第一部分

FFmpeg 的命令行使用篇

第一部分主要介绍 FFmpeg 的命令行使用，在使用 FFmpeg 命令行之前，首先需要了解 FFmpeg 的发展过程，搭建 FFmpeg 的使用环境，比如编译 FFmpeg、生成文档、查找说明文档等，相关内容在第 1 章和第 2 章均会有详细的介绍，从第 3 章开始将会进入稍微深入的使用环节，由浅入深，讲解如何使用 FFmpeg 实现流媒体应用中的常见功能。

第 1 章
FFmpeg 简介

1.1 FFmpeg 的定义

FFmpeg 既是一款音视频编解码工具，同时也是一组音视频编解码开发套件，作为编解码开发套件，它为开发者提供了丰富的音视频处理的调用接口。

FFmpeg 提供了多种媒体格式的封装和解封装，包括多种音视频编码、多种协议的流媒体、多种色彩格式转换、多种采样率转换、多种码率转换等；FFmpeg 框架提供了多种丰富的插件模块，包含封装与解封装的插件、编码与解码的插件等。

FFmpeg 中的“FF”指的是“Fast Forward”，曾经有人写信给 FFmpeg 的项目负责人，询问“FF”是不是代表“Fast Free”或者“Fast Fourier”的意思。FFmpeg 中的“mpeg”则是人们通常理解的 Moving Picture Experts Group（动态图像专家组），FFmpeg 是一个很全面的图像处理套件。其实从 2000 年发展至今，FFmpeg 中的“FF”已经可以用各种组合进行理解，因为 FFmpeg 的强大足以支撑这些意义。

1.2 FFmpeg 的历史

想要深入了解一个软件、一个系统，首先要了解其发展史，下面就来介绍一下 FFmpeg 的整体发展过程。

FFmpeg 由法国天才程序员 Fabrice Bellard 在 2000 年时开发出初版；后来发展到 2004 年，Fabrice Bellard 找到了 FFmpeg 的接手人，这个人就是至今还在维护 FFmpeg 的 Michael Niedermayer。Michael Niedermayer 对 FFmpeg 的贡献非常大，其将滤镜子系统 libavfilter 加入 FFmpeg 项目中，使得 FFmpeg 的多媒体处理更加多样、更加方便。在 FFmpeg 发布了 0.5 版本之后，很长一段时间没有进行新版本的发布，直到后来 FFmpeg 采用 Git 作为版本控制服务器以后才开始继续进行代码更新、版本发布，当然也是时隔多

年之后了；2011 年 3 月，在 FFmpeg 项目中有一些提交者对 FFmpeg 的项目管理方式并不满意，因而重新创建了一个新的项目，命名为 Libav，该项目尽管至今并没有 FFmpeg 发展这么迅速，但是提交权限相对 FFmpeg 更加开放；2015 年 8 月，Michael Niedermayer 主动辞去 FFmpeg 项目负责人的职务。Michael Niedermayter 从 Libav 中移植了大量的代码和功能至 FFmpeg 中，Michael Niedermayer 辞职的主要目的是希望两个项目最终能够一起发展，若能够合并则更好。时至今日，在大多数的 Linux 发行版本系统中已经使用 FFmpeg 来进行多媒体处理。

作为一套开源的音视频编解码套件，FFmpeg 可以通过互联网自由获取。FFmpeg 的源码 Git 库提供了多站同步的获取方式，具体如下。

- git://source.ffmpeg.org/ffmpeg.git
- http://git.videolan.org/?p=ffmpeg.git
- https://github.com/FFmpeg/FFmpeg

FFmpeg 发展至今，已经被许多开源项目所采用，如 ijkplayer、ffmpeg2theora、VLC、MPlayer、HandBrake、Blender、Google Chrome 等。DirectShow / VFW 的 ffdshow(外部工程）和 QuickTime 的 Perian（外部工程）也采用了 FFmpeg。由于 FFmpeg 是在 LGPL/GPL 协议下发布的（如果使用了 GPL 协议发布的模块则必须采用 GPL 协议），任何人都可以自由使用，但必须严格遵守 LGPL / GPL 协议。随着参与的人越来越多，FFmpeg 的发展也越来越快，至本书完稿，FFmpeg 已经发布到 3.3 版本。

