ffmpeg -i input.flv -f avi -y md5:output.avi.md5
```

- 把编码成avi文件对应的md5输出到标准输出设备上

```
ffmpeg -i input.flv -f avi -y md5:
```

注意一些格式（例如MOV）要求输出协议是可以搜索/检索（seekable）的，这时MD5输出协议将不被支持（对于这样的MD5计算要求，只能是先生成文件，再利用其它工具对文件计算MD5）。

pipe

UNIX先入先出访问协议（管道）

通过UNIX pipe读写

语法为：

```
pipe:[number]
```

这里 `number` 是对应管道文件描述符（例如0对应stdin,1对应stdout,2对应stderr）。如果 `number` 没有指定，默认为 `stdout` 被用于输出写，`stdin` 被用于输入读

例如 `ffmpeg`从stdin读取

```
cat test.wav | ffmpeg -i pipe:
```

又如写入标准输出：

```
ffmpeg -i test.wav -f avi pipe:1 | cat > test.avi
```

它等效于

```
ffmpeg -i test.wav -f avi pipe: | cat > test.avi
```

协议支持如下选项：

- `blocksize`

设置I/O操作最大块尺寸，单位byte（字节）。默认是 `INT_MAX`，表示没有限制。如果设置了这个值将改善对数据传输缓慢的情况下用户终止请求反应的时间。

注意一些格式（例如MOV）要求输出协议是可以搜索/检索（seekable）的，这时pipe输出协议将不被支持

rtmp

rtmp (Real-Time Messaging Protocol.)

RTMP被用通过TCP/IP网络流式处理多媒体内容

请求语法：

```
rtmp://[username:password@]server[:port][/app][/instance]/[playpath]
```

其中各个参数为：

- username
可选的用户名，一般在公开发布时
- password
可选的密码，一般在公开发布时
- server
RTMP服务器地址
- port
TCP端口号，默认1935
- app
它是根据应用程序的名称来访问，它通常对应于应用程序在RTMP服务器上的安装路径（例如 `/ondemand/`，`/flash/live/` 等等），你可以通过 `rtmp_app` 选项值来覆盖URI
- playpath
是通过 `app` 指定的要播放资源的路径或者名称，可能有 `mp4:` 前缀，你可以通过 `rtmp_playpath` 选项值来覆盖
- listen
作为服务器，侦听传入的连接
- timeout
设置传入连接最大等待时间，其间一直侦听

额外的，下面的参数可以通过命令行（或者在代码中通过 `AVOptions`）设置

- rtmp_app
连接到RTMP服务的程序名。它可以用于覆盖URI中对应部分
- rtmp_buffer
设置客户端缓冲时间，单位milliseconds,默认3000.
- rtmp_conn
额外的AMF连接参数，参数值是字符串，从中解析出AMF信息。例如：`B:1 S:authMe O:1 NN:code:1.23 NS:flag:ok`

O:0.每个值有一个单字符前导（B表示布尔值，N表示数字值，S表示字符串，O表示对象，Z表示null），后面加冒号“:”，对于布尔值，范围为0或者1分别对应 FALSE 和 TRUE。同样对于对象值，必须由0或者1开始或者结束。数据子项可以被命名，如果数据前面指定了N并且在值前面命名了数据（例如NB:myFlag:1），这个选项可以被多次在AMF序列中使用。

- rtmp_flashver

SWF播放器插件版本，默认 LNX 9,0,124,2. (当发布，默认为FMLE/3.0 (兼容与).)

- rtmp_flush_interval

在同一请求的数据包刷新数量(只RTMPT)。默认值是10。

- rtmp_live

指示是一个直播流。直播流不能重放和搜索。默认值为 any ,它意味着首先尝试按直播流去搜索 playpath , 如果没有找到再以记录流去搜索。其他参数值为 live 和 recorded

- rtmp_pageurl

嵌入在URL中的媒体页面。默认没有值被发送

- rtmp_playpath

要播放/发布的流的ID。这个值将覆盖URI中的对应部分

- rtmp_subscribe

直接订阅的直播流名字，默认没有值被发送。这个值仅在被指定，而且 rtmp_live 被设置为 live 时发送

- rtmp_swfhash

解压SWF文件的SHA256(32 bytes).

- rtmp_swfsize

解压SWF文件的尺寸，被 SWFVerification 要求

- rtmp_swfurl

对应媒体SWF播放器的URL。默认没有值被发送

- rtmp_swfverify

播放对应swf文件URL，会自动计算hash/size

- rtmp_tcurl

目标流URL，默认为 proto://host[:port]/app 例如利用ffplay播放 myserver 服务器上 vod 应用下的 sample 资源：

ffplay rtmp://myserver/vod/sample 发布一个资源到密码保护的服务器上

ffmpeg -re -i  -f flv -rtmp_playpath some/long/path -rtmp_app long/app/name

rtmp://username:password@myserver/ 注这里最终发布的URL

为 rtmp://username:password@myserver/long/app/name/some/long/path

rtmpe

加密的Real-Time Messaging Protocol.

通过提供的标准加密原语加密流式多媒体内容的RTMPE。由Diffie-Hellman 和 HMACSHA256作为密钥交换，产生一对RC4 密钥

rtmps

基于安全SSL的rtmp

通过SSL通道传输的rtmp

rtmpt

基于HTTP的Rtmp

由HTTP通道承载的RTMP，这样可以突破很多防火墙限制。

rtmpte

加密RTMP，且通过http传输

在HTTP通道承载下，进行加密rtmp的传输。

rtmpts

通过HTTPS传输的rtmp。

通过HTTPS进行传输的rtmp，它可以突破很多防火墙并提供安全的连接

libsmbclient

libsmbclient提供对CIFS/SMB网络资源访问支持

支持一下语法：

```
smb://[[domain:]user[:password@]]server[/share[/path[/file]]]
```

支持下面选项：

- timeout

以miliseconds为单位设置I/O操作时限。默认设置为-1，表示没有限制

- truncate

如果设置为1，表示文件存在则覆盖，否则为0，默认为1

- workgroup

设置工作组属性覆盖参数，默认未设置

更多信息参考<http://www.samba.org/>

libssh

通过libssh提供安全文件传输协议

可以通过SFTP协议读写远程资源

请求按下面语法：

```
sftp://[user[:password]@]server[:port]/path/to/remote/resource.mpeg
```

支持下面选项：

- `timeout`
以milliseconds为单位设置I/O操作时限。默认设置为-1，表示没有限制
- `truncate`
如果设置为1，表示文件存在则覆盖，否则为0，默认为1
- `private_key`
指定一个文件路径用于持续期认证，默认在 `~/.ssh/` 搜索密钥。

例如：播放一个远程服务器上资源

```
ffplay sftp://user:password@server_address:22/home/user/resource.mpeg
```

librtmp rtmp, rtmpe, rtmps, rtmpt, rtmpte

通过librtmp提供的rtmp（以及rtmpe, rtmps, rtmpt, rtmpte）

在编译时需要明确指定相应的头和库，并且在配置中通过"`--enable-librtmp`"予以编译允许，它将替代原生的RTMP协议族支持

它提供了关于（rtmp, rtmpe, rtmps, rtmpt, rtmpte）大部分客户端功能和一些服务器功能。

请求语法为：

```
rtmp_proto://server[:port][[/app][/playpath] options
```

这里 `rtmp_proto` 可以是（rtmp, rtmpe, rtmps, rtmpt, rtmpte）中的一个，此外诸如 `server`、`port`、`app`、`playpath` 都和在原生RTMP中一样。`options` 是由空格分隔的 `key=val` 值对列表

例如，在ffmpeg中流式输出到RTMP协议：

```
ffmpeg -re -i myfile -f flv rtmp://myserver/live/mystream
```

用ffplay播放这个流：

```
ffplay "rtmp://myserver/live/mystream live=1"
```

rtp

Real-time Transport Protocol——RTP，实时传输协议

请求语法：`rtp://hostname[:port][?option=val...]`

- `port`：用于指定RTP端口

下面是被支持的URL选项：

`ttl=n`

设置TTL(Time-To-Live)值，仅多播

`rtcpport=n`

设置远端RTCP端口为n。

`localrtpport=n`

设置本地RTP端口为n。

`localrtcpport=n'`

设置本地RTCP端口为n。

`pkt_size=n`

设置最大包尺寸为n，单位 bytes。

`connect=0|1`

为1则在UDP socket上连接（`connect()`），否则（为0）不连接

`sources=ip[,ip]`

列出允许的源IP地址

`block=ip[,ip]`

列出禁止的源IP地址

`write_to_source=0|1`

如果为1，将数据包发送到最新收到的源地址，否则发送到默认远程地址（为0）

localport=n

设置本地RTP端口为n

这时一个废弃选项，用`localrtpport`替代

重要提示：

1. 如果 `rtcpport` 没有被设置，则RTCP端口将是RTP端口加1
2. 如果 `localrtpport`（本地RTP端口）没有设置，则任何有效端口可能被用作本地RTP/RTCP端口
3. 如果 `localrtcpport`（本地RTCP）没有设置，则默认为本地RTP端口加1.

rtsp

Real-Time Streaming Protocol——RTSP，实时流协议

RTSP在libavformat中不是作为技术协议处理，而是作为分离器和混合器。分离器支持两种常见的RTSP（基于RTP的数据传输，被用在Apple 和Microsoft）和Real-RTSP（基于RDT的数据传输）

混合器可以用来发送流（使用 `RTSP ANNOUNCE`）给支持它的服务器（当前有 `Darwin` 流服务器和 `Mischa Spiegelmock` 的 [RTSP server](#)）

请求的语法为：

```
rtsp://hostname[:port]/path
```

一些选项可以用于ffmpeg/ffplay命令行，或者通过 `AVOptionS` 进行编码设置，或者在 `avformat_open_input` 进行编码。

下面的选项被支持：

- `initial_pause`

为1表示不立即开始，否则为0为立即开始，默认为0

- `rtsp_transport`

设置RTSP传输承载协议，有如下值：

`'udp'`

采用UDP作为底层传输协议

`'tcp'`

采用TCP作为底层传输协议（RTSP控制通道也交叉混合在一起）

`'udp_multicast'`

采用UDP多播为底层传输协议

'http'

采用HTTP隧道为底层传输协议，它通常用于代理

多个底层传输协议可以被指定，这时它们按顺序依次被尝试（一个失败就试下一个）。对应混合器，只支持'tcp'和'udp'

- rtsp_flags

设置RTSP标志，下面的值被支持：

'filter_src'

仅从对等协商的地址和端口接收数据包

'listen'

当作服务器，侦听连接

'prefer_tcp'

首先尝试采用TCP作为传输，如果TCP可用，就使用RTSP RTP 传输

默认为 'none'.

- allowed_media_types

设置从服务器端接收数据类型，下面标志被允许：

'video' 'audio' 'data'

默认接受所有类型

- min_port

设置最小的UDP端口，默认为 5000.

- max_port

设置最大的UDP端口，默认为 65000.

- timeout

设置等待数据连接的最大时间，单位秒

默认值为-1，表无限。这个选项蕴含着 `rtsp_flags` 选项被设置为'listen'.

- reorder_queue_size

设置在缓冲区处理重排序的包数量

- stimeout

设置socket TCP I/O超时时间，单位microseconds.

- user-agent

覆盖用户代理头。如果不指定，将默认为libavformat标识字符串

rtsp例子

下面是使用ffplay和ffmpeg工具的例子

- 使用UDP最大延迟0.5秒的播放

```
ffplay -max_delay 500000 -rtsp_transport udp rtsp://server/video.mp4
```

- 观看HTTP隧道流播放

```
ffplay -rtsp_transport http rtsp://server/video.mp4
```

- 发送流到RTSP服务器

```
ffmpeg -re -i input -f rtsp -muxdelay 0.1 rtsp://server/live.sdp
```

- 获取实时流

```
ffmpeg -rtsp_flags listen -i rtsp://ownaddress/live.sdp output
```

sap

Session Announcement Protocol(RFC 2974),会话通告协议。它在libavformat中不作为技术协议处理，而是作为混合器和分离器。它被用于RTP流信号，以宣布SDP流经常在一个单独的端口。

sap作为混合器

作为分离器时的SAP URL语法

```
sap://destination[:port][?options]
```

RTP数据包被送往 destination 的 port 端口，如果 port 没有指定则默认为5004. options 是一个“&”分隔的列表，允许下面的值：

- announce_addr=address

指定目的地IP地址发送公告地址，如果省略，则采用SAP多播地址 224.2.127.254 (sap.mcast.net),或者 ff0e::2:7ffe (IPv6)

- announce_port=port

指定公告发送端口，默认9875

- ttl=ttl

指定公告和RTP包的TTL（time to live）值，默认255。

- `same_port=0|1`

如果设置为1，则所有的RTP流在同一个端口发送，如果设置为0（默认），则所有流在独立的端口发送，每个流在比前一个流高2号的端口进行发送。VLC/Live555 要求设置为1，以可以获取流。`libavformat` 下的RTP则要求独立端口，即需要设置为0

sap作为分离器

作为分离器的语法：

```
sap://[address][:port]
```

这里 `address` 是侦听的多播地址，如果省略则为 `224.2.127.254` (`sap.mcast.net`)。`port` 是侦听端口，默认9875

demuxers在给定的地址和端口监听公告。一旦收到公告,它试图接受特定流

命令行例子如下：

播放常规的SAP多播地址的第一个流：

```
ffplay sap://
```

为了播放IPV6 SAP多播地址的第一个流

```
ffplay sap://[ff0e::2:7ffe]
```

sctp

Stream Control Transmission Protocol——SCTP，流控制传输协议

允许的URL语法：

```
sctp://host:port[?options]
```

协议接收如下选项：

- `listen`

可以设置为任何值，用于侦听传入连接。所有外向/传出连接都被作为默认值

- `max_streams`

设置最大数量的流，默认没有限制

srtp

Secure Real-time Transport Protocol——SRTP，安全实时传输协议

接收如下选项：

- `srtp_in_suite`
- `srtp_out_suite`

选择输入和输出编码套件

支持的值：

`'AES_CM_128_HMAC_SHA1_80'` `'SRTP_AES128_CM_HMAC_SHA1_80'` `'AES_CM_128_HMAC_SHA1_32'`
`'SRTP_AES128_CM_HMAC_SHA1_32'`

- `srtp_in_params`
- `srtp_out_params`

设置输入和输出编码参数，采用base64编码表示的二进制块。第一个16字节的块用作主要密钥，随后的14字节作为密钥的“盐”

subfile

提取一段文件或者流的模拟。必须是可以定位/搜索的底层流。

接收的选项：

- `start`

提取段的开始位置偏移，单位byte（字节）

- `end`

提取段的结束位置偏移，单位byte（字节）

例子：

- 从DVD的VOB文件中提取章节（节的开始和结束区块数乘以2048）

`subfile,,start,153391104,end,268142592,,:/media/dvd/VIDEO_TS/VIDEO_TS_08_1.VOB`

- 播放直接从TAR打包中播放AVI文件：

`subfile,,start,183241728,end,366490624,,:archive.tar`

tcp

Transmission Control Protocol——TCP，传输控制协议

请求的TCP url语法为：

```
tcp://hostname:port[?options]
```

其中 `options` 是由“&”分隔的 `key=val` 对选项列表

允许下面一些选项：

- listen=1|0

是否对传入连接侦听，默认为0，表示不侦听

- timeout=microseconds

设置超时错误抛出时间，单位毫秒 这个选项仅工作在读模式：如果超过设置时间周期没有获取到数据则抛出错误

- listen_timeout=milliseconds

设置侦听超时，单位毫秒 下面的例子展示了如何在ffmpeg中应用TCP连接，以及如何用ffplay播放一个TCP内容

```
ffmpeg -i input -f format tcp://hostname:port?listen ffplay tcp://hostname:port
```

tls

Transport Layer Security (TLS) / Secure Sockets Layer (SSL)

对于TLS/SSL的 URL 语法是：

```
tls://hostname:port[?options]
```

下面是可以在命令行中设置的选项（或者通过 `AVOptions` 在编程时设置）

- ca_file, cafile=filename

设置含证书颁发机构(CA)作为受信任的根证书的文件。如果连接的TLS库包含一个默认而不需要指定即可工作的值，但不是所有库和配置都有默认值。文件必须是OpenSSL PEM格式

- tls_verify=1|0

如果允许，则试图对等验证。注意如果使用OpenSSL，这是目前唯一支持通过的CA根证书签署数据库对等验证，但它在试图连接到一个主机时并不验证根证书匹配（GnuTLS验证了主机名）。

默认禁用，因为大多时候它需要调用者提供一个CA数据库

- cert_file, cert=filename

设置包含一个用于处理同行连接的证书。(当操作服务器,在侦听模式中,这是经常需要证书的,而客户端证书仅在某些设置中授权。)

- key_file, key=filename

设置包含证书的私钥文件

- listen=1|0

如果允许，则在提供的端口上监听连接，并假设当前为服务器角色，而不是客户端角色

命令行中的例子：

- 根据输入流创建一个TLS/SSL服务

```
ffmpeg -i input -f format tls://hostname:port?listen&cert=server.crt&key=server.key
```


- 播放一个TLS/SSL:

```
ffplay tls://hostname:port
```

UDP

User Datagram Protocol——UDP，用户数据报协议

请求UDP URL的语法：

```
udp://hostname:port[?options]
```

这里 `options` 是由"&"分隔的 `key=val` 选项列表

在启用线程模式的系统，一个循环缓存被用于存储传入的数据，它可以减少数据由于UDP套接字（socket）缓冲区溢出的损失。`fifo_size` 和 `overrun_nonfatal` 选项就是关于这个缓冲区设置的。

下面列出支持的选项：

- `buffer_size=size`

设置UDP 最大socket 缓冲区大小，单位bytes,它用于设置接收或者发生的缓冲区大小，其取决于套接字的需求，默认为64KB。也可以参看 `fifo_size`

- `localport=port`

覆盖本地UDP端口来绑定

- `localaddr=addr`

选择本地IP地址，这是对需要设置多播或者主机有多个IP时是必要的。通过设置用户可以选择通过那个IP地址作为发送接口

- `pkt_size=size`

以字节为单位设置UDP包大小

- `reuse=1|0`

显式设置允许或者不允许UDP套接字(socket)

- `ttl=ttl`

设置time to live (TTL) 值 (仅对多播).

- `connect=1|0`

如果设置为1则利用 `connect()`初始化UDP套接字。在这种情况下目的地址不能由后面的 `ff_udp_set_remote_url` 改变，如果目的地址在开始时是未知的，这个选项可以允许指定 `ff_udp_set_remote_url`，它允许根据 `getsockname` 为包找到源地地址，如果目的地址不可到达则通过 `AVERROR(ECONNREFUSED)` 返回。它可以保证只从指定的地址/端口进行数据获取

- `sources=address[,address]`

只从多播组的一个指定发送IP地址获取数据包

- block=address[,address]

忽略多播组中指定IP地址的包

- fifo_size=units

设置UDP获取循环缓冲区大小，单位为188字节的包个数。如果没有指定，默认7*4096。

- overrun_nonfatal=1|0

生存的UDP接收循环缓冲区溢出，默认为0。

- timeout=microseconds

设置抛出超时错误的时限，单位毫秒

这个选项仅与读模式相关：如果超过设置的时间没有获取到数据则抛出超时错误。

- broadcast=1|0

显式地允许或不允许UDP广播

注意,在网络广播风暴的保护环境可能无法正常工作

UDP例子

- 使用ffmpeg输出流到远程UDP端点

```
ffmpeg -i input -f format udp://hostname:port
```

- 使用ffmpeg，流式输出到UDP端点，UDP包大小是188字节，使用一个大的输入缓冲区：

```
ffmpeg -i input -f mpegts udp://hostname:port?pkt_size=188&buffer_size=65535
```

- 使用FFmpeg获取基于UDP传来的远程端点数据：

```
ffmpeg -i udp://[multicast-address]:port ...
```

UNIX

Unix本地socket（套接字）

请求Unix socket的URL语法为：

```
unix://filepath
```

下面的参数允许在命令行设置（通过 `AVOptions` 在编码时设置）：

- timeout

以ms设置超时。

- listen

以侦听模式建立一个Unix socket

25 设备选项

`libavdevice` 库提供类似 `libavformat` 的接口，即一个输入设备被认为类似一个分离器活着输出设备类似一个混合器。这些接口也类似 `libavformat` 一样提供一些常规设备选项。（参考 `ffmepeg` 格式手册）。

当然，一些输入或者输出设备还提供一些私有的选项，它们只在特定的组件中有效。

可以做 `ffmpeg` 命令行中采用 `-option value` 来设定某个选项点，或者通过 `libavutil/opt.h` 中的API设置，或者通过 `AVFormatContext` 的显式值来进行设置。

26 输入设备

FFmpeg中的输入设备配置元素用来启用对附加到您的系统一个多媒体设备访问数据。

当编译时，默认会支持所有的输入设备。你可以通过在配置脚本执行时附加 `-list-indevs` 了解到支持的设备。

可以通过 `-disable-indevs` 在编译时禁用所有输入设备，也可以在此基础上通过 `-enable-indev=INDEV` 允许个别设备，或者在默认支持基础上通过 `-disable-indev=INDEV` 禁用个别设备支持达到类似的目的。

在ff*工具集中，使用 `-devices` 可以获取当前支持的设备信息。

下面是当前可用的输入设备介绍。

alsa

ALSA (Advanced Linux Sound Architecture——高级Linux音频架构) 输入设备

为了能够使用这个设备，在你的系统上必须安装有 `libasound` 库。

这个设备允许从ALSA设备采集，设备通过名称来作为ALSA卡标识符，以进行采集。

ALSA标识语法为：

```
hw: CARD[, DEV[, SUBDEV]]
```

这里 `DEV` 和 `SUBDEV` 是可选的。通过这3个参数（`CARD`、`DEV` 和 `SUBDEV`）可以指定一个卡的序号或者标识、设备序号和子设备序号（-1意味着任何一个）

在你的系统上要列出当前可用的卡，可以通过文件：`/proc/asound/cards` and `/proc/asound/devices`

例如要利用FFmpeg采集ALSA设备（卡ID为0），你可以如下：

```
ffmpeg -f alsa -i hw:0 alsaout.wav
```

更多信息参考<http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html>

avfoundation

AVFoundation 输入设备

AVFoundation是当前Apple OSX（>=10.7）下建议的流采集框架，它在IOS上也是可用的。而老的 `qtkit` 框架从OSX10.7开始已经废弃。

这个设备作为输入文件名的语法为：

```
-i "[[VIDEO]:[AUDIO]]"
```

第一部分选择视频输入，然后选择音频输入。流必须通过设备列表中的设备名或者设备索引号来指定。或者视频和/或音频输入设备可以通过使用 `-video_device_index <INDEX>` 和/或 `-audio_device_index <INDEX>` 语法指定，它将覆盖设备名或者索引来作为输入文件名。

所有有效的设备都可以通过使用 `-list_devices true` 枚举出来，它会列出所有设备的名称以及对应的索引号。

下面是两个设备的别名：

- `default`：选择AVFoundation默认设备（类型）。
- `none`：不记录相应的媒体类型，这相当于指定一个空的设备名或者索引

译者补注：`none`可以用来在进行指定时明确表示没有某种类型，比如

```
-i "none:[AUDIO]"
```

表示没有视频只有音频

avfoundation选项

avfoundation支持如下的选项：

- `-list_devices`

如果设置为 `true` 则列出所有有效输入设备，显示设备名和对应的索引

- `-video_device_index`

通过索引指定视频设备，它将覆盖作为输入文件名

- `-audio_device_index`

通过索引指定音频设备，它将覆盖作为输入文件名

- `-pixel_format`

描述视频设备采用的像素格式，如果不知道，将列出可用设备中第一个有效的支持格式。像素格式是：`monob`, `rgb555be`, `rgb555le`, `rgb565be`, `rgb565le`, `rgb24`, `bgr24`, `0rgb`, `bgr0`, `0bgr`, `rgb0`, `bgr48be`, `uyvy422`, `yuva444p`, `yuva444p16le`, `yuv444p`, `yuv422p16`, `yuv422p10`, `yuv444p10`, `yuv420p`, `nv12`, `yuyv422`, `gray`

avfoundation例子

- 输出AVFoundation支持的设备

```
$ ffmpeg -f avfoundation -list_devices true -i ""
```

- 从视频设备0和音频设备0 采集输出到out.avi:

```
$ ffmpeg -f avfoundation -i "0:0" out.avi
```

- 从视频输入设备2和音频输入设备1采集输出到 out.avi:

```
$ ffmpeg -f avfoundation -video_device_index 2 -i "2:1" out.avi
```

- 从系统默认视频设备以bgr0像素格式采集，而不采集音频到out.avi:

```
$ ffmpeg -f avfoundation -pixel_format bgr0 -i "default:none" out.avi
```

bktr

BSD 视频输入设备

decklink

decklink输入设备提供从Blackmagic DeckLink 采集的能力

要支持这个设备，编译时需要Blackmagic DeckLink SDK，且需要采用 `--extra-cflags` 和 `--extra-ldflags` 编译选项。在 Windows，你可能需要通过 `widl` 运行IDL。

DeckLink非常挑剔支持输入格式。像素格式万恶有 `uyvy422/210`。对于视频你必须利用 `-list_formats 1` 指定一个视频画面尺寸（`-list_formats 1.`）和帧率。音频采样率被设置为48KHz。音频数可能是2、8或16

decklink选项

- `list_devices`

如果设置为 `true`，输出设备列表然后退出，默认为`false`。

- `list_formats`

如果设置为 `true` ,输出支持的格式然后退出，默认为`false`。

- `bm_v210`

如果设置为1，则视频采集采用10bit量化的uyvy422 v210标准。不是所有的Blackmagic设备都支持这个选项

decklink例子

- 列出所有输入设备:

```
ffmpeg -f decklink -list_devices 1 -i dummy
```

- 列出支持的格式:

```
ffmpeg -f decklink -list_formats 1 -i 'Intensity Pro'
```

- 采集1080i50视频格式 (format 11):

```
ffmpeg -f decklink -i 'Intensity Pro@11' -acodec copy -vcodec copy output.avi
```

- 以10bit采集1080i50视频格式:

```
ffmpeg -bm_v210 1 -f decklink -i 'UltraStudio Mini Recorder@11' -acodec copy -vcodec copy output.avi
```

- 采集720p50格式，同时采集32bit音频:

```
ffmpeg -bm_audiodepth 32 -f decklink -i 'UltraStudio Mini Recorder@14' -acodec copy -vcodec copy output.avi
```

- 采集576i50采集视频，同时采集8路音频:

```
ffmpeg -bm_channels 8 -f decklink -i 'UltraStudio Mini Recorder@3' -acodec copy -vcodec copy output.avi
```

dshow

Windows DirectShow 输入设备。

DirectShow在ffmpeg中由 `mingw-w64` 项目提供支持。当前只有音频和视频设备能够使用。

多个单独输入的设备可能被打开,但它们也可能打开相同的输入,这将改善他们之间的同步

输入名可以按格式（语法）：

```
TYPE=NAME[:TYPE=NAME]
```

这里 `TYPE` 可以是 `audio` 或者 `video`，`NAME` 是设备名或者别名。

dshow选项

如果没有特别指定，将采用设备的默认值。如果设备不支持要求的选项，则会打开失败。

- `video_size`
设置采集视频的尺寸
- `framerate`
设置采集视频的帧率
- `sample_rate`
设置采集音频的采样率（单位Hz）
- `sample_size`
设置采集音频的采样位深（单位bits）
- `channels`
选择采集音频的通道
- `list_devices`
如果为真，输出设备列表并退出
- `list_options`
如果为真，输出选择设备的选项列表并退出
- `video_device_number`
对视频设备名设置索引编号(从0开始，默认0).
- `audio_device_number`
对音频设备名设置索引编号(从0开始，默认0).
- `pixel_format`
选择用于DirectShow的像素格式。当视频编码没有设置或者设置为 `rawvideo` 时需要设置
- `audio_buffer_size`

以milliseconds为单位设置音频设备缓冲大小（它可以直接影响延迟，则取决于依赖的设备）。默认使用设备默认缓冲（通常为500ms的倍数）。这个值设置过低会降低性能。参考[http://msdn.microsoft.com/en-us/library/windows/desktop/dd377582\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/dd377582(v=vs.85).aspx))

- video_pin_name

通过pin名称（或者别名）选择视频捕获源

- audio_pin_name

通过pin名称（或者别名）选择音频捕获源

- crossbar_video_input_pin_number

从交错/交叉设备（音视频交错编码）中选择视频输入端口。可以选择交错设备的视频解码输出端。注意改变这个值将影响未来的调用（设置了一个新的默认值），直到发生系统重启

- crossbar_audio_input_pin_number

从交错/交叉设备（音视频交错编码）中选择音频输入端口。可以选择交错设备的音频解码输出端。注意改变这个值将影响未来的调用（设置了一个新的默认值），直到发生系统重启

- show_video_device_dialog

如果设为真，在开始采集前会弹出一个面向用户的对话框，以允许他们改变视频滤镜属性和一些手动配置。注意对于交错设备，可能需要同时在PAL (25 fps)和NTSC (29.97)输入帧率、尺寸、隔行等等属性。改变这些值以不同的扫描率/帧率和避免底部绿色、闪烁的扫描行等等。注意这些改变将影响未来的调用（作为新的默认值）直到系统被重启

- show_audio_device_dialog

如果为真，将在开始采集前弹出一个面向用户的对话框，以允许他们改变音频滤镜属性和一些手动配置

- show_video_crossbar_connection_dialog

如果为真，如果打开视频设备将在开始采集前弹出一个面向用户的对话框，以允许手动编辑交错设备路由

- show_audio_crossbar_connection_dialog

如果为真，如果打开音频设备将在开始采集前弹出一个面向用户的对话框，以允许手动编辑交错设备路由

- show_analog_tv_tuner_dialog

如果为真，将在开始采集前弹出一个面向用户的对话框，以允许手动调整电视频道/频率

- show_analog_tv_tuner_audio_dialog

如果为真，将在开始采集前弹出一个面向用户的对话框，以允许手动调整电视音频设置 (例如 mono与stereo, 语言 A,B 或者 C)

- audio_device_load

从文件加载一个音频捕获设备而不是根据名字搜索。它可以同时加载附加参数（如果滤镜支持）。它用于音频源必须指定为一个值，但可以是任何虚拟的

- audio_device_save

存储当前音频采集滤镜设备和他们的参数（如果滤镜支持）到一个文件。如果文件存在则被覆盖（这个文件可以

被 `audio_device_load` 加载)

- `video_device_load`

从文件加载一个视频捕获设备而不是根据名字搜索。它可以同时加载附加参数（如果滤镜支持）。它用于视频源必须指定为一个值，但可以是任何虚拟的

- `video_device_save`

存储当前视频采集滤镜设备和他们的参数（如果滤镜支持）到一个文件。如果文件存在则被覆盖（这个文件可以被 `video_device_load` 加载）

dshow例子

- 输出DirectShow支持的设备列表并退出:

```
$ ffmpeg -list_devices true -f dshow -i dummy
```

- 打开摄像头:

```
$ ffmpeg -f dshow -i video="Camera"
```

- 打开名为 `camera` 的第二个视频设备:

```
$ ffmpeg -f dshow -video_device_number 1 -i video="Camera"
```

- 打开摄像头和话筒:

```
$ ffmpeg -f dshow -i video="Camera":audio="Microphone"
```

- 输出选择设备支持的选项列表并退出:

```
$ ffmpeg -list_options true -f dshow -i video="Camera"
```

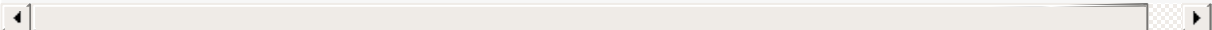
- 通过名字/别名指定pin名来采集，指定别名设备名:

```
$ ffmpeg -f dshow -audiopin_name "Audio Out" -video_pin_name 2 -i video=video="@device_pnp?\pci#ven_1a0a&dev_6200&subsys_62021461&rev_01#4&e2c7dd6&0&00e1#{65e8773d-8f56-11d0-a3b9-00a0c9223196}{ca465100-deb0-4d59-818f-8c477184adf6}":audio="Microphone"
```

- 配置交错设备，指定交错pin，允许在开始时进行视频采集属性调整:

```
$ ffmpeg -f dshow -show_video_device_dialog true -crossbar_video_input_pin_number 0
```

```
-crossbar_audio_input_pin_number 3 -i video="AVerMedia BDA Analog Capture":audio="AVerMedia BDA Analog Capture"
```



dv1394

Linux DV1394输入设备

fbdev

Linux framebuffer（Linux帧缓冲）输入设备

Linux framebuffer是一种独立于硬件的图像抽象层，它用于在计算机屏幕上显示图像,通常是在控制台（环境）。它可以通过一个文件设备节点访问，通常为：`/dev/fb0`

要了解更多信息请阅读Linux源码文件树下文档：`Documentation/fb/framebuffer.txt`

为了从 `/dev/fb0` 读取：

```
ffmpeg -f fbdev -r 10 -i /dev/fb0 out.avi
```

你可以通过下面的命令截屏：

```
ffmpeg -f fbdev -frames:v 1 -r 1 -i /dev/fb0 screenshot.jpeg
```

此外还可以在<http://linux-fbdev.sourceforge.net/>了解fbset(1)。

gdigrab

Win32 GDI 屏幕截取设备

这个设备允许你截取显示在Windows（系统）上的屏幕区域。

它有两个可选的输入文件名（形式）：`desktop` 或者 `title=window_title`

第一个可选名（`desktop`）会截取整个桌面或者桌面的指定区域，第二个可选名（根据窗口标题）会截取单独的窗口，而无论在屏幕上的位置（即即使根据某些操作，该窗口已经移除屏幕可见区域，或者被其他窗口覆盖了也可以截取到）

下面是截取整个桌面的例子：

```
ffmpeg -f gdigrab -framerate 6 -i desktop out.mpg
```

截取桌面上从点（10,20）开始的640x480大小区域

```
ffmpeg -f gdigrab -framerate 6 -offset_x 10 -offset_y 20 -video_size vga -i desktop out.mpg
```

截取名为 "Calculator"的窗口

```
ffmpeg -f gdigrab -framerate 6 -i title=Calculator out.mpg
```

gdigrab选项

- draw_mouse

为1指定是截取鼠标，0表示不截取，默认为1

- framerate

设置帧率，默认为ntsc,相应帧率为30000/1001.

- show_region

在屏幕上显示截取区域。Show grabbed region on screen.

如果指定为1，则指定的截取范围会显示在屏幕上，通过这个选项，可以很容易的知道要截取的范围，这在只截取屏幕的一部分时很有用。

注意 `show_region` 在截取单独窗口时无效（即不可用）

例如:

```
ffmpeg -f gdigrab -show_region 1 -framerate 6 -video_size cif -offset_x 10 -offset_y 20 -i desktop out.mpg
```

- `video_size`

设置视频帧尺寸，默认为屏幕（以 `desktop` 为源）或者窗口（以 `title=window_title` 为源）尺寸

- `offset_x`

当区域截取时,起点的x轴的偏移（左边距屏幕左边距离）

注意坐标系是以可见屏幕左上为原点的。如果你有一个监看对象从左边超出了屏幕可见范围，则 `offset_x` 的值为负数值。

- `offset_y`

当区域截取时,起点的y轴的偏移（上边距屏幕上边距离）

注意坐标系是以可见屏幕左上为原点的。如果你有一个监看对象从上边超出了屏幕可见范围，则 `offset_y` 的值为负数值

ieec61883

使用 `ieec61883` 的FireWire（火线） DV/HDV输入设备。

要允许这个输入设备，需要 `libieec61883`，`libraw1394` 和 `libavc1394` 被安装到系统中。此外还要在编译时配置 `--enable-libieec61883` 以支持。

`ieec61883` 支持通过 IEEE1394 (FireWire)接口连接设备获取视频(使用 `libieec61883` 和新的Linux FireWire stack (火线堆栈 juju))。从Linux Kernel 2.6.37开始它是默认 DV/HDV输入方法了，而老的 `FireWire stack` 已经被移除。

ieec61883的选项

- `dvtype`

覆盖自动检测的DV/HDV类型。它仅用于自动检测类型失败的情况，或需要禁止者需要的格式被禁止的条件。错误的指定将使设备不能正常工作。选项支持 `auto`，`dv` 和 `hdv` 为参数

- `dvbuffer`

对传入的数据设置缓冲（单位帧）。对于DV，它是一个精确的帧数，对于HDV，它不是精确的帧数，因为HDV没有一个固定的帧大小。

- `dvguid`

通过GUID来指定截取设备ID。这样捕获将仅从指定的设备，或者失败（没有指定的设备）。对于系统连接了多个可用设备的情况它非常有用。在系统的 `/sys/bus/firewire/devices` 可以找到连接设备的GUID。

iec61883的例子

- 获取播放FireWire DV/HDV

```
ffplay -f iec61883 -i auto
```

- 获取记录FireWire DV/HDV，包缓冲大小为100000个包

```
ffmpeg -f iec61883 -i auto -hdvbuffer 100000 out.mpg
```

jack

JACK输入设备。

为了使用JACK设备，需要系统上存在 `libjack`

一个JACK输入设备创建1个或者多个JACK可写客户端，每一个对应于一个音频通道，命名（指定）为 `client_name:input_N`，这里 `client_name` 由程序提供，`N` 是通道id号。每个可写客户端作为ffmpeg的输入设备发送数据。

你一次可以创建1个或者多个JACK可读客户端，来连接到1个或者多个JACK可写客户端。

可以使用 `jack_connect` 和 `jack_disconnect` 连接或者断开（不连接）JACK客户端，或者通过图形化接口实现：例如通过 `qjackctl`，

可以通过 `jack_lsp` 来列出JACK客户端和它们的属性列表。

下面的例子展示ffmpeg如何从JACK可读客户端采集数据：

```
# Create a JACK writable client with name "ffmpeg".
$ ffmpeg -f jack -i ffmpeg -y out.wav
```

```
# Start the sample jack_metro readable client.
$ jack_metro -b 120 -d 0.2 -f 4000

# List the current JACK clients.
$ jack_lsp -c
system:capture_1
system:capture_2
system:playback_1
system:playback_2
ffmpeg:input_1
metro:120_bpm

# Connect metro to the ffmpeg writable client.
$ jack_connect metro:120_bpm ffmpeg:input_1
```

更多信息参考<http://jackaudio.org/>

lavfi

`Libavfilter` 输入虚拟设备

这个输入设备可以从 `libavfilter` 滤镜链图的一个开放输出端口读取数据。

对于每个滤镜链图开放输出端口，这个输入设备将创建一个对应的流映射到这个端口进行输出。当前只支持视频数据。滤镜链图是通过选项 `graph` 描述的。

lavfi选项

- graph

描述用作输入的滤镜链图。每个视频开放输出必须由一个形如 `outN` 的独立标签命名，这里 `N` 是从0开始的数字，以指代要映射作为设备的输入流（序号）。第一个没有标签命名的输出自动被作为 `out0`，但所有其他的必须明确指定。

通过附加后缀“+subcc”可以向输出标签创建一个额外的封闭包装字幕（实验性质：现只对EIA-608 / CEA-708）。这个 `subcc` 流在所有其它常规流创建后才附加，并按对应流顺序。例如有 `"out19+subcc"`, `"out7+subcc"` 以及最高普通流`"out42"`，则43号流是 `subcc` 对应于 `out7`，44号流也是 `subcc` 流对应 `out19`

如果没有指定（选项）默认值为输入设备指定的文件名（这里文件名其实是滤镜链图描述）

- graph_file

设置通过文件读取/发送（给其他滤镜）滤镜链图的文件名。在文件中的语法与通过 `graph` 选项描述滤镜链图的语法相同。

lavi例子

- 创建一个颜色流并播放：

```
ffplay -f lavfi -graph "color=c=pink [out0]" dummy
```

- 类似前面的例子，但是有文件名来指定滤镜链图描述，并且省略了`"out0"`标签：

```
ffplay -f lavfi color=c=pink
```

- 创建3个不同的视频测试滤镜源并播放:

```
ffplay -f lavfi -graph "testsrc [out0]; testsrc,hflip [out1]; testsrc,negate [out2]" test3
```

- 从文件中使用 `amovie` 读取音频流来播放:

```
ffplay -f lavfi "amovie=test.wav"
```

- 读取音频和视频流来播放:

```
ffplay -f lavfi "movie=test.avi[out0];amovie=test.wav[out1]"
```

- 复制解码出来的帧和对应字幕到图片（实验）：

```
ffmpeg -f lavfi -i "movie=test.ts[out0+subcc]" -map v frame%08d.png -map s -c copy -f rawvideo subcc.bin
```

libcdio

基于 `libcdio` 的音乐CD输入设备。

需要系统中有 `libcdio` 才能启用，且编译时需要用 `--enable-libcdio` 配置选项允许。

设备允许从音频CD播放和获取

例如利用ffmpeg在 `/dev/sr0` 获取整个音频CD内容:

```
ffmpeg -f libcdio -i /dev/sr0 cd.wav
```

libcdio选项

- speed

设置读取速度，默认为0

这个速度指定了CD-ROM速度，它通过 `libcdio` 的 `cdio_cddap_speed_set` 函数设置。很多CD-ROM驱动器如果设置更大的值将获得更快的速度。

- paranoia_mod

设置纠偏恢复模式的标志，它接受下面的值：

'disable'
'verify'
'overlap'
'neverskip'
'full'

默认值是'disable'

关于可用纠偏模式的更多信息，请咨询纠偏项目文档

libdc1394

IIDC1394输入设备，其基于 `libdc1394` 和 `libraw1394`

编译允许需要配置 `--enable-libdc1394`

openal

这个OpenAL输入设备支持在所有实现了 `OpenAL 1.1` 的系统上进行音频捕获。

要编译使用它需要系统包含 `OpenAL` 头和 `libraries` 库，并且设置编译选项 `--enable-openal`

`OpenAL` 头和 `libraries` 库可以是你的OpenAL实现的部分，或者作为附件下载（SDK）。根据你的安装方式，你可能需要通过 `-extra-cflags` 和 `--extra-ldflags` 为编译指定本地的头文件和库文件

兼容OpenAL的实现有：

- Creative

官方的Windows实现,提供后备支持硬件加速的设备和软件，参考<http://openal.org/>

- OpenAL Soft

便携式,开源(LGPL)软件实现。包括在Windows,Linux、Solaris、BSD操作系统上提供最常见的后端声音api。参考<http://kcat.strangesoft.net/openal.html>

- Apple

OpenAL是核心音频的一部分,官方的Mac OSX音频接口。参考 <http://developer.apple.com/technologies/mac/audio-and-video.html>

这个设备允许通过OpenAL处理来捕获音频输入。

你需要通过提供文件名来指定捕获设备的名称。如果为空字符串（""），则会自动选择默认设备。你可以通过 `list_devices` 获取到支持设备列表。

openal选项

- channels

设置捕获音频的通道。只有1（单声道）和2（立体声）被支持，默认为2

- sample_size

设置音频采样位宽（单位bit——位）。当前只支持8和16，默认16

- sample_rate

设置音频采样频率（单位Hz），默认44.1k.

- list_devices

如果为真(true)，则列出系统上支持的设备并退出，默认为false.

openal例子

- 输出OpenAL支持设备列表并退出

```
$ ffmpeg -list_devices true -f openal -i dummy out.ogg
```

- 从设备 `DR-BT101 via PulseAudio` 捕获音频:

```
$ ffmpeg -f openal -i 'DR-BT101 via PulseAudio' out.ogg
```

- 从默认设备捕获音频 (注意文件名字符串为空):

```
$ ffmpeg -f openal -i " out.ogg
```

- 同时从两个设备捕获，写入不同的文件:

```
$ ffmpeg -f openal -i 'DR-BT101 via PulseAudio' out1.ogg -f openal -i 'ALSA Default' out2.ogg
```

注意不是所有的OpenAL实现设备都支持多路同时捕获。如果上面不工作，则在最新版OpenAL软件上尝试（测试）

pulse

PulseAudio（脉冲音频）输入设备

要使用须编译配置设置 `--enable-libpulse`

需要提供文件名或者"default"来指定输入源设备

通过 `pactl list sources` 可以列出所有PulseAudio设备以及属性。

更多信息参考<http://www.pulseaudio.org>

pulse选项

- server

连接到指定PulseAudio服务器，指定是用IP地址。如果没有设置就用默认服务器

- name

指定用作显示活动客户端的程序名，默认为 `LIBAVFORMAT_IDENT`

- stream_name

指定流名称Specify the stream name PulseAudio will use when showing active streams, 默认为"record"

- sample_rate

Specify the samplerate in Hz, by default 48kHz is used.

- channels

Specify the channels in use, by default 2 (stereo) is set.

- frame_size

Specify the number of bytes per frame, by default it is set to 1024.

- fragment_size

Specify the minimal buffering fragment in PulseAudio, it will affect the audio latency. By default it is unset.

pulse例子

从默认设备捕获来记录：

```
ffmpeg -f pulse -i default /tmp/pulse.wav
```

qtkit

QTKit输入设备

文件名作为设备名或者索引序号参数被传递。设备索引也可以使用 `-video_device_index` 选项来设定。一个获取的设备索引可以覆盖任何获取的设备名。如果所需的设备仅包含数字，则使用 `-video_device_index` 来识别。如果文件名为空字符串或者设备名为"default"都会选择默认设备。有效设备可以由 `-list_devices` 枚举。

```
ffmpeg -f qtkit -i "0" out.mpg
ffmpeg -f qtkit -video_device_index 0 -i "" out.mpg
ffmpeg -f qtkit -i "default" out.mpg
ffmpeg -f qtkit -list_devices true -i ""
```

sndio

sndio输入设备。

要使用它需要系统安装并配置有 `libsndio` 库

文件名作为输入设备节点，通常为 `/dev/audio0`

例如从 `/dev/audio0` 捕获音频：

```
ffmpeg -f sndio -i /dev/audio0 /tmp/oss.wav
```

video4linux2 ,v4l2

Video4Linux2 输入视频设备

"v4l2"是"video4linux2"的别名

编译需要 `v4l-utils` 支持（`--enable-libv4l2` 编译选项被配置），也可用于 `-use_libv4l2` 输入设备选项。

捕获的设备名是一个文件设备节点，通常Linux系统在设备（例如USB摄像头）插入到系统时自动创建这样的节点，会被命名为 `/dev/videoN`，`N` 是设备索引序号

Video4Linux设备通常只支持有限的分辨率（`width x height`）和帧率,通过 `-list_formats all` 选项来获取支持情况。一些设备，例如电视卡可以支持1个或者多个标准，它支持的标准可以通过 `-list_standards all` 来了解。

时间戳时基单位为1microsecond。根据内核版本和配置，时间戳可以基于实时间（real time clock——绝对时间，一种起源于Unix的表示方式）或者单调时钟（monotonic clock——通常源自启动时间，不受NTP或者手动改变）。`-timestamps abs` 或者 `-ts abs` 选择启用实时间。

在ffmpeg或ffplay使用的例子如下：

- 列出支持的设备（video4linux2）

```
ffplay -f video4linux2 -list_formats all /dev/video0
```

- 捕获并显示（对video4linux2设备）

```
ffplay -f video4linux2 -framerate 30 -video_size hd720 /dev/video0
```

- 捕获并记录输入的Grab and record the input of a video4linux2 device, leave the frame rate and size as previously set:

```
ffmpeg -f video4linux2 -input_format mjpeg -i /dev/video0 out.mpeg
```

更多关于video4linux的信息参考<http://linuxtv.org/>

video4linux,vl4的选项

- standard

设置采用的标准，必须是被支持的标准。为了获取当前支持的标准，需要使用 `list_standards` 选项

- channel

设置采用的输入通道索引，默认值为-1，表示采用前面选择的通道。

- video_size

设置视频帧尺寸 `frame size`。参数是格式为 `WIDTHxHEIGHT` 的字符串或者有效的索引

- pixel_format

选择像素格式（仅对raw视频输入有效）

- input_format

设置欲采用的像素格式(仅对raw视频格式)或者编码名。这个选项允许选择一个输入格式（当有多个有效值时）

- framerate

设置首选帧率

- list_formats

列出有效的格式 (支持像素格式、编码和帧尺寸)然后退出

有效值:

‘all’

显示有效可能（压缩和未压缩的）格式

‘raw’

仅显示raw video（非压缩）格式

‘compressed’

仅显示压缩格式

- list_standards

列出支持的标准然后退出

有效值:

‘all’

显示所有支持的标准

- timestamps, ts

设置捕获帧的时间戳标准

有效值:

‘default’

根据核心的默认值

‘abs’

使用绝对时间戳(时间时钟)。

'mono2abs'

强制从单调时间转换为绝对时间戳

默认值是 default

vfwcap

vfw（Video for Windows）捕获输入端

文件名必须是捕获设备索引，范围0-9可以用 `list` 作为文件名，将输出一个设备列表。任何数字外其它文件名被视作设备索引0。

x11grab

X11 视频输入设备

使用需要 `libxcb` 库，它会在编译时自动检测。

另外，配置 `--enable-x11grab` 以对应遗留的Xlib用户。

这个设备允许捕获X11显示区域。

作为输入的文件名语法为：

```
[hostname]:display_number.screen_number[+x_offset,y_offset]
```

`hostname:display_number.screen_number` 指定了要捕获的X11显示屏幕名，`hostname` 可以省略则默认为"localhost"。环境变量 `DISPLAY` 可以指定默认显示名。`x_offset, y_offset` 指定捕获偏移，是对于左上建立的X11屏幕，默认为0。

通过X11文档（`man x`）来了解更详细信息。

使用 `xdpyinfo` 程序来获得关于你X11显示的基本属性信息（配合 `grep "name" 或者 "dimensions"`）

例如使用ffmpeg捕获 `:0.0`：

```
ffmpeg -f x11grab -framerate 25 -video_size cif -i :0.0 out.mpg
```

捕获坐标 10, 20

```
ffmpeg -f x11grab -framerate 25 -video_size cif -i :0.0+10,20 out.mpg
```

X11grab选项

- `draw_mouse`

指定是否捕获鼠标，0表示不，1为默认表示要

- `follow_mouse`

随鼠标定义捕获区域。参数可以是 `centered` 或者像素值 `PIXELS`

当设定为"centered", 捕获区域跟随鼠标指针保持指针所指在区域中, 否则捕获区仅当鼠标指向达到距边缘 `PIXELS` (大于0)像素值离区域内边缘时

例如:

```
ffmpeg -f x11grab -follow_mouse centered -framerate 25 -video_size cif -i :0.0 out.mpg
```

遵循只有当鼠标指针达到100像素内边缘:

```
ffmpeg -f x11grab -follow_mouse 100 -framerate 25 -video_size cif -i :0.0 out.mpg
```

- `framerate`

设置捕获帧率。默认ntsc,为30000/1001.

- `show_region`

显示捕获区域

如果 `show_region` 设置为1, 则区域将显示在屏幕上, 通过这个选项可以很容易的判断哪些内容将被捕获

- `region_border`

设置 `-show_region` 设置为1时使用的区域边框线粗细, 值范围1-128,默认为3 (仅XCB-based x11grab).

例如:

```
ffmpeg -f x11grab -show_region 1 -framerate 25 -video_size cif -i :0.0+10,20 out.mpg
```

设置了 `follow_mouse` :

```
ffmpeg -f x11grab -follow_mouse centered -show_region 1 -framerate 25 -video_size cif -i :0.0 out.mpg
```

- `video_size`

设置帧尺寸, 默认vga.

- `use_shm`

对共享内存使用MIT-SHM扩展, 默认为1, 可能需要禁用远程显示器(仅legacy x11grab).

x11grab 的`grab_x,grab_y` AV选项

语法:

```
-grab_x x_offset -grab_y y_offset
```

设置区域坐标。它们表示抵消X11左上角。默认值为0。

27 输出设备

输出设备是可配置用于ffmpeg写入多媒体数据的元素，其附加到系统的输出设备。

在编译配置ffmpeg时，所有支持的输出设备都被默认允许。你可以使用配置选项 `-list-outdevs` 了解有哪些设备。

你可以通过 `-disable-outdevs` 禁止编译所有输出设备，然后再通过 `-enable-outdev=OUTDEV` 以支持个别的设备，也可以通过默认配置，再添加 `-disable-outdev=OUTDEV` 来禁用个别设备。

在ff*工具集中，`-devices` 可以显示当前允许的输出设备。

当前有效的输出设备介绍见下。

alsa

ALSA(Advanced Linux Sound Architecture) 音频输出设备

alsa例子

- 在默认ALSA设备播放:

```
ffmpeg -i INPUT -f alsa default
```

- 在声卡1的7音频设备播放:

```
ffmpeg -i INPUT -f alsa hw:1,7
```

caca

CACA输出设备

这个输出设备允许在CACA窗口显示视频流。每个程序仅有一个CACA窗口。所以在一个实例中你只能有一个CACA输出。

要允许这个输出设备，需要编译时配置 `--enable-libcaca`，`libcaca` 是一个输出文本而不是像素的图形库。

关于libcaca的更多信息参考<http://caca.zoy.org/wiki/libcaca>

caca选项

- `window_title`

设置CACA窗口标题,如果没有设置，文件名作为默认值指定到输出设备

- `window_size`

设置CACA窗口尺寸, 参数可以是 `widthxheight` 或者一个视频大小缩写。如果没有指定，则以输入视频为默认

- `driver`

设置显示驱动

- `algorithm`

设置抖动算法。有时抖动是必要的，其可以让感觉到的颜色大于调色板。其接受 `-list_dither` 列出的算法

- antialias

设置抗锯齿算法。抗锯齿可柔滑渲染后的图像,并避免常见的楼梯效果。其接受 `-list_dither` 列出的抗锯齿算法

- charset

设置哪些字符将显示文本时使用。它接受 `-list_dither` 列出的字符集

- color

设置将用于渲染的颜色, 接受 `-list_dither` 列出的颜色

- list_drivers

如果设置为真, 输出可用的设备并退出

- list_dither

抖动可用选项列表相关的参数。参数为 `algorithms`, `antialiases`, `charsets`, `colors`

caca例子

- 下面的命令将强制一个80x25的CACA窗口输出:

```
ffmpeg -i INPUT -vcodec rawvideo -pix_fmt rgb24 -window_size 80x25 -f caca -
```

- 列出有效设备并退出:

```
ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_drivers true -
```

- 列出有效抖动颜色并退出:

```
ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_dither colors -
```

decklink

在Blackmagic DeckLink设备输出回放。

编译需要Blackmagic DeckLink SDK, 以及配置 `--extra-cflags` 和 `--extra-ldflags` 以允许。在Windows下, 你需要通过 `widl` 运行 `IDL` 文件。

DEckLink非常挑剔所支持的格式。像素格式必须是uyvy422, 帧率和视频分辨率必须是设备 `-list_formats 1` 列出的值, 音频采样频率必须是48kHz。

decklink选项

- list_devices

如果设置为true, 将列出设备并退出。默认为false.

- list_formats

如果设置为true,将列出支持的格式并退出, 默认为false.

- preroll

以秒为单位设置预卷视频的时间，默认为0.5

decklink例子

- 列出输出设备:

```
ffmpeg -i test.avi -f decklink -list_devices 1 dummy
```

- 列出支持的格式:

```
ffmpeg -i test.avi -f decklink -list_formats 1 'DeckLink Mini Monitor'
```

- 播放视频片段:

```
ffmpeg -i test.avi -f decklink -pix_fmt uyvy422 'DeckLink Mini Monitor'
```

- 按非标准的帧率和视频大小播放视频:

```
ffmpeg -i test.avi -f decklink -pix_fmt uyvy422 -s 720x486 -r 24000/1001 'DeckLink Mini Monitor'
```

fbdev

Linux framebuffer（linux帧缓冲）输出设备

Linux framebuffer是独立于硬件的计算机屏幕显示图形的抽象层，通常用于控制台，它通过文件设备点访问，通常为 `/dev/fb0`

更多信息阅读Linux源码树中的 `Documentation/fb/framebuffer.txt`文档。

fbdev选项

- xoffset
- yoffset

设置左上角的x/y坐标，默认为0

fbdev例子

在设备/dev/fb0播放文件，需要的像素格式依赖于当前framebuffer设置

```
ffmpeg -re -i INPUT -vcodec rawvideo -pix_fmt bgra -f fbdev /dev/fb0
```

更多请参考<http://linux-fbdev.sourceforge.net/>的 `fbset(1)`

opengl

OpenGL输出设备

编译允许配置选项 `--enable-opengl`

这个输出设备允许渲染输出OpenGL内容。内容可以是由程序提供或者默认创建的SDL窗口。

当设备呈现到外部环境时，程序必须实现处理如下的消息：

- AV_DEV_TO_APP_CREATE_WINDOW_BUFFER - 在当前线程创建OpenGL 环境
- AV_DEV_TO_APP_PREPARE_WINDOW_BUFFER - OpenGL当前上下文（环境）
- AV_DEV_TO_APP_DISPLAY_WINDOW_BUFFER - 交换缓冲区
- AV_DEV_TO_APP_DESTROY_WINDOW_BUFFER - 分解/摧毁OpenGL环境
- AV_APP_TO_DEV_WINDOW_SIZE - 告知相关设备（更新信息，程序向设备的）

opengl选项

- background

设置背景颜色，默认为 `Black`（黑色）

- no_window

非0表示禁止默认的SDL窗口。程序必须提供OpenGL环境（上下文）同时设置

`window_size_cb` 与 `window_swap_buffers_cb` 两个回调

- window_title

设置SDL窗口标题,如果没有指定将以指代输出设备的文件名作为默认。当 `no_window` 设置时会被忽略。

- window_size

设置首选窗口尺寸，可以是形如 `widthxheight` 的字符串参数或者视频尺寸短语。如果不指定则默认以输入视频尺寸进行等比例缩放（让高或者宽恰好等于窗口最大可能且完全展示的尺寸）。如果 `no_window` 没有设置可用

opengl例子

使用OpenGL渲染播放到SDL窗口

```
ffmpeg -i INPUT -f opengl "window title"
```

OSS

OSS（open Sound System）输出设备

pulse

PulseAudio输出设备

编译选项开关 `--enable-libpulse`

更多关于PulseAudio信息参考<http://www.pulseaudio.org>

pulse选项

- server

以IP地址描述的用于连接到的PulseAudio服务器，如果不提供则为默认服务器（其他地方进行配置）

- name

将显示在活动客户端的应用名，默认为 `LIBAVFORMAT_IDENT`

- `stream_name`

显示为活动流的流名称，默认为指定的输出名

- `device`

指定设备。如果不指定则采用默认设备。可以用命令 `pactl` 列出输出设备

- `buffer_size`

- `buffer_duration`

控制缓冲尺寸和持续时间。一个小的缓冲区提供了更多的控制，当需要更频繁的更新。

`buffer_size`是按字节指定

`buffer_duration`以milliseconds为单位指定

当两个都被指定时，使用更大的值(持续时间将按使用流的参数重新计算——即按流参数可以把 `buffer_size` 转换成 `buffer_duration` 来比较). 如果设置为0 (默认值), 将采用默认的PulseAudio持续时间，即2秒。

- `prebuf`

指定pre-buffering尺寸，单位字节。它指定服务器开始播放前至少缓冲的量。默认会同于 `buffer_size` 或 `buffer_duration` (中大的一个)

- `minreq`

指定最小请求尺寸，单位字节。即如果数据量达到这里指定的值，就可向服务器发送请求，而不是达到缓冲区填满。它不被建议设置，由服务器来初始化这个值更明智

pulse例子

在默认服务器的默认设备上播放文件

```
ffmpeg -i INPUT -f pulse "stream name"
```

sdl

SDL (Simple DirectMedia Layer) 输出设备

其可以允许在SDL窗口上显示视频流。每个进程仅能创建一个SDL窗口所以你的程序实例只有一个SDL设备输出。

编译需要 `libsdl` 库。

关于SDL的更多信息参考 <http://www.libsdl.org/>

sdl选项

- `window_title`

设置SDL窗口标题，如果没有指定，则用输出文件名

- `icon_title`

置图标化SDL窗口的名称，如果没有指定则采用和 `window_title`

- `window_size`

设置SDL窗口尺寸，可以是 `widthxheight` 格式，也可以是视频尺寸短语。如果没有指定则以输入文件的等比例填充放大最大可能值（某边和屏幕窗口边重合）

- `window_fullscreen`

非0则设置全屏模式，默认为0

sdl交换命令

- 窗口创建的设备可以通过下面的交互式控制命令；

Quit the device immediately.

sdl例子

下面强制以 `qcif` 尺寸标准中SDL窗口上显示图像

andio

sndio 音频输出设备

xv

XV (XVideo) 输出设备

这个X环境设备允许在Xwindow系统的一个窗口上显示视频流

xv选项

- `display_name`

指定用在显示的硬件名，它决定了显示和通信

显示名或者 `DISPLAY` 环境变量值是一个格式字符串 `hostname[:number[.screen_number]]`

`hostname` 是指定了主机的物理连接，`number` 指明了在主机上显示服务索引号，`screen_number` 指定了服务上的那个屏幕

如果不指定，则采用 `DISPLAY` 环境变量值

例如：`dual-headed:0.1` 指定了是 `dual-headed` 主机上的0号显示服务的1号屏幕

通过X11介绍了解更多关于显示名的格式信息

- `window_id`

为非0值表示不创建新窗口而是使用已有的 `window_id` 窗口（如果该 `window_id` 窗口已经存在）。默认为0表示创建自己的窗口。

- `window_size`

设置窗口尺寸，参数可以是 `widthxheight` 或者视频尺寸短语。如果不指定，则默认以输入视频尺寸，当 `window_id` 被设置时忽略

- `window_x`
- `window_y`

设置创建窗口的坐标偏移。默认都为0.它可能被窗口管理器忽略。当 `window_id` 被设置后被忽略。

- `window_title`

设置窗口标题，如果不设置默认以输出文件名作为值，当 `window_id` 被设置后被忽略

xv例子

- 同时解码、显示和编码输入

```
ffmpeg -i INPUT OUTPUT -f xv display
```

- 解码显示输入视频到多个X11窗口:

```
ffmpeg -i INPUT -f xv normal -vf negate -f xv negated
```

28 重采样选项

音频重采样支持下面一些选项。

选项可以在ffmpeg工具集中采用 `-option value` 的形式进行设置，或者在 `aresample` 滤镜中以 `option=value` 形式设置，也可以通过 `libavutil/opt.h` 的API或明确设置在 `SwrContext` 选项中。

- `ich, in_channel_count`

设置输入通道序数。默认为0。如果 `in_channel_layout` 被设置，则并不强制要求设置这个值。

- `och, out_channel_count`

设置输出通道序数，默认为0。如果 `out_channel_layout` 被设置，则并不强制要求设置这个值。

- `uch, used_channel_count`

设置使用的输入通道序数，默认为0。这个选项仅用于指定重新映射

- `isr, in_sample_rate`

设置输入采样率，默认为0

- `osr, out_sample_rate`

设置输出采样率，默认为0

- `isf, in_sample_fmt`

设置输入采样格式，默认为none

- `osf, out_sample_fmt`

设置输出采样格式，默认为none

- `tsf, internal_sample_fmt`

设置内部采样格式。默认值为none，当不显式设置时它会自动被选中

- `icl, in_channel_layout`

- `ocl, out_channel_layout`

设置输入/输出通道布局

参考FFmpeg工具集(ffmpeg-utils)通道布局章节手册（ffmpeg-utils(1)）以了解要求语法。

- `clev, center_mix_level`

设置中心混合水平。这是个用分贝（decibel）表示的值，范围在 [-32,32].

- `slev, surround_mix_level`

设置环绕混合水平。这是个用分贝（decibel）表示的值，范围在 [-32,32].

- `lfe_mix_level`

设置LFE混合成非LFE水平。它表示输入为LFE，但输出没有LFE的处理。这是个用分贝（decibel）表示的值，范围在 [-32,32]

- `rmvol, rematrix_volume`

设置 `rematrix` 值（声音的放缩处理），默认1.0.

- `rematrix_maxval`

设置 `rematrix` 处理的最大值。它用来防止声音放缩为1.0时被裁剪

- `flags, swr_flags`

设置为采用转换的标志，默认为0.

它支持下面的标志：

- `res`

强制重采样。这个标志将强制重采样，即使输入和输出的采样频率一样。

- `dither_scale`

设置抖动率。默认1. 注关于 `dither` : Dither是数字音乐处理上非常神奇的技巧，目的是通过用少数的Bit达到与较多Bit同样的听觉效果，方法是在最后一个Bit(LSB)上动“手脚”。例如用16Bit记录听起来好似20Bit的信息，听到原先16Bit无法记录的微小信息。举例来说，现在我有20Bit的采样信息，现在想将其存为16Bit的信息格式，最简单的转换方式就是直接把后面4个Bit去掉，但是这样就失去用20Bit录音/混音的意义。比较技巧性的方法是在第17~20Bit中加入一些噪音，这段噪音就叫做Dither。这些噪音加入后，可能会进位而改变第16个Bit的信息，然后我们再把最后4个Bit删掉，这个过程我们称为redithering，用意是让后面4个Bit的数据线性地反映在第16个Bit上。由于人耳具有轻易将噪音与乐音分离的能力，所以虽然我们加入了噪音，实际上我们却听到了更多音乐的细节。

- `dither_method`

设置抖动方法，默认 0.

支持的值：

- `'rectangular'`

选择rectangular抖动方法

- `'triangular'`

选择triangular抖动方法

- `'triangular_hp'`

对高频采用triangular抖动方法

- `'lipshitz'`

选择lipshitz噪音塑造抖动方法

- `'shibata'`

选择shibata噪音塑造抖动方法

- 'low_shibata'

选择低shibata噪音塑造抖动方法

- 'high_shibata'

选择高shibata噪音塑造抖动方法

- 'f_weighted'

选择f-weighted噪音塑造抖动方法

- 'modified_e_weighted'

选择modified-e-weighted噪音塑造抖动方法

- 'improved_e_weighted'

选择improved-e-weighted噪音塑造抖动方法

- resampler

设置重采样技术。默认为 `swr`。

支持的值:

- 'swr'

选择原生SW重采样; 滤镜选项 `precision` 和 `cheby` 在这种情况下不可用

- 'soxr'

选择SoX重采样 (如果可用)。 `compensation`, 和滤镜选项 `filter_size`, `phase_shift`, `filter_type` 和 `kaiser_beta` 在这种情况下不可用

- filter_size

仅对 `swr` 有效, 设置重采样滤镜尺寸, 默认为32

- phase_shift

仅对 `swr` 有效, 设置重采样相移, 默认为10, 范围 [0,30].

- linear_interp

如果为1则采用线性插值。默认为0

- cutoff

设置截止频率 (swr: 6dB ; soxr: 0dB) 比例;必须是一个0到1的浮点数。默认为0.97 (对swr模式) 和0.91 (对soxr) (这意味着采样率为44100, 就可以保存整个20KHz的频带)

- precision

仅对soxr有效,重采样信号精度位计算。默认值为20 (配合合适的抖动, 适合一个16bit位深的目标), 适合于高品质的 `sox`, 可以设置为28以获得非常高品质的 `sox`

- cheby

仅对soxr有效, 选择通频带滚边切除(Chebyshev)和高精度近视'无理数'比率。默认为0

- async

仅对 swr 有效, 为1则可以采用伸展、挤压、填充和修剪等方法实现同步, 默认为0, 表示没有任何补偿用于同步音频时间戳

- first_pts

仅对 swr 有效,假定第一个 pts (这里表示数据包时间戳)是这个值。时间单位是1/采样率。这允许对流填充/切边。默认, 没有假设第一帧预期时间, 所以没有填充或者切边。例如这个值设置为0, 表示如果有编码器延迟, 音频流与视频流本身是同步的, 则先静音直到二者同步开始

- min_comp

仅对 swr 有效, 设置时间戳与音频数据最小差值, 单位秒, 以此来触发拉伸/压缩/填充或调整的数据匹配的时间戳。默认值为(min_comp = FLT_MAX), 表示禁用拉伸/压缩/填充或调整的数据匹配到时间戳

- min_hard_comp

仅对 swr 有效, 设置时间戳与音频数据最小差值, 单位秒, 以此来触增加/抛弃以匹配时间戳。它设置了一个阈值来选择有效的硬(裁剪/填充)补偿和软(压缩/拉伸)补偿.注意补偿默认是禁止的。而这个选项默认值是0.1.

- comp_duration

仅对 swr 有效, 设置拉伸/压缩数据匹配的时间戳的持续时间。必须是非负双精度浮点数, 默认为1.0.

- max_soft_comp

仅对 swr 有效, 设置在拉伸/压缩以匹配时间戳的最大系数。必须是非负双精度浮点数, 默认为0.

- matrix_encoding

选择立体声编码矩阵

接收如下值:

- 'none'

选择none

- 'dolby'

选择Dolby

- 'dplii'

选择Dolby Pro Logic II

默认为none.

- filter_type

仅对 swr 有效,选择重采样滤镜类型, 仅用于重采样操作。

接收如下值:

- 'cubic'

选择cubic

- 'blackman_nuttall'

选择Blackman Nuttall Windowed Sinc

- 'kaiser'

选择Kaiser Windowed Sinc

- kaiser_beta

仅对 `swr` 有效, 设置Kaiser Window Beta值, 必须是整数, 范围[2,16], 默认为9.

- output_sample_bits

仅对 `swr` 有效, 采用输出采样抖动。必须为整数, 范围 [0,64], 默认为0, 表示不采用

29 放缩选项

视频支持下面的一些选项。

选项可以在ffmpeg工具集中采用 `-option value` 的形式进行设置，或者在 `aresample` 滤镜中以 `option=value` 形式设置，也可以通过 `libavutil/opt.h` 的API或明确设置在 `SwrContext` 选项中。

- `sws_flags`

设置放缩标志。也用于设置放缩算法，仅有一个算法能被选中。

接受如下值：

- `'fast_bilinear'`

快速双线性缩放算法

- `'bilinear'`

双线性缩放算法

- `'bicubic'`

双三次的缩放算法.

- `'experimental'`

实验缩放算法.

- `'neighbor'`

近邻取样缩放算法

- `'area'`

平均区域尺度缩放算法.

- `'bicublin'`

对亮度采用双三次的缩放算法，对色度采用双线性缩放算法

- `'gauss'`

高斯缩放算法

- `'sinc'`

辛格缩放算法

- `'lanczos'`

兰索斯分块缩放算法

- `'spline'`

自然双三次的样条插值缩放算法

- 'print_info'

允许输出/调试日志

- 'accurate_rnd'

允许精度舍入

- 'full_chroma_int'

允许完整的色度插值

- 'full_chroma_inp'

选择完整的浓度输入

- 'bitexact'

允许bitexact（位精确算法）输出

- srcw

设置源宽度

- srch

设置源高度

- dstw

设置目标宽度

- dsth

设置目标高度

- src_format

设置源像素格式 (必须表示为整数).

- dst_format

设置目标像素格式 (必须表示为整数).

- src_range

选择源区域范围

- dst_range

选择目标区域范围

- param0, param1

设置缩放算法参数。指定的值是特定缩放算法适用的可能被别的算法忽略。值为浮点数

- `sws_dither`

设置抖动算法。接收如下值，默认为 'auto'.

- 'auto'

自动选择

- 'none'

没有抖动

- 'bayer'

bayer抖动

- 'ed'

error diffusion (误差扩散) 抖动

- 'a_dither'

arithmetic (算术) 抖动,基于加法

- 'x_dither'

arithmetic (算术) 抖动, 基于xor (异或) (比 `a_dither` 有更多的随机性/更少的模式化).

30 滤镜入门

FFmpeg通过 libavfilter 库实现滤镜功能。

在 libavfilter 中，一个滤镜可以有多个输入和多个输出。为了尽可能介绍清楚，我们假定有下面的滤镜链图。

```

      [main]
input --> split -----> overlay --> output
      |                   ^
      |[tmp]              [flip]|
      +-----> crop --> vflip -----+

```

在这个滤镜链图中，利用 split 滤镜把输入流分离成了两路流，其中一路通过 crop 滤镜和 vflip 滤镜的同一路级联应用，再同另外一路一起通过 overlay 滤镜处理的流合成进行输出。则可以采用如下的命令行实现：

```
ffmpeg -i INPUT -vf "split [main][tmp]; [tmp] crop=iw:ih/2:0:0, vflip [flip]; [main][flip] overlay=0:H/2" OUTPUT
```

这样最终输出将是视频上部是原始，下部是上部的镜像。（倒影效果）

同一路的滤镜间用逗号(',')进行分割，不同路的滤镜间用分号(';')进行分割。在这个例子里面 crop 和 vflip 是在同一路中的滤镜，split 和 overlay 则不是同一路的（同一路的级联是对连续的视频进行，如果涉及到一输多、多输一或者多输多则都不是在同一路的，即不是同一路级联）。可以通过在方括号 '[' 中的标签名来命名处理的链路。这个例子里，split 滤镜生成了两路就通过 [main] 和 [tmp] 进行了标签命名以方便后续处理。

其中被 split 处理输出的第二路流被命名为 [tmp]，它又被 crop 滤镜处理裁去下半部视频，然后通过 vflip 进行了水平镜像（垂直翻转，即把视频镜像到下半部了）。这是整个输出被命名为 flip。再把 [main] 与 flip 通过 overlay 进行覆盖合成，即把源输入通过 split 滤镜获得的 [main] 的上半部分覆盖到由 crop 和 vflip 滤镜级联处理的输出（这里的 [flip]）上最终得到了镜像结果。

一些滤镜支持参数列表：滤镜名=由冒号(':')隔开的多个参数

还存在所谓的源过滤器（即没有输入音频/视频的过滤器），以及槽过滤器（即没有任何音频/视频输出的过滤器）

31 graph2dot

FFmpeg工具目录下包含一个 `graph2dot` 程序可以用来分析滤镜链图描述并产生用 `dot` 语言描述的对文本表示。

调用命令：

```
graph2dot -h
```

可以了解如何使用 `graph2dot`

你可以把 `dot` 语言描述用于 `dot` 程序（`graphviz` 程序套件中），并获取到滤镜链图的图形表示。

例如命令序列：

```
echo GRAPH_DESCRIPTION | \
tools/graph2dot -o graph.tmp && \
dot -Tpng graph.tmp -o graph.png && \
display graph.png
```

就用来创建和显示一个由 `GRAPH_DESCRIPTION` 字符串定义的滤镜链图图示。注意这里表示滤镜链图的字符串必须是能完整独立的表达的图，其显示定义了输入和输出。例如对于下面的命令形式：

```
ffmpeg -i infile -vf scale=640:360 outfile
```

你的 `GRAPH_DESCRIPTION` 字符串需要为：

```
nullsrc,scale=640:360,nullsink
```

你可能需要用 `nullsrc` 参数以及添加 `format filter` 来模拟指定一个输入文件。

滤镜链图介绍

一个滤镜链图（filtergraph）是连接滤镜的有向图。它可以包含循环动作，也可以在多个滤镜间形成链路，每个链接都有一个连接到滤镜的输入和一个连接到滤镜的输出。

滤镜链图中的每个滤镜都是一个滤镜注册类应用程序的实例，它定义了滤镜的功能、输入接口和输出接口。

如果滤镜没有输入端（接口），则被称作“源”，如果滤镜没有输出端则被称作“槽”（这样的滤镜用于描述/测试等场景，而不用于实际处理）

滤镜链图语法

滤镜链图采用文本表示，其有由一些ffmpeg和ffplay通用的选项 `-filter/-vf/-af` 和 `-filter_complex`（ffmpeg）以及 `-vf/-af`（ffplay）外加定义与 `libavfilter/avfilter.h` 的 `avfilter_graph_parse_ptr()` 等来描述。

一个滤镜链包含序列链接起来的滤镜，这个序列由“，”分隔各个滤镜描述

一个滤镜链图包含序列滤镜链，这个序列有“；”分隔各个滤镜链描述

一个滤镜由一个字符串表单表示：`[in_link_1]...[in_link_N]filter_name=arguments[out_link_1]...[out_link_M]`

这里 `filter_name` 是滤镜类名字，用于指定滤镜实例（它注册于程序中）。其后的 `=arguments` 用于指定滤镜选项。`arguments` 是用于初始化滤镜的参数，它可能有下面两类表单中的一个：

- 一个“：”分隔的 `key=value` 列表
- 一个“：”分隔的列表 `value` 值，在这种情况下，键（key）假定为选项名声明顺序，如 `fade` 滤镜按顺序声明了3个选项 `type`、`start_frame` 和 `nb_frames`，则参数 `in:0:30` 意味着 `type` 为 `in`，`start_frame` 为 `0`，`nb_frames` 为 `30`。
- 前面二者的混合。这种情况下，键值对必须在前，然后接遵循相同约束的若干值。在键值对中可以按任意顺序设置优先顺序。（后接值按最后一个键值对顺序延续设置）。

如果选项的值本身就是一个列表（例如 `format` 滤镜有一个像素格式列表选项），则这种列表通常用“|”分隔。

列表参数可以被‘（单引号）包含起来。字符 \ 作为转义字符用于引号包含的文本。否则参数字符串在遇到特殊字符（例如 `[]=,:;'`）处被看作终止。

在滤镜名和参数前 和 后 有一个连接标签列表。一个连接标签允许命名1个名字的连接，其作为滤镜的输入或者输出端口。以下预订标签 `in_link_1 ... in_link_N` 作为滤镜的输入端，`out_link_1 ... out_link_M` 作为滤镜的输出端。

当中一个滤镜链图中找到相同名字的连接标签时，一个在相应输入端和输出端之间的连接被创建（即认为它们是连接在一起的，如果用做一个滤镜的输出，又用着一个滤镜的输入，则表示从前一个滤镜输出到后一个滤镜）

如果一个输出端没有命名标签，它默认连接到滤镜链上后面滤镜中第一个没有命名标签的输入端。例如：

```
nullsrc, split[L1], [L2]overlay, nullsink
```

这里 `split` 有两路输出，`overlay` 有两路输入，`split` 的第一路输出被命名为标签“L1”，`overlay` 的第一路输入被命名为标签“L2”。则 `split` 的第二路输出链接到 `overlay` 的第二路输入（它们都没有用标签命名）。

在一个滤镜描述中，如果第一个滤镜的输入没有指定，则假定为“in”，如果最后一个滤镜输出没有指定，则假定为“out”

在一个完整的滤镜链上，所有未标签命名的输入和输出必须被连接（匹配）。滤镜链图中如果所有滤镜的输入和输出都被连接则被认为是有效的。

如果格式要求，则 `libavfilter` 会自动插入 `scale` 滤镜。对于滤镜链图描述，可以通过 `sws_flags=flags` 来指定 `swscale` 标志实现自动插入放缩。

这里有一个 BNF 描述的滤镜链图语法：

```
NAME ::= sequence of alphanumeric characters and '_'
LINKLABEL ::= "[" NAME "]"
LINKLABELS ::= LINKLABEL [LINKLABELS]
FILTER_ARGUMENTS ::= sequence of chars (possibly quoted)
FILTER ::= [LINKLABELS] NAME ["=" FILTER_ARGUMENTS] [LINKLABELS]
FILTERCHAIN ::= FILTER [,FILTERCHAIN]
FILTERGRAPH ::= [sws_flags=flags;] FILTERCHAIN [;FILTERGRAPH]
```

注意滤镜链图中的转义

滤镜链图成分需要包含多层的转义。参考ffmpeg-utils(1)手册中的“引用与转义”章节（"Quoting and escaping"）了解更多关于转义过程的信息。

第一层的转义效果在每个滤镜选项值中，其可能包含特殊字符":"来分隔值，或者一个转义符"

第二层的转义在整个滤镜描述，其可能包含转义符\"或者特殊字符[,]，它们被用于滤镜链图的描述

实际上，当你在shell命令行上描述滤镜链图时还可能遇见第三层的转义（处理shell中需要转义的字符）

例如下面的字符需要嵌入 `drawtext` 滤镜描述的 `text` 值

```
this is a 'string': may contain one, or more, special characters
```

这个字符串包含 ' 字符需要被转义，还有 : 这里也需要被转义，方法是：

```
text=this is a \'string\': may contain one, or more, special characters
```

当嵌入滤镜描述时发生第二层的转义，为了转义所有的滤镜链图特殊字符，需要按下例处理：

```
drawtext=text=this is a \\\'string\\\'\: may contain one\, or more\, special characters
```

（注意对于\转义中的每个特殊字符都需要再次转义，就成了\\）

最终当在shell命令行中写这个滤镜链图时再次转义。例如需要对每个特殊字符和转义处理的 \ 等进行转义，最终为：

```
-vf "drawtext=text=this is a \\\\'string\\\\\'\\\: may contain one\\, or more\\, special characters"
```


33 时间线编辑

一些滤镜支持常规的 `enable` 选项。对于支持时间线编辑的滤镜，这个选项可以被设置为一个表达式，其通过评估之前的情况来决定是否把帧画面发送给滤镜。如果表达式计算结果为非0值，则表明滤镜被使用，否则滤镜将被跳过（即把帧画面直接送到滤镜链图的下一个滤镜中）

表达式中可以出现下面的值：

- 't'

时间戳，单位秒。如果输入时间戳未知则为 `NAN`

- 'n'

输入帧的序数，从0开始计数

- 'pos'

输入帧在文件中的偏移位置，如果未知则为 `NAN`

- 'w'

- 'h'

视频输入帧的宽和高

此外，这些滤镜 `enable` 选项状态也可以用于表达式。

类似其他选项，这个 `enable` 选项有相同的规则。

例如，要在10秒到3分钟允许一个blur（模糊）滤镜（`smartblur`），然后 `curves` 滤镜在3秒之后：