1.3　FFmpeg 的基本组成

FFmpeg 框架的基本组成包含 AVFormat、AVCodec、AVFilter、AVDevice、AVUtil 等模块库，结构如图 1-1 所示。

图 1-1　FFmpeg 基本组成模块

下面针对这些模块做一个大概的介绍。

（1）FFmpeg 的封装模块 AVFormat

AVFormat 中实现了目前多媒体领域中的绝大多数媒体封装格式，包括封装和解封装，如 MP4、FLV、KV、TS 等文件封装格式，RTMP、RTSP、MMS、HLS 等网络协议封装格式。FFmpeg 是否支持某种媒体封装格式，取决于编译时是否包含了该格式的封装库。根据实际需求，可进行媒体封装格式的扩展，增加自己定制的封装格式，即在 AVFormat 中增加自己的封装处理模块。

（2）FFmpeg 的编解码模块 AVCodec

AVCodec 中实现了目前多媒体领域绝大多数常用的编解码格式，既支持编码，也支持解码。AVCodec 除了支持 MPEG4、AAC、MJPEG 等自带的媒体编解码格式之外，还支持第三方的编解码器，如 H.264（AVC）编码，需要使用 x264 编码器；H.265（HEVC）编码，需要使用 x265 编码器；MP3（mp3lame）编码，需要使用 libmp3lame 编码器。如果

希望增加自己的编码格式，或者硬件编解码，则需要在 AVCodec 中增加相应的编解码模块，关于 AVCode 的更多相关信息以及使用信息将会在后面的章节中进行详细的介绍。

（3）FFmpeg 的滤镜模块 AVFilter

AVFilter 库提供了一个通用的音频、视频、字幕等滤镜处理框架。在 AVFilter 中，滤镜框架可以有多个输入和多个输出。我们参考下面这个滤镜处理的例子，如图 1-2 所示。

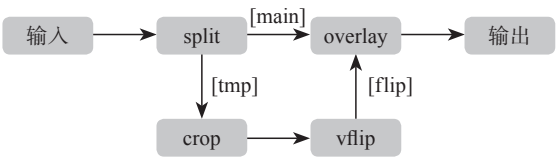

图 1-2　AVFilter 使用样例

图 1-2 所示样例中的滤镜处理将输入的视频切割成了两部分流，一部分流抛给 crop 滤镜与 vflip 滤镜处理模块进行操作，另一部分保持原样，当 crop 滤镜与 vflip 滤镜处理操作完成之后，将流合并到原有的 overlay 图层中，并显示在最上面一层，输出新的视频。对应的命令行如下：

```
    ./ffmpeg -i INPUT -vf "split [main][tmp]; [tmp] crop=iw:ih/2:0:0, vflip [flip];
[main][flip] overlay=0:H/2" OUTPUT
```