```
smartblur = enable='between(t,10,3*60)',
curves    = enable='gte(t,3)' : preset=cross_process
```

34 音频滤镜

当你配置编译FFmpeg时，先采用 `--disable-filters` 可以禁止所有的滤镜，然后显式配置想要支持的滤镜。

下面是当前可用的音频滤镜

adelay

延迟一个或者多个音频通道

它接受如下选项：

- delays

参数是以 `|` 分隔的列表字符串，分别用于指明对应各个通道延迟的微秒（milliseconds）数。应提供至少一个大于0的延迟。未使用的延迟将被静默忽略。如果延迟值数量小于通道数量，则剩余通道不会被延迟。

adelay例子

- 第一通道延迟1.5秒，第三通道0.5秒（其它通道均不延迟变化）

```
adelay=1500|0|500
```

aecho

重复应用于音频输入（回声效果）滤镜

回声反射声音,可以自然发生在山区大型建筑(有时)谈话时,或者大叫时，数字回声效果模拟这种行为，通常用来帮助填补一个乐器或声音的（回声）。原始信号和发射信号的时差就是 `delay`（延迟），而反射信号的响度是 `decay`（衰减）。多个回声可以有不同的延迟和衰减。

要描述一个回声效果需要如下的参数值（注意下面的参数之间用 `:` 分隔）：

- in_gain

设置输入获得的反射信号强度，默认0.6.

- out_gain

设置输出增加反射信号强度，默认0.3.

- delays

一个由 `|` 分隔原始信号和反射作用的指定延迟时间的字符串列表，单位是微秒（milliseconds）。每个延迟允许范围（0-90000.0），默认为1000

- decays

设置一个由 `|` 分隔的反映信号响度衰减值的列表，每个衰减值范围是（0-1.0），默认为0.5.

aecho例子

- 让一个声音听起来像两倍

```
aecho=0.8:0.88:60:0.4
```

- 如果延迟十分短，那听起来像一个机器人（金属）音乐

```
aecho=0.8:0.88:6:0.4
```

- 一个十分长延迟（回声）的声音好像在一个空旷山谷里听露天音乐会。

```
aecho=0.8:0.9:1000:0.3
```

- 同上，但不只一座山的效果（多次反射回音）

```
aecho=0.8:0.9:1000|1800:0.3|0.25
```

aeval

根据指定的表达式修改（改变、变化）一个音频信号

这个滤镜接受一个或者多个表达式（对每个通道），这些表达式计算用于相应的音频信号。

它接受下面的参数：

- exprs

设置一个用 `|` 分隔的对应于各个通道的表达式。如果输入通道数量比表达式数量大，则最后指定的表达式用于其余通道

- channel_layout, c

设置输出通道布局。如果不指定，通道布局采用通道布局数值表达式。如果设置为 `same` 则采用输入通道相同的布局（这是默认值）

帧各通道的计算表达式中，下面的项目被允许。：

- ch

当前表达式对应通道索引 expression

- n

评估样本数量，从0开始

- s

采样率

- t

一秒内采样点数量。nb_in_channels nb_out_channels

输入和输出通道索引 val(CH)

the value of input channel with number CH

注意这个滤镜比较慢，要快速处理你可能需要 `dedicated` 滤镜

aeval例子

- 一半音量

```
aeval=val(ch)/2:c=same
```

- 转化相位的第二个通道::

```
aeval=val(0)]-val(1)
```

afade

对输入音频应用淡入淡出效果

下面是跟上来的滤镜参数：

- type, t

指定滤镜效果，可以是 fade-in ,或者 fade-out

- start_sample, ss

指定的数量开始样品开始应用褪色的效果。默认是0S

- nb_samples, ns

指定实现淡入/淡出效果的样品数量，最终淡入效果输出的音频音量同于输入，而淡出将静音。默认音频采样率为44100。

- start_time, st

指定淡入/淡出效果开始的时间，默认为0。这个值必须被以[持续时间](#)语法来描述。它可以用来替代 start_sample 选项。

- duration, d

指定淡化效果持续时间。以[持续时间](#)语法来描述。在效果的最后，淡入使得音量同于输入音频，淡出则静音。默认持续时间由 nb_samples 定义。这里设置了就替代了 nb_samples

- curve

设置曲线过渡衰减，接受下面的值：

tri

```
选择三角形, 线性斜坡(默认)
```

qsin

```
选择正弦波
```

hsin

```
选择正弦波的一半
```

esin

选择指数正弦

log

选择对数

ipar

选择倒抛物线

qua

选择二次

cub

选择立方

squ

选择平方根

cbr

选择立方根

par

选择抛物线

exp

选择指数

iqsin

选择正弦波反季

ihsin

选择倒一半的正弦波

dese

选择双指数插值

desi

选择双指数S弯曲

afade例子

- 15秒的一个音频淡入

```
afade=t=in:ss=0:d=15
```

- 25秒的音频淡出

```
afade=t=out:st=875:d=25
```

aformat

让输入音频约束成为指定格式。该框架会采用最合适（少）的格式转换

它接受下面的参数：

- sample_fmts

一个用 `|` 分隔的列表，列出了采样格式

- sample_rates

一个用 `|` 分隔的列表，列出了采样率

- channel_layouts

一个用 `|` 分隔的列表指定通道布局。

参考[通道布局](#)了解通道布局相关语法。

如果一个参数被省略，所有的值都是允许的。

强制输出为8位 或者16位 立体声

```
aformat=sample_fmts=u8|s16:channel_layouts=stereo
```

allpass

应用一个两极（two-pole）全通（all-pass）滤波器的中心频率(Hz)的 `frequency` ,和`filter-width`值 `width` 。一个 `allpass` 滤镜可以改变音频的频率相位关系而不改变其频率振幅关系。（可以实现移相）

滤镜接受下面的选项：

- frequency, f

设置频率，单位Hz.

- width_type

设置带宽滤波器的带宽单位，有下面的类型

h

```
Hz
```

q

```
Q-Factor
```

o

```
octave—8度 音阶
```

s

```
slope
```

- width, w

指定一个过滤器的带宽width_type单位

amerge

合并两个或两个以上的音频流到一个多通道流

滤镜接受下面的选项：

- inputs

设置输入数量，默认为2

如果输入的通道布局是不相交的,因此可兼容,输出将设置相应的通道布局和渠道，并在必要时重新排序。如果输入的通道布局是不可分离的，则输出将会是第一个输入的所有通道，然后第二个输入的所有通道，在这种顺序下，输出的通道布局将默认通道数设为总数。

例如：如果第一个输入是 2.1 （ FL+FR+LF ）和第二个输入为 FC+BL+BR，则输出是 5.1 通道布局，并且按： a1, a2, b1, a3, b2, b3 设置输出通道布局（这里a1是第一个输入的第一个通道 FL， b1是第二个输入的第一个通道 FC）

在另外的应用中，如果两个输入都是立体声，则输出会默认为： a1, a2, b1, b2，即输出流显示为一个4通道音频流，这可能是一个非预期的值。

所有的输入必须有相同的采样率和格式。

如果输入没有相同的持续时间，输出将在最短时间停止。

amerge例子

- 合并两个单声道为立体声

```
amovie=left.wav [l] ; amovie=right.mp3 [r] ; [l] [r] amerge
```

- 合并多个到1个视频和6个音频流

```
ffmpeg -i input.mkv -filter_complex "[0:1][0:2][0:3][0:4][0:5][0:6] amerge=inputs=6" -c:a pcm_s16le output.mkv
```

amix

混合多个音频输入到单路音频输出（叠加混合音频，不同于前面的amerge）

注意这个滤镜只支持浮动采样（amerge 和 pan 音频滤镜支持很多格式）。如果 amix 滤镜输入有一个整数采样，则 aresample 滤镜会自动插入转换成浮动采样。

例如：

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex amix=inputs=3:duration=first:dropout_transition=3 OUTPUT
```

会把3个输入音频流混合成一个输出流，持续时间采用第一个输入流的持续时间并且有3秒的结束过渡。

它支持下面的参数：

- inputs

输入数，如果没有指定则默认为2

- duration

确定流结束的方法，有：longest

按最长持续时间输入（默认）

shortest

按最短持续时间输入

first

按第一个输入持续时间

- dropout_trnsition

过渡时间，单位秒，指一个输入流结束时音量从正常到无声渐止效果，默认为2秒

anull

输入音频源不变的到输出

apad

在一个音频流的末尾添加无声。

它可以用来同 `ffmpeg -shortest` 一起把最短的音频流延长到视频相同长度。

选项介绍见下：

- `packet_size`

设置垫的包大小字节，默认4096.

- `pad_len`

设置要添加到最后的采样点数量（实为时间的一种表达，采样率一定则采样点数决定了持续时间，这里只设置了差值）。值达到后终止。它与 `whole_len` 互斥

- `whole_len`

设置最小的音频流总输出样本点数（就是直接设置总持续时间的一种方式），如果这个值大于输入音频数，则垫上差值到最后。它与 `pad_len` 互斥

如果既不设置 `pad_len` 也不设置 `whole_len`，则接在后面的静音将一直持续。

apad例子

- 添加1024个静音样本点到输入末尾

```
apad=pad_len=1024
```

- 让输出至少有10000个样本点，不足就添加静音样本点到末尾

```
apad=whole_len=10000
```

- 利用ffmpeg添加静音样本点，让视频和音频有同样长的持续时间（以视频时间为准的）。

```
ffmpeg -i VIDEO -i AUDIO -filter_complex "[1:0]apad" -shortest OUTPUT
```

aphaser

添加一个移相到输入音频

移相器滤镜创建一系列的波峰和波谷的频谱。波峰和波谷的位置调制,这样他们会随着时间变化,建立一个全面的效果。

可接受参数介绍见下：

- `in_gain`

设置输入增益，默认 0.4.

- `out_gain`

设置输出增益，默认0.74

- delay

设置延迟，单位微秒，默认3.0.

- decay

设置衰减权重，默认0.4.

- speed

设置调制速度，单位Hz，默认 0.5.

- type

设置调制类型，默认 triangular.

它接受下面的值：

'triangular, t' 'sinusoidal, s'

aresample

对输入音频按指定的参数重采样，使用了 `libswresample` 库，如果没有特殊设定，将自动在输入和输出设置间转换。

这个滤镜还可以用于拉伸/压缩音频数据，使其匹配时间戳，或者通过注入静音/剪切来匹配时间戳

滤镜接受的语法：`[sample_rate:]resampler_options`，这里 `sample_rate` 是新采样率，`resampler_options` 是一个由 `:` 分隔的 `key=value` 选项参数值对列表。参考ffmpeg重采样手册完整了解支持的选项。

aresample例子

- 重采样为44100Hz:

`aresample=44100`

- 拉伸/压缩采样来适应时间戳，最大1000个样本点每秒:

`aresample=async=1000`

asetnsamples

设置每个输出音频帧中样本点个数

除了最后一个输出包括有包含不同数量的样本点外，这个滤镜使得持续中的每个数据包中包含一致数量的样本点。

滤镜接受下面的选项：

- nb_out_samples, n

设置每个输出音频帧中样本点个数，它替代作为每个通道样本点个数，默认1024

- pad, p

如果设置为1，则滤镜在最后一个音频帧中补0填充，这样所有帧都有一样的样本点个数。默认为1

例如，设置每帧样本点个数为1234，且禁止最后帧补齐

```
asetnsamples=n=1234:p=0
```

asetrate

重新设置采样率而不改变PCM数据。这将导致速度和音调的变化。

滤镜接受下面的选项：

- `sample_rate, r`

设置输出的采样率，默认为44100Hz

ashowinfo

对每个输入流音频帧显示其所含各种信息。输入音频不被改变

信息由一个序列的键值对构成, 键值对的格式为 `key:value`

下面的信息将被显示（作为键值对的键名）：

- `n`

当前的输入（音频）帧序号，从0开始计数

- `pts`

输入帧的时间戳，按时基计数，时基依赖于滤镜输入，通常为 `1/sample_rate`

- `pts_time`

输入帧时间戳按秒的表示

- `pos`

输入帧中输入流中的偏移（文件读写指针位置），-1表示该信息不可用 和/或者 无意义 (例如合成音频)

- `fmt`

采样格式

- `chlayout`

通道布局

- `rate`

音频帧的采样率

- `nb_samples`

每帧中（每个通道）采样点数

- `checksum`

音频数据Adler-32校验和(以十六进制数据形式输出)，对于连续音频数据，数据被看作是都连接在一起的。

- plane_checksums

一个Adler-32校验和列表，对应于每个数据块

astats

显示音频通道的时域统计信息。统计计算和显示每个音频通道，值适合情况下还提供整体图。

它接受下面的选项：

- length

按秒给出的小窗口长（指统计动态移动窗），用于RMS测量波峰和波谷。默认为0.05（50微秒），允许值范围为：[0.1-10]

每个显示用选项的介绍见下A：

- DC offset

从0振幅位移

- Min level

最小采样点 水平

- Max level

最大采样点 水平

- Peak level dB
- RMS level dB

标准的峰值和有效值测量，单位dBFS

- RMS peak dB
- RMS trough dB

在短窗口中波峰和波谷RMS值水平测量

- Crest factor

标准比率RMS的峰值水平 (注意不是dB值了).

- Flat factor

平整度(即连续样本具有相同值)信号的峰值水平(即最小水平或最大级别)。

- Peak count

计数多次(而不是样品的数量),信号达到最小水平或最大水平

astreamsync

将两个音频流控制的发到缓冲区

滤镜接受下面的选项：

- `expr, e`

设置一个用于判断哪个流被送出的表达式。如果结果为负则第一个流被转发，否则如果为非负则第二个流被转发。表达式中允许下面的变量：

`b1 b2`

分别指代缓冲区中每个输入音频流到目前为止转发数量

`s1 s2`

到目前为止，已经转发的每个流的采样点数量

`t1 t2`

当前时间每个流的时间戳 `stream`

默认表达式为 `t1-t2`，它意味着把时间戳小的流进行转发

astreamsync例子

压力测试 `amerge` 通过随机发送给缓冲区作为有错误的输入，同时避免太多同步失锁：

```
amovie=file.ogg [a] ; amovie=file.mp3 [b] ;
[a] [b] astreamsync=(2*random(1))-1+tanh(5*(t1-t2)) [a2] [b2] ;
[a2] [b2] amerge
```

asyncts

通过需要 压缩/拉伸 和/或 采样点/填补静音 来让音频数据和时间戳同步。

这个滤镜不默认编译，请使用[aresample](#)来压缩/拉伸。

它接受下面的参数：

- `compensate`

若允许则通过拉伸/压缩来让数据匹配时间戳。默认禁止。当禁止时数据对时间戳将以静音补齐

- `min_delta`

触发数据丢弃/补齐的时间戳与音频数据最小差异（按秒为单位）默认为0.1。如果默认值还是不完美同步，可以尝试设置为0

- `max_comp`

当 `compensate=1` 时，每秒最大补齐样本点数，默认为500.

- `first_pts`

这时一个时基单位（其实设置在前面补齐/去除时间），第一个PTS为 `1/sample_rate`，它允许了在开始补齐/去除的时间

量。默认情况下，没有假定一个，所有没有补齐/调制。例如 要让一个音频和另外的视频同步，可能需要在前面加上/或者减去 一些样本点

atempo

调整音频节奏(变奏)

滤镜接受1个参数，表示音频节奏。如果不知道，则默认为1.0，表示不变，参数值范围为 `[0.5, 2.0]`

atempo例子

- 减慢为80%

```
atempo=0.8
```

- 加快为125%

```
atempo=1.25
```

atrim

建设连续输入中的部分作为输出。

接受下面的参数：

- start

以秒为单位的开始时间戳。即所指时间样本点将作为输出的第一个样本点

- end

以秒为单位的结束时间戳。即所指时间前最后一个样本点将作为输出的最后样本点，其所指样本点及其后的均被丢弃。

- start_pts

类似 start，除了它的不是以秒为单位

- end_pts

类似 end，除了它不是一秒为单位

- duration

输出的持续时间

- start_sample

要输出的第一个样本点序号

- end_sample

要丢弃的第一个样本点序号，（其前的最后一个样本点是输出的最后一个样本点）

其中 start，end 和 duration 采样[持续时间](#)格式描述，参考相关章节以了解详情。

注意前面的 start / end 和 duration 是看帧的时间戳，而有 _sample 的选项则只是简单的对传入数据的样本点计数。所有

如 `start / end_pts` 和 `start / end_sample` 会造成不同的结果（当时间戳不准确、或从0开始）。还要注意这个滤镜并不修改时间戳。如果你想让输出时间从0开始，则在其后插入 `atrim` 滤镜

如果同时有多个 `start` 或 `end` 选项被设置，滤镜尝试（贪婪算法）保留尽量多的样本点作为输出（即 `start` 和 `end` 差最大的）。如果想保留多个块，需要连接多个 `atrim` 滤镜来应用（多次输入源，接 `atrim` 后再连接起来）

默认所有输入被保留。所有它可被配置为保留结束前的一切。

例子：

- 丢弃指定时间外的输入：

```
ffmpeg -i INPUT -af atrim=60:120
```

- 仅保留开始的1000帧

```
ffmpeg -i INPUT -af atrim=end_sample=1000
```

bandpass

应用一个通过中心点频率 `frequency` 定义的两极Butterworth（巴特沃斯）带通滤波器，其有3dB的带宽。`csg` 选项指定一个常数作为默认增益（峰值增益 `Q` ,默认为0）。滤镜到期后每个8分音度有6dB的衰减。

滤镜接受下面的选项：

- `frequency, f`

设置滤镜的中心点频率，默认3000.

- `csg`

设置增益常数，若想增益倍值为1（不变化），则值默认为0。

- `width_type`

设置指定带通滤波的类型：

h

```
Hz
```

q

```
Q-Factor
```

o

```
octave-8分音度
```

s

```
slope-斜率
```

- width, w

设置带通滤波带宽（单位为 width_type ）

bandreject

应用一个通过中心点频率 frequency 定义的两极Butterworth（巴特沃斯）带通滤波器，其有单侧3dB的带宽。滤镜到期后每个8分音度有6dB的衰减。

- frequency, f

设置滤镜的中心点频率，默认3000.

- width_type

设置指定带通滤波的类型：

h

Hz

0

octave-8分音度

s

slope-斜率

- width, w

设置带通滤波带宽（单位为 width_type ）

bass

使用双刀搁置滤波器增加或减少低音(低)音频的频率响应，类似于一个标准的高保真的音控。这也被称为搁置平衡(EQ)。

滤镜接受下面选项：

- gain, g

设置在0Hz的增益，其可用的范围约为-20（大振幅）+20。要合适的值以防被削波（振幅过大超出说了样本格式允许值范围就削波）

- frequency, f

设置滤镜的中心点频率，默认100 Hz.

- width_type

设置指定带通滤波的类型：

h

```
Hz
```

o

```
octave-8分音度
```

s

```
slope-斜率
```

- width, w
设置带通滤波带宽（单位为 `width_type`）

biquad

应用一个 `biquad` IIR（无限冲激响应）滤镜，它有b0、b1、b2和a0、a1、a3 分别作为分子和分母。

bs2b

Bauer（鲍尔）立体声双声道的转换,使耳机听立体声音频记录

它接受下面的参数：

- profile
预定义横向进给水平

default

```
默认水平 (fcut=700, feed=50)。
```

cmoy

```
Chu Moy circuit (fcut=700, feed=60)。
```

jmeier

```
Jan Meier circuit (fcut=650, feed=95)。
```

- fcut
截至频率，单位Hz
- feed

进给水平，单位Hz

channelmap

重新映射输入通道

它接受下面的参数：

- channel_layout

输出流通道布局

- map

从输入到输出的通道映射。参数值是一个由 | 分隔的映射关系描述列表。每个为 in_channel-out_channel 或者 in_channel 格式。in_channel 可以用于输入通道名称（例如FL表示左前）或者在输入通道中的索引数。out_channel 可以用输出通道名称或者索引数。如果 out_channel 被省略，则从0开始递增映射每个输出通道

如果没有设置 map，滤镜将隐式映射，保留所指（通道布局中对应通道）

例如，要把一个5.1 声道的MOV文件下变换

```
ffmpeg -i in.mov -filter 'channelmap=map=DL-FL|DR-FR' out.wav
```

则将创建一个输出WAV文件，其只有2个声道（立体声）。

又如要修复一个5.1声道AAC编码中不当的通道顺序

```
ffmpeg -i in.wav -filter 'channelmap=1|2|0|5|3|4:channel_layout=5.1' out.wav
```

channelsplit

把输入音频流的每个通道分开作为多个输出流

它接受下面的参数：

- channel_layout

指定输入通道布局，默认为 "stereo"

例如从输入MP3文件中分离立体声

```
ffmpeg -i in.mp3 -filter_complex channelsplit out.mkv
```

会创建一个包含2个音频流的Matroska文件，其中一个对应于原来的左声道，另外一个对应于右声道。

划分5.1声道的WAV文件：

```
ffmpeg -i in.wav -filter_complex
'channelsplit=channel_layout=5.1[FL][FR][FC][LFE][SL][SR]'
-map '[FL]' front_left.wav -map '[FR]' front_right.wav -map '[FC]'
front_center.wav -map '[LFE]' lfe.wav -map '[SL]' side_left.wav -map '[SR]'
side_right.wav
```

chorus

给声音添加合唱效果

可以让独唱变得像合唱，但也可以用于仪表。

合唱与回声效应都有短延迟,但是回波延迟是常数,合唱则采用不同的正弦或三角调制。调制深度范围定义了调制延迟(播放之前或之后的延迟)。因此延迟的声音听起来较慢或更快,这是原来周围的延迟调整声音,像是有一个与合唱整体略微差异。

它接受下面的参数（每个参数项如果有多个可能值用 | 分隔）：

- in_gain

设置输入增益，默认0.4.

- out_gain

设置输出增益，默认0.4.

- delays

设置延迟，延迟通常在40ms - 60ms

- decays

设置衰减

- speeds

设置速度

- depths

设置深度

chorus例子

- 一个延迟（二人合唱效果）：

chorus=0.7:0.9:55:0.4:0.25:2

- 2个延迟（三人合唱效果）：

chorus=0.6:0.9:50|60:0.4|0.32:0.25|0.4:2|1.3

- 3个延迟（四人及更多合唱效果）：

chorus=0.5:0.9:50|60|40:0.4|0.32|0.3:0.25|0.4|0.3:2|2.3|1.3

compand

音频压缩或扩展的动态范围（动态压缩）

它接受下面的参数（参数值有多个时用 | 分隔，各个参数间用 : 分隔）：

- attacks
- decays

一个以秒计时的单通道输入信号计量瞬时水平平均值的计算窗口宽度列表。attacks 用于指示增加，decays 用于指示衰减。对于大多数情况,增强时间(响应音频声)应该比衰减时间短,因为人类的耳朵感觉中，突然大声的音频录音比突然减弱更敏感。典型的对增强采用0.3秒，对衰减采样0.8秒。

- points

一个要进行传输的点的列表，以dB指定相对于最大可能信号的振幅。每个关键节点由下面的语法描述：

`x0/y0|x1/y1|x2/y2|...` 或者 `x0/y0 x1/y1 x2/y2 ...` 即由 | 或者空格分隔的列表。

所有的输入值（频点）必须按递增排序（传递函数——放大倍数，不需要单调递增），点值 0/0 表示可能覆盖(对应0/输出的按dB值增益)。典型值有 `-70/-70|-60/-20`

- soft-knee

设置对所有关节的曲线半径，默认为 0.01

- gain

以dB为单位设置附加增益，对应于所有设置为需要获得传输的点。这允许调整整体增益，默认为0

- volume

以dB为单位设置初始化音量，其作为开始是每个通道的假设值，即允许用户提供一个名义上的初始值。例如一个非常大的增益并不适用于初始信号在开始运作时有没有压缩。一个典型的表示最初十分安静的值是-90dB，默认为 0.

- delay

以秒为单位设置延迟。立即输入音频分析,但音频延迟之前美联储音量调节器。指定一个延迟约等于增强/衰减时间允许滤镜有效地预测而不是被动的模式运作。它默认为0。

compand例子

- 为音乐适合在嘈杂环境中听时让音乐更安静和响亮:

```
compand=.3|.3|.1|1:-90/-60|-60/-40|-40/-30|-20/-20:6:0:-90:0.2
```

另外的例子是耳语和爆炸部分音频:

```
compand=0|0:1|1:-90/-900|-70/-70|-30/-9|0/-3:6:0:0:0
```

- 一个噪声门，对于噪声对于音频有较低水平信号:

```
compand=.1|.1|.2|.2:-900/-900|-50.1/-900|-50/-50:.01:0:-90:.1
```

- 另外一个噪声门，这次噪声对于音频有更高水平信号。this time for when the noise is at a higher level than the signal (使它在某些方面,类似于压制):

```
compand=.1|.1|.1|.1:-45.1/-45.1|-45/-900|0/-900:.01:45:-90:.1
```

dcshift

将直流转换应用到音频。

这可以有助于消除直流偏置(可能由硬件问题引起的记录链)的音频。直流偏置的影响是减少空间,因此体积。 `astats` 滤镜可以用来确定一个信号直流偏移。

- `shift`

设置直流偏置,允许值为[-1, 1].它叠加到音频上

- `limitergain`

可选, 它应该有一个远低于1的值(例如 0.05 或0.02)用来防止裁剪。

earwax

让声音更容易在耳机听

这个为44.1kHz立体声 (即CD音频格式) 添加 `cues` (线索)。让声音听起来像离开了耳机, 是在扬声器前面 (标准应该播放的环境)。

它从SoX移植来。

equalizer

应用一个两极平衡 (EQ) 峰值滤镜。通过这个滤镜, 信号电平值在选定的频率可以增强或者衰减 (不想 `bandpass` 和 `bandreject` 滤镜), 而其它频率不变。

为了产生复杂的平衡曲线,这个过滤器可以被使用几次,每一个都有不同的中心频率。

滤镜接受如下选项:

- `frequency, f`

设置中心频点, 单位 Hz.

- `width_type`

设置带通滤波宽度定义类型

h

Hz

q

Q-Factor

o

octave

s

slope

- width, w

设置带通滤波宽度，其关联 width_type

- gain, g

设置对应增益，单位dB

equalizer例子

- 1000 Hz 增加为10dB，带宽200HZ

equalizer=f=1000:width_type=h:width=200:g=-10

- 在1000Hz处增加为2dB,带宽Q 1，在100Hz处增加为5dB，带宽Q2:

equalizer=f=1000:width_type=q:width=1:g=2,equalizer=f=100:width_type=q:width=2:g=-5

flanger

为音频增加翻边效果

滤镜接受下面的选项：

- delay

以微秒为单位设置延迟，范围0-30，默认为0

- depth

以微秒为单位设置swep延迟，范围0-10，默认2

- regen

设置再生百分比(延迟信号反馈)，范围-95-95，默认0

- width

设置的延迟信号与原始混合比例。从0到100不等。默认值是71。

- speed

设置每秒扫描（Hz），范围0.1-10，默认0.5

- shape

设置波形，可能值为 triangular 或者 sinusoidal，默认 sinusoidal

- phase

对多个通道设置波形转换百分比，范围0-100，默认25

- interp

设置延迟线内插方法，可能值为 `linear` 或 `quadratic`，默认 `linear`

highpass

指定频率3dB的高通滤波器。这个滤波器可以是单极或者双极（默认），滤波器每极有6dB倍频（每极10倍频是20dB）

滤镜接受下面选项：

- frequency, f

设置频点，默认3000.

- poles, p

设置极数，默认2

- width_type

设置带宽计算模式.

h

Hz

q

Q-Factor

o

octave

s

slope

- width, w

设置带宽，其根据 `width_type` 计数，仅对双极滤镜，有0.707q的.巴特沃斯响应

join

把多个输入流连接成一个多通道流

它接受下面参数：

- inputs

输入流数，默认2

- channel_layout

通道布局，默认 `stereo`

- `map`

从输入流映射通道，参数是'|'分隔的字符串。每个 `input_idx.in_channel-out_channel` 都是映射自输入流。`input_idx` 是0起始的输入了流索引。`stream.in_channel` 设置输出流的标识，（例如：`FL`对应于左前退），其选定需要更多的音视频出入。

这个滤镜可用于未显式映射的一些应用。首先试图找到一个匹配行，如果没有就把第一个未使用的视频的音频信号插入。

- 连接3个输入（正确设置通道布局）

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex join=inputs=3 OUTPUT
```

- 从6路音频合成为5.1输出到

```
ffmpeg -i fl -i fr -i fc -i sl -i sr -i lfe -filter_complex 'join=inputs=6:channel_layout=5.1:map=0.0-FL|1.0-FR|2.0-FC|3.0-SL|4.0-SR|5.0-LFE' out
```

ladspa

加载一个LADSPA(Linux音频开发人员的简和插件API)插件

编译选项：`--enable-ladspa`，接受下面的值。

- `file, f`

描述LADSPA库所在文件，如果 `LADSPA_PATH` 环境变量被定义，`LADSPA` 插件将在被逗号分隔的各个路径中查找（`LADSPA_PATH`中的）。否则采样标准查找顺序查找：`HOME/.ladspa/lib/`，`/usr/local/lib/ladspa/`，`/usr/lib/ladspa/`。

- `plugin, p`

指定库中的插件。有些库只有一个插件，而另外一些可能有多个。如果没有设定，则指定库中所有的插件将被列出

- `controls, c`

有由'|'分隔的浮点数指列表，以确定加载的行为（例如延迟、阈值或者增益）。控制参数需要由语法：`c0=value0|c1=value1|c2=value2|...`，这样来设置第*i*个选项值。如果设置为 `help` 将输出有效的控制器和可用的打印控制参数

- `sample_rate, s`

指定采样率，默认44100，仅用于插件有0号输入。

- `nb_samples, n`

设置每个通道每个输出帧中包含的样本点数。默认1024，仅用于插件有0号输入。

- `duration, d`

设置最小持续时间。参考[持续时间](#)语法来描述.注意实际返回的结果持续时间可能大于指定时间，生成的音频总是少一个完整的帧。如果没有特别指定，或者表示持续时间的值为负数，则表明生成的音频持续不断。仅用于插件有0号输入。

ladspa例子

- 列出amp（LADSPA e例子插件）库中有效插件：


```
ladspa=file=amp
```

- 列出所有有效的控制项和有效值范围（对vcf_notch插件，其在VC库中）：

```
ladspa=f=vcf:p=vcf_notch:c=help
```

- 利用计算机音乐工具包(CMT)插件库模拟低质量的音频设备：

```
ladspa=file=cmt:plugin=lofi:controls=c0=22|c1=12|c2=12
```

- 使用TAP-plugins添加混响的音频(汤姆-Tom-的音频处理插件):

```
ladspa=file=tap_reverb:tap_reverb
```

- 产生白噪声,有.2振幅:

```
ladspa=file=cmt:noise_source_white:c=c0=.2
```

- 利用Metronome from the C Audio Plugin Suite (CAPS)库中的C Click插件，产生20bmp的内容:

```
ladspa=file=caps:Click:c=c1=20'
```

- 应用C* Eq10X2 - Stereo 10段均衡效应:

```
ladspa=caps:Eq10X2:c=c0=-48|c9=-24|c3=12|c4=2
```

Commands

它支持下面的命令：

- cN

编辑 N-th 控制值

如果指定的值无效，会忽略它。

lowpass

应用3dB频点倍带宽的低通滤波器。它可以是单极或者双极的（默认）。滤镜每个8度有6dB的衰减（20dB 则是10倍）

滤镜接受下面的选项：

- frequency, f

设置频点，默认500.

- poles, p

设置极数，默认2

- width_type

设置带宽计算模式.

h

Hz

q

Q-Factor

O

octave

S

slope

- width, w
设置带宽，其根据 width_type 计数，仅对双极滤镜，有0.707q的.巴特沃斯响应

pan

按指定的增益关系混合。滤镜接受通道布局 and 一组通道定义

这个滤镜也可以有效的重新映射通道音频流。

滤镜接受的参数格式为：“|outdef|outdef|...”

- l
输出通道布局或者通道号
- outdef
输出通道指定，格式: "out_name=[gain]in_name[+gain]in_name..."
- out_name
输出通道名，每个通道 (FL, FR, 等) 或者通道索引数 (c0, c1, etc.)
- gain
增益倍数，1表示不变
- in_name
采用的输入通道，参考 out_name 的介绍。它不能是混合了名字和索引号的输入通道（描述）

如果在通道描述中有‘=’而不是‘<’,则表明对指定通道总是按1倍重整（表明不变），从而避免削波噪音

pan的混合例子

例如，如果想下变换立体声为单声道，而且更大的权重是在左声道：

```
pan=1c|c0=0.9*c0+0.1*c1
```

一个定制的下变换工作与 3- 4- 5- 和7- 通道环绕

```
pan=stereo| FL < FL + 0.5*FC + 0.6*BL + 0.6*SL | FR < FR + 0.5*FC + 0.6*BR + 0.6*SR
```

pan的再映射例子

通道再映射仅在下面的情况起效：

- 增益为0或者1
- 每个输入仅有一个通道输出

如果这些条件都满足,滤镜将通知用户("Pure channel mapping detected"-“纯通道映射发现”),并使用一个优化和无损方法重新映射。

例如：如果有一个5.1声道要映射立体声，去除扩展通道：

```
pan="stereo| c0=FL | c1=FR"
```

对于同一个源，你也可以交换左前和右前，保持输入布局：

```
pan="5.1| c0=c1 | c1=c0 | c2=c2 | c3=c3 | c4=c4 | c5=c5"
```

如果是立体声源，要对左前通道静音（但保持立体声通道布局）：

```
pan="stereo|c1=c1"
```

仍然是立体声，把右前用两次：

```
pan="stereo| c0=FR | c1=FR"
```

replaygain

ReplayGain扫描仪滤镜。这个滤镜以一个音频流作为输入和输出也不改变。在过结束后显示 track_gain 和 track_peak。

resample

转换音频采样格式，采样率和通道布局，它一般不直接使用。

silencedetect

检测一个音频流中的静音。

这个滤镜是在检测到输入音频小于或等于一个噪音公差值，且持续时间大于或等于最低噪音持续时间时输出日志消息

输出的时间和持续时间以秒为单位

滤镜接受下面的选项：

- duration, d

设置需通告的静默持续时间(默认为2秒).

- noise, n

设置噪声限，可以采样为dB描述 (指附加值的dB表示) 或者振幅比，默认为-60dB或者0.001 0.001.

silencedetect例子

- 检测5秒静默，-50dB的噪音限

```
silencedetect=n=-50dB:d=5
```

- 噪音限为0.0001 检测静默（静默时长2秒）

```
ffmpeg -i silence.mp3 -af silencedetect=noise=0.0001 -f null -
```

silenceremove

从音频的开始、中间或者结束删除静默

滤镜接受如下选项：

- start_periods

这个值用来指定在开始时应该作为静默削减的音频幅度，0表示不削减，指定一个非0值则表示直到找到一个非静默值时的都被削减掉。通常该值为1，而更高的值甚至可以削减掉所有的音频。默认为0

- start_duration

指定一个非静默持续时间阈值，如果非静默时间超过该阈值则不被削减，否则记为连续静默中的脉冲噪音。通过增加这个值，脉冲噪声可以视为静默被修剪掉，缺省为0

- start_threshold

这个用于指出该作为静默的样本值。对于数字音频，值为0肯定很合适作为表明是静默的，但对于模拟音频（ADC获取的），你可能希望增加这个值（作为背景噪音），可以指定dB值或者振幅比，默认为0

- stop_periods

指定为从音频削减沉默数。为了从中间削减沉默，需要为 `stop_periods` 指定一个负数值。这个值被视为有积极的价值,用于显示效果应该重启`start_periods`规定的处理,使其适合于去除的沉默的音频。默认值是0。

- stop_duration

指定一个时间的沉默之前,必须存在音频不是复制，通过指定一个更高的持续时间，沉默会更多的留在音频中，默认为0.

- stop_threshold

这个值类似 `start_threshold` ,但是从音频末尾削减。也可以用dB值或者振幅比指定，默认为0

- leave_silence

这表明 `stop_duration` 长度的音频应该原封不动的在每个周期的开始沉默，例如如果你想删除长单词之间的停顿,但不想完全删除停顿。默认值是0。

silenceremove例子

- 下面的示例说明了如何使用这个过滤器开始录音,而不包含通常发生的按键后的延迟按，即按下键到开始记录之间的静默：

`silenceremove=1:5:0.02`

treble

对频点的3倍（上下）利用双刀搁置（two-pole shelving）滤镜增加或者减少频率响应，类似于高保真的音控，也被称为搁置平衡（EQ）

滤镜接受下面选项：

- gain, g

Give the gain at whichever is the lower of ~22 kHz and the Nyquist frequency. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.

- frequency, f

设置频点，默认3000Hz.

- width_type

设置带宽计算模式.

h

Hz

q

Q-Factor

o

octave

s

slope

- width, w

设置带宽，其根据 `width_type` 计数，仅对双极滤镜，有0.707q的.巴特沃斯响应

volume

调整输入音量

接受下面的参数：

- volume

设置音频音量表达式

输出值是剪辑的最大值

输出音频音量有如下关系：

$$\text{output_volume} = \text{volume} * \text{input_volume}$$

默认为 "1.0". precision

这个参数指出数学（计算）精度

它决定哪些输入样本格式将被允许,这影响伸缩量的精度.

fixed

8-bit 整形, 它限于输入是 U8, S16, 和 S32.

float

32-bit 浮点, 它限于输入格式是FLT, 这是默认值

double

64-bit 浮点; 它限于输入格式是DBL

replaygain

选择当输入帧数据遇到`ReplayGain`（重演设置—音量增益）时的处理

drop

丢掉ReplayGain, 按原始标准（默认值）.

ignore

忽略ReplayGain侧数据, 但是离开它的框架

track

采用轨道设置, 如果存在

album

采用专辑设置, 如果存在

replaygain_preamp

在应用`replaygain`的前置放大增益, 单位dB

replaygain_preamp默认值为0.0.

eval

设置如果计算音量表达式
它接受以下值：
‘once’
 仅在初始化时计算一次，或者`volume`命令被发生时
‘frame’
 在每个输入帧都重新计算
默认‘once’.

音量表达式支持下面的参数：.

- n
 帧数 (从0开始计数)
- nb_channels
 通道数
- nb_consumed_samples
 总的经过滤镜的样本点数
- nb_samples
 当前帧中样本点数
- pos
 在文件中的帧偏移
- pts
 帧的PTS
- sample_rate
 采样率
- startpts
 流开始来的PTS计数
- startt
 流开始来的时间
- t

帧时间

- `tb`

时间戳时基

- `volume`

最近设置的音量值

注意如果计算模式是 `once`，则除了 `sample_rate` 和 `tb` 有效外其他都无效而等于 `NAN`

volume命令

滤镜支持下面的命令：

- `volume`

编辑音量表达式。这个命令接受相同选项和语法作为命令参数

如果描述的表达式无效，它将保持当前值

- `replaygain_noclip`

通过限制防止剪裁

默认对于 `replaygain_noclip` 为1

volume例子

- 调整输入音量

`volume=volume=0.5` `volume=volume=1/2` `volume=volume=-6.0206dB`

在上面的例子中所指定的选项名 `volume` 可以省略，例如：

`volume=0.5`

- 输入音频功率增加6dB，使用定点精度

`volume=volume=6dB:precision=fixed`

- 一个音频在10秒后5秒内逐渐削弱效果

`volume='if(lt(t,10),1,max(1-(t-10)/5,0))':eval=frame`

volumedetect

检测输入音频音量

滤镜没有参数，输入也不会被编辑。统计数据将在输入流结束后输出到日志中。

特别是它显示平均音量（均方根），最大音量（每个样本基础上）和开始来的音量直方图（从最大音量累计1/1000样本）

所有的音量都是相对于最大PCM值的

volumedetect例子

这里有一个输出实例：

```
[Parsed_volumedetect_0 0xa23120] mean_volume: -27 dB
[Parsed_volumedetect_0 0xa23120] max_volume: -4 dB
[Parsed_volumedetect_0 0xa23120] histogram_4db: 6
[Parsed_volumedetect_0 0xa23120] histogram_5db: 62
[Parsed_volumedetect_0 0xa23120] histogram_6db: 286
[Parsed_volumedetect_0 0xa23120] histogram_7db: 1042
[Parsed_volumedetect_0 0xa23120] histogram_8db: 2551
[Parsed_volumedetect_0 0xa23120] histogram_9db: 4609
[Parsed_volumedetect_0 0xa23120] histogram_10db: 8409
```

它意味着：

- 平均音量是-27dB,或 10.....-2.7
- 最大音量点为-4dB或者说介于-4dB 到-5dB
- 有6个样本点是-4dB，62个-5dB，286个-6dB 等等

换句话说，提供+4dB的音量不会引起任何剪裁（削波），而提高+5dB就有6个地方会削波。

35 音频源

下面介绍当前可用的音频源

abuffer

缓冲音频帧，作为滤镜链图中有效的组成（起点）

它主要编程使用，特别是通过 `libavfilter/asrc_abuffer.h` 中的接口进行调用。

接受如下参数：

- `time_base`
用于提交帧的时间戳时基。是浮点数或者分数形式。
- `sample_rate`
进入音频缓冲的采样率。
- `sample_fmt`
进入音频缓冲的采样格式。 `libavutil/samplefmt.h` 下 `AVSampleFormat` 枚举值中的一个格式名称或者对应的整数
- `channel_layout`
进入音频缓冲的通道布局。为 `libavutil/channel_layout.c` 中的 `channel_layout_map` 定义的布局名称或者 `libavutil/channel_layout.h` 中 `AV_CH_LAYOUT_*` 类宏（对应的整数表示）
- `channels`
进入缓冲的通道数。如果 `channels` 和 `channel_layout` 同时被设置，则二者必须一致。

abuffer例子

`abuffer=sample_rate=44100:sample_fmt=s16p:channel_layout=stereo` 会用来指出源接受16位（信号）立体声（采样率44100Hz）。 `sample_fmt` 为 `s16p` 即 6， `channel_layout` 为 `stereo`，即 `0x3`， `sample_rate` 为 44100，它等效于

`abuffer=sample_rate=44100:sample_fmt=6:channel_layout=0x3`

aevalsrc

按表达式生成一个音频信号（信号发生器）

它接受一个或者多个表达式（每个对应一个通道），根据表达式计算产生相应的音频信号。

接受如下的选项：

- `exprs`
由'|'分隔的表达式列表，每个表达式对应一个通道。以防 `channel_layout` 没有指定选项，选中的通道布局取决于提供的数量表达式。否则最后指定表达式应用于剩下的输出通道。
- `channel_layout, c`

设置通道布局。这里的通道数必须等于表达式数量。

- duration, d

设置源音频持续时间。参考 `ffmpeg-utils` 工具集手册（`ffmpeg-utils(1)`）中的[持续时间](#)章节内容以了解语法。注意由此生成的音频持续时间可能会超过这里指定的时间，因为生成的音频最少是一个完整的帧内容。

如果不指定，或者指定一个非负数，表面会持续生成音频信号。

- nb_samples, n

设置每个输出帧中每个通道的样例数量，默认1024。

- sample_rate, s

指定采样频率，默认44100。

每个表达式可以包含下面的常量：

- n

评估样本的数量，从0开始计数

- t

样本时间表示，从0开始计时

- s

样本采样率

aevalsrc例子

- 生成静音（无声）：

```
aevalsrc=0
```

- 生成频率为440Hz的正弦波，采样频率8000Hz：

```
aevalsrc="sin(4402PI*t):s=8000"
```

- 生成双路信号，这里指定为（中前和中后），表达式为：

```
aevalsrc="sin(4202PI*t)|cos(4302PI*t):c=FC|BC"
```

- 生成白噪声：

```
aevalsrc="-2+random(0)"
```

- 生成一个振幅调制信号：

```
aevalsrc="sin(102PI*t)sin(8802PI*t)"
```

- 生成2.5赫兹双耳节拍在360赫兹的载体：

```
aevalsrc="0.1sin(2PI(360-2.5/2)t) | 0.1sin(2PI(360+2.5/2)t)"
```

anullsrc

null（空）音频源会产生未处理的音频帧。它一般用于分析/调试，或作为滤镜可忽略的输入源（例如 `sox` 合成滤镜）

这个源接受下面选项：

- `channel_layout, cl`

指定通道布局，可以是整数或对应的短语，默认为"stereo".

检查定义在 `libavutil/channel_layout.c` 了解短语和数字值对应关系。

- `sample_rate, r`

指定采样率,默认 44100.

- `nb_samples, n`

设置每帧中样例数量

anullsrc例子

- 以采样率48000 Hz，单声道(`AV_CH_LAYOUT_MONO`).

```
anullsrc=r=48000:cl=4
```

- 同上的效果（采样短语定义布局）：

```
anullsrc=r=48000:cl=mono
```

所有选项参数都必须明确定义。

flite

使用libflite库合成声音话语

编译选项是 `--enable-libflite`

注意 `flite` 库不是线程安全的。

接受如下选项：

- `list_voices`

如果为1，列出有效的语音并退出，默认0.

- `nb_samples, n`

设置每个帧最大样例数量，默认512.

- `textfile`

设置要朗读的文件名

- `text`

设置要朗读的文本

- voice, v

设置语音合成的声音，默认 `kal`。参考 `list_voices` 选项

flite例子

- 从文件speech.txt读，使用标准声音合成：

```
flite=textfile=speech.txt
```

- 读取指定文本，并用 `slt` 语音合成：

```
flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
```

- 作为ffmpeg输入：

```
ffmpeg -f lavfi -i flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
```

- 播放合成语音：

```
ffplay -f lavfi flite=text='No more be grieved for which that thou hast done.'
```

关于 `libflite` 库的更多信息，确认<http://www.speech.cs.cmu.edu/flite/>

sine

生成一个音频信号的振幅的正弦波1/8

是一个bit-exact音频信号（脉冲？）

接受如下选项：

- frequency, f

设置载波频率，默认 440 Hz.

- beep_factor, b

每个 `beep_factor` 倍载波频率周期产生一个 `beep`，默认为0，表示 `beep` 被禁止

- sample_rate, r

指定采样率，默认44100.

- duration, d

指定产生音频持续时间

- samples_per_frame

设置每帧样例数，默认1024

sine例子

- 产生440Hz的 `sine` 波Generate a simple 440 Hz sine wave:

sine

- 产生220Hz sine 波，且880Hz产生一个 beep，持续5秒:

sine=220:4:d=5 sine=f=220:b=4:d=5 sine=frequency=220:beep_factor=4:duration=5

36 音频槽

下面介绍当前有效的音频皮肤。

abuffersink

缓冲音频帧，并可作为滤镜的结束。

这个槽主要用于编程使用，特别是通过 `libavfilter/buffersink.h` 的接口或选择操作系统

它接受指向 `AVABufferSinkContext` 结构的指针，用于定义传入缓冲区的格式，作为不透明参数传递给 `avfilter_init_filter` 以初始化。

anullsink

Null（空）音频槽，绝对没有输入的音频。它主要用作模板以分析/调试工具。

37 视频滤镜

在配置编译FFmpeg时可以通过 `--disable-filters` 来禁止所有滤镜的编译。也可以配置编译脚本来输出所有包含进编译的滤镜信息。

下面是当前可用的视频滤镜介绍。

alphaextract

把输入视频作为灰度视频来提取透明通道，它通常和 `alphamerge` 滤镜联用。

alphamerge

通过添加或者替换透明通道，让主要视频与另外一路视频混合。这里主要是使用 `alphaextract` 来让不支持透明通道的视频成为允许传输或存储帧透明的帧序列

例如：为了重建完整的帧，让一个普通的YUV编码视频和利用了 `alphaextract` 的一个单独的视频混合：

```
movie=in_alpha.mkv [alpha]; [in][alpha] alphamerge [out]
```

因为这个滤镜专为重建帧设计，所以它的运行过程中没有考虑时间戳，而是在输入达到流结束时终止。这可能将导致编码管道丢帧，如果你仅想实现一个图像叠加到视频，建议采用 `overlay` 滤镜。

ass

类似 `subtitles`，除了它不需要 `libavcodec` 和 `libavformat`。另一方面，它受限于ASS（Advanced Substation Alpha-高级子透明）字幕格式文件

这个滤镜除了 `subtitles` 滤镜选项外还接受下面选项：

- `shaping`

设置图像引擎，有效的值有：

`'auto'`

默认的 `'libass'` 图像引擎，它是最合适的值

`'simple'`

快，仅采用 `font-agnostic` 图像

`'complex'`

慢，采用 `OpenType`。

默认为 `auto`。

bbox

在输入的亮度平面上计算非黑边缘。

这个滤镜按设定的最小亮度值计算平面上所有像素图像的编译（按亮度——即按阈值连接起来构成多个区块，阈值连线即边缘）。描述边缘的参数会记录到滤镜日志中。

滤镜接受下面的选项：

- min_val

设置最小的亮度值（其下认为是黑的，其上为非黑），默认16

blackdetect

检测视频（几乎）完全是黑色的时间间隔（段）。它可以用于检测章间的过渡、广告或者无效的录制。输出是以秒为单位的检测出黑场时间段的开始、结束和持续时间线信息。

为了显示（输出）时间线信息，你需要设置日志层次为 `AV_LOG_INFO`。

滤镜接受下面的选项：

- black_min_duration, d

以秒为单位设置最小黑场持续时间（小于这个的不被检出），它必须是非负浮点数，默认为2.0

- picture_black_ratio_th, pic_th

设置画面黑场检验标准，即图像中黑色像素占画面所有像素比的最小值：

$\text{nb_black_pixels} / \text{nb_pixels}$

如果画面中黑色像素占比大于这个值则判断当前画面是黑场画面，默认值为0.98

- pixel_black_th, pix_th

设置判断像素是黑色像素的标准

采用像素亮度值来判断，只有像素亮度不大于一个设定的值（所确定的标准，见后）即为黑色像素。其依据下面的方程计算亮度：

$\text{absolute_threshold} = \text{luminance_minimum_value} + \text{pixel_black_th} * \text{luminance_range_size}$

`luminance_range_size` 和 `luminance_minimum_value` 依赖于输入视频格式，范围为 对YUV全范围格式视频[0-255]，非全范围格式[16-235]

默认值0.10（对于YUV全范围格式，则计算值为 $0+0.1255=25.5$ ，对于非YUV全范围则为 $16+0.1(235-16)=37.9$ ）

下面的例子设置黑像素标准为最小值，黑场时长大于等于2秒：

```
blackdetect=d=2:pix_th=0.00
```

blackframe

判断（几乎）黑帧。它可以用于检测章间的过渡、广告等。输出内容包括检测到的黑帧数、占比和文件中的偏移（对文件格式

式支持检测，如果不支持检测为-1）和以秒为单位的时间戳。

为了显示这些输出信息，需要设置日志层次为 `AV_LOG_INFO`

它接受下面的参数：

- amount

像素为黑在帧中的占比百分比数，默认98

- threshold, thresh

像素为黑的判断标准，默认为32

blend, tblend

混合两个视频帧

其中 `blend` 混合两路输出1路流，第一个输入为 `top` 层，二个路为 `bottom` 层，输出以输入短的为结束。而 `tblend`（时间混合）需要从一个单独视频流的连续两帧，让新帧在上叠加在老帧上。

接受选项的介绍如下：

- c0_mode
- c1_mode
- c2_mode
- c3_mode
- all_mode

设置混合模式（对指定像素或者所有像素——利用 `all_mode`），默认值是 `normal`

当前有效的混合模式如下:

'addition' 'and' 'average' 'burn' 'darken' 'difference' 'difference128' 'divide' 'dodge' 'exclusion' 'glow' 'hardlight' 'hardmix' 'lighten' 'linearlight' 'multiply' 'negation' 'normal' 'or' 'overlay' 'phoenix' 'pinlight' 'reflect' 'screen' 'softlight' 'subtract' 'vividlight' 'xor'

- c0_opacity
- c1_opacity
- c2_opacity
- c3_opacity
- all_opacity

设置特定像素的透明度，或者设置整个透明度（利用 `all_opacity`），仅用于组合像素混合模式 `blend` 滤镜.

- c0_expr
- c1_expr
- c2_expr
- c3_expr
- all_expr

设置特定像素混合表达式或所有像素混合表达式（`all_expr`），注意如果它们被设定，则相关模式选项被忽略

表达式可以采用下面的变量:

- 从左到右揭开的效果:

```
blend=all_expr='if(gte(N*SW+X,W),A,B)'
```

- 从上到下揭开效果:

```
blend=all_expr='if(gte(Y-N*SH,0),A,B)'
```

- 从右下向左上揭开效果:

```
blend=all_expr='if(gte(TSH40+Y,H)gte((T40SW+X)W/H,W),A,B)'
```

- 显示当前和前一帧之间的差异:

```
tbblend=all_mode=difference128
```

boxblur

对输入视频应用边缘虚化

接受下面的参数：

- luma_radius, lr
- luma_power, lp
- chroma_radius, cr
- chroma_power, cp
- alpha_radius, ar
- alpha_power, ap

接受选项的介绍见下：

- luma_radius, lr
- chroma_radius, cr
- alpha_radius, ar

在输入平面，设置用于边缘模糊的像素半径表达式

这个半径必须是非负数，且必须在亮度和透明度（平面/通道上）大于 $\min(w,h)/2$ ，在色度平面上大于 $\min(cw,ch)/2$

`luma_radius` 默认为2，如果没有指定 is "2". `chroma_radius` 和 `alpha_radius` 默认是根据 `luma_radius` 值的默认集

表达式中允许下面内容：

w h

输入视频宽和高

cw ch

输入色度图像的高和宽

hsub vsub

水平和垂直采样的色彩浓度值，例如对于 "yuv422p"像素格式，`hsub`为2，`vsub`为1

- luma_power, lp
- chroma_power, cp
- alpha_power, ap

指定应用于相应平面的边缘虚化滤镜次数。

默认 luma_power 为2， chroma_power 和 alpha_power 的默认值是根据 luma_power 的对于默认值集

如果有1个值为0则禁止了效果

boxblur例子

- 对亮度、色度和透明都以2的半径进行边缘虚化（模糊）：

boxblur=luma_radius=2:luma_power=1 boxblur=2:1

- 设置亮度编译模糊半径为2，而透明和色度模糊半径为0:

boxblur=2:1:cr=0:ar=0

- 设置亮度和色度模糊半径是视频维度的小部分（这里是1/10）：

boxblur=luma_radius=min(h\,w)/10:luma_power=1:chroma_radius=min(cw\,ch)/10:chroma_power=1

codeview

可视化展示一些编码的信息

一些编码可以在图像的一端或者其他地方输出信息。例如一些 MPEG 编码器就可以通过设置 flags2 标志为 export_mvs 来输出运动矢量检测信息

滤镜接受下面选项：

- mv

设置运动矢量检测来可视化，

对 mv 选项有效的标志有：

'pf'

基于P帧的前向预测

'bf'

基于B帧的前向预测

'bb'

基于B帧的后向预测

codecview例子

- 基于P帧和B帧，可视化多径向运动矢量检测

```
ffmpeg -flags2 +export_mvs input.mpg -vf codecview=mv=pf+bf+bb
```

colorbalance

对输入帧编辑主要的颜色（红、绿和蓝）强度

滤镜用于对输入帧调整阴影、中间调或高亮区域实现 red-cyan（红和蓝绿），green-magenta（绿和品红）或blue-yellow（蓝和黄）的平衡。

一个正向的调整值对应于平衡主要颜色，一个负数值则对应于补色。

滤镜接受下面的选项：

- rs
- gs
- bs

调整red, green和blue的阴影(黑暗像素).

- rm
- gm
- bm

调整red, green和blue的中间调 (中亮度像素).

- rh
- gh
- bh

调整red, green和blue的高亮 (亮的像素).

所有值的取值范围为[-1.0, 1.0]，默认为0（不调整）

colorbalance例子

- 添加影子的红色

```
colorbalance=rs=.3
```

colorkey

RGB颜色键控

滤镜接受下面的选项：

- color

设置被作为透明的颜色

- similarity

设置对色键的相似性百分比（表示也作为色键的颜色范围）

0.01匹配表示只有色键，而1表示所有颜色（相当于直接是透明了）

- blend

混合百分比

0.0 表示像素完全透明或者完全不透明

更高的值导致半透明像素，对于更高的透明度相当于像素的颜色更接近于色键

colorkey例子

- 让所有的绿色像素透明

```
ffmpeg -i input.png -vf colorkey=green out.png
```

- 在绿屏上显示背景图片

```
ffmpeg -i background.png -i video.mp4 -filter_complex "[1:v]colorkey=0x3BBD1E:0.3:0.2[ckout];[0:v][ckout]overlay[out]"
-map "[out]" output.flv
```

colorlevels

使用层来调整输入视频

滤镜接受下面的选项：

- rmin
- gmin
- bmin
- amin

调整输入红、绿、蓝和透明的黑（标准），允许范围为[-1.0,1.0]，默认为0

- rmax
- gmax
- bmax
- amax

调整输入红、绿、蓝和透明的白（标准），允许范围为[-1.0,1.0]，默认为1.

输入水平被用来减轻突出（明亮的色调），变暗阴影（暗色调），改变亮和暗的平衡

- romin
- gomin
- bomin
- aomin

调整输出红、绿、蓝和透明的黑（标准），允许范围为[-1.0,1.0]，默认为0

- romax
- gomax

- bomax
- aomax

调整输出红、绿、蓝和透明的白（标准），允许范围为[-1.0,1.0]，默认为1.

用来手动调整输出电平范围

colorlevels例子

- 让视频输出暗色调

```
colorlevels=rimin=0.058:gimin=0.058:bimin=0.058
```

- 增加对比度

```
colorlevels=rimin=0.039:gimin=0.039:bimin=0.039:rimax=0.96:gimax=0.96:bimax=0.96
```

- 让视频更亮

```
colorlevels=rimax=0.902:gimax=0.902:bimax=0.902
```

- 增加亮度

```
colorlevels=romin=0.5:gomin=0.5:bomin=0.5
```

colorchannelmixer

通过重新混合颜色通道调整视频输入帧

这个滤镜通过在不同颜色通道对同一个像素添加与其他通道相关的值进行颜色调整，例如为了修改红色，输出会是：

```
red=red*rr + blue*rb + green*rg + alpha*ra
```

滤镜接受下面选项：

- rr
- rg
- rb
- ra

为调整输出的红色通道而进行参数设置（分别对应于根据红、绿、蓝和透明通道的权值），rr默认为1，其他默认为0

- gr
- gg
- gb
- ga

为调整输出的绿色通道而进行参数设置（分别对应于根据红、绿、蓝和透明通道的权值），gg默认为1，其他默认为0

- br
- bg
- bb
- ba

为调整输出的蓝色通道而进行参数设置（分别对应于根据红、绿、蓝和透明通道的权值），bb默认为1，其他默认为0

- ar
- ag
- ab
- aa

为调整输出的透明通道而进行参数设置（分别对应于根据红、绿、蓝和透明通道的权值），aa默认为1，其他默认为0

各个值的运行范围为[-2.0, 2.0].

colorchannelmixer例子

- 转换源到灰度模式

```
colorchannelmixer=.3:.4:.3:0:.3:.4:.3:0:.3:.4:.3
```

- 模拟墨色应用

```
colorchannelmixer=.393:.769:.189:0:.349:.686:.168:0:.272:.534:.131
```

colormatrix

颜色转换矩阵（常用于颜色标准转换）

滤镜接受下面的选项：

- src
- dst

分别指定源和目标的颜色矩阵模式。两个值都必须设置。

接受的值为：

‘bt709’

BT.709

‘bt601’

BT.601

‘smpte240m’

SMPTE-240M

‘fcc’

FCC

例如要转换BT.601 到 SMPTE-240M需要：

```
colormatrix=bt601:smpte240m
```

copy

复制输入源，而不改变的输出，常用于测试。

crop

裁剪输入

接受下面的参数：

- w, out_w

输出视频的宽，默认为 iw。这个表达式只有过滤器配置是进行一次求值

- h, out_h

输出视频的高，默认为 ih。这个表达式只有过滤器配置是进行一次求值

- x

输入的水平坐标，默认为(in_w-out_w)/2，每帧都计算

- y

输入的垂直坐标，默认为(in_h-out_h)/2，每帧都计算

- keep_aspect

如果设置为1，则强制输出采样输入一样的长宽比例，默认为0

这里 out_w, out_h, x, y 参数都可以是表达式，包含下述意义：

- x

- y

计算x和y，通常每帧计算

- in_w

- in_h

输入的宽和高

- iw

- ih

同于 in_w 和 in_h

- out_w

- out_h

输出的（裁剪后的）宽和高

- ow
- oh

同于 out_w 和 out_h

- a

同于 `iw / ih`

- sar

输入的样本点（像素）长宽比

- dar

输入显示样本点长宽比它同于 `(iw / ih) * sar`

- hsub
- vsub

水平和垂直颜色分量值。例如对于"yuv422p"的像素， hsub 为2， vsub 为1

- n

输入帧的序数，从0计数

- pos

输入帧在文件中的偏移，如果未知则为 NAN

- t

秒为单位的输入帧时间戳，如果时间戳未知则为 NAN

crop例子

- 从(12,34)开始裁剪出一个100x100的图像

crop=100:100:12:34

- 使用命名选项，同上例一样：

crop=w=100:h=100:x=12:y=34

- 在输入中心裁剪出100x100:

crop=100:100

- 裁剪输入的2/3区域:

crop=2/3in_w:2/3in_h

- 裁剪输入中心区域:

crop=out_w=in_h crop=in_h

- 据边缘100裁剪出中间部分.

```
crop=in_w-100:in_h-100:100:100
```

- 据左右10像素，上下20像素的中间部分

```
crop=in_w-210:in_h-220
```

- 保留右下角（四分后的右下部分）

```
crop=in_w/2:in_h/2:in_w/2:in_h/2
```

- 按黄金分割裁剪

```
crop=in_w:1/PHI*in_w
```

- 应用抖动效果

```
crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)sin(n/10):(in_h-out_h)/2+((in_h-out_h)/2)sin(n/7)
```

- 取决于时间戳的不定照相效果:

```
crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)sin(t10):(in_h-out_h)/2+((in_h-out_h)/2)sin(t13)"
```

- 让x依赖于y

```
crop=in_w/2:in_h/2:y:10+10*sin(n/10)
```

cropdetect

自动检测裁剪尺寸(自动去除边缘的黑部)

它计算必要的剪切参数并通过日志系统输出推荐的参数。检测尺寸对应于输入视频的黑色区域

它接受下面的参数：

- limit

设置更高的黑色阈值。可以选择从0到所有（255，基于8位格式）。一组强度值更大的价值被认为是黑色。它默认为24。你也可以指定一个值在0.0和1.0之间,将按比例根据像素格式的位深计算

- round

设置可行（分割出）的宽/高分割基数，默认为16。偏移量自动调整到视频的中心，对于平面维度采用2的倍数（4：2：2视频需要的），16已经可以满足大多数编码要求了

- reset_count, reset

设置在处理多少帧后计数器重置以重新检测最佳裁剪区域。默认为0。

它被用于通道标志混乱的视频。0表示不复位，并返回已播放视频的最大区域

curves

利用curves（曲线）实现颜色调整

这个滤镜模拟Adobe Photoshop 和GIMP的 `curves` 工具。每个分量（红、绿和蓝）都由定义的 `N` 个节点相互作用成为光滑的曲线来调整输入到输出的关系。其中x轴是像素从输入帧获取的值，而y轴则是对应作为输出帧的值（从而实现输入到输出的

映射)。

默认原始的曲线实为 (0, 0) 和 (1, 1) 连接的直线。这种“变化”关系就是输出=输入，意味着没有变化。

滤镜允许根据设置的两点或者更多点产生一个新的分量关系曲线（采用自然三次样条插值法依次连接各个点形成的平滑曲线）。所有的点坐标中x和y值范围在[0,1]，超出定义域的点都被抛弃。

如果定义中没有关键点定义在x=0，则自动添加点(0,0)，类似，如果没有关键点定义在x=1，则自动添加一个 (1, 1) 点。

下面是滤镜允许的选项：

- preset

选定一个颜色预置（配置）。这个选项用在除了r, g, b选项的设置。在这种情况下，在后的选择作为实际预置。可用的预置有：

'none' 'color_negative' 'cross_process' 'darker' 'increase_contrast' 'lighter' 'linear_contrast' 'medium_contrast' 'negative' 'strong_contrast' 'vintage'

默认为none.

- master, m

设置主要的关键点。这些关键点用于定义第二个通过映射。它们有时调用"luminance" 或者 "value"映射。它们可以被用于r, g, b 或者 all，因为它们更像一个后处理LUT。

- red, r

对红色分量指定关键点

- green, g

对绿色分量指定关键点

- blue, b

对蓝色分量指定关键点

- all

设置针对所有分量的关键点（对复合信号，但不包括主要的——即对单独设置了的分量外的其他分量的处理），在这种情况下，没有单独设置的分量都采用这里的设置

- psfile

指定一个Photoshop曲线文件(.asv)来导入设置

curves例子

- 略有增加中间层次的蓝色:

curves=blue='0.5/0.58'

- 复古的效果:

curves=r='0/0.11 .42/.51 1/0.95':g='0.50/0.48':b='0/0.22 .49/.44 1/0.8'

在这里,我们为每个分量设置以下坐标:

red

```
(0;0.11) (0.42;0.51) (1;0.95)
```

green

```
(0;0) (0.50;0.48) (1;1)
```

blue

```
(0;0.22) (0.49;0.44) (1;0.80)
```

- 前面的例子也可以通过 预置实现:

```
curves=preset=vintage
```

- 或者更简单的:

```
curves=vintage
```

- 采用Photoshop预置并应用于绿色分量:

```
curves=psfile='MyCurvesPresets/purple.asv':green='0.45/0.53'
```

dctdnoiz

采用2D DCT来降噪（频域滤波）

滤镜不能实时应用。

滤镜接受下面的选项：

- sigma, s

设置固定的噪声标准差（西格玛——sigma）

这里的 `sigma` 用于确定一个固定的阈值 `3*sigma`，每个DCT系数（绝对值）如果低于这个阈值则被丢弃（认为是噪声）

如果你需要更多高级的过滤特性，则需要 `expr`

默认为 0

- overlap

为每个块重叠的像素数。如果滤镜非常慢，你可以降低这个值，从而在较少的成本下有效过滤各种纹理

如果重叠值不允许处理整个输入宽度或高度,将显示一个警告，且在边界不会运用

默认为blocksize-1,它通常是一个最佳设置

- expr, e

设置系数表达式

对于每个DCT块的系数，表达式会计算得出一个乘数系数值

如果这里被设置，则 `sigma` 选项被忽略

系数的绝对值可以通过 `c` 变量访问到

- `n`

按位宽设置块尺寸。1远远小于 n ($1 < n$) ,这决定于处理块的宽度和高度

默认值是3(对应于8x8)和 4 (对应于16x16) , 注意改变这个值将极大影响处理速度, 同时一个更大的块大小并不一定有好的降噪效果

dctdnoiz例子

- 以 `sigma` 为4.5降噪

```
dctdnoiz=4.5
```

- 同上效果，但是采用了 `expr`

```
dctdnoiz=e='gte(c, 4.5*3)'
```

- 块大小设置为16x16

```
dctdnoiz=15:n=4
```

decimate

定期重复的帧数丢弃

接受下面的选项：

- `cycle`

指示要被丢弃的帧。设置为 `N` 表示如果一帧已经被重复 `N` 次就被丢弃，默认为5

- `dupthresh`

设置检验重复的阈值。如果帧间变化量小于等于阈值则被认为是重复的。默认为1.1.

- `scthresh`

设置场景变化阈值，默认为15

- `blockx`
- `blocky`

设置x或y轴大小，其用于度量计算内存。更大的内存块可以更好的实现噪声抑制，但小的运动检测效果也越差。必须是2的幂。，默认为32

- `ppsrc`

标记一个输入是有预处理的，还有一个是原始源。它允许通过输入（经各种滤镜）预处理的源和原始源来更好的无损保

持内容。设置为1时,第一个输入是经过预处理的输入流,第二个流是清洁源,其保留了原始的帧（信息）。默认为0。

- chroma

设置颜色通道是否也用这个矩阵计算。默认为1

dejudder

删除部分隔行电视电影的内容产生的颤动

颤动可被引发,例如通过 `pullup` 滤镜,如果原始内容经 `pullup` 产生向上的爬动效应, `dejudder` 可以引入动态帧频来消除。它可能会改变容器的帧频记录,除了这个改变,滤镜不会影响固定帧频视频的其他地方。

下面的选项被允许：

- cycle

指定颤抖的重复窗口的长度

接受一个大于1的整数,常用值有:

‘4’

常用于把24帧电影转换为30帧的NTSC信号

‘5’

用于把25帧的PAL转换为30帧的NTSC

‘20’

前俩个的混合

默认为‘4’。

delogo

通过一个简单的插值周围像素来抑制台标。仅需要在台标周围设置一个矩形遮盖就可以消除台标（有时会出现莫名的东西）

它接受下面的参数：

- x
- y

指定台标覆盖的x和y坐标,必须被设置

- w
- h

指定覆盖的宽和高,必须被设置（联合前面的x和y就定义了一个矩形覆盖区）

- band, t

指定矩形的边缘模糊厚度（添加到w和h），默认为4

- show

但设置为1时，一个绿色的矩形被简单填充到覆盖区。默认为0，这时采用的区域内像素填充计算方法为：矩形内，画在最外层的像素将(部分)内插替换值。下一个像素的值在每个方向用于计算矩形内的内插像素值。

delogo例子

- 采用默认方法，在坐标（0,0）采用100x77，模糊10的台标覆盖：

```
delogo=x=0:y=0:w=100:h=77:band=10
```

deshake

尝试在小的范围内去除/修复 水平/垂直变化。用于去除手持、或者旋转三脚架以及在移动车辆上拍摄的抖动（防抖效果）

接受下面的选项

- x
- y
- w
- h

定义一个处理区域，它限定了动态监测搜索范围。它定义了有限运动矢量的左上角位置以及宽度和高度。这些参数随着 `drawbox` 滤镜一样有相同的意义，可以用来界定想象的边界位置

对于同时运动（比如在车上拍摄），它十分有用，因为主题框架内混淆了本身有益的运动和无意的抖动，通过限定可以有效区分，减少搜索难度。

如果这些参数中的一个或者所用被设置为-1则意味着在全帧应用。这使得后来选项设置不指定的边界框运动矢量搜索

默认为搜索整个帧

- rx
- ry

指定最大程度x和y方向运动范围，值范围为0-64像素，默认为16.

- edge

指定如何生成像素来填补空白的边缘。可用值：

'blank, 0'

在空白的地方填零

'original, 1'

填充原始图像（背景部分） locations

'clamp, 2'

在空白地方挤压边缘值

'mirror, 3'

在空白位置反映边缘

默认为 'mirror'.

- **blocksize**

指导用于运动检测的块尺寸，范围4-128像素，默认8.

- **contrast**

指定块的对比度阈值。只有块超过指定的对比(黑暗和轻的像素)将被考虑。范围1-255,默认125。

- **search**

指定的搜索策略。可用值:

'exhaustive, 0'

穷举搜索

'less, 1'

不详尽的搜索

默认'exhaustive'.

- **filename**

如果设置,那么运动搜索的详细日志写入指定的文件

- **opencl**

如果设置为1时,指定使用OpenCL功能,只在编译FFmpeg时配置了 `—enable-opencl` 可用。默认值是0。

detelecine

使用电视电影的逆操作。它需要一个预定的模板指定使用过的模式（必须同原始操作一样的参数）

滤镜接受下面的选项：

- **first_field**

'top, t'

顶场优先

'bottom, b'

底场优先

默认为top.

- pattern

一串数字表示您想要应用的下拉模式。默认值是23。

- start_frame

许多代表第一帧的位置对电视电影模式,这用于流被剪辑过。默认值是0。

drawbox

在输入图像上画一个颜色区块。

它接受下面的参数：

- x
- y

指定区块的x和y坐标，可以是表达式，默认为0

- width, w
- height, h

指定表示区块宽和高的表达式，如果为0表示整个输入的宽和高，默认为0

- color, c

指定填充颜色。语法详见[颜色/color](#)中的介绍。如果指定的值是 `invert`，则区块边缘颜色采用视频该处颜色的反亮（这样可以衬托出区域）。

- thickness, t

设置边缘宽度的表达式，默认为3.

各个表达式接受的值见下面的介绍。

对于 x, y, w 和 h 以及t是表达式时，可以包括下面的内容:

- dar

输入的显示长宽比，它即为 (w / h) * sar.

- hsub
- vsub

水平和垂直色度量量化分量值，例如对于"yuv422p"像素格式，hsub为2，vsub为1

- in_h, ih
- in_w, iw

输入的宽和高

- sar

输入样本点的长宽比

- x
- y

区块的x和y坐标偏移

- w
- h

区块的宽和高

- t

区块边缘厚度

这些x, y, w, h 和 t常（变）量都允许相互在表达式中引用，例如可以设置 $y=x/dar$ 或 $h=w/dar$.

drawbox例子

- 在输入图像上画一个黑块

drawbox

- 采用50%透明的红色画一个区块

drawbox=10:20:200:60:red@0.5

也可以写为

drawbox=x=10:y=20:w=200:h=60:color=red@0.5

- 采用 pink 填充一个区块

drawbox=x=10:y=10:w=100:h=100:color=pink@0.5:t=max

- 画一个2像素红，有2.40:1的遮盖区域

drawbox=x=t:y=0.5(ih-iw/2.4)-t:w=iw+t*2:h=iw/2.4+t*2:t=2:c=red

drawgrid

在输入图像上画上网格线（外框）

接受下面的参数：

- x
- y

指定区块的x和y坐标，可以是表达式，默认为0

- width, w

- height, h

指定表示区块宽和高的表达式，如果为0表示整个输入的宽和高采用最小的 `thickness` 划线，默认为0.

- color, c

指定填充颜色。语法详见 [颜色/color](#) 中的介绍。如果指定的值为 `invert`，则网格线采用视频该处的反亮颜色绘制。

- thickness, t

设置边缘宽度的表达式，默认为1.

各个表达式接受的值见下面的介绍。

对于 x, y, w 和 h 以及t是表达式时，可以包括下面的内容:

- dar

输入的显示长宽比，它即为 $(w / h) * sar$.

- hsub
- vsub

水平和垂直色度量化分量值，例如对于"yuv422p"像素格式，hsub为2，vsub为1

- in_h, ih
- in_w, iw

输入的宽和高

- sar

输入样本点的长宽比

- x
- y

区块的x和y坐标偏移

- w
- h

区块的宽和高

- t

区块边缘厚度

这些x, y, w, h 和 t常（变）量都允许相互在表达式中引用，例如可以设置 $y=x/dar$ 或 $h=w/dar$.

drawgrid例子

- 以2像素宽度绘制一个100x100的网格，颜色是红色，透明度50%

`drawgrid=width=100:height=100:thickness=2:color=red@0.5`

- 在图像上绘制3x3网格，透明度50%

```
drawgrid=w=iw/3:h=ih/3:t=2:c=white@0.5
```

drawtext

在视频的上绘制文本或者描述于文件的文本块，使用了 `libfreetype` 库。

为了允许这个滤镜，在编译时需要配置 `--enable-libfreetype`，为了允许默认的字体回调和 `font` 选项，还需要配置 `--enable-libfontconfig`。为了允许 `text_shaping` 选项，还需要配置 `--enable-libfribidi`

drawtext语法

它接受如下参数：

- `box`

设置是否在文本下衬一个背景颜色，1为要，0为不要，默认为0

- `boxborderw`

设置背景块边缘厚度（用于在背景块边缘用 `boxcolor` 再围绕画一圈），默认为0

- `boxcolor`

设置用于绘制文本底衬的颜色。语法详见[颜色/color](#)中的介绍。

默认为"white".

- `borderw`

使用 `bordercolor` 颜色绘制的文字边缘厚度，默认为0

- `bordercolor`

绘制文本衬底的颜色。语法详见[颜色/color](#)中的介绍。

默认为"black".

- `expansion`

设置文本扩展模式。可以为 `none`，`strftime`（已弃用了）或 `normal`（默认）。见后面[文本扩展](#)中的详细介绍

- `fix_bounds`

如果为true，检查和修复文本坐标来避免剪切

- `fontcolor`

设置文本颜色。语法详见[颜色/color](#)中的介绍。

默认为"black".

- `fontcolor_expr`

用于计算获得动态 `fontcolor` 值的字符串表达式。默认为空，即不处理。当被设置时将把计算结果覆盖 `fontcolor` 选项

- `font`

设置选用的字体，默认为Sans.

- fontfile

指定字体文件。信息中包括路径。这个参数在 `fontconfig` 被禁用时必须设置

- draw

这个选项已不存在，参考 `timeline` 系统（时间线）

- alpha

设置绘制的文本透明度，值范围为0.0-1.0。而且还可以接受x和y的变量。请参阅 `fontcolor_expr`

- fontsize

设置字体大小，默认为16 drawing text. The default value of fontsize is 16.

- text_shaping

如果设置为1，则试图整理文本排列顺序（例如阿拉伯语是按从右到左排序），否则按给定的顺序从左到右排，默认为1

- ft_load_flags

这些标志用于加载字体

这些标志对应于 `libfreetype` 支持的标志，并结合下面的值：

default no_scale no_hinting render no_bitmap vertical_layout force_autohint crop_bitmap pedantic
ignore_global_advance_width no_recurse ignore_transform monochrome linear_design no_autohint

默认值为 "default".

要了解更多信息，请参考文档中 `libfreetype` 标志的 `FT_LOAD_*` 部分。

- shadowcolor

阴影颜色。语法详见[颜色/color](#)中的介绍。

默认为"black".

- shadowx

- shadowy

这里的x和y是字阴影对于字本体的偏移。可以是正数或者负数（决定了偏移方向），默认为0

- start_number

起始帧数，对于 `n/frame_num` 变量。默认为0

- tabsize

用于呈现的区域数量空间大小，默认为4

- timecode

设置初始的时间码，以"hh:mm:ss[.].ff"格式。它被用于有或者没有 `text` 参数，此时 `timecode_rate` 必须被指定

- `timecode_rate`, `rate`, `r`

设置时间码 `timecode` 的帧率（在 `timecode` 指定时）

- `text`

要被绘制的文本。文本必须采用 UTF-8 编码字符。如果没有指定 `textfile` 则这个选项必须指定

- `textfile`

一个文本文件，其内容将被绘制。文本必须是一个 UTF-8 文本序列

如果没有指定 `text` 选项，则必须设定。

如果同时设定了 `text` 和 `textfile` 将引发一个错误

- `reload`

如果设置为1，`textfile` 将在每帧前加载。一定要自动更新它，或者它可能是会被读取的或者失败

- `x`
- `y`

指定文本绘制区域的坐标偏移。是相对于图像顶/左边的值

默认均为"0".

下面是接受的常量和函数

对于x和y是表达式时，它们接受下面的常量和函数:

- `dar`

输入显示接受的长宽比，它等于 $(w / h) * sar$

- `hsub`
- `vsub`

水平和垂直色度分量值。例如对于"yuv422p"格式像素，`hsub` 为2，`vsub` 为1

- `line_h`, `lh`

文本行高

- `main_h`, `h`, `H`

输入的高

- `main_w`, `w`, `W`

输入的宽

- `max_glyph_a`, `ascent`

从基线到最高/上字形轮廓高点（所有内容的最高点）的最大距离。必须是一个正值，因为网格与Y轴方向关系使然

- `max_glyph_d`, `descent`

从基线到最低/下字形轮廓高点（所有内容的最高点）的最大距离。必须是一个负值，因为网格与Y轴方向关系使然

- `max_glyph_h`

最大字形高度，限定了所有单个字的高度，如果设置值小于实际值，则输出可能覆盖到其他行上

- `max_glyph_w`

最大字形宽度，限定了所单个字显示的宽度，如果设置值小于实际值，则发生字重叠

- `n`

输入帧序数，从0计数

- `rand(min, max)`

返回一个min和max间的随机数

- `sar`

输入样本点的长宽比

- `t`

以秒计的时间戳。如果无效则为 `NAN`

- `text_h, th`

渲染文本的高

- `text_w, tw`

渲染文本的宽

- `x`

- `y`

文本的x和y坐标。

所有参数都允许 `x` 和 `y` 被引用，例如 `y=x/dar`

文本扩展

如果 `expansion` 设置为 `strftime`，则滤镜会接受 `strftime()` 序列提供的文本并进行相应扩展。检查 `strftime()` 的文档。这个特性现在是弃用的。

如果 `expansion` 设置为 `none`，则文本都是直接打印文本（即直接以文本内容不扩展进行输出）

如果 `expansion` 设置为 `normal`（它是默认值），将应用下面的扩展规则。

序列形式 `${...}` 的内容将被扩展。大括号之间的文本是一个函数的名字，可能紧随其后是一些用 `:` 隔开的参数。如果包含特殊字符或分隔符（这里是 `:` 或者 `}`），它们应该被转义。

注意对于在作为滤镜参数的 `text` 选项值，或者滤镜链图中的参数（多个滤镜连接时）以及是在 `shell` 环境中使用，则可能需要4层转义。使用文本文件可以避免这些问题（减少转义的使用）。

下面是有效的函数（功能）：

- `expr, e`

计算结果的表达式

它必须有一个参数来计算，接受计算x和y相同的常数和函数。注意并不是所有的常数都适合，例如 `text_w` 和 `text_h` 在此时还是一个未定义值（因为这两个值依赖于这里计算结果）

- `expr_int_format, eif`

把表达式求值和输出格式化为整数

第一个参数是用于计算的表达式，就像是 `expr` 函数（包括了变量/常量等），第二个参数指定输出格式，允许 'x', 'X', 'd' 和 'u'，其意义同于 `printf` 函数（C语言）中的意义。第三个参数是可选的，用来设置格式化为固定位数，左边可以用0来填补。

- `gmtime`

设置滤镜运行时间，是UTC时间。它接受一个 `strftime()` 格式字符串参数

- `localtime`

滤镜运行的本地时间，它以本地时区表示的时间。它接受一个 `strftime()` 格式字符串参数

- `metadata`

帧的元数据。它必须有一个指定元数据键的参数

- `n, frame_num`

帧序号，从0开始计数

- `pict_type`

一个字符描述当前图片类型

- `pts`

当前帧的时间戳。它可以有2个参数

第一个参数是时间戳格式，默认为 `flt` 是秒格式，其有微秒级精度；`hms` 则代表 `[-]HH:MM:SS.mmm` 格式时间戳表示，其也有毫秒精度。

第二个参数是添加到时间戳的偏移量

drawtext例子

- 以FreeSerif字体绘制 "Test Text"，其他可选参数均为默认值