下面看一下具体的执行情况，以验证该命令的可行性：

```
    ffmpeg version n3.3.2 Copyright (c) 2000-2017 the FFmpeg developers
        built with Apple LLVM version 8.1.0 (clang-802.0.42)
        configuration: --disable-yasm
        libavutil      55. 58.100 / 55. 58.100
        libavcodec     57. 89.100 / 57. 89.100
        libavformat    57. 71.100 / 57. 71.100
        libavdevice    57.  6.100 / 57.  6.100
        libavfilter     6. 82.100 /  6. 82.100
        libswscale      4.  6.100 /  4.  6.100
        libswresample   2.  7.100 /  2.  7.100
    Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'input.mp4':
        Metadata:
            major_brand     : isom
            minor_version   : 1
            compatible_brands: isomavc1
            creation_time   : 2015-02-02T18:19:19.000000Z
        Duration: 00:45:02.06, start: 0.000000, bitrate: 2708 kb/s
            Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 1280x714
[SAR 1:1 DAR 640:357], 2576 kb/s, 25 fps, 25 tbr, 25k tbn, 50 tbc (default)
            Metadata:
                creation_time   : 2015-02-02T18:19:19.000000Z
                handler_name    : GPAC ISO Video Handler
            Stream #0:1(und): Audio: aac (LC) (mp4a / 0x6134706D), 48000 Hz, stereo,
fltp, 127 kb/s (default)
            Metadata:
                creation_time   : 2015-02-02T18:19:23.000000Z
                handler_name    : GPAC ISO Audio Handler
```

```
    Stream mapping:
        Stream #0:0 -> #0:0 (h264 (native) -> mpeg4 (native))
        Stream #0:1 -> #0:1 (aac (native) -> aac (native))
    Press [q] to stop, [?] for help
    Output #0, mp4, to 'output.mp4':
        Metadata:
            major_brand     : isom
            minor_version   : 1
            compatible_brands: isomavc1
            encoder         : Lavf57.71.100
            Stream #0:0(und): Video: mpeg4 ( [0][0][0] / 0x0020), yuv420p
(progressive), 1280x714 [SAR 1:1 DAR 640:357], q=2-31, 200 kb/s, 25 fps, 12800 tbn,
25 tbc (default)
            Metadata:
                creation_time   : 2015-02-02T18:19:19.000000Z
                handler_name    : GPAC ISO Video Handler
                encoder         : Lavc57.89.100 mpeg4
            Side data:
                cpb: bitrate max/min/avg: 0/0/200000 buffer size: 0 vbv_delay: -1
             Stream #0:1(und): Audio: aac (LC) ([64][0][0][0] / 0x0040), 48000 Hz,
stereo, fltp, 128 kb/s (default)
            Metadata:
                creation_time   : 2015-02-02T18:19:23.000000Z
                handler_name    : GPAC ISO Audio Handler
                encoder         : Lavc57.89.100 aac
    frame= 729 fps= 85 q=31.0 size= 4332kB time=00:00:29.31 bitrate=1210.7kbits/s
dup=2 drop=0 speed=3.41x
```

以上内容输出完成，该命令将自动退出，生成的视频结果是保留视频的上半部分，同时上半部分会镜像到视频的下半部分，二者合成之后作为输出视频，如图 1-3 所示。

图 1-3　Filter 运行前后对比

下面详细说明一下规则，具体如下。

- 相同的 Filter 线性链之间用逗号分隔
- 不同的 Filter 线性链之间用分号分隔

在以上示例中，crop 与 vflip 使用的是同一个滤镜处理的线性链，split 滤镜和 overlay 滤镜使用的是另外一个线性链，一个线性链与另一个线性链汇合时是通过方括号“[]”括起来的标签进行标示的。在这个例子中，两个流处理后是通过 [main] 与 [flip] 进行关联汇合的。

split 滤镜将分割后的视频流的第二部分打上标签 [tmp]，通过 crop 滤镜对该部分流进行处理，然后进行纵坐标调换操作，打上标签 [flip]，然后将 [main] 标签与 [flip] 标签进行合并，[flip] 标签的视频流从视频的左边最中间的位置开始显示，这样就出现了镜像效果，如图 1-3 所示。

（4）FFmpeg 的视频图像转换计算模块 swscale

swscale 模块提供了高级别的图像转换 API，例如它允许进行图像缩放和像素格式转换，常见于将图像从 1080p 转换成 720p 或者 480p 等的缩放，或者将图像数据从 YUV420P 转换成 YUYV，或者 YUV 转 RGB 等图像格式转换。

（5）FFmpeg 的音频转换计算模块 swresample

swresample 模块提供了高级别的音频重采样 API。例如它允许操作音频采样、音频通道布局转换与布局调整。

1.4　FFmpeg 的编解码工具 ffmpeg

ffmpeg 是 FFmpeg 源代码编译后生成的一个可执行程序，其可以作为命令行工具使用。本节将通过实际的示例分析，对 ffmpeg 编解码工具的使用方法进行详细的介绍。