```
drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text'"
```

- 以FreeSerif字体，24大小在x=100 和y=50位置 绘制 'Test Text'，字体采用黄色且还有红色边缘，字体和衬底军为20%透明度

```
drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text':\
```

```
x=100: y=50: fontsize=24: fontcolor=yellow@0.2: box=1: boxcolor=red@0.2"
```

注意如果不是参数列表则不一定采用双引号来标识范围

- 把字显示在视频中间（计算位置）

```
drawtext="fontsize=30:fontfile=FreeSerif.ttf:text='hello world':x=(w-text_w)/2:y=(h-text_h-line_h)/2"
```

- 在视频帧的最后一排从右向左滑动显示一个文本行。假设文件LONG_LINE仅包含一行且没有换行。

```
drawtext="fontsize=15:fontfile=FreeSerif.ttf:text=LONG_LINE:y=h-line_h:x=-50*t"
```

- 从下向上显示文本行内容

```
drawtext="fontsize=20:fontfile=FreeSerif.ttf:textfile=CREDITS:y=h-20*t"
```

- 在视频中间以绿色绘制单独的字符"q"。文本基线放置在视频半高位置

```
drawtext="fontsize=60:fontfile=FreeSerif.ttf:fontcolor=green:text=q:x=(w-max_glyph_w)/2:y=h/2-ascent"
```

- 每3秒显示文本1秒

```
drawtext="fontfile=FreeSerif.ttf:fontcolor=white:x=100:y=x/dar:enable=lt(mod(t\,3)\,1):text='blink'"
```

- 采用 fontconfig 来设置字体。注意冒号需要转义

```
drawtext='fontfile=Linux Libertine O-40\:style=Semibold:text=FFmpeg'
```

- 输出实时编码日期（参考 `strftime(3)`）：

```
drawtext='fontfile=FreeSans.ttf:text=%{localtime\:%a %b %d %Y}'
```

- 以淡入淡出显示文本（显示/消失）

!/bin/sh

```
DS=1.0 # display start
DE=10.0 # display end
FID=1.5 # fade in duration
FOD=5 # fade out duration
ffplay -f lavfi "color,drawtext=text=TEST:fontsize=50:fontfile=FreeSerif.ttf:fontcolor_expr=ff0000%{eif\}:
clip(255(1between(t\, $DS + $FID\, $DE - $FOD) + ((t - $DS)/$FID)between(t\, $DS\, $DS + $FID) + (-(t -
$DE)/$FOD)between(t\, $DE - $FOD\, $DE) )\, 0\, 255) \: x\: 2 }"
```

关于 libfreetype，参考<http://www.freetype.org/>

对于 fontconfig，参考<http://freedesktop.org/software/fontconfig/fontconfig-user.html>。

对于 libfribidi，参考<http://fribidi.org/>。

edgedetect

检测边缘。滤镜采用精明边缘检测(Canny Edge Detection)算法

接受下面的选项：

- low
- high

设置检测算法边缘探测的低和高的阈值。

high 阈值选择 强有力 的边缘像素，然后通过8向邻接检测 软弱 到 low 阈值，从而标注出边缘

low 和 high 的值范围为[0,1]，且 $low \leq high$

默认 low 为 20/255, high 为 50/255.

- mode

定义绘制(边缘)模式

‘wires’

绘制一个 white/gray 间隔线在黑背景上

‘colormix’

混合颜色创建一个油漆/卡通效果

默认为 wires.

edgedetect例子

- 采用了自定义滞后阈值的标准边缘检测

edgedetect=low=0.1:high=0.4

- 绘画效果没有阈值（因为high设置为0，low必须小于等于high则只能为0了）

edgedetect=mode=colormix:high=0

eq

设置亮度、对比度、饱和度和近似伽马(gamma)调整

滤镜支持下面选项：

- contrast

设置contrast表达式，值必须是一个-2.0-2.0间的浮点数，默认为0

- brightness

设置brightness表达式.值必须是一个-1.0-1.0间的浮点数，默认为0

- saturation

设置saturation表达式. 值必须是一个0-3.0间的浮点数，默认为1

- gamma

设置gamma表达式，值必须是一个0.1-10.0间的浮点数，默认为1

- gamma_r

设置gamma表达式，对红色. 值必须是一个0.1-10.0间的浮点数，默认为1

- gamma_g

设置gamma表达式，对绿色. 值必须是一个0.1-10.0间的浮点数，默认为1

- gamma_b

设置gamma表达式，对蓝色. 值必须是一个0.1-10.0间的浮点数，默认为1

- gamma_weight

设置gamma权重表达式. 它可以用来减少高伽马值在明亮的图像区域影响,例如只是普通的白色放大，而其它保持不变。值必须是一个在0.0到1.0范围的浮点数。值为0.0时把伽马校正效果最强，为1.0没有效果。默认设置是“1”。

- eval

设置brightness, contrast, saturation 和 gamma是表达式时的计算模式S

它接受下面值:

'init'

仅在滤镜初始化或者命令被处理时计算

'frame'

每帧计算

默认为'init'.

下面是表达式中接受的参数:

- n

帧序数，从0计数

- pos

当前包在文件中的偏移，如果没有定义则为 NAN

- r

输入视频帧率，如果无效则为 NAN

- t

以秒计的时间戳，如果输入时间戳无效则为 NAN

eq命令

滤镜也接受下面的命令：

- contrast

设置contrast表达式

- brightness

设置brightness表达式

- saturation

设置saturation表达式

- gamma

设置gamma表达式

- gamma_r

设置gamma_r表达式

- gamma_g

设置gamma_g表达式

- gamma_b

设置gamma_b表达式

- gamma_weight

设置gamma_weight表达式

命令接受对应选项中相同的语法

如果指定的表达式是无效的，则保持当前值

extractplanes

从输入流分离单独的颜色通道成为灰度视频流

滤镜接受下面选项：

- planes

设置要提取的通道

接受下面的值（标识通道）：

'y' 'u' 'v' 'a' 'r' 'g' 'b'

选择无效的值会产生错误。这也意味着同时你只能选择 `r, g, b`和`y`，或者 `y, u, v`

extractplanes例子

- 提取亮度和 `u` 和 `v` 颜色分量到3个灰度输出。

```
ffmpeg -i video.avi -filter_complex 'extractplanes=y+u+v[y][u][v]' -map '[y]' y.avi -map '[u]' u.avi -map '[v]' v.avi
```

elbg

应用多色调分色印效果，使用了ELBG(增强型LBG)算法。（构建颜色模板）

对每个输入图像，滤镜会对于给定编码长度计算最优的从输入到输出的映射，它们对应于不同的输出颜色的数量

滤镜接受下面的选项：

- `codebook_length, l`

设置编码长度,值必须是正整数，代表不同的输出颜色数量，默认为256

- `nb_steps, n`

设置计算最优映射的最大迭代数。值越高，结果越好，但越耗时。默认为1

- `seed, s`

设置一个随机种子，必须是0-UINT32_MAX间的整数，如果不指定，或者设置为-1，则会自动选择一个好的随机值

fade

应用淡入/淡出

它接受下面参数：

- `type, t`

指定类型是 `in` 代表淡入，`out` 代表淡出，默认为 `in`

- `start_frame, s`

指定应用效果的开始时间，默认为0.

- `nb_frames, n`

应用效果的最后一帧序数。对于淡入，在此帧后将以本身的视频输出，对于淡出此帧后将以设定的颜色输出，默认25.

- `alpha`

如果设置为1，则只在透明通道实施效果（如果只存在一个输入），默认为0

- `start_time, st`

指定按秒的开始时间戳来应用效果。如果 `start_frame` 和 `start_time` 都被设置，则效果会在更后的时间开始，默认为0

- `duration, d`

按秒的效果持续时间。对于淡入，在此时后将以本身的视频输出，对于淡出此时后将以设定的颜色输出。如果 `duration` 和 `nb_frames` 同时被设置，将采用 `duration` 值。默认为0（此时采用 `nb_frames` 作为默认）

- `color, c`

设置淡化后（淡入前）的颜色，默认为"black".

fade例子

- 30帧开始淡入

```
fade=in:0:30
```

- 等效上面

```
fade=t=in:s=0:n=30
```

- 在200帧视频中从最后45帧淡出

```
fade=out:155:45 fade=type=out:start_frame=155:nb_frames=45
```

- 对1000帧的视频25帧淡入，最后25帧淡出：

```
fade=in:0:25, fade=out:975:25
```

- 让前5帧为黄色，然后在5-24淡入：

```
fade=in:5:20:color=yellow
```

- 仅在透明通道的第25开始淡入

```
fade=in:0:25:alpha=1
```

- 设置5.5秒的黑场，然后开始0.5秒的淡入：

```
fade=t=in:st=5.5:d=0.5
```

fftfilt

在频域内应用任意表达式于样品

- dc_Y

调整亮度dc值（增益），范围0-1000，默认为0

- dc_U

调整色度第1分量dc值（增益），范围0-1000，默认为0

- dc_V

调整色度第2分量dc值（增益），范围0-1000，默认为0

- weight_Y

设置对于亮度的频域权重表达式

- weight_U

设置对于色度第1分量的频域权重表达式

- weight_V

设置对于色度第2分量的频域权重表达式

滤镜接受下面的变量:

- X
- Y

对应当前样本点的坐标

- W
- H

当前图像的宽和高

fftfilt例子

- 高通:

```
fftfilt=dc_Y=128:weight_Y='squish(1-(Y+X)/100)'
```

- 低通:

```
fftfilt=dc_Y=0:weight_Y='squish((Y+X)/100-1)'
```

- 锐化:

```
fftfilt=dc_Y=0:weight_Y='1+squish(1-(Y+X)/100)'
```

field

使用步算法从隔行图像中提取单个场来避免浪费CPU时间。标记为逐行输出帧。

滤镜接受下面选项：

- type

指定是 `top`（或者0）还是 `bottom`（或者1）类型的场

fieldmatch

场匹配用于反转电视（隔行）电影。它为了从电视流中建立起逐帧电影，需要过滤保留部分重复帧，为了更好的对压缩转换电视 `fieldmatch` 滤镜需要跟随一个 `decimate` 滤镜之类的抽取滤镜。

为了更好的分离和抽取，需要在两个滤镜间插入一个反交错滤镜。如果源是混合了电视电影和现实的内容，则单独的 `fieldmatch` 滤镜不足以分离出交错内容，所以需要一个诸如 `yadif` 之类的反交错滤镜来进一步标记剩余的交错内容，以便于后面的抽取。

除了各种配置选项，`fieldmatch` 可以通过 `ppsrc` 可选项启用第二个流。如果被允许，将基于第二个流进行帧的重建。它允许第一个流作为预处理来帮助各种滤镜算法实现无损输出（减少能正确匹配）。通常一个 `field-aware` 降噪，或者亮度/对比度调整可以实现这一的帮助。

注意滤镜使用如TIVTC/TFM（AviSynth项目）和VIVTC/VFM（VapourSynth项目）的相同算法。`fieldmatch` 有一点克隆TFM的意味，除了一些行为和选项不同外，语义和用法很接近。

当前 `decimate` 滤镜仅工作在固定帧率视频。在输入视频是混合了电视电影和逐场内容且是变帧率时不能使用 `fieldmatch` 和 `decimate`

滤镜接受下面的选项：

order

指定输入流的场序。可用值：

`'auto'`

自动探测（采用FFmpeg内部校验值）。

`'bff'`

设置为下场优先

`'tff'`

设置为上场优先

****注意****不要太信任流的宣称值（即需要尝试探测）

默认为`auto`。

mode

设置匹配模式或采用的策略。`pc`模式是最安全的，不会产生抖动，但其对于错误编辑或者混合会被输出，而实际有更好的匹配结果（即检测不出最好匹配结果）

更多关于p/c/n/u/b 的有效性见 p/c/n/u/b 意义部分。

有效值有：

`'pc'`

2路匹配（p/c）

`'pc_n'`

2路匹配，并尝试第3路（p/c + n）

`'pc_u'`

2路匹配，并尝试第3路（同`order`）对应（p/c + u）

`'pc_n_ub'`

2路匹配，并尝试第3路，并可尝试第4/第5匹配(p/c + n + u/b)

`'pcn'`

3路匹配（p/c/n）

`'pcn_ub'`

3路匹配，并尝试第4/第5匹配（p/c/n + u/b）

括号中的模式匹配是假设order=tff（场序是自动或者上场优先）。

pc模式最快，而pcn_ub则最慢

默认为pc_n。

ppsrc

标记主要输入作为预处理输入，而且允许第二个输入流作为干净源。参考滤镜介绍以了解更多详细信息。它类似于VFM/TFM 的clip2特性。

默认为0（表示禁止）。

field

设置场序，它建议设置为同`order`，除非你尝试的结果是失败。在某些情况下改变设置可以产生很大的匹配性能影响，可用值有：

`'auto'`

自动（同于`order`中的介绍）。

'bottom'

下场优先

'top'

上场优先

默认为auto。

mchroma

设置色度信号是否包含在比较判断中。大多数情况下建议不专门设置。仅当影片中包含环的色度问题如有大片的彩虹或者其他工件时才该设置为0。设置为0也可以

默认为1。

y0 y1

这些定义用来明确一个范围，以除去（忽略）字幕、台标或者其他可能干扰匹配的东西。其中y0设置扫描的开始行，y1设置扫描的结束行（包括y0和y1，之外的

scthresh

设置在亮度上场景变化最大百分比数，好的值范围为[8.0, 14.0]，场景变化检测只是在combmatch=sc有效。可用值范围[0.0, 100.0]。

默认为12.0。

combmatch

当设置为非`none`，`fieldmatch`会在更多的情况中梳理出合适的结果。有效值：

'none'

梳理多种可能但没有最优结果

'sc'

在采用了场景改变检测基础上梳理结果

'full'

全时梳理结果

默认为sc。

combdbg

强制`fieldmatch`对某些指标匹配并输出。这个设置在TFM/VFM中被称为`micout`，有效值有：

'none'

没有强制

'pcn'

强制p/c/n

'pcnub'

强制p/c/n/u/b。

默认none。

cthresh

用于控制梳理检测的阈值。它实质上控制“强烈”或“可见”的交错必须被检测到。大的值意味着必须有更多差异才能被检测到，小的值则允许很少的交错（量）就会被检测到。

默认为9。

chroma

决定是否把色度检测包含在匹配模式中。如果源中存在色度问题（彩虹）则需要禁用。实际上设置`chroma=0`通常就足够可靠了，除非你确实源中包含了色度问题。

默认为0。

blockx blocky

设置检测窗口的x和y轴尺寸。它协同`combpel`领域像素一起来声明梳理框架。参考`combpel`介绍了解更多信息。可能值是2的幂数，从4到512。

默认为16。

combpel

在梳理检测窗口中梳理像素的数量。虽然`cthresh`控制了必须是“可见”的才被梳理（即精度），而这个参数控制了局部（检测窗口）中有“多少”则被检出，最小值是1。

默认为80。

p/c/n/u/b的意义

p/c/n

我们假定有下如下的电视电影流：

```
Top fields:      1 2 2 3 4
Bottom fields:  1 2 3 4 4
```

对应的数字对应需处理的场。在这里前2帧是需要的，3和4则被梳理，等等

当 `fieldmatch` 被配置运行在 `field=bottom` 时，输入流被如下传输：

输入流：

T	1	2	2	3	4
B	1	2	3	4	4

<-- 匹配参考

匹配检测： c c n n c

输出流：

T	1	2	3	4	4
B	1	2	3	4	4

作为一个场匹配的结果，我们可以看到一些帧被复制了。为了完整实现电视电影的逆转换，你需要依靠后续的滤镜去除掉重复的帧。参考 `decimate` 滤镜。

如果相同的处理但配置为 `field=top`，结果会：

```
输入流：
  T      1 2 2 3 4      <-- 匹配参考
  B      1 2 3 4 4

匹配检测： c c p p c

输出流：
  T      1 2 2 3 4
  B      1 2 2 3 4
```

在这些例子里，我们可以看出 `p`、`c` 和 `n` 的意义。基本上，它们指出参考帧的位置（关系）

- `p`，匹配于前一帧
- `c`，匹配于当前帧
- `n`，匹配于下一帧

u/b

这里的 `u` 和 `b` 是在匹配帧基础上的位级描述。在下面的例子中，我们假定当前匹配了2个帧（`Top:2 ,bottom:2`），根据匹配，一个 `x` 是在每个匹配场景的上方和下方：

对于 `field=bottom` 的匹配

匹配：	c	p	n	b	u
	x	x	x	x	x
Top	1 2 2	1 2 2	1 2 2	1 2 2	1 2 2
Bottom	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3
	x	x	x	x	x
输出帧：					
	2	1	2	2	2
	2	2	2	1	3

对于 `field=top` 的匹配

匹配：	c	p	n	b	u
	x	x	x	x	x
Top	1 2 2	1 2 2	1 2 2	1 2 2	1 2 2
Bottom	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3
	x	x	x	x	x
输出帧：					
	2	2	2	1	2
	2	1	3	2	2

fieldmatch例子

- 简单的IVTC 有上场优先的视频电影流：

```
fieldmatch=order=tff:combmatch=none, decimate
```

- 高级IVTC，由后续 `yadif` 继续处理：

```
fieldmatch=order=tff:combmatch=full, yadif=deint=interlaced, decimate
```

fieldorder

改变输入的场序

接受下面的参数：

- order

输出场序。有效值有 `tff` 对应的上场优先和 `bff` 对应的下场优先

默认是 `tff`

转换是将图像内容向上或者向下一行，并填充其余部分以符合图像内容。该方法适合大多数广播电视的场序转换

如果输入视频没有标记为交错，或者已经标记为所需场序，则滤镜不会改变输入视频

它常用于转换到PAL DV格式，它是下场优先的。

例如：

```
ffmpeg -i in.vob -vf "fieldorder=bff" out.dv
```

fifo

缓冲输入并在需要时送出

它常用于 `libavfilter` 的自动插入（保证一些连接有效）

它没有参数

find_rect

找到一个矩形对象

它接受下面的选项：

- object

对象图像的文件路径，需要是gray8.

- threshold

探测阈，默认0.5.

- mipmaps

最小的贴图，默认为3.

- xmin, ymin, xmax, ymax

指定矩形对象检测的范围（xmin,ymin）(xmax,ymax)定义的一个矩形中

find_rect例子

- 对视频生产一个调色板视频

```
ffmpeg -i file.ts -vf find_rect=newref.pgm,cover_rect=cover.jpg:mode=cover new.mkv
```

cover_rect

覆盖一个矩形对象

它接受下面的选项：

- cover

用作覆盖的图像路径，需要是yuv420.

- mode

设置覆盖模式

接受下面的值：

‘cover’

蒙在表面

‘blur’

同周边插值来覆盖

默认blur.

cover_rect例子

- 对视频生产一个调色板视频

```
ffmpeg -i file.ts -vf find_rect=newref.pgm,cover_rect=cover.jpg:mode=cover new.mkv
```

format

转换输入视频为指定的像素格式。libavfilter 尝试为下一个滤镜输入选择一个合适的输出（而自动采用）。

它接受下面的参数：

- pix_fmts

一个用 | 分隔的像素格式名列表，例如"pix_fmts=yuv420p|monow|rgb24".

format例子

- 转换输出为 yuv420p 格式

```
format=pix_fmts=yuv420p
```

转换输入到列表的任何格式之一

```
format=pix_fmts=yuv420p|yuv444p|yuv410p
```

fps

通过复制或丢弃帧来把帧率调整为一个近似固定值

它接受下面的参数：

- fps

目标帧率，默认25

- round

舍入方法.

可能值:

zero

对0舍入（去小数），绝对值最小

inf

从0开始的圆

down

向负无穷舍入（去除小数）

up

向正无穷舍入（见小数加1）

near

舍入到近似值（命名的频率）

默认为near.

- start_time

假设第一个 PTS 是一个按秒的给定值。它允许填补/去除流的开始。默认没有需要填补/去除。例如设置为0，将会在视频流后于音频流时在前面添加黑帧，否则去除早于音频流的负帧。

另外，选项可以指定为一个平面字符串形式：fps[:round]

参考 [setpts](#) 滤镜

fps例子

- 输出25帧频

```
fps=25
```

- 输出24帧频，使用了帧频缩写名和舍入方法为最接近

```
fps=film:round=24
```

framepack

包两个不同的视频流到立体视频,设置适当的元数据支持的编解码器。两个视角视频需要有相同的尺寸与帧频以及以短的視頻为停止时间。注意你可能需要预先通过 `scale` 和 `fps` 调整。

接受下面的参数：

- format

设置包装格式，支持:

sbs

一个视图左另一个在右(默认)。

tab

一个视图在上，一个视图在下

lines

视图按线交错（按行）

columns

视图按列交错

frameseq

视图都暂时交错

一些例子：

- 把左/右视图以 frameseq 模式打包成立体视频

```
ffmpeg -i LEFT -i RIGHT -filter_complex framepack=frameseq OUTPUT
```

- 把视图以 sbs 模式交错，还进行了放缩预处理

```
ffmpeg -i LEFT -i RIGHT -filter_complex [0:v]scale=w=iw/2[left],[1:v]scale=w=iw/2[right],[left][right]framepack=sbs OUTPUT
```

framestep

每 `N` 帧选择1帧

它接受下面选项：

- `step`

设置间隔的 `N` 值。必须大于0，默认为1（这样相当于不处理，完全输出）

frei0r

对视频采用 `frei0r` 效果

编译配置参数 `--enable-frei0r`

接受下面的参数：

- `filter_name`

设置加载的 `frei0r` 效果名。如果环境变量 `FREI0R_PATH` 被定义，则将在其所指目录搜索。`FREI0R_PATH` 是有 `,` 分隔的多个路径。通常 `frei0r` 的搜索路径是: `HOME/.frei0r-1/lib/`, `/usr/local/lib/frei0r-1/`, `/usr/lib/frei0r-1/`。

- `filter_params`

一个由 `|` 分隔的参数列表来传递给 `frei0r` 效果

一个 `frei0r` 效果的参数是布尔值（为 `y` 或者 `n`）、双精度数、颜色值（以R/G/B形式描述，其中R、G和B是0.0-1.0间的浮点数）或者在 `颜色/color` 中定义的颜色名、位置量（以X/Y形式描述，X和Y均是浮点数）和/或 字符串

参数的数量和类型要根据加载的效果，如果一个效果参数没有指定则选用默认设置

frei0r例子

- 采用 `distort0r` 效果，参数有2个值。

```
frei0r=filter_name=distort0r:filter_params=0.5|0.01
```

- 应用 `colordistance` 效果，有一个颜色值作为参数（下面3个形式等效）

```
frei0r=colordistance:0.2|0.3|0.4 frei0r=colordistance:violet frei0r=colordistance:0x112233
```

- 应用 `perspective` 效果，指定了图像的左上和右上位置

```
frei0r=perspective:0.2|0.2|0.8|0.2
```

关于 `frei0r` 的更多信息参考<http://frei0r.dyne.org>

fspp

应用快速简单的后处理，这是 `spp` 滤镜的快速版本

它分离 (I)DCT为水平/垂直 来传递，不同于简单的 `post` 滤镜，其中每个块执行一次，而不是每个像素，则允许更快的速度。

滤镜接受下面的选项：

- quality

设置品质水平，它定义了平均水平数，接受4-5的整数，默认为4

- qp

强制包含一个不断量化参数，它接受0-63间的整数，如果不设定，滤镜会使用视频流的 qp （如果有效）

- strength

设置滤镜强度。它接受-15-30间的整数。越低表示更多细节但需要更多工作，高的值则图像平滑（模糊）也更快，默认值为0——PSNR最佳

- use_bframe_qp

为1则允许从B帧使用QP。使用它在大的QP时可能导致B帧闪烁。默认为0（不允许）

geq

滤镜接受下面的选项：

- lum_expr, lum

设置亮度表达式

- cb_expr, cb

设置色度分量中蓝色表达式

- cr_expr, cr

设置色度分量中红色表达式

- alpha_expr, a

设置透明通道表达式Set the alpha expression.

- red_expr, r

设置红色表达式

- green_expr, g

设置绿色表达式

- blue_expr, b

设置蓝色表达式

根据指定的选项来确定颜色空间。如果 lum_expr , cb_expr , 或者 cr_expr 中的一个被定义，则滤镜自动选择 YCbCr 颜色空间，如果 red_expr , green_expr , 或 blue_expr 中有一个被定义则选择 RGB 颜色空间

如果其中一个颜色分量选项没有被定义，则它等于前一个谷底值。如果 alpha_expr 没有被定义则认为是不透明的。如果没有任何颜色分量被定义，它将只计算亮度表达式

表达式接受下面变量和函数:

- N

帧序数，从0开始计数 from 0.

- X
- Y

当前样本坐标

- W
- H

图像宽和高

- SW
- SH

依赖当前滤镜的放缩宽和高。它根据当前像素亮度数和当前平面的比例。例如对于YUV4:2:0给我饿死，这个值是1,1对应于亮度还有0.5，0.5 的颜色分量

- T

按秒当前帧时间

- p(x, y)

返回当前帧平面 (x,y) 点的像素值

- lum(x, y)

返回当前帧平面 (x,y) 点的像素亮度值

- cb(x, y)

返回当前帧平面 (x,y) 点的像素色度分量差蓝色值,0表示没有该分量

- cr(x, y)

返回当前帧平面 (x,y) 点的像素色度分量差红色值,0表示没有该分量

- r(x, y)
- g(x, y)
- b(x, y)

返回当前帧平面 (x,y) 点的像素红/绿/蓝值，为0表示没有该颜色

- alpha(x, y)

返回当前帧平面 (x,y) 点的像素透明通道值，为0表示没有该值

对于函数，如果x和y超出了范围，则值自动由影片边缘值代替

geq例子

- 水平翻转图像 $geq=p(W-X\backslash,Y)$
- 生成一个二维的正弦波,角 $\pi/3$ 和100像素的波长:

```
geq=128 + 100sin(2(PI/100)(cos(PI/3)(X-50T) + sin(PI/3)Y)):128:128
```

- 生成一个花哨的神秘的光:

```
nullsrc=s=256x256,geq=random(1)/hypot(X-cos(N0.07)W/2-W/2,Y-sin(N0.09)H/2-H/2)^21000000sin(N*0.02):128:128
```

- 生成一个快速浮雕效果:

```
format=gray,geq=lum_expr='(p(X,Y)+(256-p(X-4,Y-4)))/2'
```

- 根据像素的位置修改RGB分量:

```
geq=r='X/Wr(X,Y)':g='(1-X/W)g(X,Y)':b='(H-Y)/H*b(X,Y)'
```

- 创建一个径向渐变,是相同的大小作为输入(也见 [vignette](#) 滤镜):

```
geq=lum=255gauss((X/W-0.5)3)gauss((Y/H-0.5)3)/gauss(0)/gauss(0),format=gray
```

- 创建一个线性渐变使用作为另一个滤镜的蒙版,然后用叠加组成。在本例中,视频会从底部到顶部的y轴定义的线性梯度逐渐变得更加模糊:

```
ffmpeg -i input.mp4 -filter_complex "geq=lum=255*(Y/H),format=gray[grad];[0:v]boxblur=4[blur];[blur]
[grad]alphamerge[alpha];[0:v][alpha]overlay" output.mp4
```

gradfun

解决条带效果,这是由于对于8位颜色深度有时会被就近截断成平面(化),通过插入渐变来柔化它们

这个滤镜仅用于回放。在有损压缩前也不要使用它,因为压缩本身就会损伤细节(渐变)而成为条带

它接受下面的参数:

- strength

滤镜对任何像素最大的改变值。这也是检测(颜色)平坦区域的阈值。取值范围是.51 至64,默认为1.2.超出范围的值会被被裁减为有效值

- radius

指定合适的修正梯度。一个大的 radius 值会产生更平滑的过渡,也防止滤镜修改处理区域附近的像素。接受的范围为8-32,默认为16,超出范围的值会被裁减以符合

另外,选项可以采用平面字符串的形式指定: `strength[:radius]`

gradfun例子

- 以3.5的 strength 和8的 radius 值应用滤镜:

```
gradfun=3.5:8
```

- 指定 radius, 省略 strength (会采用默认值为1.2):

```
gradfun=radius=8
```

haldclut

对视频流采用 `Hald CLUT`

第一个输入是要处理的视频，第二个则是 `Hald CLUT`，这个 `Hald CLUT` 输入可以是一张简单的图片或者复合视频信号

滤镜接受下述选项：

- `shortest`

强制以最短输入来总作为整个数出。默认为0

- `repeatlast`

在结束后继续以 `CLUT` 处理。值为0则禁止。默认为1.

`haldclut` 也有类似 `lut3d` 相同的选项（两个滤镜共享相同的内部结构）。

关于 `Hald CLUT` 可以通过 [Eskil Steenberg](http://www.quelsolaar.com/technology/clut.html)（`Hald CLUT`的作者）的网站，在<http://www.quelsolaar.com/technology/clut.html>

haldclut工作流程例子

Hald CLUT视频流

- 生成一个 `Hald CLUT` 的流，流改变各种效果。

```
ffmpeg -f lavfi -i haldclutsrc=8 -vf "hue=H=2Pi*t:s=sin(2Pi*t)+1, curves=cross_process" -t 10 -c:v ffv1 clut.nut
```

注意：确认你选用的是无损编码

- 然后把滤镜（随 `haldclut`）应用在随机流。

```
ffmpeg -f lavfi -i mandelbrot -i clut.nut -filter_complex '[0][1] haldclut' -t 20 mandelclut.mkv
```

这里 `Hald CLUT` 被用于前10秒（持续时间由`clut.nut`定义）。然后其最后的CLUT图片应用到继续的的 `mandelbrot` 流上

带预览的Hald CLUT

一个 `Hald CLUT` 支持作为有多层（`Level*Level*Level`）像素的多层（`Level*Level*Level`）图像。对于一个给定的 `Hald CLUT`，`FFmpeg`尽可能在图像的左上开始选择最大可能，将选择尽可能最大的广场在图片的左上角开始。剩下的填充像素(底部或右)将被忽略。这个区域可以用来添加一个预览 `Hald CLUT`。

通常，下面会利用 `haldclutsrc` 生成一个支持 `haldclut` 滤镜的 `Hald CLUT` 图：

```
ffmpeg -f lavfi -i haldclutsrc=8 -vf "
    pad=iw+320 [padded_clut];
    smptebars=s=320x256, split [a][b];
    [padded_clut][a] overlay=W-320:h, curves=color_negative [main];
    [main][b] overlay=W-320" -frames:v 1 clut.png
```

它包含原始和CLUT的预览效果：SMPTE颜色条被显示在右上，其下显示相同的颜色处理的颜色变化

然后,这Hald CLUT效果可以可视化：

```
ffplay input.mkv -vf "movie=clut.png, [in] haldclut"
```

hflip

水平翻转输入视频

例如利用ffmpeg水平翻转输入视频：

```
ffmpeg -i in.avi -vf "hflip" out.avi
```

histeq

这个过滤器适用于每帧的基础上的全局颜色直方图均衡化

它被用于产生压缩了像素强度的正确视频。滤镜在强度范围内重新分配像素强度分布。它可被视为“自动调整对比度滤镜”。滤镜只适用于纠正退化或者较差质量的视频采集

接受下面的选项：

- strength
确定的数量均衡。随着参数的降低，像素强度的分布在输入帧中越来越多。值为浮点数，范围为[0,1]，默认0.200.
- intensity
设置在生成的输出中最大可能强度。 strength 设置表面了期望，而 intensity 的设置强调了限制，从而避免了出现错误。值为浮点数，范围为[0,1]，默认0.210.
- antibanding
设置antibanding级别。如果启用，滤镜将通过随机小批量改变输出像素的亮度直方图避免产生条带。允许的值有 none，weak 或 strong，默认为 none。

histogram

对输入视频计算并绘制一个颜色分布直方图

它计算的直方图代表了各种颜色分量在图像中的分布情况。

滤镜接受下面的选项：

- mode
设置直方图模式。
有下面的可能值：
'levels'

显示图像颜色分量的标准直方图。它显示每个颜色分量图形。依据输入视频可以显示当前帧的`Y, U, V, A`或者`R, G, B`分量图形。其下是每个图

'color'

在二维图（这被称为向量监视器）中显示色度通道值（U / V颜色位置）。像素矢量表示亮度，每个点对应代表输入中的各个像素（都有其色度分量值）

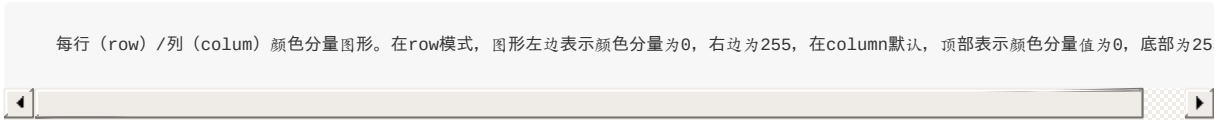
图中白色像素的位置对应一个像素的色度值输入。因此该图可以用来了解色相（颜色味道）和饱和度（颜色）的情况其反映了原始图像的主导色调。颜色的色



'color2'

类似color以矢量监视器显示，不过增加了实际色度值的显示。

'waveform'



默认为 levels 模式

- level_height

在 levels 模式中设置图形高，默认200，允许[50, 2048].

- scale_height

在 levels 设置颜色放缩高，默认12，允许[0, 40].

- step

对 waveform 模式设置步长。小的值用于更多了解在相同亮度下颜色分布情况，默认10，允许 [1, 255] 。

- waveform_mode

对 waveform 设置 row 或者 column ，默认 row

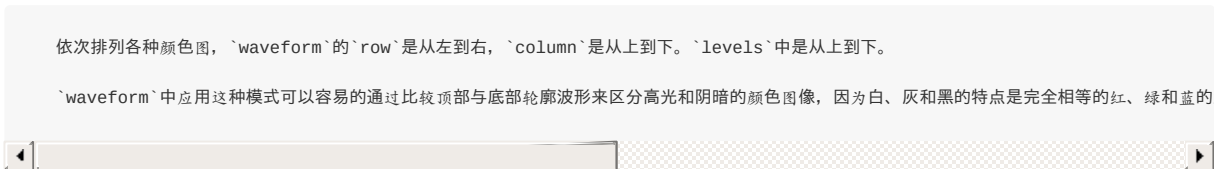
- waveform_mirror

对 waveform 设置镜像模式，0表示不镜像（默认），1表示镜像。在镜像模式中，对 row 高值在左边，对 column 高值在上面

- display_mode

设置 waveform 和 levels 的显示模式，它接受:

'parade'



'overlay'

除了表示颜色的图形组件直接叠加在一起外，与`parade`模式展示的信息相同
在`waveform`中这样的显示模式更易于发现颜色的相对差异或相似。因为重叠区域的分量应该是相同的，如中性的白色、灰和黑

默认为 parade

- `levels_mode`

对 `levels` 设置模式，可以是 `linear` 或 `logarithmic`，默认 `linear`

histogram例子

- 计算并绘制柱状图：

```
ffplay -i input -vf histogram
```

hqdn3d

这是一个高精度/质量的3D降噪滤镜。它的目的是减少图像噪声,产生平滑的图像和让静止图像保存原样。它可以提高压缩率。

接受下面可选参数:

`luma_spatial`

非负浮点数来指明亮度强度。默认为 `4.0`

`chroma_spatial`

非负浮点数来指明亮色强度，默认为 ``3.0*luma_spatial/4.0``。

`luma_tmp`

一个浮点数指明亮度临时强度。默认为 ``6.0*luma_spatial/4.0``

`chroma_tmp`

一个浮点数指明色度临时强度。默认为 ``luma_tmp*chroma_spatial/luma_spatial``

hqx

应用一个高质量的像素放大滤镜。这个滤镜最初由 Maxim Stepin 创建。

它接受下面的选项：

- `n`

设置缩放尺度。2 对应hq2x, 3 对应hq3x, 4对应hq4x，默认为3。

hue

编辑或者设定颜色的饱和度

接受下面的参数:

- `h`

指定色度角的度数，接受表达式，默认为0

- s

指定饱和度，范围[-10,10]，接受表达式，默认为"1".

- H

指定色调角的弧度，接受表达式，默认为"0".

- b

指定亮度，范围[-10,10]。接受表达式，默认为"0".

`h` 和 `H` 互斥，不能同时设定

其中 `b` , `h` , `H` 和 `s` 表达式允许下面内容:

- n

从0开始的帧序数

- pts

按时间基单位的输入帧时间戳

- r

输入视频帧率，如果无效则为 `NAN`

- t

按秒的时间码，如果无效则为 `NAN`

- tb

输入视频时基

hue例子

- 设置色度角90度，饱和度为1:

```
hue=h=90:s=1
```

- 同上，但以色度角弧度值进行设置:

```
hue=H=PI/2:s=1
```

- 旋转色相,以及让饱和度在0-2间变化:

```
hue="H=2PI/t: s=sin(2PI/t)+1"
```

- 从0开始应用一个3秒饱和度淡入效果:

```
hue="s=min(t/3,1)"
```

- 一般淡入表达式可以为:

```
hue="s=min(0\, max((t-START)/DURATION\, 1))"
```

- 从5秒开始的饱和度淡出:

```
hue="s=max(0\, min(1\, (8-t)/3))"
```

- 一般淡出表达式为:

```
hue="s=max(0\, min(1\, (START+DURATION-t)/DURATION))"
```

hue命令

滤镜还支持下面的命令：

- b
- s
- h
- H

它们分别编辑色度 和/或 饱和度 和/或 亮度。命令接受对应选项一样的语法。

如果指定的表达式是无效的，则采用当前值（不变化）

idet

检测视频交错类型。

这个滤镜试图检测如果输入帧交错,逐行,顶部或底部优先（对交错视频）。它还将尝试检测相邻帧之间的字段重复(电视电影的标志)。

单帧检测时只考虑当前与相邻帧每一帧类型。多帧检测结合的之前的帧类型历史。

滤镜输出日志有下面的元数据值:

- single.current_frame

对当前帧单帧检测结果，有如下值:“tff” (上场优先), “bff” (下场优先), “progressive” (逐行) , 和“undetermined” (不能检测出)

- single.tff

累积的以单帧检测检测出的上场优先数

- multiple.tff

累积的以多帧检测检测出的上场优先数

- single.bff

累积的以单帧检测检测出的下场优先数

- multiple.current_frame

对当前帧多帧检测结果，有如下值:“tff” (上场优先), “bff” (下场优先), “progressive” (逐行) , 和“undetermined” (不能检测出, 不定)

- multiple.bff

累积的以多帧检测检测出的下场优先数

- single.progressive

累积的以单帧检测检测出的逐行帧数

- multiple.progressive

累积的以多帧检测检测出的逐行帧数

- single.undetermined

累积的以单帧检测检测出的不定帧数

- multiple.undetermined

累积的以多帧检测检测出的不定帧数

- repeated.current_frame

指示当前帧是从最近帧的重复情况,为“neither”(表示不是重复), “top”,或者 “bottom”.

- repeated.neither

累积的 `neither` 重复情况.

- repeated.top

累积的 `top` 重复情况

- repeated.bottom

累积的 `bottom` 重复情况

滤镜接受下面选项:

- intl_thres

设置交错阈值

- prog_thres

设置逐行阈值

- repeat_thres

重复检测阈值

- half_life

设定指定的帧数之后,一个给定的数据帧的贡献是减半（即只占0.5的类型）。默认为0意味着所有的帧永远是有1.0的权重

- analyze_interlaced_flag

当设置为非0数时，`idet` 将使用指定的帧数来确定交错标记是准确的，它不会对不能检测帧计数。如果发现标记是正确的使用它没有进一步的计算，即使发现标记不正确，也只是将它(标记)清除而没有进一步的计算。它插入 `idet` 滤镜作为低计算方法来清除交错标记。

il

反转非交错或者交错

这个滤镜可以让非交错图像变成交错的，把交错图像变成非交错的。非交错的图像被分裂为2部分（称作半图），其中奇数行移到输出图像的上部，偶数行移到下半部分。你可以利用滤镜连续处理2次（相当于没有效果）。

滤镜接受下面的选项：

- `luma_mode, l`
- `chroma_mode, c`
- `alpha_mode, a`

对 `luma_mode`，`chroma_mode` 和 `alpha_mode` 的可能值有：

'none'

什么都不做

'deinterleave, d'

非交错部分，放置在其它上

'interleave, i'

交错部分，反向非交错效果

默认值为 `none`

- `luma_swap, ls`
- `chroma_swap, cs`
- `alpha_swap, as`

交换 luma/chroma/alpha 部分。交换奇数和偶数行。默认0.

interlace

简单的从逐行转交错滤镜。它把奇数帧交错上（或下）行，同时把偶数帧作为下（或上）行。同时减半帧率和保持图像高度。

Original Frame 'j'	Original Frame 'j+1'	New Frame (tff)
=====	=====	=====
Line 0	----->	Frame 'j' Line 0
Line 1	Line 1 ---->	Frame 'j+1' Line 1
Line 2	----->	Frame 'j' Line 2
Line 3	Line 3 ---->	Frame 'j+1' Line 3
...

新的帧 + 1 会以帧'j+2'和帧'j+3'这样依次生成

它接受下面的选项参数:

- scan

它决定了交错帧模式是 even (tff - 默认) 还是 odd (bff)

- lowpass

允许(默认)或禁止垂直低通滤波器来避免 `twitter` 交错和减少波纹。

kerndeint

通过Donald Graft的自适应内核deinterling反交错视频。其使交错视频转换成逐行视频

下面是接受参数的介绍.

- thresh

设置一个滤镜阈值用于确定哪些像素会被处理。值为[0,255]间整数，默认为10.如果为0则处理所有像素

- map

设置如果超过阈值的处理模式，为1则为白色，默认为0

- order

设置字段顺序。如果为1则交换字段，否则为0，默认为0.

- sharp

为1则附加锐化，默认为0.

- twoway

为1则尽量锐化，默认为0

kerndeint例子

- 以默认值应用:

`kerndeint=thresh=10:map=0:order=0:sharp=0:twoway=0`

- 允许附加锐化:

`kerndeint=sharp=1`

- 超阈值后刷成白色

`kerndeint=map=1`

lenscorrection

径向透镜畸变修正

这个滤镜可以用来修正广角镜头产生的径向畸变，从而修正图像。要找到合适的参数，可以使用工具，例如 `opencv` 或者简单的多次试错尝试。利用 `opencv` 源码中的校准样例（在 `samples/cpp`）并且从结果矩阵中提取 `k1` 和 `k2` 系数。

注意相同效果滤镜在开源KDE项目工具 `Krita` 和 `Digikam` 中同样是有用的。

这个滤镜还可以同 `vignette` 滤镜联合使用，来补偿透镜错误，它修正滤镜处理图像的失真，而 `vignette` 滤镜纠正亮度分布，所以你可能需要同时使用两个滤镜，不过你还要注意点二者的顺序，即到底是哪个滤镜先使用

lenscorrection选项

滤镜接受下面选项：

- `cx`

图像的焦点相对x坐标,从而扭曲的中心。这个值区间[0,1],用分数表示图像的宽度比

- `cy`

图像的焦点相对y坐标,从而扭曲的中心。这个值区间[0,1],用分数表示图像的高度比

- `k1`

二次修正系数。0.5意味着没有修正

- `k2`

双二次修正系数。0.5意味着没有修正。

生成修正的公式:

```
r_src = r_tgt * (1 + k1 * (r_tgt / r_0)^2 + k2 * (r_tgt / r_0)^4)
```

这里 `r_0` 是减半的图像对角，`r_src` 和 `r_tgt` 分别是源和目标图像中相对于焦点的距离。

lut3d

对视频应用3D LUT滤镜。

滤镜接受下面的选项：

- `file`

设置3D LUT文件名，该文件支持的类型有：

'3dl'

AfterEffects 的类型

'cube'

Iridas 的类型

‘dat’

DaVinci 的类型

‘m3d’

Pandora 的类型

- interp

选择插值模式，有效值有：

‘nearest’
选用离定义点最近的

‘trilinear’
采用8点多维设置来定义插入值

‘tetrahedral’
使用一个四面体插入值

lut, lutrgb, lutyuv

根据每个像素的分量数据查表选择输出值的滤镜。

其中 `lutyuv` 应用于 `YUV` 输入视频，而 `lutrgb` 应用于 `RGB` 输入视频

滤镜接受下面的参数：

- c0
设置第一个像素分量表达式
- c1
设置第二个像素分量表达式set second pixel component expression
- c2
设置第三个像素分量表达式
- c3
设置第四个像素分量表达式
- r
设置红色分量表达式
- g
设置绿色分量表达式
- b

设置蓝色分量表达式

- a

设置透明分量表达式

- y

设置亮度 (Y) 分量表达式

- u

设置色度U/Cb分量表达式

- v

设置色度V/Cr分量表达式

它们每个人都指定使用的表达式计算像素对应分量（查找表方法）。

具体的分量关联到每个 `c*` 选项的输入格式。

其中 `lut` 滤镜允许输入像素格式是 `YUV` 或者 `RGB`，`lutrgb` 只允许 `RGB` 格式，`lutyuv` 只允许 `YUV` 格式

表达式支持下列常量和函数：

- w
- h

输入的宽和高

- val

输入像素分量值

- clipval

输入值，并且裁剪到 `minval-maxval` 范围内

- maxval

输入像素分量的最大值

- minval

输入像素分量的最小值 component.

- negval

“负片”表示的分量值，它被裁剪到 `minval-maxval` 范围内，对应于 `maxval-clipval+minval` .

- clip(val)

对于 `val` 的计算值,裁剪到 `minval-maxval` 范围内

- gammaval(gamma)

伽马校正计算值像素分量的值，裁剪到 `minval-maxval` 范围内，其对应于 `pow((clipval-minval)/(maxval-minval)\, gamma) * (maxval-minval)+minval`

所有表达式默认为 "val"

lut, lutrgb, lutyuv例子

- 输入图像的负片效果

```
lutrgb="r=maxval+minval-val:g=maxval+minval-val:b=maxval+minval-val" lutyuv="y=maxval+minval-val:u=maxval+minval-val:v=maxval+minval-val"
```

等效于:

```
lutrgb="r=negval:g=negval:b=negval" lutyuv="y=negval:u=negval:v=negval"
```

- 亮度负片效果

```
lutyuv=y=negval
```

- 移除色度分量，转换成灰度图像:

```
lutyuv="u=128:v=128"
```

- 应用一个亮度燃烧效果:

```
lutyuv="y=2*val"
```

- 移除绿色和蓝色分量（红色灰度图）:

```
lutrgb="g=0:b=0"
```

- 设定固定的透明通道效果:

```
format=rgba,lutrgb=a="maxval-minval/2"
```

- 以系数0.5进行伽玛亮度校正:

```
lutyuv=y=gammaval(0.5)
```

- 丢弃的亮度低有效位（减少细节，亮块化）:

```
lutyuv=y='bitand(val, 128+64+32)'
```

mergeplanes

从一些视频流中混合颜色通道

滤镜最多接受4路输入流，然后混合选用的(颜色)平面来输出。

滤镜接受下面的选项：

- mapping

设置输入到输出颜色映射，默认为0 mapping. Default is 0.

这个映射由一个位映射指定，它被描述为一个格式为 `0xAa[Bb[Cc[Dd]]]` 的十六进制数。其中 Aa 描述第一个用作输出的

(颜色)平面, `A` 设置采用那个输入流 (0-3), `a` 指定是输入流的那个分量 (0-3, 因为一个输入视频流最多有4个输入分量)。此后的 `Bb`、`Cc` 和 `Dd` 以此类推, 分别指定第2到第4个输出映射关系。

- `format`

设置输出像素格式, 默认为 `yuva444p`

mergeplanes例子

- 从三个灰度视频流混合为单个视频流 (有相同的图像尺寸):

```
[a0][a1][a2]mergeplanes=0x001020:yuva444p
```

- 混合第一路的 `yuv444p` 和第二路的灰度视频到一个 `yuva444p`:

```
[a0][a1]mergeplanes=0x00010210:yuva444p
```

- 在 `yuva444p` 交换Y和A通道:

```
format=yuva444p,mergeplanes=0x03010200:yuva444p
```

- 在 `yuv420p` 交换U和V输出到 `yuv420p`:

```
format=yuv420p,mergeplanes=0x000201:yuv420p
```

- 转换 `rgb24` 到 `yuv444p`

```
format=rgb24,mergeplanes=0x000102:yuv444p
```

mcdeint

应用反交错运动补偿

它每帧需要一个输入字段, 所以必须同 `yadif=1/3` 或者等效 一同使用。

率接受下面的选项:

- `mode`

设置反交错模式, 有下列有效值:

`'fast'` `'medium'` `'slow'`

使用迭代的运动估计

`'extra_slow'`

类似 `'slow'`, 但使用多个参考帧

默认 `'fast'`.

- `parity`

设置输入视频图片字段校验。它必须以下值之一:

'0, tff'

对应上场优先

'1, bff'

对应下场优先

默认 'bff'.

- qp

设置内部使用的编码器按块量化参数设置(QP)

更高的值会导致一个更平滑的运动向量场但不最优个体向量。默认值是1。

mpdecimate

减少对前帧变化不大的帧,以减少帧速率

这个滤镜的主要作用是对非常低码率的编码进行预处理（例如面向拨号调整解调器应用的），不过理论上还可以用于修复发转的电视电影影片（影片转换成电视又转换回来）

下面是选项介绍：

- max

设置连续帧的最大数量,可以删除(如果正),或删除帧之间的最小间隔帧(如果负的)，如果为0，则删除帧和前面删除帧没有关联（关系）

默认为0.

- hi
- lo
- frac

设置删除阈值。

其中 hi 和 lo 是对于一个 8x8 像素块和代表实际像素值差异，所以阈值64对应与每个像素都有1个单位的差异，或者被完全不同的块覆盖

一个帧作为候选（被丢弃）则它没有超过 hi 个不同的 8x8 块和没有超过 frac 量的（1意味着整个图像）超过 lo 个不同的 8x8 块

默认值为 hi 是 64*12，lo 为 64*5，frac 为 0.33

negate

否定输入视频

它接受一个整数（参数），如果非零它否定透明分量(如果可用)。默认值是0

noformat

强制 `libavfilter` 不采用指定的像素格式来输入到下一个滤镜

接受的参数为：

- `pix_fmts`

是一个由 '|' 分隔的像素格式名列表，例如 `pix_fmts=yuv420p|monow|rgb24`

noformat 例子

- 强制不采用 `yuv420p` 的像素格式来输出到 `vflip` 滤镜

```
noformat=pix_fmts=yuv420p,vflip
```

- 转换输入到不是指定的像素格式

```
noformat=yuv420p|yuv444p|yuv410p
```

noise

给视频添加噪点

滤镜接受下面的选项：

- `all_seed`
- `c0_seed`
- `c1_seed`
- `c2_seed`
- `c3_seed`

对指定像素分量或者整个像素设置噪点种子，默认为123457.

- `all_strength, alls`
- `c0_strength, c0s`
- `c1_strength, c1s`
- `c2_strength, c2s`
- `c3_strength, c3s`

对指定像素分量或者整个像素设置噪点强度，默认为0，值范围[0, 100].

- `all_flags, allf`
- `c0_flags, c0f`
- `c1_flags, c1f`
- `c2_flags, c2f`
- `c3_flags, c3f`

设置分量或者所有的标志，可能的标志有：

'a'

平均时间的噪点(平滑)

'p'

混合随机噪点(半)规律

't'

时间噪点（噪点模式在帧间变化）

'u'

统一噪点（gaussian外的）

noise例子

- 添加时间和统一的噪点给图像

```
noise=all=20:allf=t+u
```

null

不改变输入进行输出

OCV

申请使用 `libopencv` 进行视频转换。

要启用需要处理编译参数 `--enable-libopencv`，且系统中要有 `libopencv` 的头和库。

接受下面的参数：

- `filter_name`

指定要使用的 `libopencv` 滤镜名

- `filter_params`

对滤镜指定参数，如果不指定则采用具体滤镜默认参数

在 [libopencv 官方文档](#) 了解更多 `libopencv` 精确信息

在 `libopencv` 中下面的滤镜被支持。

dilate

这个滤镜使用特定结构化的元素来扩张图像，其对应于 `libopencv` 中的函数 `cvDilate`

它接受的参数形式为：`struct_el|nb_iterations`

其中 `struct_el` 代表了一个结构化元素，语法为：`colsxrows+anchor_xanchor_y/shape`，其中 `cols` 和 `rows` 是结构元素的行和列数，`anchor_x` 和 `anchor_y` 是锚点的坐标值，`shape` 是结构化元素的图形，`shape` 可能值是"rect"、"cross"、"ellipse"和"custom"。如果 `custom` 被设置，它还必须跟一个格式为 `=filename` 来指定一个位图，其每个可打印字符对应于一个亮的像素。当定制的图形被使用，`cols` 和 `rows` 被忽略，行和列的数有读取的文件决定。

`struct_el` 的默认值为 `3x3+0x0/rect`

`nb_iterations` 指定迭代的次数，默认为1

一些例子：

- 采用默认值

```
ocv=dilate
```

- Dilate采用了一个5x5的结构元素，且迭代2次

```
ocv=filter_name=dilate:filter_params=5x5+2x2/cross|2
```

- 从文件 `diamond.shape` 读取图像，迭代2次

其中 `diamond.shape` 文件的内容为： ``

```
*
```

```
*
```

因为描述为定制，所以原来指定的行和列值被忽略：

```
ocv=dilate:0x0+2x2/custom=diamond.shape|2
```

```
#### erode ####
```

通过使用特定的结构化元素侵蚀一个图像。它对应于`libopencv`中的`cvErode`函数。

它接受参数为：`struct_el:nb_iterations`，解释同[`dilate`](#dilate)

```
#### smooth ####
```

平滑输入视频

滤镜接受的参数为：`type|param1|param2|param3|param4`

其中`type`是应用的平滑类型，有 "blur", "blur_no_scale", "median", "gaussian", 或者"bilateral".默认为"gaussian"

接下来的参数`param1`，`param2`，`param3`和`param4`依赖于所选的平滑类型，其中`param1`，`param2`接受正整数或0，`param3`和`param4`接受浮点值。

这些参数对应于`libopencv`中的`cvSmooth`函数。

```
### overlay ###
```

把一个视频覆盖在另外一个上面

它有两个输入，其中第一个输入是主要的输入会覆盖到第二个输入上

它接受下面的参数（介绍见下）：

```
- x
- y
```

设置主要视频（在被覆盖视频上）的x和y坐标表达式，默认为0，如果表达式无效则会设置为一个巨大的值（在这种情况下相当于没有覆盖，即覆盖出现在非覆盖区域）

在操作时遇到第二路输入的`EOF`信号时的处理，它接受下面的值之一：

```
repeat
```

重复最后一帧（默认）

```
endall
```

同时结束两个流

```
pass
```

把第一路输入作为输出

```
- eval
```

它设置何时计算x和y坐标值

它接受下面的值：

‘init’

仅在滤镜初始化或命令处理时计算一次

‘frame’

每帧计算

默认‘frame’。

- shortest

如果设置为1，强制以最短的输入时间终止输出，默认为0

- format

设置输出格式，允许：

‘yuv420’

强制为YUV420

‘yuv422’

强制为YUV422

‘yuv444’

强制为YUV444

‘rgb’

强制为RGB

默认 ‘yuv420’。

- rgb（弃用的）

如果设置为1，强制滤镜接受输出是RGB颜色空间。默认为0，它已经被弃用，而是使用格式来代替。

- repeatlast

如果设置为1，强制滤镜绘制最后一个覆盖帧直到结束流。如果设置为0则禁止这样的行为，默认为1

值‘x’和‘y’的表达式中接受下面的参数：

- main_w, W

- main_h, H

主要输入流的宽和高

- overlay_w, w

- overlay_h, h

被覆盖流的宽和高

- x

- y

对于‘x’和‘y’的计算值，它在每帧中都计算

- hsub

- vsub

输出格式中的水平和垂直色度通道分量值，例如对于‘yuv422p’像素格式，‘hsub’是2，‘vsub’是1

- n

帧序数，从0开始计数

- pos

输入帧在文件当中的偏移，如果未知则为‘NAN’

- t

时间码，以秒计。如果时间码未知则为‘NAN’

overlay命令

滤镜支持下面的命令：

- x

- y

修改覆盖输出的‘x’和‘y’坐标。它接受的语法同于前面对应的选项

如果指定的表达式无效，则保持当前值

overlay例子

- 在右下10像素位置绘制主要视频：

```
overlay=main_w-overlay_w-10:main_h-overlay_h-10
```

采用名字选项，则上面的变成：

```
overlay=x=main_w-overlay_w-10:y=main_h-overlay_h-10
```

- 插入一个透明的PNG标记到右下。协同`ffmpeg`工具集利用`-filter_complex`选项来实现：

```
ffmpeg -i input -i logo -filter_complex 'overlay=10:main_h-overlay_h-10' output
```

- 插入2个不同的透明PNG标志，（第二个在右下角）：

```
ffmpeg -i input -i logo1 -i logo2 -filter_complex 'overlay=x=10:y=H-h-10,overlay=x=W-w-10:y=H-h-10' output
```

- 在视频上面添加一个透明颜色层，其中`WxH`指定了输入视频的宽和高：

```
color=color=red@.3:size=WxH [over]; [in][over] overlay [out]
```

- 同时播放原始和过滤版（协同了`deshake`滤镜）：

```
ffplay input.avi -vf 'split[a][b]; [a]pad=iw*2:ih[src]; [b]deshake[filt]; [src][filt]overlay=w'
```

同上效果，但利用命令完成

```
ffplay input.avi -vf 'split[b], pad=iw*2[src], [b]deshake, [src]overlay=w'
```

- 从2秒开始一个从左到右的滑动叠加：

```
overlay=x='if(gte(t,2), -w+(t-2)*20, NAN)':y=0
```

- 组合2个输入，一个放置在一边：

```
ffmpeg -i left.avi -i right.avi -filter_complex "
nullsrc=size=200x100 [background];
[0:v] setpts=PTS-STARTPTS, scale=100x100 [left];
[1:v] setpts=PTS-STARTPTS, scale=100x100 [right];
[background][left] overlay=shortest=1 [background+left];
[background+left][right] overlay=shortest=1:x=100 [left+right]
"
```

- 在10-20秒应用一个`delogo`滤镜

```
ffmpeg -i test.avi -codec:v:0 wmv2 -ar 11025 -b:v 9000k
```

```
-vf '[in]split[split_main][split_delogo];[split_delogo]trim=start=360:end=371,delogo=0:0:640:480[delogoed];[split_main]
masked.avi
```

- 串联多个`overlays`滤镜：

```
nullsrc=s=200x200 [bg];
testsrc=s=100x100, split=4 [in0][in1][in2][in3];
[in0] lutrgb=r=0, [bg] overlay=0:0 [mid0];
[in1] lutrgb=g=0, [mid0] overlay=100:0 [mid1];
[in2] lutrgb=b=0, [mid1] overlay=0:100 [mid2];
[in3] null, [mid2] overlay=100:100 [out0]
```

owdenoise

应用超完备小波降噪

滤镜接受下面的选项：

- depth

设置深度

大的值将在低频部分降噪明显，但速度很慢

值范围8-16，默认为8

- luma_strength, ls

设置亮度强度

为0-1000的双精度值，默认为1.0

- chroma_strength, cs

设置色度强度

为0-1000的双精度值，默认为1.0

pad

在原始输入的`x`和`y`坐标上填充输入图像（多处部分用颜色填充）

它接受下面参数：

width, w
height, h

指定输出尺寸的表达式。如果值为0，则输入图像尺寸作为输出尺寸

在`width`表达式中可以引用`height`值，反之亦然

默认都是0

x
y

指定输入图像在输出中放置的坐标据上边和左边的值

其中`x`可以引用`y`值计算，反之亦然

默认为0.

color

指定添加区域的颜色。语法参考[颜色/color](ffmpeg-doc-cn-07.md#颜色Color)章节的介绍

默认为"black".

对于`width`，`height`，`x`和`y`选项的表达式，可以包含下面的常量：

in_w
in_h

输入宽和高

iw
ih

同于`in_w`和`in_h`

out_w
out_h

输出的宽和高（输出添加的区域），它由`width`和`height`表达式指定

ow
oh

同于`out_w`和`out_h`

x
y

指定的x和y的偏移（在另外一个表达式中），如果不指定则为`NAN`

a

同于`iw / ih`

sar

输入样本点长宽比

dar

输入视频长宽比，它等于`(iw / ih) * sar`

hsub
vsub

水平和垂直色度分量值，例如对`yuv422p`像素格式，`hsub`为2，`vsub`为1

pad例子

- 在 (0, 40) 填充`violet`颜色到输入视频，输出视频为`640x480`

pad=640:480:0:40:violet

其等效于：

pad=width=640:height=480:x=0:y=40:color=violet

- 把图像放置在原始输入扩大3/2倍的衬底中间位置：

pad="3/2*iw:3/2*ih:(ow-iw)/2:(oh-ih)/2"

- 输出到一个正方形衬底上，衬底的变长是输入图像宽和高中的大的一个值，放置在正中：

pad="max(iw\,ih):ow:(ow-iw)/2:(oh-ih)/2"

- 输出成为有16:9的长宽比的衬底上，水平中置图像（其他比例视频转16:9视频，但不拉伸放缩的效果）：

pad="ih*16/9:ih:(ow-iw)/2:(oh-ih)/2"

- 在合成视频的情况下，为了正确设置显示，有必要利用`sar`设置表达式：

```
(ih * X / ih) * sar = output_dar
X = output_dar / sar
```

因此需要修正前例为：

```
pad="ih*16/9/sar:ih:(ow-iw)/2:(oh-ih)/2"
```

- 双倍输入视频尺寸，把输入放置在右下区域（占输出区域的右下1/4）：

```
pad="2*iw:2*ih:ow-iw:oh-ih"
```

```
### palettegen ###
```

对整个视频产生一个调色板

它接受下面的选项：

- max_colors

设置调色板最大数量。****注意****调色板仍包含256个色彩，只是没有用到的色彩都被设置为黑色了

- reserve_transparent

创建一个255色调色板，最后一个存储颜色。GIF优化保留透明的颜色是非常有用的。如果没有设置，那么最大的颜色调色板将达到256。你可能对于某个独立

- stats_mode

设置统计模式

接受下面值：

```
'full'
```

计算全帧直方图

```
'diff'
```

只计算与前一帧的差异部分。这将更关注输入中变化的部分（运动的部分），如果背景是静态的。

默认为full。

```
#### palettegen例子 ####
```

- 生成一个调色板

```
ffmpeg -i input.mkv -vf palettegen palette.png
```

```
### paletteuse ###
```

使用一个调色板来处理输入视频流中的样本点转换

滤镜接受2个输入，一个视频输入流和一个调色板。调色板必须是256个像素的图像（即有256个颜色）

它接受下面选项：

- dither

选择抖动模式，可用算法有：

```
'bayer'
```

顺序8x8 bayer抖动(确定的)

```
'heckbert'
```

由Paul Heckbert定义与1982年的抖动算法（简单的错误扩散）****注意****这个抖动有时被认为是“错误”的，而仅作参考

```
'floyd_steinberg'
```

Floyd 和Steingberg抖动(错误扩散)

```
'sierra2'
```

Frankie Sierra抖动第二版(错误扩散)

```
'sierra2_4a'
```

Frankie Sierra抖动第二版的"简化"（错误扩散）

默认为sierra2_4a。

- bayer_scale

当bayer抖动算法被选取，这个选项将定义用作调色板规模（多少交叉——混合是可见的）。一个小的值意味着更多可见的且更少的条纹，高的值意味更少可见

值范围为[0,5]，默认为2

- diff_mode

如果设置，定义区域过程

`'rectangle'`

只有改变的矩形区域会进行再加工。这类似于GIF裁剪/抵消压缩机制。它用来加快速度，对于只要部分改变时，如用例，只有一个矩形移动（边界）区域。

默认为none。

paletteuse例子

- 利用`palette`（调色板可由[`palettegen`](#palettegen)产生）编码输出GIF

```
ffmpeg -i input.mkv -i palette.png -lavfi paletteuse output.gif
```

perspective

采用正确的视角记录视频，而不是垂直于平面

接受参数的介绍如下：

- x0
- y0
- x1
- y1
- x2
- y2
- x3
- y3

对左上、右上、左下和右下各点设置表达式。默认为`0:0:W:0:0:H:W:H`表示不改变视角。如果场景选项是对源设置（`sense=0`），那么将发送指定点

表达式接受下面的变量：

W
H

视频帧的宽和高

- interpolation

为透视校正设置插值

允许下面的值：

`'linear'`
`'cubic'`

默认为`linear`。

- sense

设置协调选项的解释。

它接受值为：

`'0, source'`

表明前面`x0y0x1y1x2y2x3y3`是对源设置的

`'1, destination'`

表明前面`x0y0x1y1x2y2x3y3`是对目标设置的

默认为`source`。

phase

延迟隔行视频一段时间以便现场秩序变化

用于解决PAL制下电影到视频转换中的场序问题

接受参数介绍见下：

- mode

设置相位方法，它允许的值为：

`'t'`

捕获是上场优先，要转换为下场优先3，滤镜会延迟下场

`'b'`

捕获是下场优先，要转为上场优先，滤镜会延迟上场。

`'p'`

```

    捕获和输出相同场序。这个模式存在文档中引用的其他选项，如果你真的选择它，滤镜将什么也不做。
    'a'

    捕获字段自动确定场序标志，转换则相反。滤镜根据相应标识在逐帧的基础上自动选择' t '和' b '。如果没有包含有效的指示字段，则同于`u`
    'u'

    捕获模式未知或者不定，转换时则相反。滤镜通过分析自动选择 't' 和 'b'实现最佳匹配
    'T'

    捕获是上场优先，转换未知或者不定。滤镜通过图像分析选择`t`和`p`
    'B'

    捕获是下场优先，转换未知或者不定。滤镜通过图像分析选择`b`和`p`
    'A'

    捕获根据标志确定，滤镜由此选择`t`、`b`和`p`，如果没有有效的标志信息，则同于`U`，这是默认模式
    'U'

    捕获和转换都未知，根据检测自动选择`t`、`b`和`p`以使最佳匹配

### pixdescstest ###
像素格式检测测试滤镜，主要用于内部测试。输出视频等于输入视频

例如：

    format=monow, pixdescstest

可以用来测试monowhite像素格式描述是否符合定义

### pp ###
使用指定的`libpostproc`后处理`subfilters`链。这个库会自动选择一个`GPL`编译（--enable-gpl）。`subfilters`必须是由`/`分隔，可以利用`.
滤镜接受下面的选项：

- subfilters

    指定subfilters字符串

所有subfilters有共同选项来确定其范围，它们是：

a/autoq

    对subfilter的质量等级
c/chrom

    同时做色差和亮度(默认)。
y/nochrom

    只做亮度过滤（无色差处理）。
n/noluma

    只做色差过滤（无亮度处理）。

这些选项可以通过`|`附加在`subfilter`名后面

有效的`subfilter`有：

hb/hdeblock[|difference[|flatness]]

    水平解封滤镜

    difference

        差异因素, 高值意味着更多的解封(默认值:32)。
    flatness

        平面度阈值, 降低值意味着更多的解封(默认值:39)。

vb/vdeblock[|difference[|flatness]]

    垂直解封滤镜

    difference

        差异因素, 高值意味着更多的解封(默认值:32)。
    flatness
```

平面度 阈值, 降低值意味着更多的解封 (默认值:39)。

ha/hadepblock[|difference[|flatness]]

准确的水平解封滤镜

difference

差异因素, 高值意味着更多的解封 (默认值:32)。

flatness

平面度 阈值, 降低值意味着更多的解封 (默认值:39)。

va/vadepblock[|difference[|flatness]]

准确的垂直解封滤镜

difference

差异因素, 高值意味着更多的解封 (默认值:32)。

flatness

平面度 阈值, 降低值意味着更多的解封 (默认值:39)。

水平和垂直解封过滤器共享`difference`和`flatness`, 因此不能设置平面度值不同的水平和垂直的阈值。

h1/x1hdeblock

实验的水平解封滤镜

v1/x1vdeblock

实验的垂直解封滤镜

dr/dering

去振铃滤镜

tn/tmpnoise[|threshold1[|threshold2[|threshold3]]], temporal noise reducer

threshold1

larger -> stronger filtering /大->强滤镜

threshold2

larger -> stronger filtering /大->强滤镜

threshold3

larger -> stronger filtering /大->强滤镜

al/autolevels[:f/fullyrange], automatic brightness / contrast correction

f/fullyrange

拉伸亮度到0-255范围

lb/linblenddeint

通过`(1 2 1)`滤镜线性混合`deinterlacing`滤镜的去交错块

li/linipoldeint

每秒通过线性插值`deinterlacing`滤镜的去交错块

ci/cubicpoldeint

每秒通过立方插值`deinterlacing`滤镜的去交错块

md/mediandeint

每秒通过中值滤波`deinterlacing`滤镜的去交错块

fd/ffmpegdeint

每秒通过线性 $(-1\ 4\ 2\ 4\ -1)$ 滤波来处理`deinterlacing`滤镜的去交错块

l5/lowpass5

通过 $(-1\ 2\ 6\ 2\ -1)$ 滤波处理`deinterlacing`滤镜的去交错块

fq/forceQuant[|quantizer]

覆盖指定一个从输入到输出指示不变的量化器

quantizer

使用的Quantizer (量化器)

de/default

默认的`pp`滤镜组合 (hb|a,vb|a,dr|a)
fa/fast

快的`pp`滤镜组合 (h1|a,v1|a,dr|a)
ac

高品质`pp`滤镜组合(ha|a|128|7,va|a,dr|a)

pp例子

- 应用水平和垂直解封，去交错和自动亮度/对比度：

pp=hb/vb/dr/al

- 应用默认的滤镜但不包括自动亮度/对比度：

pp=de/-al

- 应用默认滤镜且包括瞬时降噪A：

pp=default/tmpnoise|1|2|3

- 应用亮度解封，自动根据可用的CPU时间开关垂直解封：

pp=hb|y/vb|a

pp7

应用`Postprocessing`滤镜7.它是[`spp`](#spp)滤镜的变通，类似于 spp =6 或者7的点DCT。是有中心样本使用者IDCT后

滤镜接受下面的选项：

- qp

强制设置的量化参数，接受一个0-63的整数，如果没有设置，将使用视频流的`QP`值（如果可用）

- mode

设置阈值模式，可用的是：

‘hard’

设置为硬阈值

‘soft’

设置为软阈值（更好的de-ringing效果,但可能变得更模糊）

‘medium’

设置中度阈值（最好的效果，默认）

psnr

两个输入视频之间的平均,最大和最小峰值信噪比(PSNR-Peak Signal to Noise Ratio)表示。

滤镜接受2路输入视频，其中第一个输入被认为是“主要”的源，将不改变的进行输出，第二输入作为“参考”视频进行PSNR计算

两个视频必须有相同的分辨率和像素格式才能正常工作。而且假定了有相同的输入帧来进行比较。

获得的平均PSNR会被输出到日志系统

滤镜存储每帧积累MSE（均方误差），并在处理末尾记录帧平均值，其公示为：

$$\text{PSNR} = 10 * \log_{10}(\text{MAX}^2 / \text{MSE})$$

这里`MAX`是图像中每个分量最大值的平均值。

滤镜接受参数的介绍见下：

- stats_file, f

如果指定，滤镜将采用指定的文件来保存每个帧的PSNR值

在`stats_file`指定的文件中将包含一个`key/value`序列，其是每两个比较帧的相应值

`stats_file`指定的文件中各个key的介绍如下：

- n

帧序号，从1计数

- mse_avg

```
对比帧逐像素均方误差，平均值是基于整个图像所有分量
- mse_y, mse_u, mse_v, mse_r, mse_g, mse_g, mse_a

对比帧逐像素分量均方误差，后缀就是分量类型
- psnr_y, psnr_u, psnr_v, psnr_r, psnr_g, psnr_b, psnr_a

对比帧分量峰值信噪比，后缀表明分量类型

例如：

movie=ref_movie.mpg, setpts=PTS-STARTPTS [main];
[main][ref] psnr="stats_file=stats.log" [out]

在这个例子中，输入和`ref_movie.mpg`文件比较，每个单独帧的`PSNR`存储在`stats.log`中

### pullup ###
下拉转换（逆电视电影）滤镜。能够处理混合`hard-telecine`，24000/1001帧率逐行和30000/1001帧率逐行内容。

这个`pullup`滤镜利用上下文进行决策。它是无状态的，不锁定模式，但不期待有以下字段可以进行匹配识别和重建逐行帧。

为了能产生一个包含偶数帧率的内容，在`pullup`后插入一个滤镜，如果输入帧率是29.97fps使用`fps=24000/1001`，如果输入是30fps或者25fps则`fp

滤镜接受下面的选项：

- jl
- jr
- jt
- jb

这些选项设置“垃圾”部分（会被忽略的左、右、上和下部分），左和右是8像素单位的整数倍，上和下是2行的整数被。默认每边8像素
- sb

设置严格的打断。设置为1将减少滤镜生成一个偶尔不匹配框架的机会，但也可能在高运动序列下导致帧数下降。设置为-1将更容易过滤匹配，则可以帮助处
- mp

设置度量标准平面（通道），允许值有：

‘l’

采用亮度通道

‘u’

采用色差蓝通道

‘v’

采用色差红

这个选项被设置为使用指定的通道作为滤镜计算平面，而替换默认的亮度通道计算。它可能会提高精度（如果源材料十分干净），但更可能是降低精度，特别

为了更好的效果（而不在输出中复制帧）且尽量让输出帧率接近，例如要反转一个NTSC输入：

ffmpeg -i input -vf pullup -r 24000/1001 ...

### qp ###
改变视频的量化参数（QP）

滤镜接受下面选项：
- qp

设置量化参数表达式

这个表达式通过`eval` API计算，可以包含以下常量：

- known

1表示索引不是129，否则为0
- qp

从-129到128的时序索引

#### qp例子 ####
- 一个方程：

qp=2+2*sin(PI*qp)

### removelogo ###
抑制台标，使用一个图像文件来确定哪些像素组成台标。它通过采用台标相近像素来填充台标位置。
```


率接受下面选项：

- filename, f

设置一个过滤位图文件，它指示了要抑制的台标图像信息，格式可以是`libavformat`中任何有效格式。图像的宽和高必须匹配要处理的视频流。

像素依据这样的规则指示台标：为0的位置不属于台标部分，非0的值则是台标的部分。如果使用白色（255）黑色（0）来标志是最安全的。过滤位图建议是带黑边框。

若必要，可以手动固定小斑点。*记住*如果台标像素没有被标注到，则滤镜质量会大大降低。标记过多的像素不会有太多伤害，但会增加需要模糊处理信息，从而降低性能。

repeatfields

这个滤镜根据其价值使用视频`ES`头中和重复字段中的`repeat_field`标志

rotate

任意旋转视频角度（以弧度值表示）

可选参数介绍如下：

- angle, a

设置一个表示要旋转角度的弧度表达式（中心旋转）。负值表示逆时针旋转，默认为0

这个表达式会每帧计算

- out_w, ow

设置输出宽表达式，默认为"iw",表达式只在配置时计算一次

- out_h, oh

设置输出高表达式，默认为"ih",表达式只在配置时计算一次

- bilinear

如果为1则允许双线性插值，为0禁用，默认为1

- fillcolor, c

设置填补颜色（图像旋转后水平），颜色的语法见[颜色/color](ffmpeg-doc-cn-07.md#颜色Color)章节的介绍，如果指定为`none`表示没有背景色

默认为"black".

角度和输出大小的表达式可以包含以下常量和函数：

- n

输入帧序数，从0计数（开始滤镜时），如果早于滤镜第一帧则为`NAN`

- t

输入帧按秒时间，当滤镜被配置时为0，早于则为`NAN`

- hsub

- vsub

水平和垂直色度分量。例如对像素格式`yuv422p`，`hsub`为2，`vsub`为1

- in_w, iw

- in_h, ih

输入的视频宽和高

- out_w, ow

- out_h, oh

输出的宽和高，它指定了要填补的宽和高表达式

- rotw(a)

- roth(a)

计算要完整包含旋转`a`弧度图像的最小的宽和高

它只能用于计算`out_w`和`out_h`的表达式

rotate例子

- 顺时针旋转PI/6:

rotate=PI/6

- 逆时针旋转PI/6 :

rotate=-PI/6

- 顺时针旋转45度:

rotate=45*PI/180

- 应用一个恒定选择周期T,开始角度为PI/3:

rotate=PI/3+2*PI*t/T

- 让输入旋转振荡T秒，有A弧度振幅：

```
rotate=A*sin(2*PI/T*t)
```

- 旋转视频，输出能包含所有内容：

```
rotate='2*PI*t:ow=hypot(iw,ih):oh=ow'
```

- 旋转视频，减少输出大小且没有背景：

```
rotate=2*PI*t:ow='min(iw,ih)/sqrt(2)':oh=ow:c=none
```

rotate命令

滤镜接受下面的命令：

- a, angle

设置表示角度的弧度表达式，命令接受同对于选项相同的语法。

如果指定的表达式无效，则保持当前值（不旋转）

sab

应用形状自适应模糊

滤镜接受下面选项：

- luma_radius, lr

设置亮度模糊强度，值范围0.1-4.0，默认1.0。更大的值会导致图像更模糊，但更慢

- luma_pre_filter_radius, lpfr

设置亮度预处理半径，值范围为0.1-2.0，默认值1.0。

- luma_strength, ls

设置被认为是同一像素的最大亮度区别。值范围0.1-100.0，默认1.0。

- chroma_radius, cr

设置色差模糊强度，值范围0.1-4.0，默认1.0。更大的值会导致图像更模糊，但更慢

- chroma_pre_filter_radius, cpfr

设置色差预处理半径，值范围为0.1-2.0

- chroma_strength, cs

设置被认为是同一像素的最大色差区别。值范围0.1-100.0

每个色度选项值，如果没有明确指定，则选用对应亮度选项值

scale

对输入视频放缩（修改尺寸），利用了`libswscale`库。

这个`scale`滤镜强制输出与输入有相同的长宽比，但改变输出样本点长宽比。

如果输入图像格式不同于下一个滤镜要求的格式，这个`scale`滤镜可以转换以符合要求。

scale选项

滤镜接受下面介绍的选项或任何`libswscale`放缩支持的选项。

参考 (ffmpeg-scaler)ffmpeg-scaler手册中以了解完整的放缩选项。

- width, w
- height, h

设置输出视频尺寸表达式，默认是输入尺寸。

如果值为0，则输入宽被用作输出

如果其中一个值被设置为-1，`scale`滤镜将以另外一个值在保持输入长宽比的基础上计算该值。如果两个都设置为-1则以输入尺寸作为输出

如果一个值设置为-n，且n>1，滤镜将以另外一个值为基础，保持输入长宽比进行计算，然后确保计算出的值是可整除n的最接近值

看下面对于表达式中允许内容的介绍

- interl

设置交错模式，允许：

'1'

强制交错放缩

'0'

不应用交错放缩Do not apply interlaced scaling.

‘-1’

根据输入源帧标志中是否有`interlaced`来决定是否采用交错放缩

默认为‘0’。

- flags

设置`libswscale`放缩标志，参考ffmpeg-scaler文档来了解完整的值列表。如果没有显式指定将采用默认值

- size, s

设置视频尺寸，语法同于[视频尺寸（分辨率）](ffmpeg-doc-cn-07.md#视频尺寸（分辨率）)

- in_color_matrix

- out_color_matrix

设置输入/输出`YCbCr`颜色空间类型

这将设定一个值来覆盖且强制用于输出和编码器

如果不指定，则颜色空间类型依赖于像素格式。

可能值：

‘auto’

自动选择。

‘bt709’

格式符合国际电信联盟(International Telecommunication Union-ITU) 推荐标准 BT.709。

‘fcc’

格式符合美国联邦通信委员会 (United States Federal Communications Commission-FCC)的美国联邦法规 (Code of Federal Regula

‘bt601’

设置颜色空间符合：

格式符合国际电信联盟(International Telecommunication Union-ITU) 推荐标准 BT.601

ITU-R Rec. BT.470-6 (1998)系统的B, B1, 和 G 以及电影与电视工程师学会 (

Society of Motion Picture and Television Engineers-SMPTE)的 ST 170:2004

‘smpte240m’

颜色空间符合SMPTE ST 240:1999。

- in_range

- out_range

设置输入/输出`YCbCr`样本范围

它设置一个值来覆盖且强制用于输出和编码。如果不知道，则范围依赖于输入像素格式，可能值是：

‘auto’

自动选择。

‘jpeg/full/pc’

设置完整的范围 (在8-bit亮度上是0-255)。

‘mpeg/tv’

设置"MPEG"范围(在8-bit亮度上是16-235)。

- force_original_aspect_ratio

设置在减少或者增加视频高/宽时是否保持原来的宽高比模式。可能值：

‘disable’

禁用这个特性而完全依据设置的宽/高尺寸

‘decrease’

输出视频尺寸将自动减少 (以保持宽高比)

‘increase’

输出视频尺寸将自动增加 (以保持宽高比)

关于这个选项的有用例子是：当你知道一个特定设备的最大允许尺寸时，可以使用这个来限制输出视频，同时保持宽高比。例如设备允许1280x720，你的视

****请注意****它与让w或者h为-1有些不同，要使用这个选项，你需要完整指定w和h为实际需要的尺寸才能工作

在上述表达式中允许下面的内容：