首先列举一个简单的例子：

```
./ffmpeg -i input.mp4 output.avi
```

这条命令行执行过程输出如下：

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'input.mp4':
    Metadata:
        major_brand     : isom
        minor_version   : 1
        compatible_brands: isomavc1
        creation_time   : 2015-02-02T18:19:19.000000Z
    Duration: 00:45:02.06, start: 0.000000, bitrate: 2708 kb/s
        Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 1280x714
[SAR 1:1 DAR 640:357], 2576 kb/s, 25 fps, 25 tbr, 25k tbn, 50 tbc (default)
        Metadata:
            creation_time   : 2015-02-02T18:19:19.000000Z
            handler_name    : GPAC ISO Video Handler
        Stream #0:1(und): Audio: aac (LC) (mp4a / 0x6134706D), 48000 Hz,
stereo, fltp, 127 kb/s (default)
        Metadata:
            creation_time   : 2015-02-02T18:19:23.000000Z
            handler_name    : GPAC ISO Audio Handler
Stream mapping:
```

```
        Stream #0:0 -> #0:0 (h264 (native) -> mpeg4 (native))
        Stream #0:1 -> #0:1 (aac (native) -> ac3 (native))
    Press [q] to stop, [?] for help
    Output #0, avi, to 'output.avi':
        Metadata:
            major_brand     : isom
            minor_version   : 1
            compatible_brands: isomavc1
            ISFT            : Lavf57.71.100
            Stream #0:0(und): Video: mpeg4 (FMP4 / 0x34504D46), yuv420p
(progressive), 1280x714 [SAR 1:1 DAR 640:357], q=2-31, 200 kb/s, 25 fps, 25 tbn, 25
tbc (default)
            Metadata:
                creation_time   : 2015-02-02T18:19:19.000000Z
                handler_name    : GPAC ISO Video Handler
                encoder         : Lavc57.89.100 mpeg4
            Side data:
                cpb: bitrate max/min/avg: 0/0/200000 buffer size: 0 vbv_delay: -1
             Stream #0:1(und): Audio: ac3 ([0] [0][0] / 0x2000), 48000 Hz, stereo,
fltp, 192 kb/s (default)
            Metadata:
                creation_time   : 2015-02-02T18:19:23.000000Z
                handler_name    : GPAC ISO Audio Handler
                encoder         : Lavc57.89.100 ac3
    frame=  786 fps=111 q=31.0 size=  5187kB time=00:00:31.71 bitrate=1340.1kbits/s
speed=4.47x
```

这是一条简单的 ffmpeg 命令，可以看到，ffmpeg 通过 -i 参数将 input.mp4 作为输入源输入，然后进行转码与转封装操作，输出到 output.avi 中，这条命令主要做了如下工作。

1）获得输入源 input.mp4。

2）转码。

3）输出文件 output.avi。

看似简单的两步主要的工作，其实远远不止是从后缀名为 MP4 的文件输出成后缀名为 AVI 的文件，因为在 ffmpeg 中，MP4 与 AVI 是两种文件封装格式，并不是后缀名就可以决定的，例如上面的命令行同样可以写成这样：