```
- in_w
- in_h

    输入的宽和高
- iw
- ih

    同于`in_w`和`in_h`.
- out_w
- out_h

    输出 (放缩后)宽和高
- ow
- oh

    同于`out_w`和`out_h`
- a

    等于`iw / ih`
- sar

    输入的样本点宽高比
- dar

    输入的宽高比, 计算自`(iw / ih) * sar`
- hsub
- vsub

    水平和垂直输入色差分量值, 例如对于`yuv422p`像素格式, `hsub`为2, `vsub`为1
- ohsb
- ovsb

    水平和垂直输出色差分量值, 例如对于`yuv422p`像素格式, `hsub`为2, `vsub`为1

#### scale例子 ####
- 放缩输入为200x100

    scale=w=200:h=100

    等效于:

    scale=200:100

    或者:

    scale=200x100
- 指定输出为缩写尺寸名:

    scale=qcif

    也可以写为:

    scale=size=qcif
- 放大为输入的2倍:

    scale=w=2*iw:h=2*ih
- 上面等效于:

    scale=2*in_w:2*in_h
- 放大2倍, 且采用交错放大模式:

    scale=2*iw:2*ih:interl=1
- 缩小为一半:

    scale=w=iw/2:h=ih/2
- 增大宽度, 且高度保持:

    scale=3/2*iw:ow
- 黄金分割比:

    scale=iw:1/PHI*iw
    scale=ih*PHI:ih
- 增加高度, 且让宽是高的3/2:

    scale=w=3/2*oh:h=3/5*ih
- 增加尺寸, 让尺寸是一个色差分量的整数倍:
```

```
scale="trunc(3/2*iw/hsub)*hsub:trunc(3/2*ih/vsub)*vsub"
```

- 增加最大为500像素，保持输入的宽高比：

```
scale=w='min(500\, iw*3/2):h=-1'
```

```
### separatefields ###
```

这个`separatefields`滤镜以视频帧为基础，将每个帧分成多个分量，产生一个半高而两有两倍帧率和帧数。

这个滤镜默认使用在帧中的场序标志信息来决定那些先输出。如果结果不对，在之前使用[`setfield`](#setfield)

```
### setdar, setsar ###
```

其中`setdar`滤镜设置输出的宽高比

是按下式通过改变指定的样本（即像素）宽高比来实现的：

$$\text{DAR} = \text{HORIZONTAL_RESOLUTION} / \text{VERTICAL_RESOLUTION} * \text{SAR}$$

记住，`setdar`滤镜不修改视频帧的像素尺寸（画面尺寸，即还是w×h）。设定的显示长宽比也可能改变其后的滤镜链中的滤镜，例如在放缩滤镜或者另一个“se

而`setsar`滤镜设置输出视频的像素宽高比

****注意****有用`setsar`滤镜的应用，输出显示的宽高比将根据前面的方程变化。

****记住****这里改变了样本（像素）宽高比将影响到后续滤镜，如另外一个“setdar”或“setsar”滤镜。

它接受下面参数：

- r, ratio, dar (setdar only), sar (setsar only)

设置指定的宽高比

可以是浮点数或者表达式或者一个`num:den`形式的字符串（其中分别对应分子和分母，表示`num/den`）。如果没有指定则假定为0。在使用`num:den`
- max

设置可改变的最大整数的值，其用于保证宽高比为设置值，而允许在分子（对应宽）和分母（对应高）减少的数。默认为100

参数表达式接受下面的内容：

- E, PI, PHI

一些可用常量，其中`E`为自然对数的底，`PI`为 π 值，以及`PHI`为黄金比例

- w, h

输入的宽和高

- a

等于`w / h`

- sar

输入的样本点宽高比

- dar

输入的宽高比，计算自`(iw / ih) * sar`

- hsub, vsub

水平和垂直输入色差分量值，例如对于`yuv422p`像素格式，`hsub`为2，`vsub`为1

```
#### setdar, setsar例子 ####
```

- 设置输出为16:9，可以是

```
setdar=dar=1.7777
```

```
setdar=dar=16/9
```

```
setdar=dar=1.7777
```

- 设置样本点宽高比为10:11

```
setsar=sar=10/11
```

- 设置输出为16:9，并设置了`max`值，采用命令

```
setdar=ratio=16/9:max=1000
```

```
### setfield ###
```

强制输出帧的场序

这里`setfield`把交错场提供给输出帧，它不改变输入帧，只是设置相应的属性，影响到后续的滤镜（如果 [`fieldorder`](#fieldorder)和 [`yadif`](#yadif)

滤镜接受下面选项：

```

- mode

    有效的值是：

    'auto'

    保持输入属性。
    'bff'

    设置为下场优先
    'tff'

    设置为上场优先
    'prog'

    设置为逐行

#### showinfo ####
不改变输入而在行中显示每帧信息。

显示的信息以`key/value`的序列形式给出

下面是将显示在输出中的值：

- n

    帧序数，从0开始计数
- pts

    输入帧的时间戳，以时基为单位，时间依赖于输入
- pts_time

    按秒计的时间戳
- pos

    输入帧在输入流中的偏移定位，-1表示信息不可用和/或无意义（例如合成视频中）
- fmt

    像素格式名
- sar

    输入帧的宽高比，表示为`num/den`格式
- s

    输入帧尺寸，语法同于[视频尺寸（分辨率）](ffmpeg-doc-cn-07.md#视频尺寸（分辨率）)
- i

    交错模式（"P"对应"逐行"，"T"对应上场优先，"B"为下场优先t）
- iskey

    为1表示是关键帧，0则不是
- type

    输入帧图片类型（"I"对应I帧，"P"对应P帧，"B"对应B帧，或者"?"对应未知类型）。参考定义与`libavutil/avutil.h`中的`av_get_picture_type`
- checksum

    输入帧所有信息内容的 Adler-32校验值（以16进制输出）
- plane_checksum

    输入帧所有信息内容的 Adler-32校验值（以16进制输出），以格式"[c0 c1 c2 c3]"显示

#### showpalette ####
显示每个帧的256色模板。滤镜是关于`pal8`像素格式帧的。

它接受下面选项：

-s

    设置调色板中颜色数量宽，默认为30（对应于30x30`pixel`）

#### shuffleplanes ####
重排和/或复制视频通道

它接受下面参数：

- map0

    被用作第一个输出通道的输入通道

```

- map1

被用作第二个输出通道的输入通道

- map2

被用作第三个输出通道的输入通道

- map3

被用作第四个输出通道的输入通道

第一个通道序数为0，默认是保持输入不变。

交换第二和第三通道：

```
ffmpeg -i INPUT -vf shuffleplanes=0:2:1:3 OUTPUT
```

```
### signalstats ###
```

评估各种视觉指标, 协助确定问题与模拟视频媒体的数字化。

默认滤镜会以日志记录这些元数据值：

- YMIN

显示输入帧中最小的Y值（亮度），范围为[0-255]

- YLOW

显示10%(表示有10%的像素点Y值低于这个值)的Y值, 范围[0-255]

- YAVG

显示输入帧的平均Y值，范围[0-255]

- YHIGH

显示90%(表示有90%的像素点Y值低于这个值)的Y值，范围[0-255]

- YMAX

显示输入帧中最大Y值，范围[0-255]

- UMIN

显示输入帧中最小的U值（色差U），范围为[0-255]

- ULOW

显示10%(表示有10%的像素点U值低于这个值)的U值, 范围[0-255]

- UAVG

显示输入帧的平均U值，范围[0-255]

- UHIGH

显示90%(表示有90%的像素点U值低于这个值)的U值，范围[0-255]

- UMAX

显示输入帧中最大U值，范围[0-255]

- VMIN

显示输入帧中最小的V值（色差V），范围为[0-255]

- VLOW

显示10%(表示有10%的像素点V值低于这个值)的V值, 范围[0-255]

- VAVG

显示输入帧的平均V值，范围[0-255]

- VHIGH

显示90%(表示有90%的像素点U值低于这个值)的U值，范围[0-255]

- VMAX

显示输入帧中最大V值，范围[0-255]

- SATMIN

显示输入帧最小色包含度值，范围[0-~181.02]。

- SATLOW

显示输入帧10%(表示有10%的像素点该类值低于这个值)色包含度值，范围[0-~181.02]。

- SATAVG

显示输入帧最平均包含度值，范围[0-~181.02]。

- SATHIGH

显示输入帧90%(表示有90%的像素点该类值低于这个值)色包含度值，范围[0-~181.02]。

```
- SATMAX

    显示输入帧最大色包含度值，范围[0-181.02]。
- HUEMED

    显示输入帧中的颜色中值，范围[0-360]。
- HUEAVG

    显示输入帧中颜色中值平均，范围[0-360]。
- YDIF

    显示当前帧与前帧的像素平均Y值差，范围[0-255]。
- UDIF

    显示当前帧与前帧的像素平均U值差，范围[0-255]。
- VDIFF

    显示当前帧与前帧的像素平均V值差，范围[0-255]。

滤镜接受下面的选项：

- stat
- out

    `stat`指定一个额外形式的图像分析，`out`则对指定类型像素在输出视频中高亮（突出）显示

这两个选项都接受下面的值：

'tout'

    识别时间离群值（瞬时差异）像素。一个时间离群是像素不同于其周围的像素有同样的分量特性（趋势）。例如视频的中断、头阻塞、磁带跟踪问题
'vrep'

    识别垂直线重复。垂直线重复即帧内包含类似的行像素。在数码视频中垂直线重复较常见的，但这种情况在数字化模拟源时是罕见的，当出现则表明信号
'brng'

    识别像素超出范围的

- color, c

    指定高亮的颜色，默认为黄色

#### signalstats例子 ####
- 输出各种数据指标

    ffmpeg -f lavfi movie=example.mov,signalstats="stat=tout+vrep+brng" -show_frames

- 输出每帧Y的最小和最大值：

    ffmpeg -f lavfi movie=example.mov,signalstats -show_entries frame_tags=lavfi.signalstats.YMAX,lavfi.signalstats.YMIN

- 播放视频，并以红色突出标识超出范围的像素

    ffmpegplay example.mov -vf signalstats="out=brng:color=red"

- 播放视频，并与标志元数据一同展示：

    ffmpegplay example.mov -vf signalstats=stat=brng+vrep+tout,drawtext=fontfile=FreeSerif.ttf:textfile=signalstat_drawtext.txt

    使用的命令后signalstat_draw.text的内容：
```

time %pts:hms} Y (%metadata:lavfi.signalstats.YMIN)-%metadata:lavfi.signalstats.YMAX)) U (%
{metadata:lavfi.signalstats.UMIN}-%metadata:lavfi.signalstats.UMAX)) V (%metadata:lavfi.signalstats.VMIN)-%
{metadata:lavfi.signalstats.VMAX}) saturation maximum: %metadata:lavfi.signalstats.SATMAX} ``

smartblur

在不影响轮廓的基础上模糊视频

它接受下面的选项：

- luma_radius, lr

设置亮度半径，为浮点数，范围[0.1,5.0],用于指示高斯滤波模糊的方差值（越大越慢），默认为1.0

- luma_strength, ls

设置亮度强度。为浮点数，范围[-1.0,1.0],用于配置模糊强度，在[0.0,1.0]内的值将模糊图像，而[-1.0,0.0]内的值将锐化图像，默认为1.0

- luma_threshold, lt

设置亮度阈值作为系数来判断一个像素是否该模糊。为[-30,30]内的整数，如果值在[0,30]则过滤平面，如果值在[-30,0]则过滤边缘。默认为0

- chroma_radius, cr

设置色度半径，为浮点数，范围[0.1,5.0],用于指示高斯滤波模糊的方差值（越大越慢），默认为1.0

- chroma_strength, cs

设置色度强度。为浮点数，范围[-1.0,1.0],用于配置模糊强度，在[0.0,1.0]内的值将模糊图像，而[-1.0,0.0]内的值将锐化图像，默认为1.0

- chroma_threshold, ct

设置色度阈值作为系数来判断一个像素是否该模糊。为[-30,30]内的整数，如果值在[0,30]则过滤平面，如果值在[-30,0]则过滤边缘。默认为0

如果色度（差）选项没有显式设置，将采用对应的亮度值来设定

stereo3d

在不同的立体图像格式之间进行转换

滤镜接受下面选项：

- in

设置输入格式类型

有效的是:

'sbsl'

并排，左眼在左，右眼在右

'sbsr'

并排，右眼在左，左眼在右

'sbs2l'

半宽并排，左眼在左，右眼在右

‘sbs2r’

半宽并排，右眼在左，左眼在右

‘abl’

上下排布，左眼上，右眼下

‘abr’

上下排布，右眼上，左眼下

‘ab2l’

半高上下排布，左眼上，右眼下

‘ab2r’

半高上下排布，右眼上，左眼下

‘al’

交替帧，左眼前，右眼后

‘ar’

交替帧，右眼前，左眼后
默认是‘sbs1’。

● out

设置输出排布类型：

有效的在输入格式基础上还有：

‘arbg’

浮雕红/蓝通道（红色左眼，蓝色右眼）

‘argg’

浮雕红/绿通道（红色左眼，绿色右眼）

‘arcg’

浮雕红/青通道（红色左眼，青色右眼）

‘arch’

浮雕红/青一半（红色左眼，青色右眼）

‘arcc’

浮雕红/青通道（红色左眼，青色右眼）

‘arcd’

浮雕红/青最小二乘优化杜布瓦投影（红色左眼，青色右眼）

‘agmg’

浮雕绿/品红通道（绿色左眼，品红色右眼）

‘agmh’

浮雕绿/品红半色通道（绿色左眼，品红色右眼）

‘agmc’

浮雕绿/品红通道（绿色左眼，品红色右眼）

‘agmd’

浮雕绿/品红最小二乘优化杜布瓦投影（绿色左眼，品红色右眼）

‘aybg’

浮雕黄/蓝通道（黄色左眼，蓝色右眼）

‘aybh’

浮雕黄/蓝半色通道（黄色左眼，蓝色右眼）

‘aybc’

浮雕黄/蓝通道（黄色左眼，蓝色右眼）

‘aybd’

浮雕黄/蓝最小二乘优化杜布瓦投影（黄色左眼，蓝色右眼）

‘irl’

隔行扫描的行（左眼上行，右眼下一行）

‘irr’

隔行扫描的行（右眼上行，左眼下一行）

‘ml’

单独输出左眼

‘mr’

单独输出右眼

默认是‘arcd’.

stereo3d例子

- 将并排3d格式转换为浮雕黄/绿通道混合3d:

stereo3d=sbsl:aybd

- 将上下排布3d转换为并排3d

stereo3d=abl:sbsr

spp

应用一个简单的后处理滤镜，压缩或者解压缩图像（或-quality 为6）变化提升平均结果。

滤镜接受下面选项：

- quality

设置质量系数。它是一个平均水平数，范围0-6，为6意味着有最高质量，每个增量大约对应处理速度2倍降低，默认为3

- qp

强制量化参数。如果不设置，将从输入流中提取 qp 值（如果可用）

- mode

设置阈值模式，允许：

'hard'

硬阈值（默认）

'soft'

软阈值（更好的效果，也可能更模糊）。

- use_bframe_qp

如果为1将支持使用B帧的QP，使用这个选项如果B帧有大的QP则可能导致闪烁默认为0（不允许）

subtitles

利用 libass 库在输入视频上添加字幕

编译配置选项 `--enable-libass`。滤镜还需要编译包含 `libavcodec` 和 `libavformat` 以支持把字幕文件转换成ASS(Advanced Substation Alpha) 字幕格式

滤镜接受下面选项：

- filename, f

设置要读取的字幕文件，它必须指定

- original_size

指定原始视频分辨率，这个视频是要组合ASS文件的。其语法见[视频尺寸（分辨率）](#)部分。为了应对ASS中长宽比例不良设计，放缩字体长宽比是必要的

- charenc

设置输入字符编码，只对字幕过滤，仅当编码不是UTF-8时使用

- stream_index, si

设置字幕流索引数，仅在 `subtitles` 滤镜中使用

- force_style

覆盖 `subtitles` 的默认样式或脚本信息参数。它接受一个有，隔开的 `KEY=VALUE` 字符串

如果第一个键没有指定，则假定第一个值是描述 `filename`

例如渲染字幕文件sub.srt到输入视频顶部采用命令：

```
subtitles=sub.srt
```

它等效于:

```
subtitles=filename=sub.srt
```

渲染mkv文件中的字幕:

```
subtitles=video.mkv
```

渲染从文件活动的字幕文件中提取的字幕:

```
subtitles=video.mkv:si=1
```

为字幕添加一个透明的绿 DejaVer 衬线:

```
subtitles=sub.srt:force_style='FontName=DejaVu Serif,PrimaryColour=&HAA00FF00'
```

super2xsai

使用滤镜来平滑放大2倍（放大和插入）像素扩展算法。

通常用于在不降低锐度基础上放大图像。

swapuv

交换U和V分量值

telecine

对视频应用 telecine 处理

滤镜接受下面选项：

- first_field

可能值：

‘top, t’

```
上场优先（默认）
```

‘bottom, b’

```
下场优先
```

- pattern

要应用的下拉模式。默认为23

一些典型的模式是

NTSC输出(30i):

```
27.5p: 32222
24p: 23 (classic)
24p: 2332 (preferred)
20p: 33
18p: 334
16p: 3444
```

PAL输出(25i):

```
27.5p: 12222
24p: 22222222223 ("Euro pulldown")
16.67p: 33
16p: 33333334
```

thumbnail

在一个给定的连续帧序列中选定一个最具代表性的帧（缩略图，抽帧效果形成跳帧序列）。

滤镜接受下面选项：

- n

设置要分析的帧批量大小。设置为 `n` 帧，则滤镜会从中挑选出一帧，然后处理下面 `n` 帧，直到最后，默认为100

因为滤镜跟踪整个帧序列，一个更大的 `n` 将占用更多的内存，所有不推荐设置太高的值

thumbnail例子

- 每50帧抽取一个

```
thumbnail=50
```

- 应用ffmpeg来输出一个缩略图序列

```
ffmpeg -i in.avi -vf thumbnail,scale=300:200 -frames:v 1 out.png
```

tile

让连续几帧磁贴拼接

滤镜接受下面选项：

- layout

设置网格布局模式（行和列数）。语法同于[视频尺寸（分辨率）](#)部分

- nb_frames

设置最大用作渲染的区域尺寸，必须小于或者等于 `wxh`（这里是 `layout` 值），默认为0，表示最大（所有）尺寸

- margin

设置外边界边缘像素

- padding

设置内边界边缘厚度（布局渲染后两个帧间的像素个数）。更大的填补选项（表示有更多的值用于边缘）将由滤镜填充（颜色）

- color

设置未使用区域的颜色。描述语法同于颜色/Color部分。默认为"black".

tile例子

- 产生8x8的PNG关键帧标题（-skip_frame nokey）

```
ffmpeg -skip_frame nokey -i file.avi -vf 'scale=128:72,tile=8x8' -an -vsync 0 keyframes%03d.png
```

这里 -vsync 0 是必要的，以防止ffmpeg以原始帧率复制每个输出帧

- 每个区域显示5个图片（在3x2的 layout 值下），间隔7像素且有2像素的外边缘，使用了省略（关键名）和命名选项

```
tile=3x2:nb_frames=5:padding=7:margin=2
```

tinterlace

执行各种类型的时间交错

帧计数从1开始，所以第一帧为奇数帧。

滤镜接受下面选项：

- mode

指定交错模式。这个选项也可以单独指定一个值。关于值的列表见下。

有效的值：

```
'merge, 0'
```

移动奇数帧到上场，偶数帧到下场，产生双倍高度的帧，但是只要一半的帧率。

```
-----> time
输入：
Frame 1      Frame 2      Frame 3      Frame 4

11111        22222        33333        44444
11111        22222        33333        44444
11111        22222        33333        44444
11111        22222        33333        44444

输出：
11111                33333
22222                44444
11111                33333
22222                44444
11111                33333
22222                44444
11111                33333
22222                44444
```


‘drop_odd, 1’

```
只输出偶数帧，奇数帧被丢弃，不改变高但只要一半帧率

-----> time
输入：
Frame 1      Frame 2      Frame 3      Frame 4

11111        22222        33333        44444
11111        22222        33333        44444
11111        22222        33333        44444
11111        22222        33333        44444

输出：
                22222                44444
                22222                44444
                22222                44444
                22222                44444
```

‘drop_even, 2’

```
只输出奇数帧，丢弃了偶数帧，不改变高，但只有一半帧率

-----> time
输入：
Frame 1      Frame 2      Frame 3      Frame 4

11111        22222        33333        44444
11111        22222        33333        44444
11111        22222        33333        44444
11111        22222        33333        44444

Output:
11111                33333
11111                33333
11111                33333
11111                33333
```

‘pad, 3’

```
扩展每个帧的高度，每行用黑色间隔，生成有2倍的高，和相同的帧率

-----> time
输入：
Frame 1      Frame 2      Frame 3      Frame 4

11111        22222        33333        44444
11111        22222        33333        44444
11111        22222        33333        44444
11111        22222        33333        44444

输出：
11111        .....        33333        .....
.....        22222        .....        44444
11111        .....        33333        .....
.....        22222        .....        44444
11111        .....        33333        .....
.....        22222        .....        44444
11111        .....        33333        .....
.....        22222        .....        44444
```

‘interleave_top, 4’

```
奇数帧在上保留奇数行，偶数帧在下保留偶数行的交错，不改变高，有一半的帧率

-----> time
```

```
输入：
Frame 1      Frame 2      Frame 3      Frame 4

11111<-      22222      33333<-      44444
11111      22222<-      33333      44444<-
11111<-      22222      33333<-      44444
11111      22222<-      33333      44444<-

输出：
11111      33333
22222      44444
11111      33333
22222      44444
```

‘interleave_bottom, 5’

```
偶数帧在上保留奇数行，奇数帧在下保留偶数行的交错，不改变高，有一半的帧率

-----> time
输入：
Frame 1      Frame 2      Frame 3      Frame 4

11111      22222<-      33333      44444<-
11111<-      22222      33333<-      44444
11111      22222<-      33333      44444<-
11111<-      22222      33333<-      44444

输出：
22222      44444
11111      33333
22222      44444
11111      33333
```

‘interlacedx2, 6’

```
双倍帧率不改变高，帧间插入一个两帧的混合，混合模式（依赖于`top_field_first`标志）可是后帧的奇数行和前帧的偶数行交错。这对没有同步的

-----> time
输入：
Frame 1      Frame 2      Frame 3      Frame 4

11111      22222      33333      44444
 11111      22222      33333      44444
11111      22222      33333      44444
 11111      22222      33333      44444

输出：
11111 22222 22222 33333 33333 44444 44444
 11111 11111 22222 22222 33333 33333 44444
11111 22222 22222 33333 33333 44444 44444
 11111 11111 22222 22222 33333 33333 44444
```

单独的数字值被弃用，但这里给出是为了向后兼容

默认为 merge .

● flags

指定标记影响筛选过程.

有效值对 flags 是:

low_pass_filter, vlfp

使用垂直低通滤波。在从逐行源（且包含高频垂直细节）创建交错视频时为了减少‘twitter’和云纹图案现象需要使用这个滤波器

垂直低通滤波仅在`interleave_top`和`interleave_bottom`模式时允许使用

transpose

对输入转置行和列来可选的翻转（图像）

它接受下面的参数：

- dir

指定翻转方向，可以采用下面的值：

‘0, 4, cclock_flip’

逆时针90度和垂直翻转（默认），它的效果是：

L.R	L.l
. . -> . .	
l.r	R.r

‘1, 5, clock’

顺时针90度旋转，效果是：

L.R	l.L
. . -> . .	
l.r	r.R

‘2, 6, cclock’

逆时针90度旋转，效果是：

L.R	R.r
. . -> . .	
l.r	L.l

‘3, 7, clock_flip’

顺时针90度旋转且垂直翻转，效果是：

L.R	r.R
. . -> . .	
l.r	l.L

对于4-7的值，表示转换是对于肖像或风景的。这些值都是被弃用的，应该采用 passthrough 设定来替代

数值是弃用,应该被删除,取而代之的是符号常量

- passthrough

如果输入的几何匹配（超出）一个指定的形式部分将不进行转换。有效值是：

‘none’

一直适用

'portrait'

保护肖像的几何 (对于 `height >= width`)，指裁剪一定的转换宽度，保护高度不变（等比例）

'landscape'

保护景观几何 (对于 `width >= height`)，表示裁剪一定的转换高度，保持宽度不变（等比例）

默认为none.

例如，要顺时针旋转90度且保护肖像布局:

```
transpose=dir=1:passthrough=portrait
```

以命令的形式实现同样效果:

```
transpose=1:portrait
```

trim

减少输入，输出包含一个连续输入的组成部分

它接受下面参数：

- start

指定开始部分时间的，即帧时间戳开始将输出第一帧

- end

指定结束部分时间，即帧的时间戳达到的前一帧是输出的最后一帧。

- start_pts

同于 `start`，只是以时基为时间单位替代秒

- end_pts

同于 `end`，只是以时基为时间单位替代秒

- duration

按秒最大持续时间 seconds.

- start_frame

开始的帧序数，该帧开始被输出

- `end_frame`

结束的帧序数，该帧开始被丢弃（不被输出）

其中 `start`、`end` 和 `duration` 采用持续时间表示格式，其语法见[持续时间](#)

注意 `start` / `end` 开头的选项和 `duration` 都仅关注帧的时间戳，而带 `_frame` 后缀的版本则仅仅通过帧简单计数。还要注意滤镜并不改变时间戳，如果你想输出是以0开始的时间戳，则需要在滤镜后接一个 `setpts` 滤镜来处理。

如果有多个开始或者结束时间，滤镜将采取贪婪算法，视图在匹配至少一个约束（开始/结束）的情况下尽量多的输出。想保留多个约束部分组成，需要链式采用多个裁剪过滤器（可能还需要结合其他滤镜来拼接）

默认是保留所有输入。所以它可以仅设置一个结束值来保留之前的一切。

例如：

- 只保留第二分钟

```
ffmpeg -i INPUT -vf trim=60:120
```

- 保留第一秒内的

```
ffmpeg -i INPUT -vf trim=duration=1
```

unsharp

锐化或者模糊输入视频

它接受下面的参数：

- `luma_msize_x`, `lx`

设置亮度矩阵水平尺寸。它必须是3-63的奇数值，默认5

- `luma_msize_y`, `ly`

设置亮度矩阵垂直尺寸，它必须是3-63的奇数值，默认5

- `luma_amount`, `la`

设置亮度效果强度，合理值为-1.5 - 1.5的浮点数（可超出前范围）。

负数值表明视频会被模糊，正数值则会被锐化，0则没有效果

默认为1.0.

- `chroma_msize_x`, `cx`

设置色度矩阵水平尺寸。它必须是3-63的奇数值，默认5

- `chroma_msize_y`, `cy`

设置色度矩阵垂直尺寸。它必须是3-63的奇数值，默认5.

- `chroma_amount`, `ca`

设置色度效果强度，合理值为-1.5 - 1.5的浮点数（可超出前范围）。

负数值表明视频会被模糊，正数值则会被锐化，0则没有效果

默认为0.0.

- opengl

如果为1，指定采用OpenCL能力，要求编译时启用了 `--enable-opengl`，默认0.

所有参数都是可选的默认值等效于'5:5:1.0:5:5:0.0'字符串

unsharp例子

- 加强亮度锐化效果

```
unsharp=luma_msize_x=7:luma_msize_y=7:luma_amount=2.5
```

- 加强亮度和色度的模糊

```
unsharp=7:7:-2:7:7:-2
```

uspp

应用超慢/简单的后处理,压缩和解压图像(或对应于 `quality` 中水平为8的完全处理)变化和平均结果。

它不同于 `spp`，实际上 `uspp` 编码和解码每个 `libavcodec` 块（Snow），而 `spp` 使用一个内部简化的8x8 DCT，其相似于 MJPEG的DCT

滤镜接受下面选项：

- quality

设置质量水平值。它是平均水平值数字，范围0-8，如果为0，则滤镜没有效果，设置为8将有最好的效果。每增加1级大约速度减慢2倍，默认为3

- qp

强制设定质量参数，如果不设置，将采用输入流中的QP值（如果可用）

vidstabdetect

分析视频的静止/不晃动，两步过程中的第1步，下一步是 `vidstabtransform`。

这个滤镜生成一个文件，指定相对平移和旋转变换后续帧的信息，它用于 `vidstabtransform` 滤镜

为了编译支持它需要设置 `--enable-libvidstab`

滤镜接受下面选项：

- result

指定保存转换信息的文件路径。默认为 `is transforms.trf. shakiness`

设置摄像头如何快速设置来满足晃动的视频,值范围是1-10整数，1意味着很小的晃动，10意味着强烈晃动，默认为5
`accuracy`

设置检测过程的准确性,值范围为1-15，1表示低精度，15表示高精度。默认15 `stepsize`

设置搜索过程的间隔值（扫描尺度）。最低是1像素分辨率扫描，默认为6 mincontrast

设置最低对比度。低于这个值一个本地测量领域会被丢弃。为范围在0-1的浮点数，默认为0.3. tripod

设置参考帧数三脚架模式

如果允许，对帧运动的比较将以一个参考过滤流相比进行，从中指定一个。这样可以补偿或多或少的静态帧中的所有动作，保持相机视图绝对静止

如果设为0则禁用，帧数从1开始计数 show

显示字段和转换生成的帧，接受一个0-2间的整数，默认为0，它禁止任何可视内容。

vidstabdetect例子

- 使用默认值：

```
vidstabdetect
```

- 分析晃动视频的强度，把结果放置在mytransforms.trf:

```
vidstabdetect=shakiness=10:accuracy=15:result="mytransforms.trf"
```

- 把内部转换生成的视频显示出来（可视化）：

```
vidstabdetect=show=1
```

- 在ffmpeg中分析中等强度晃动：

```
ffmpeg -i input -vf vidstabdetect=shakiness=5:show=1 dummy.avi
```

vidstabtransform

视频静止/不晃动，两步过程的第二步，其第一步是 `vidstabdetect`

从一个文件读取每一帧需要应用/补偿的信息，与 `vidstabdetect` 一起使用来稳定视频，参看<http://public.hronopik.de/vid.stab> 来了解更多。见下，它对于使用 `unsharp` 是很重要的。

为了使用它需要允许编译设置 `--enable-libvidstab`

vidstabtransform选项

- input

设置读取转换信息的文件，默认为transforms.trf.

- smoothing

设置帧数，其值以表达式 (value*2 + 1)用作低通来滤除摄像机运动，默认为10.

例如对于设置为10则意味着21帧被使用（过去10帧和接下来10帧）来平滑摄像机移动。更大的值可以得到一个更平滑视频，但限制摄像机加速度(平底锅摇/倾斜 移动)。0表示摄像机是静止的

- optalgo

设置相机路径优化算法

接受值:

‘gauss’

镜头运动采用高斯低通滤波器内核 (默认)

‘avg’

转换平均值

- maxshift

设置帧中最大转换像素值，默认为-1，表示没有限制

- maxangle

设置最大帧旋转角度（弧度值，度*PI/180），默认为-1，表示没有限制

- crop

指定如何处理边界,由于运动补偿可能可见

有效值:

‘keep’

从以前帧保持图像信息（默认）

‘black’

填充黑色边

- invert

为1则转化转换。默认值为0

- relative

为1表示转换是相对于前帧，0表示绝对的（不和前帧相关），默认为0

- zoom

设置放大比例。正数则相对于推进效果，负数相当于拉远效果，默认为0（不变）

- optzoom

设置最佳缩放以避免边界

可能值:

‘0’

禁止

‘1’

确定最优静态缩放值 (只有很强的运动将导致可见边界) (默认)

‘2’

确定最优自适应缩放值 (没有边界可见), 参见 ‘zoomspeed’

注意这里的 zoom 值被添加到一个计算中

- zoomspeed

设置每帧放大的最大百分比限度值 (当 optzoom 被设置为2时), 范围为0-5, 默认为0.25

- interpol

指定插值类型

有效值是:

‘no’

不插值

‘linear’

水平线性插值

‘bilinear’

在两个方向上线性插值 (默认)

‘bicubic’

在两个方向上立方插值 (慢)

- tripod

如果为1启用虚拟三脚架模式, 其等效于 relative=0:smoothing=0 默认为0 Default value is 0.

它要求在 vidstabdetect 中也启用 tripod

- debug

为1增加日志记录按冗长形式。也检测全局运动写入到临时文件 global_motions.trf, 默认为0

vidstabtransform例子

- 帧ffmpeg使用默认典型的稳定系数:

```
ffmpeg -i inp.mpeg -vf vidstabtransform,unsharp=5:5:0.8:3:3:0.4 inp_stabilized.mpeg
```

注意一直建议使用 `unsharp`

- 从给定文件加载转换数据来放大一点:

```
vidstabtransform=zoom=5:input="mytransforms.trf"
```

- 使视频更平滑:

```
vidstabtransform=smoothing=30
```

vflip

让输入垂直翻转

例如：利用ffmpeg垂直翻转视频

```
ffmpeg -i in.avi -vf "vflip" out.avi
```

vignette

使或扭转自然渐晕效应

滤镜接受下面选项：

- angle, a

以弧度表示的镜头组角度

值范围为 [0,PI/2]

默认为: "PI/5"

- x0
- y0

设置中心坐标表达式，默认分别是"w/2" and "h/2"

- mode

设置向前/向后模式

有效值为：

'forward'

中心点的距离越大, 图像的颜色越深

`'backward'`

中心点的距离越大, 图像越亮。这可以用于扭转装饰图案效果, 虽然没有自动检测提取镜头角度和其他设置。它也可以用来创建一个燃烧的效果。

默认为`'forward'`。

- `eval`

设置表达式计算模式(对于`angle`, `x0`, `y0`)。

有效值为:

`'init'`

只在初始化时计算一次

`'frame'`

每帧计算, 它的速度远低于`'init'`模式, 因为它需要每帧计算所有表达式, 但这允许了先进的动态表达式(完成一些特效)

默认为`'init'`。

- `dither`

为1(默认)则启用抖动减少循环条带效应

- `aspect`

设置插图像素长宽比。此设置将允许调整插图形状, 设置值对于输入 `SAR` (样本长宽比) 将调整矩形光损失后的尺寸

默认为1/1。

vignette表达式

这里有 `angle` (原文误为`alpha`), `x0` 和 `y0`表达式允许包含的参数

- `w`
- `h`

输入的宽和高

- `n`

输入帧序号, 从0开始计

- `pts`

以时基单位计的PTS (作品时间戳), 未定义则为 `NAN`

- `r`

输入视频帧率, 未知则为 `NAN`

- t

以秒计的PTS (作品时间戳)，未定义则为 `NAN`

- tb

输入视频时基

vignette例子

- 应用简单的强大的渐变效应:

`vignette=PI/4`

- 做一个闪烁的光损失:

`vignette='PI/4+random(1)*PI/50':eval=frame`

w3fdif

反交错的输入视频(“w3fdif”代表“韦斯顿3场反交错滤波器——Weston 3 Field Deinterlacing Filter”)。

基于英国广播公司（BBC R&D）的马丁•韦斯顿（Martin Weston）研发，并由吉姆·伊斯特布鲁克（Jim Easterbrook）实现的反交错算法。这个滤镜使用的滤波系数是BBC研发的

它有两组滤波系数，被称为"simple"（简单）和 "complex"（复杂）。使用那个滤波系数可以通过参数设置。

- filter

设置采用的滤波系数，允许值为：

`'simple'`

简单滤波器系数。

`'complex'`

复杂滤波器系数

默认 `'complex'`。

- deint

指定帧反交错，接受值为：

`'all'`

反交错所有帧

`'interlaced'`

仅反交错设置为交错的帧

默认 'all'.

xbr

对像素应用一个xBR高质量放大滤镜，它遵循一套边缘检测规则，详情见<http://www.libretro.com/forums/viewtopic.php?f=6&t=134>

接受选项：

- n

设置放缩尺寸，2对应于2xBR，3对应于3xBR，4对应于4xBR，默认为3

yadif

反交错输入视频（yadif 意味着另外一个反交错滤镜）

它接受下面的参数：

- mode

采用隔行扫描模式。它接受下列值之一：

0, send_frame

对每帧都输出

1, send_field

对每场都输出一帧

2, send_frame_nospatial

类似`send_frame`，但跳过交错检查

3, send_field_nospatial

类似`send_field`，但跳过交错检查

默认为 `send_frame`

- parity

假定输入隔行视频的模式，它接受下列值：

0, tff

假定为上场优先

1, bff

假定为下场优先

-1, auto

自动侦测

默认为 auto ,如果交错模式未知或者不能正确处理则假定为 tff

- deint

指定哪些帧需要反交错，接受下列值：

0, all

所有帧

1, interlaced

仅标记为交错的帧

默认为所有

zoompan

应用放大和摇镜头效果

滤镜接受下面选项：

- zoom, z

设置放大系数表达式，默认为1

- x
- y

设置x和y表达式，默认为0

- d

设置持续帧数，这设置有多少数量的帧受到影响

- s

设置输出图像尺寸，默认为 'hd720'. 每个表达式接受下列参数:

- in_w, iw

输入的宽

- in_h, ih

输入高

- out_w, ow

输出宽

- out_h, oh

输出高

- in

输入帧计数

- on

输出帧计数

- x
- y

最后计算的x和y对于当前输入帧的x和y表达式。

- px
- py

之前输入帧对应的最后输出帧最后计算'x'和'y'，或者为0（第一个输入帧）

- zoom

当前输入帧对应的最后 z 表达式计算得出的放大系数

- pzoom

前一输入帧前最后输出帧计算的放大系数

- duration

当前输入帧对应的输出帧数。对每个输入帧计算 d 值

- pduration

前一输入帧之前创建输出帧的数量

- a

有理数 = iw/ih

- sar

样本长宽比

- dar

显示长宽比

zoompan例子

- 推进到1.5 并且同时在中心附近摇的效果:

```
zoompan=z='min(zoom+0.0015,1.5)':d=700:x='if(gte(zoom,1.5),x,x+1/a)':y='if(gte(zoom,1.5),y,y+1)':s=640x360
```

- 推进到1.5 并且同时以中心摇的效果:

```
zoompan=z='min(zoom+0.0015,1.5)':d=700:x='iw/2-(iw/zoom/2)':y='ih/2-(ih/zoom/2)'
```


38 视频源

下面是当前有效的视频源

buffer

缓冲视频帧，其可以作为滤镜链图的环节

它通常用于编程，特别是通过 `libavfilter/vsrc_buffer.h` 的接口。

接受如下参数：

- `video_size`
指定视频尺寸，(同时指定width 和 height)。语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节。
- `width`
输入视频宽
- `height`
输入视频高
- `pix_fmt`
像素格式描述，可以是一个表征像素格式的号码或者名称
- `time_base`
指定时间戳时基
- `frame_rate`
指定帧率 stream.
- `pixel_aspect, sar`
输入视频的像素长宽比。
- `sws_param`
指定一个可选参数用于在自动检测到输入视频大小或者格式变化时插入放缩滤镜。

例如：

```
buffer=width=320:height=240:pix_fmt=yuv410p:time_base=1/24:sar=1
```

其指定源为 320x240 分辨率，采样 `yuv410p` 的像素格式，时基为1/24秒，采样1：1的像素比。因为 `yuv410p` 对应的像素格式序号为 6（`libavutil/pixfmt.h` 中的 `AVPixelFormat` 枚举中定义），所以上面又等效于

```
buffer=size=320x240:pixfmt=6:time_base=1/24:pixel_aspect=1/1
```

另外,选项可以字符串（以':'分隔）直接指定（没有选项名,按顺序给选项赋值（下面介绍选项的顺序））,但不建议使用这个语法:

```
width:height:pix_fmt:time_base.num:time_base.den:pixel_aspect.num:pixel_aspect.den[:sws_param]
```

cellauto

创建一个模式生成的细胞自动发生器（就是细胞变化样的图）

细胞自动发生器的初始状态可以通过 `filename` 选项和 `pattern` 选项的模式来定义,如果不知道则是随机初始状态。

每个新帧中的一个新行视频充满了下一代细胞自动发生器的结果。当 `scroll` 选项被指定时,整个帧会被滚动填充。

接受如下选项:

- `filename, f`

用于读取细胞自动发生器初始状态的文件。在文件中第一行从行首开始每个非空字符被认为是活的细胞直到换行,更多行则被忽略

- `pattern, p`

用于定义细胞自动发生器初始状态,从指定字符串开始作为起始行

每个非空字符作为一个细胞直到换行（或者字符串结束）,更多的行被忽略

- `rate, r`

设置视频帧率,默认25

- `random_fill_ratio, ratio`

设置初始随机填充率,是浮点数,范围0-1,默认 $1/\text{PHI}$

此选项在指定了初始文件或模式时被忽略

- `random_seed, seed`

设置随机填充初始种子,必须是整数,范围0- `UINT32_MAX`。不指定或显式指定为-1,将尝试使用一个更好的随机种子

- `rule`

设置细胞自动发生规则,是0-255间数,默认110

- `size, s`

设置输出视频尺寸,语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节。

如果 `filename` 或者 `pattern` 指定了,则尺寸默认为初始化行的宽度 `width`,高为 `width * PHI`

如果尺寸被设置,其宽必须匹配 `pattern` 字符串中最大行。

如果 `filename` 和 `pattern` 都没有指定,则默认为 320x518 (用作随机生成初始化)

- `scroll`

如果为1，向上滚出已经填满的行。如果为0，到最后一行后，新行将覆盖第一行，默认为1

- start_full, full

如果设置为1，则需要完全填满后才输出第一帧，这时默认行为，设置为0则禁用

- stitch

如果设置为1，左和右连接在一起，这是默认行为，为0则禁用

cellauto例子

- 从 pattern 读取初始化，输出为 200x400.

```
cellauto=f=pattern:s=200x400
```

- 随机化输出初始化，宽200个细胞，填充率为2/3:

```
cellauto=ratio=2/3:s=200x200
```

- 以规则18创建一个由单细胞开始，初始化宽度为100的源：

```
cellauto=p=@:s=100x400:full=0:rule=18
```

- 指定一个详细的初始模式:

```
cellauto=p='@@ @ @':s=100x400:full=0:rule=18
```

mandelbrot

生成一个曼德尔勃特（Mandelbrot）集合分形，它逐渐从点(start_x,start_y)放大

支持下列选项：

- end_pts

设置终端 pts 值，默认400.

- end_scale

设置终端缩放值，必须是浮点数，默认0.3.

- inner

设置内部着色模式，该算法用于绘制曼德布洛特分形内部区域.

允许下面的值：

black

设置black模式.

convergence

设置时间收缩模式

mincol

设置基于点的颜色最接近的起源迭代

period

设置时间模式

默认为mincol.

- bailout

设置bailout值，默认为10

- maxiter

设置最大迭代执行的渲染算法，默认7189.

- outer

设置外部着色模式，允许下面的值：

iteration_count

设置为迭代计算模式

normalized_iteration_count

设置为规范化的迭代计算模式

默认为normalized_iteration_count.

- rate, r

设置帧率，可以是表达式和每秒帧数，默认为25

- size, s

设置帧尺寸，语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节. 默认"640x480".

- start_scale

设置初始化放大值，默认为3.0.

- start_x

设置初始化点的x坐标，必须是-100 到100间的浮点数，默认为 -0.743643887037158704752191506114774.

- start_y

设置初始化点的y坐标，必须是-100 到100间的浮点数，默认为-0.131825904205311970493132056385139.

mptestsrc

生成各种测试模式，以作为MPlayer测试滤镜。

生成视频是固定的256x256分辨率。这个源用于在特定编码功能测试

支持下面选项：

- rate, r

指定帧率，是默认每秒帧数数字。也可以以 `frame_rate_num/frame_rate_den` 格式设定整数和浮点数以及帧频短语都是有效值，默认25

- duration, d

设置持续时间秒数，语法同于[持续时间](#)章节，

如果不指定或者指定为负数，表示持续不断

- test, t

设置测试项的数字或者名称，允许下面的值：

`dc_luma dc_chroma freq_luma freq_chroma amp_luma amp_chroma cbp mv ring1 ring2 all`

默认为"all",表示都要测试

例如：

```
mptestsrc=t=dc_luma
```

将进行 `dc_luma` 测试

frei0r源

提供一个frei0r源

编译需要 frei0r 头以及配置项 `--enable-frei0r`，接受如下参数：

- size

生成视频大小。语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节。

- framerate

设置帧率，值为数字字符串，或者 `num/den` 形式字符串或者帧率短语

- filter_name

这个名字frei0r源到负载。获得有关frei0r的更多信息以及如何设置参数,读取文档中的frei0r视频滤镜部分。

- filter_params

由'|'分隔的参数列表传递给 `frei0r` 源

例如：要产生一个200x200分辨率，帧率为10，产生一个frei0r源用作 `partik01`

```
frei0r_src=size=200x200:framerate=10:filter_name=partik01:filter_params=1234 [overlay]; [in][overlay] overlay
```

life

产生life模式

这个源基于John Conway's life游戏

源输入一个网格、每个像素（代表细胞）可以有2个状态，活或者死。每个细胞有8个邻国水平、垂直或对角相邻。

根据采用的规则发展网格,它指定邻居活细胞的数量会使细胞生存或出生，这里 `rule` 选项在下面介绍。

这个源支持下面的选项：

- `filename, f`

设置读取初始化网格的文件。在文件中每个非空字符代表存活的细胞，换行结束一行。

如果没有指定则随机生成

- `rate, r`

设置视频帧率，默认25.

- `random_fill_ratio, ratio`

设置随机初始化随机网格填充率，值为0-1的浮点数，默认为 $1/\text{PHI}$ ，在设置了 `filename` 时忽略

- `random_seed, seed`

设置随机种子，值为0 - `UINT32_MAX` 如果设置为-1或者不设置，表示尽量用优化的种子

- `rule`

设置规则

规则可以是指定代码的形式"`SNS/BNB`"，这里 `NS` 和 `NB` 是0-8的数字序数，`NS` 在一个存活细胞周围还存活的细胞数，`NB` 指定周围要新生的细胞数，`s` 和 `b` 分别是 `S` 和 `B` 的替代

另外一个规则可以被描述为18位的整数。其中高段9位表示存活细胞周围存活细胞状态数，低段9位则为要新生的细胞状态数。例如数字6153=(12<<9)+9，表示细胞周围有12个存活细胞，新生为9的规则，其等效于"`S23/B03`".

默认为"`S23/B3`",它是原始的Conway's 游戏规则。如果它周围有2或者3个细胞将新生细胞，否则将死亡细胞

- `size, s`

设置输出视频分辨率，语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节

当 `filename` 被设定，则默认会采用输入文件的最大行宽。如果设置了这个值则需与输入文件相匹配。

如果没有设置 `filename` 则默认为 "320x240" (用于随机初始化模式).

- `stitch`

如果设置为1，则左右网格边和上下网格边缝合在一起（连续面），默认为1

- `mold`

设置细胞分解速度。如果设置，则为死细胞将从 `death_color` 在 `mold` 步骤内转变为 `mold_color` 的速度。范围0-255。

- `life_color`

设置存活的细胞颜色 (或新生)

- `death_color`

设置死亡细胞颜色。如果 `mold` 被设置，则为死亡后第一个颜色

- `mold_color`

设置分解后颜色，作为绝对死亡或已被分解的细胞颜色

前面3个颜色选项其语法可同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节

life源例子

- 从模板读取一个网格，分辨率为300x300:

```
life=f=pattern:s=300x300
```

- 填充率2/3的随机初始化，尺寸200x200, :

```
life=ratio=2/3:s=200x200
```

- 指定一个规则的随机初始化和生成:

```
life=rule=S14/B34
```

- 前面所有例子，且还伴有 `mold` （分解）效果，在ffplay中播放:

```
ffplay -f lavfi
```

```
life=s=300x200:mold=10:r=60:ratio=0.1:death_color=#C83232:life_color=#00ff00,scale=1200:800:flags=16
```

color, haldclutsrc, nullsrc, rgbtestsrc, smptebars, smptehdbars, testsrc

- `color` 源提供一致的颜色输入
- `haldclutsrc` 源提供 哈尔德（Hald）CLUT输入，参考[haldclut](#)滤镜
- `nullsrc` 源返回未处理的视频帧，它主要用于分析/调试调试，或者作为滤镜中可以忽略的输入数据
- `rgbtestsrc` 源产生 RGB 测试模板，用于检测对比 RGB 与 BGR 问题，应该可以看到一个红色、绿色和蓝色的从上到下条纹。
- `smptebars` 源产生颜色条模板，基于 SMPTE Engineering Guideline EG 1-1990 标准
- `smptehdbars` 源产生颜色条模板，基于 SMPTE RP 219-2002 标准
- `testsrc` 源产生测试视频模板，显示颜色模板和滚动的梯形以及时间戳。主要用于测试目的。

这些源支持下面一些参数：

- `color, c`

指定源颜色，仅作 `color` 源中有效，语法同于[颜色](#)

- level

指定Hald CLUT的层次。仅在 `haldclutsrc` 有效。S。level 中的 N 用于生成一个 $N*N*N$ 像素为单位矩阵用于三维查找表。每个组件都是编码在 $1/(N * N)$ 范围内。

- size, s

指定源视频尺寸。语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节，默认值为 `320x240`。

这个选项在 `haldclutsrc` 中无效

- rate, r

设置帧率，语法同于 `ffmpeg-utils` 手册中的[视频帧率](#)章节，默认为25

- sar

设置样品长宽比（像素点长宽比）

- duration, d

设置源视频持续时间，语法同于[持续时间](#)章节。

如果不设置或者设置为负数，表示持续存在。

- decimals, n

设置屏幕时间戳的小数数字显示，仅在 `testsrc` 源有效

显示的时间戳值将对应于原来的时间戳值乘以 $10^{\text{X次方数}}$ 的指定值。默认为0

例如：

```
testsrc=duration=5.3:size=qcif:rate=10
```

产生5.3秒的视频，视频尺寸是176x144，帧率为10

下面则产生红色源，有0.2透明度，尺寸为 `qcif`（176x144），帧率为10

```
color=c:red@0.2:s=qcif:r=10
```

如果输入内容会被忽略，`nullsrc` 可以被使用，下面的命令通过 `geq` 滤镜产生飞机飞过水稻的噪音

```
nullsrc=s=256x256, geq=random(1)*255:128:128
```

额外命令

`color` 源还支持下面的命令：

- c, color

设置颜色来创建一个图像。其支持的语法同于[颜色](#)中的介绍。

39 视频槽

下面是当前有效的视频槽（池）介绍。

buffersink

缓冲视频帧，可作为滤镜链图中有效的结束点。

这个槽主要用于编程使用，特别是通过 `libavfilter/buffersink.h` 的接口或选择操作系统

它接受指向 `AVABufferSinkContext` 结构的指针，用于定义传入缓冲区的格式，作为不透明参数传递给 `avfilter_init_filter` 以初始化。

nullsink

Null（空）视频槽，绝对没有输入的视频。它主要用作模板以分析/调试工具。

40 多媒体滤镜

下面介绍当前有效的多媒体滤镜

avectorscope

转换输入音频到视频输出以代表音频矢量范围（一种图形化音频处理）

这个滤镜用来测量立体声音频中两路音频间的区别。如果是单声道信号做成的2个声道（左右耳声道），因为两路完全相同（其实只有1路），所以输出是一个垂直的直线（表示二者无差别）。如果是立体声信号（两路肯定或多或少有差别），则创建一个利萨如（Lissajous）图形，其水平看，线长度与相位等表征了两个声道差异情况。

滤镜接受如下选项：

- mode, m

设置矢量显示模式，有效值为：

'lissajous'

利萨如(Lissajous)旋转45度。

'lissajous_xy'

同上，但不旋转。

默认'lissajous'。

- size, s

设置输出视频尺寸。语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节。默认400x400。

- rate, r

设置输出帧率，默认25。

- rc
- gc
- bc

分别用于指定红、绿、蓝颜色（表征对比图形的颜色）。分别默认为 40, 160 和80. 各个值的范围在 [0, 255].

- rf
- gf
- bf

分别指定淡入淡出时红、绿、蓝颜色， 分别默认值为15, 10和5.分别值范围为[0, 255].

- zoom

设置缩放因子，默认为1，允许[1, 10].

avectorscope例子

- 利用ffplayer播放

```
ffplay -f lavfi 'amovie=input.mp3, asplit [a][out1];
```

```
[a] avectorscope=zoom=1.3:rc=2:gc=200:bc=10:rf=1:gf=8:bf=7 [out0]'
```

concat

连接音频和视频流,把加入的一个接一个的在一起

这个滤镜用于按段同步视频和音频流。所有的段都必须有相同的流（类型和个数），输出也是相同流（类型和个数）

滤镜支持如下选项：

- n

设置段的数量，默认为2

- v

设置输出中视频流数量，则每个段中必须有输入视频流的数量。默认为1

- a

设置输出中音频流数量，则每个段中必须有输入音频流的数量，默认为0

- unsafe

激活不安全模式，这时如果段中有不同格式不会失败

滤镜有 `v+a` 的输出：先是一个视频输出，然后是音频输出。

有 `n x (v+a)`：有n段输出，每段都是 `v+a`。

相关的流并不总是有相同的时间，由于各种原因还包括不同的编解码帧大小或创作草稿。因此相关同步流（视频和对应音频）要连接，`concat` 滤镜将选择持续最长的流（视频的）为基准（除最后段的流），在每个流播放时通过让音频流垫长（重复部分）或者静默（截断）来实现视频流连续。

为了让滤镜工作正常，所有段都必须以0为时间戳开始。

所有应用的流在所有共同的领域必须有相同的参数，滤镜会自动选择一个通用的像素格式（色彩标注、编码颜色的标准和位深等），以及音频采样率和通道布局。但其他设置如视频分辨率必须由用户显式转换。

不同的帧率是可以接受的，但会导致输出帧率的变化，一定要配置输出文件来处理。

concat例子

- 连接一个有双语版的多段视频成为整体（视频中0号流，音频在1号和2号流）

```
ffmpeg -i opening.mkv -i episode.mkv -i ending.mkv -filter_complex \
'[0:0] [0:1] [0:2] [1:0] [1:1] [1:2] [2:0] [2:1] [2:2] concat=n=3:v=1:a=2 [v] [a1] [a2]' \
-map '[v]' -map '[a1]' -map '[a2]'
output.mkv
```

- 连接2个部分，分别处理音频和视频，使用电影来源标准调整分辨率。

```
movie=part1.mp4, scale=512:288 [v1] ; amovie=part1.mp4 [a1] ;movie=part2.mp4, scale=512:288 [v2]
;movie=part2.mp4 [a2] ;[v1] [v2] concat [outv] ; [a1] [a2] concat=v:0:a=1 [outa]
```

注意在开始的不同步现象会发生在音频和视频没有完全相同持续时间情况下。

ebur128

EBU R128 扫描滤镜，这个滤镜需要一个音频流，但会原样输出。默认情况下，会显示10H更新频率下，的瞬时响度（M）、短期响度（S）、集成响度（I）和响度范围（LRA）

滤镜有一个实时视频输出，展示响度变化。因为上述图像不停更新，所以它不可打印化输出，除非详细日志被设置。主要的绘图区域包含短期响度（3秒分析），以及其后的瞬时响度（400毫秒）

关于EBU R128响度滤镜的更多信息参考<http://tech.ebu.ch/loudness>

滤镜介绍如下选项：

- video

激活视频输出。无论怎么设置，音频流在过程中都不会变化（从输入到输出），如果激活，则视频将作为第一个输出流。默认为0

- size

设置视频尺寸。语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节。默认640x480.

- meter

设置EBU的规模计。默认为9，通用值是9和18，对应于规模计+9和+18.任何非负整数是可以的。

- metadata

设置添加的元数据。如果为1，音频输入将被划分为100毫秒的输出帧。他们每个都包含各种响应信息元数据。所有的元数据键前又缀有lavfi.r128..

默认为0.

- framelog

强制帧日志层次，允许值：

'info'

信息日志层次

'verbose'

冗长日志层次

默认为 `info`，如果设置视频或者元数据则选择 `verbose`

- peak

设置峰顶模式

有效的模式可以累积（可选标志类型），可能的值：

‘none’

没有任何峰顶模式（默认）

‘sample’

允许sample-peak模式 mode。

简单的峰值模式，其只寻找高样本值。在日志中有一个`sample-peak`消息（标记有SPK）。

‘true’

允许true-peak模式

如果启用，峰值查找是一个over-sampled版本的输入流，其峰值精度更佳，在日志中有一个`true-peak`消息（标记有TPK），且每个帧有一个`true`

ebur128例子

- 利用EBU 放缩规模计+18d 的实时图像

```
ffplay -f lavfi -i "amovie=input.mp3,ebur128=video=1:meter=18 [out0][out1]"
```

- 在ffmpeg中分析

```
ffmpeg -nostats -i input.mp3 -filter_complex ebur128 -f null -
```

interleave和aintterleave

从多个输入中暂时交错帧，`interleave` 用于视频输入，`aintterleave` 用于音频输入。

这个滤镜从多个输入读取帧，然后然后按一定顺序（最老的先发送）发送给输出。

输入流必须有定义良好的、递增的帧时间戳。

为了提交一帧输出,滤镜需至少在一个框架内为每个输入排序,所以不能工作在一个输入还没有终止情况,也不会接收传入帧（持续生存的）

例如考虑一个输入是 `select` 滤镜其可以按帧丢弃，`interleave` 滤镜会保持从输入读取，但不发送任何帧到输出直到输入发送来 `end-of-stream` 信号。

另外，根据输入同步，滤镜将丢帧，以防止一个输入收到比其他帧(时间戳)偏离太多，和队列满时

滤镜接受下面选项：

- `nb_inputs, n`

设置不同数量的输入，默认为2

interleave和ainterleave例子

- 使用ffmpeg交错不同的流：

```
ffmpeg -i bambi.avi -i pr0n.mkv -filter_complex "[0:v][1:v] interleave" out.avi
```

- 添加闪烁模糊效果：

```
select='if(gt(random(0), 0.2), 1, 2)':n=2 [tmp], boxblur=2:2, [tmp] interleave
```

perms和aperms

为输出帧设置读/写权限

这些滤镜主要针对开发人员测试下面的滤镜链图的直接路径滤镜

这个滤镜接受下面选项：

- mode

选择权限模式，接受下面的值：

'none'

什么都不做，是默认值

'ro'

设置所有输出只读

'rw'

设置所有输出可读写

'toggle'

翻转，即原来是只读则为可读写，如果原来是可读写则为只读

'random'

随机为只读或者可读写

- seed

设置 mode 为 random 时的随机种子。必须为0- `UINT32_MAX` 间的整数。如果不指定，或者指定为-1，则滤镜尝试自动选择一个好的随机种子。

注意实际在许可- permission 滤镜和其跟随的滤镜之间可能会自动插入滤镜，则许可滤镜可能不会有预期的效果给随后的滤镜，要避免这样的问题，可以在许可滤镜之前插入 `format` 或者 `aformat` 滤镜（分别针对 `perms` / `aperms`）。

select和aselect

选择一些帧来输出

滤镜接受下面的选项：

- `expr, e`

设置一个表达式，它拥有对每个输入帧进行评估（计算）

如果表达式计算结果为0，则该帧被丢弃

如果计算结果为负或者NaN，则帧被发送给第一个输出，否则发送给指定数 $\text{ceil}(\text{val})-1$ 的输出,假设输入索引从0开始。

例如 值为1.2会被发送给 $\text{ceil}(1.2)-1 = 2-1 = 1$, 即第2路输出。

- `outputs, n`

设置输出路数。输出对应于选择帧输出的结果，默认为1

表达式允许下面的内容:

- `n`

顺序化的滤镜帧，从0开始

- `selected_n`

被选择的（顺序化）帧，从0开始计数

- `prev_selected_n`

前一选择（顺序）帧，如果未定义则为 `NAN`。

- `TB`

输入时间戳的时间基准

- `pts`

PTS，滤镜后视频帧（表示时间戳）分，采用TB为单位，如果未定义则为 `NAN`

- `t`

PTS按秒为单位，滤镜后视频帧（时间戳），如果为定义则为 `NAN`

- `prev_pts`

前一滤镜后视频帧PTS，如果为定义则为 `NAN`

- `prev_selected_pts`

前面最后的滤镜视频帧的PTS，如果为定义则为 `NAN`

- `prev_selected_t`

前面最后被选择的滤镜后视频帧的PTS，如果为定义则为 `NAN`

- start_pts

视频中第一个视频帧的PTS， 如果为定义则为 NAN

- start_t

视频的起始时间. 如果为定义则为 NAN

- pict_type (video only)

滤镜帧的类型,它可以是下面的值:

I P B S SI SP BI

- interlace_type (video only)

帧交错类型，可以是下面的值:

PROGRESSIVE

逐行（非交错）.

TOPFIRST

top-field-first 类型帧

BOTTOMFIRST

bottom-field-first 类型帧

- consumed_sample_n (audio only)

当前帧之前选定的样本的数量

- samples_n (audio only)

样品在当前帧的数量

- sample_rate (audio only)

输入样本率

- key

为1则 滤镜框架是一个帧， 否则为0

- pos

滤镜帧在文件中的偏移位置， -1为不可用（如合成视频）

- scene (video only)

值在0-1间， 表面一个新场景。 较低的值反映当前帧的低概率引入一个新场景， 更高的值意味着更可能是同一个（场景，

见下面的例子)

表达式的默认值是1

select和aselect例子

- 选择所有的输入

```
select
```

这个例子等效于:

```
select=1
```

- 跳过所有的帧:

```
select=0
```

- 仅选择I帧:

```
select='eq(pict_type\,I)'
```

- 每100帧选择1帧:

```
select='not(mod(n\,100))'
```

- 只选择10-20时间间隔的帧:

```
select=between(t\,10\,20)
```

- 只选择10-20时间间隔的I帧:

```
select=between(t\,10\,20)*eq(pict_type\,I)
```

- 选择帧的最小距离为10秒:

```
select='isnan(prev_selected_t)+gte(t-prev_selected_t\,10)'
```

- 使用 `aselect` 选择包含采样大于100的音频帧:

```
aselect='gt(samples_n\,100)'
```

- 创建一个马赛克开始场景:

```
ffmpeg -i video.avi -vf select='gt(scene\,0.4)',scale=160:120,tile -frames:v 1 preview.png
```

对比场景对一个值在0.3和0.5之间通常是一个合理的选择

- 奇数和偶数帧分别发送给不同的输出，并组合:

```
select=n=2:e='mod(n, 2)+1' [odd][even]; [odd] pad=h=2*ih [tmp]; [tmp][even] overlay=y=h
```

sendcmd和asendcmd

在滤镜链图中向滤镜发送命令

这个滤镜读取命令来发送给滤镜链图中的滤镜

其中 `sendcmd` 用于两个视频滤镜间，`asendcmd` 用于两个音频滤镜间。除此之外它们采用相同方式。

命令可以是规范的滤镜 `commands` 选项的参数或者指定在 `filename` 文件选项所指文件中。

它接受如下选项：

- `command, c`

设置要发送的命令

- `filename, f`

指定一个文件，读取其中命令进行发送

sendcmd和asendcmd命令语法

命令是一个序列间隔规则（由 `;` 分隔），包括一个特定事件时段执行的命令列表。触发事件通常是当前帧的时间达到或者离开一个给定的时间段。

一个时间段命令由以下语法描述：

```
START[-END] COMMANDS;
```

时间段描述由 `START` 和 `END`（可选，默认为最大时间）指定

判断当前帧是否在指定时间段 `[START,END)`，既判断当前帧的时间是否大于等于 `START`，且小于 `END`。

这里 `COMMANDS` 是对应于时间段的一个或者由 `;` 分隔的多个命令序列。一个命令的语法规范是：

```
[FLAGS] TARGET COMMAND ARG
```

这里 `FLAGS` 是可选的，用于指定事件类型与指定时间段关系的（命令发生发送频次），必须由非空标识符，以及 `+` 或 `|` 和 `[]` 封闭的字符，标志有：

- `enter`

命令在当前时间戳达到时间段时发送，换句话说命令发送的前一帧时间戳还不满足时间段要求，而当前已经是满足要求的

- `leave`

命令在当前时间戳离开时间段时发送，换句话说，命令发送时的前一帧还满足时间段要求，而当前已经不满足了

如果 `FLAGS` 没有指定，则默认为 `[enter]`。

这里 `TARGET` 是指定命令发送的目标，通常为滤镜类型名字或者滤镜实例名字。而 `COMMAND` 是发送给目标的命令名字，`ARG` 是对应的选项参数。

一个简单的BNF 规格的命令描述语法如下：

```
COMMAND_FLAG ::= "enter" | "leave"
COMMAND_FLAGS ::= COMMAND_FLAG [(+|"|")COMMAND_FLAG]
COMMAND ::= "[" COMMAND_FLAGS "]" TARGET COMMAND [ARG]
COMMANDS ::= COMMAND [,COMMANDS]
```

```
INTERVAL      ::= START[ -END] COMMANDS
INTERVALS     ::= INTERVAL[ ;INTERVALS]
```

sendcmd和asendcmd例子

- 指定音频节奏（tempo）变化从4秒开始：

```
asendcmd=c='4.0 atempo tempo 1.5',atempo
```

- 帧文件中指定一个 drawtext 和 hue 命令

```
# 在时间5-10间显示一个文字 5.0-10.0 [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=hello world',
```

```
[leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=';
```

```
# 在时间15-20淡化图像 15.0-20.0 [enter] hue s 0,
```

```
[enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=nocolor',
[leave] hue s 1,
[leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=color';
```

```
# 从时间25开始应用指数饱和和渐变效果 25 [enter] hue s exp(25-t)
```

滤镜链图中读取然后处理前面的命令列表（存在 test.cmd 文件中），可以是如下处理：

```
sendcmd=f=test.cmd,drawtext=fontfile=FreeSerif.ttf:text=",hue
```

setpts和asetpts

修改发布时间戳（PTS-presentation timestamp），对于输入帧。

其中 setpts 对于视频帧， asetpts 对于音频帧

滤镜允许如下选项：

- expr

指定用于计算时间戳的表达式

表达式通过 eval API和下面一些常量来计算：

- FRAME_RATE

帧率，仅对于指定了帧率的视频

- PTS

输入的PTS

- N

对输入视频帧计数或已经消耗的样本计数，不包括音频的当前帧，从0开始

- NB_CONSUMED_SAMPLES

采样数（因为音频是按固定频率采样，则一个采样其实就是自然的计时单位——类似帧率），不包括当前帧（仅对音频）

- NB_SAMPLES, S

当前帧中的采样数 (仅音频)

- SAMPLE_RATE, SR

音频采样率

- STARTPTS

第一帧的PTS

- STARTT

第一帧的按秒时间

- INTERLACED

指示当前帧是否交错

- T

当前帧的按秒时间

- POS

初始位置在文件中的偏移，为当前帧，如果为定义则本值也为定义

- PREV_INPTS

前一个帧的PTS.

- PREV_INT

前一帧按秒时间

- PREV_OUTPTS

前一帧的输出PTS.

- PREV_OUTT

前一帧按秒输出时间

- RTCTIME

时间单位为微秒-microseconds. 现在被弃用，使用时间(0)时

- RTCSTART

影片开始时间以微秒为单位

- TB

输入时间戳时基

setpts和asetpts例子

- 从0开始计数PTS

`setpts=PTS-STARTPTS`

- 应用快速监看效果

`setpts=0.5*PTS`

- 应用慢速监看效果

`setpts=2.0*PTS`

- 强制为25的帧率:

`setpts=N/(25*TB)`

- 设置有抖动的25帧率:

`setpts='1/(25*TB) (N + 0.05 sin(N*PI/25))'`

- 应用一个10秒的输入偏置:

`setpts=PTS+10/TB`

- 从直播源和变基生成时间戳转换到当前时基时间戳:

`setpts='(RTCTIME - RTCSTART) / (TB * 1000000)'`

- 按当前采样率生成时间戳:

`asetpts=N/SR/TB`

settb和asettb

设置输出帧时间戳的时基。它主要用于测试时基配置。

其可以接受如下选项：

- `expr, tb`

用于计算输出时基的表达式或者值

这里的 `tb` 值是一个可得出有理数值的表达式或者值。常数可以包含“AVTB”(默认时基),“intb”(输入时基)和“sr”(采样率,只音频)。默认值是“intb”

settb和asettb例子

- 设置时基为1/25

`settb=expr=1/25`

- 设置时基为1/10

settb=expr=0.1

- 设置时基为1001/1000

settb=1+0.001

- 设置时间为2倍intb

settb=2*intb

- 设置为默认值

settb=AVTB

showcqt

转换音频输入为一个频谱视频输出（用恒Q变换Brown-Puckette算法），乐音的规模从 E0 到 D#10 (10个8度)

滤镜接受如下选项：

- volume

指定转换表达式（对音量的乘数），可含如下变量：

frequency, freq, f

转换中评估的频率

timeclamp, tc

timeclamp选项值

和下面的函数：

a_weighting(f)

一个权重等响度A-weighting

b_weighting(f)

一个权重等响度B-weighting

c_weighting(f)

一个权重等响度C-weighting

默认值16

- tlength

指定转换长度表达式，表达式可包含如下变量：

frequency, freq, f

转换中评估的频率

timeclamp, tc

timeclamp选项值

默认值 $384/f*tc/(384/f+tc)$.

- timeclamp

指定转换timeclamp，对于低频，精度在时间域和频率域间权衡，如果timeclamp较低,在时间域表示更准确(如快速低音鼓),否则在频域表示(如低音吉他)更准确。可接受的值是[0.1,1.0]。默认值是0.17。

- coeffclamp

指定转换coeffclamp. 如果coeffclamp低，转换更准确,否则转换速度更快。可接受的值是[0.1,10.0]。默认值是1.0。

- gamma

指定gamma。低的 gamma 值有更多比较细节高的则有更大的比较范围，可接受值范围 [1.0, 7.0]默认为3.0.

- gamma2

指定采用柱状图像gamma，接受值范围 [1.0, 7.0]. 默认为 1.0.

- fontfile

指定freetype字体文件，如果不知道则采用嵌入的字体

- fontcolor

指定文字颜色。是一个返回整数的算术表达式值-0xRRGGBB可以包含下面的变量：

frequency, freq, f

转换中评估的频率

timeclamp, tc

timeclamp选项值

和函数：

midi(f)

midi数频率，一些midi数为：E0(16), C1(24), C2(36), A4(69)

$r(x)$, $g(x)$, $b(x)$

红、绿、蓝的值

默认值为 $st(0, (midi(f)-59.5)/12)$; $st(1, \text{if}(\text{between}(ld(0), 0, 1), 0.5-0.5\cos(2\pi*ld(0)), 0))$; $r(1-ld(1)) + b(ld(1))$

- fullhd

如果为1 (默认值), 视频分辨率为1920x1080 (full HD), 如果设置为0, 则视频分辨率为960x540, 使用这个值可以降低CPU使用

- fps

指定视频帧率——fps, 默认25.

- count

指定每帧中的转换数, *fpscount*是1秒中的转换数。注意音频数据率必须是*fpscount*的整数倍。默认为6

showcqt例子

- 播放音频显示频谱：

```
ffmpeg -f lavfi 'amovie=a.mp3, asplit [a][out1]; [a] showcqt [out0]'
```

- 同上，但使用30的帧率：

```
ffmpeg -f lavfi 'amovie=a.mp3, asplit [a][out1]; [a] showcqt=fps=30:count=5 [out0]'
```

- 频谱为960x540和降低CPU使用率：

```
ffmpeg -f lavfi 'amovie=a.mp3, asplit [a][out1]; [a] showcqt=fullhd=0:count=3 [out0]'
```

- A1 和其谐波: A1, A2, (near)E3, A3:

```
ffmpeg -f lavfi 'aevalsrc=0.1sin(2PI55t)+0.1sin(4PI55t)+0.1sin(6PI55t)+0.1sin(8PI55t),
```

```
asplit[a][out1]; [a] showcqt [out0]'
```

- 类似上面，但帧频域更准确（更慢）：

```
ffmpeg -f lavfi 'aevalsrc=0.1sin(2PI55t)+0.1sin(4PI55t)+0.1sin(6PI55t)+0.1sin(8PI55t),
```

```
asplit[a][out1]; [a] showcqt=timeclamp=0.5 [out0]'
```

- B-weighting等响度

```
volume=16*b_weighting(f)
```

- 低的Q因子

```
tlength=100/f*tc/(100/f+tc)
```

- 定制字体颜色， C-note为green,其他为blue

```
fontcolor='if(mod(floor(midi(f)+0.5),12), 0x0000FF, g(1))'
```

- 定制gamma,现在有光谱线性振幅

```
gamma=2:gamma2=2
```

showspectrum

转换音频输入为一个视频频谱

滤镜接受如下选项：

- size, s

指定视频尺寸。语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节。默认为640x512.

- slide

指定窗口中光谱如何滑动，有如下可能值：

‘replace’

```
当滑动到最右边后又从左开始 reach the right
```

‘scroll’

```
向左滚动
```

‘fullframe’

```
帧在时间到达时正确生成
```

默认为replace.

- mode

指定显示模式，接受如下值

‘combined’

```
所有的通道在一个行中显示
```

‘separate’

```
通道显示在各自行中
```

默认为‘combined’.

- color

指定显示的颜色模式，接受下面的值：

‘channel’

每个通道采样指定的颜色显示

‘intensity’

每个通道采样相同的配色方案

默认为‘channel’.

- scale

指定用于计算强度的颜色值，允许如下的值：

‘lin’

线性

‘sqrt’

平方根, 默认

‘cbrt’

立方根

‘log’

对数

默认‘sqrt’.

- saturation

设置显示颜色饱和度修饰符，负值提供可供选择的配色方案，0没有饱和，值范围为[-10.0, 10.0]，默认为1.

- win_func

设置窗口函数，接受如下值：

‘none’

没有预处理（这时最快的了）

`'hann'`

Hann窗口

`'hamming'`

Hamming窗口

`'blackman'`

Blackman窗口

默认为 hann

它类似于 `showwaves` 滤镜，看下面的例子

showspectrum例子

- 采样对数颜色放缩的大窗口

```
showspectrum=s=1280x480:scale=log
```

- 使用ffplay每通道颜色和滑动频谱的完整示例

```
ffplay -f lavfi 'amovie=input.mp3, asplit [a][out1];
```

```
[a] showspectrum=mode=separate:color=intensity:slide=1:scale=cbrt [out0]'
```

showwaves

转换音频输入为视频输出（代表采样波形），接受如下选项：

- size, s

指定视频尺寸，语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节。默认为600x240.

- mode

设置显示模式，允许下面的值：

`'point'`

每个采样画一个点。

`'line'`

每个采样画一条直线

'p2p'

每个采样画一个点，且用直线连起来。

'cline'

每个采样画一个中垂直线

默认为 `point`

- `n`

设置一列中采样数量。一个更大的值将降低帧率。必须是正整数。它可以根据帧率计算而不用显式指定

- `rate, r`

设置输出帧率(近视)，其可以起到 `n` 选项左右，默认为25

- `split_channels`

是否分别通道或者覆盖，默认为0

showwaves例子

- 同时输出音频和其对应视频表示:

```
amovie=a.mp3,asplit[out0],showwaves[out1]
```

- 利用 `showwaves` 创建一个同步信号并显示，帧率为30:

```
aevalsrc=sin(12PIt)sin(8802PIt):cos(2PI200t),asplit[out0],showwaves=r=30[out1]
```

showwavespic

转换音频输入为视频输出（代表采样波形），接受如下选项：

- `size, s`

指定视频尺寸，语法同于 `ffmpeg-utils` 手册中的[视频尺寸](#)章节。默认为600x240.

- `split_channels`

是否分别通道或者覆盖，默认为0

showwavespic例子

- 利用ffmpeg按通道提取波形，并展示在1024x800图片中

```
ffmpeg -i audio.flac -lavfi showwavespic=split_channels=1:s=1024x800 waveform.png
```

split和asplit

从输入中选择一些来输出

其中 `asplit` 对于音频工作，`split` 对于视频工作

这个滤镜接受单个指定输出个数的参数，如果不指定，默认为2

split和asplit例子

- 对一个输入创建2给相同的输出:

```
[in] split [out0][out1]
```

- 创建3给或者更多的输出，你需要为每个输出指定标签，如下:

```
[in] asplit=3 [out0][out1][out2]
```

- 创建2个输出，其中一个作为 `cropped` 的输入，一个作为 `pad` 的输入:

```
[in] split [splitout1][splitout2]; [splitout1] crop=100:100:0:0 [cropout]; [splitout2] pad=200:200:100:100 [padout];
```

- 对输入音频制作5份拷贝:

```
ffmpeg -i INPUT -filter_complex asplit=5 OUTPUT
```

zmq和azmq

接受从 `libzmq` 客户端发送来的命令，并将它们发送给滤镜链图中的滤镜。

这里 `zmq` 和 `azmq` 都是直通滤镜，只是 `zmq` 工作于两个视频滤镜间，而 `azmq` 工作于两个音频滤镜间。

编译需要 `libzmq` 库以及头文件，并采样 `--enable-libzmq` 配置

关于 `libzmq` 的更多信息参考<http://www.zeromq.org/>

滤镜 `zmq` 和 `azmq` 都需要 `libzmq` 服务，它通过网络来发送定义于 `bind_address` 选项的接收消息接口

接收的消息必须有如下格式：

```
TARGET COMMAND [ARG]
```

这里 `TARGET` 指定命令目标，通常是滤镜类名或者滤镜实例名，`COMMAND` 为命令，`ARG` 为可选的参数列表（对于 `COMMAND`）。

在接收,处理信息和相应的命令，并注入滤镜链图，根据结果，滤镜发送一个返回信息给客户端，采样的格式为：

```
ERROR_CODE ERROR_REASON
MESSAGE
```

其中 `MESSAGE` 是可选的。

zmq和azmq例子

参考 `tools/zmsend` 中的一个 `zmq` 客户端例子，它被用来向这些的滤镜发送命令

考虑以下用于`ffplay`的滤镜链图：

```
ffplay -dumpgraph 1 -f lavfi "  
color=s=100x100:c=red [l];  
color=s=100x100:c=blue [r];  
nullsrc=s=200x100, zmq [bg];  
[bg][l] overlay [bg+l];  
[bg+l][r] overlay=x=100 "
```

为了改变视频左边的颜色，下面的命令被使用：

```
echo Parsed_color_0 c yellow | tools/zmqsend
```

如果要改变右边：

```
echo Parsed_color_1 c pink | tools/zmqsend
```

41 多媒体源

下面是目前可用的多媒体源的描述

amovie

它同于 `movie` 源，除了它选择一个默认音频流。

movie

从影片内容中读取音频和/或视频流

接受下面的参数：

- filename

要被读取的资源名（不限于文件，可以是设备或者一些协议下的流）。

- format_name, f

对要读取的影片指定格式，可以是容器或者输入设备，如果没有指定，将从影片名中猜测。

- seek_point, sp

指定定位点，单位秒。表示输出的开始点。这个参数与 `av_strtod` 评估，所以数字可能后缀一个 `IS` 后缀。默认为0

- streams, s

指定要读取的流。一些流可以被指定，以 + 分隔。这时按顺序源有多个输出。语法同于ffmpeg手册中的[流说明符](#)章节。两个特殊明智 `dv` 和 `da` 指定默认的（最合适）的视频和音频流。在滤镜中调用 `amovie` 时默认为 `dv` 或者 `da`

- stream_index, si

指定要读取的视频流索引号。如果值为 -1，则最适合的视频流被自动选择。默认为 -1。现在已弃用。如果滤镜中调用 `amovie` 将自动选择带音频的视频。

- loop

指定循环次数，如果超过1，则流会重复读取处理指定次数，默认为1。

注意当影片重新读取时内置的时间戳并不改变，所以会产生非递增的时间戳。

movie源例子

- 在文件in.avi中跳过3.2秒，并覆盖输入标签"in":

```
movie=in.avi:seek_point=3.2, scale=180:-1, setpts=PTS-STARTPTS [over]; [in] setpts=PTS-STARTPTS [main]; [main]
[over] overlay=16:16 [out]
```

- 从 `video4linux2` 设备读取，并覆盖输入标签"in":

```
movie=/dev/video0:f=video4linux2, scale=180:-1, setpts=PTS-STARTPTS [over]; [in] setpts=PTS-STARTPTS [main];
[main][over] overlay=16:16 [out]
```


- 从dvd.vob读取0号流（视频流）和id为 0x81 的音频流，视频流连接到标签 video，音频连接到标签 audio：

```
movie=dvd.vob:s=v:0+#0x81 [video] [audio]
```

42 参考/看

可以单独的看FFmpeg工具集中各个单独工具的介绍，主要有

- ffmpeg
- ffplay
- ffprobe
- ffserver
- ffmpeg-utils
- ffmpeg-scaler
- ffmpeg-resampler
- ffmpeg-codecs
- ffmpeg-bitstream-filters
- ffmpeg-formats
- ffmpeg-devices
- ffmpeg-protocols
- ffmpeg-filters

43 开发者

是FFmpeg的开发者

关于作者的详细信息，可以观看项目（[git://source.ffmpeg.org/ffmpeg](https://source.ffmpeg.org/ffmpeg)）的 `git` 历史，或者项目中目录中使用命令 `git log` 了解，或者浏览在线的源码(<http://source.ffmpeg.org>)

源代码树中维护者文件(`MAINTAINERS`)列出了特定组件的维护人员

另本文档英文版本由 `makeinfo` 于2015年6月16日生成

汉化翻译补充说明

由xdsnet（xdsnet at gmail dot com）在2015年6月16日英文版基础上进行汉化翻译。

- 因为个人水平有限，翻译作品仅为粗略的参考以进行交流，因为ffmpeg文档以英语写作，所以建议以-英语原意为准。本人对因参照本翻译的原因造成的一切后果概不负责。
- 因为ffmpeg是还在发展的项目，所以不保证本文档适用于ffmpeg及其文档新版本。
- 本翻译项目部署在<https://github.com/xdsnet/other-doc-cn/>，是一个完全公开的项目，且不以盈利为目的。欢迎转载，希望完整保留“汉化翻译补充说明”部分（即使是部分提取个别章节也请补充上）
- 欢迎就翻译进行交流，联系方式见前。