```
./ffmpeg -i input.mp4 -f avi output.dat
```

这条命令行执行过程输出如下：

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'input.mp4':
    Metadata:
        major_brand     : isom
        minor_version   : 1
        compatible_brands: isomavc1
        creation_time   : 2015-02-02T18:19:19.000000Z
    Duration: 00:45:02.06, start: 0.000000, bitrate: 2708 kb/s
        Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p,
1280x714 [SAR 1:1 DAR 640:357], 2576 kb/s, 25 fps, 25 tbr, 25k tbn, 50 tbc (default)
        Metadata:
            creation_time   : 2015-02-02T18:19:19.000000Z
```

```
            handler_name    : GPAC ISO Video Handler
        Stream #0:1(und): Audio: aac (LC) (mp4a / 0x6134706D), 48000 Hz, stereo,
fltp, 127 kb/s (default)
        Metadata:
            creation_time   : 2015-02-02T18:19:23.000000Z
            handler_name    : GPAC ISO Audio Handler
    Stream mapping:
        Stream #0:0 -> #0:0 (h264 (native) -> mpeg4 (native))
        Stream #0:1 -> #0:1 (aac (native) -> ac3 (native))
    Press [q] to stop, [?] for help
    Output #0, avi, to 'output.dat':
        Metadata:
            major_brand     : isom
            minor_version   : 1
            compatible_brands: isomavc1
            ISFT            : Lavf57.71.100
            Stream #0:0(und): Video: mpeg4 (FMP4 / 0x34504D46), yuv420p
(progressive), 1280x714 [SAR 1:1 DAR 640:357], q=2-31, 200 kb/s, 25 fps, 25 tbn, 25
tbc (default)
        Metadata:
            creation_time   : 2015-02-02T18:19:19.000000Z
            handler_name    : GPAC ISO Video Handler
            encoder         : Lavc57.89.100 mpeg4
        Side data:
            cpb: bitrate max/min/avg: 0/0/200000 buffer size: 0 vbv_delay: -1
         Stream #0:1(und): Audio: ac3 ([0] [0][0] / 0x2000), 48000 Hz, stereo,
fltp, 192 kb/s (default)
        Metadata:
            creation_time   : 2015-02-02T18:19:23.000000Z
            handler_name    : GPAC ISO Audio Handler
            encoder         : Lavc57.89.100 ac3
    frame=  711 fps=108 q=31.0 size=  4678kB time=00:00:28.83 bitrate=1329.0kbits/s
speed= 4.4x
```

这条 ffmpeg 命令相对于前面的那条命令做了一些改变，加了一个“-f”进行约束，“-f”参数的工作非常重要，它制定了输出文件的容器格式，所以可以看到输出的文件为 output.dat，文件后缀名为 .dat，但是其主要工作依然与之前的指令相同。

分析以上两个输出信息中的 Output #0 部分，可以看到输出的都是 AVI，只是输出的文件名不同，其他内容均相同。

ffmpeg 的主要工作流程相对比较简单，具体如下。

1）解封装（Demuxing）。

2）解码（Decoding）。

3）编码（Encoding）。

4）封装（Muxing）。

其中需要经过 6 个步骤，具体如下。

1）读取输入源。

2）进行音视频的解封装。

3）解码每一帧音视频数据。

4）编码每一帧音视频数据。

5）进行音视频的重新封装。

6）输出到目标。

ffmpeg 整体处理的工作流程与步骤如图 1-4 所示。

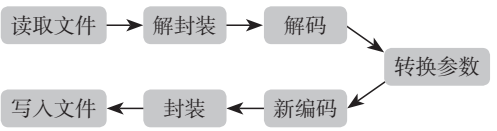

图 1-4　ffmpeg 转码工作流程

从图 1-4 所示的工作流程可以看出，ffmpeg 首先读取输入源；然后通过 Demuxer 将音视频包进行解封装，这个动作通过调用 libavformat 中的接口即可实现；接下来通过 Decoder 进行解码，将音视频通过 Decoder 解包成为 YVU 或者 PCM 这样的数据，Decoder 通过 libavcodec 中的接口即可实现；然后通过 Encoder 将对应的数据进行编码，编码可以通过 libavcodec 中的接口来实现；接下来将编码后的音视频数据包通过 Muxer 进行封装，Muxer 封装通过 libavformat 中的接口即可实现，输出成为输出流。

1.5　FFmpeg 的播放器 ffplay

FFmpeg 不但可以提供转码、转封装等功能，同时还提供了播放器相关功能，使用 FFmpeg 的 avformat 与 avcodec，可以播放各种媒体文件或者流。如果想要使用 ffplay，那么系统首先需要有 SDL 来进行 ffplay 的基础支撑。

ffplay 是 FFmpeg 源代码编译后生成的另一个可执行程序，与 ffmpeg 在 FFmpeg 项目中充当的角色基本相同，可以作为测试工具进行使用，ffplay 提供了音视频显示和播放相关的图像信息、音频的波形信息等。

> **注意：**
>
> 有时通过源代码编译生成 ffplay 不一定能够成功，因为 ffplay 在旧版本时依赖于 SDL-1.2，而 ffplay 在新版本时依赖于 SDL-2.0，需要安装对应的 SDL 才能生成 ffplay。

1.6　FFmpeg 的多媒体分析器 ffprobe

ffprobe 也是 FFmpeg 源码编译后生成的一个可执行程序。ffprobe 是一个非常强大的多媒体分析工具，可以从媒体文件或者媒体流中获得你想要了解的媒体信息，比如音频的参数、视频的参数、媒体容器的参数信息等。

例如它可以帮助分析某个媒体容器中的音频是什么编码格式、视频是什么编码格式，同时还可以得到媒体文件中媒体的总时长、复合码率等信息。

使用 ffprobe 可以分析媒体文件中每个包的长度、包的类型、帧的信息等。后面章节将会对 ffprobe 进行详细的介绍，下面列举一个简单的例子，以对 ffprobe 有一个基本的概念：

```
./ffprobe -show_streams output.mp4
```

命令行执行之后将会输出如下内容：

```
[STREAM]
index=0
codec_name=mpeg4
codec_long_name=MPEG-4 part 2
profile=Simple Profile
codec_type=video
codec_time_base=1/25
codec_tag_string=mp4v
codec_tag=0x7634706d
width=1280
height=714
coded_width=1280
coded_height=714
has_b_frames=0
sample_aspect_ratio=1:1
display_aspect_ratio=640:357
pix_fmt=yuv420p
level=1
color_range=N/A
chroma_location=left
field_order=unknown
timecode=N/A
refs=1
quarter_sample=false
divx_packed=false
r_frame_rate=25/1
avg_frame_rate=25/1
time_base=1/12800
start_pts=0
start_time=0.000000
duration_ts=170496
duration=13.320000
bit_rate=1146797
max_bit_rate=1146797
bits_per_raw_sample=N/A
nb_frames=333
nb_read_frames=N/A
nb_read_packets=N/A
[/STREAM]
[STREAM]
index=1
codec_name=aac
codec_long_name=AAC (Advanced Audio Coding)
profile=LC
codec_type=audio
codec_time_base=1/48000
codec_tag_string=mp4a
codec_tag=0x6134706d
sample_fmt=fltp
sample_rate=48000
channels=2
```

```
channel_layout=stereo
bits_per_sample=0
id=N/A
r_frame_rate=0/0
avg_frame_rate=0/0
time_base=1/48000
start_pts=0
start_time=0.000000
duration_ts=643056
duration=13.397000
bit_rate=128213
max_bit_rate=128213
bits_per_raw_sample=N/A
nb_frames=629
nb_read_frames=N/A
nb_read_packets=N/A
[/STREAM]
```

根据输出内容可以看到，使用 ffprobe 能够查看 MP4 文件容器中的流的信息，其包含了一个视频流，由于该文件中只有视频流，流相关的信息是通过 [STREAM][/STREAM] 的方式展现出来的，在 [STREAM] 与 [/STREAM] 之间的信息即为该 MP4 文件的视频流信息。当视频文件容器中包含音频流与视频流或者更多路流时，会通过 [STREAM] 与 [/STREAM] 进行多个流的分隔，分隔后采用 index 来进行流的索引信息的区分。

1.7 FFmpeg 编译

FFmpeg 在官方网站中提供了已经编译好的可执行文件。因为 FFmpeg 是开源的，所以也可以根据自己的需要进行手动编译。FFmpeg 官方建议用户自行编译使用 FFmpeg 的最新版本，因为对于一些操作系统，比如 Linux 系统（无论是 Ubuntu 还是 RedHat），如果使用系统提供的软件库安装 ffmpeg 时会发现其版本相对比较老旧，比如使用 apt-get install ffmpeg 或者 yum install ffmpeg 安装 ffmpeg，那么默认支持的版本都很老，有些新的功能并不支持，如一些新的封装格式或者通信协议。因此初学者学会编译 FFmpeg 就至关重要了，因此可以方便以后根据自己的需求进行功能的裁剪。

1.7.1 FFmpeg 之 Windows 平台编译

FFmpeg 在 Windows 平台中的编译需要使用 MinGW-w64，MinGW 是 Minimalist GNU for Windows 的缩写，它提供了一系列的工具链来辅助编译 Windows 的本地化程序，它的详细介绍和安装方法可以参照 http://www.mingw.org/。如果不希望使用 MinGW 而使用 Visual Studio 的话，则需要消耗很多时间来支持 Visual Studio 平台，感兴趣的读者可以在网上查找一下支持的方法。截至本书结稿之时，官方提供的 Windows 的开发包是使用 MinGW-w64 工具链编写的。

MinGW-w64 单独使用起来会比较麻烦，但是其可以与 MSYS 环境配合使用，MSYS

是 Minimal SYStem 的缩写，其主要完成的工作为 UNIX on Windows 的功能。显而易见，这是一个仿生 UNIX 环境的 Windows 工具集，它的详细介绍和使用方法可以参照 http://www.mingw.org/wiki/MSYS。

MinGW-w64 + MSYS 环境准备好之后，我们就可以正式进入编译的环节了。

1）进入 FFmpeg 源码目录，执行“./configure”，如果一切正常，我们会看到如下信息：

```
install prefix            /usr/local
source path               .
C compiler                gcc
C library                 mingw64
ARCH                      x86 (generic)
big-endian                no
runtime cpu detection     yes
yasm                      yes
MMX enabled               yes
MMXEXT enabled            yes
3DNow! enabled            yes
3DNow! extended enabled   yes
SSE enabled               yes
SSSE3 enabled             yes
AESNI enabled             yes
AVX enabled               yes
XOP enabled               yes
FMA3 enabled              yes
FMA4 enabled              yes
i686 features enabled     yes
CMOV is fast              no
EBX available             yes
EBP available             yes
debug symbols             yes
strip symbols             yes
optimize for size         no
optimizations             yes
static                    yes
shared                    no
postprocessing support    no
network support           yes
threading support         w32threads
safe bitstream reader     yes
texi2html enabled         no
perl enabled              yes
pod2man enabled           yes
makeinfo enabled          yes
makeinfo supports HTML    no
```

2）configure 成功后执行 make，在 MinGW 环境下编译 ffmpeg 是一个比较漫长的过程。

3）执行 make install，到此为止，FFmpeg 在 Windows 上的编译已全部完成，此时我们可以尝试使用 FFmpeg 命令行来验证编译结果。执行“./ffmpeg.exe –h”：

```
    ffmpeg version n3.3.2  Copyright (c) 2000-2017 the FFmpeg developers
        built with gcc 4.9.2 (i686-posix-dwarf-rev1, Built by MinGW-W64 project)
        configuration: --enable-gpl
        libavutil      55. 58.100 / 55. 58.100
        libavcodec     57. 89.100 / 57. 89.100
        libavformat    57. 71.100 / 57. 71.100
        libavdevice    57.  6.100 / 57.  6.100
        libavfilter      6. 82.100 /  6. 82.100
        libswscale      4.  6.100 /  4.  6.100
        libswresample   2.  7.100 /  2.  7.100
    Hyper fast Audio and Video encoder
    usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options]
outfile}...
```

注意：

以上编译配置方式编译出来的 ffmpeg 仅仅只是最简易的 ffmpeg，并没有 H.264、H.265、加字幕等编码支持，如果需要支持更多的模块和参数，还需要进行更加详细的定制，后面会有详细的介绍。

1.7.2 FFmpeg 之 Linux 平台编译

前面介绍过，很多 Linux 的发行版本源中已经包含了 FFmpeg，如 Ubuntu / Fedora 的镜像源中包含了 FFmpeg 的安装包，但是版本相对来说比较老旧，有些甚至还不支持 H.264、H.265 编码，或者不支持 RTMP 等，为了支持这些协议格式和编码格式，需要自己手动编译 FFmpeg，默认编译 FFmpeg 的时候，需要用到 yasm 汇编器对 FFmpeg 中的汇编部分进行编译。如果不需要用到汇编部分的代码，则可以不安装 yasm 汇编器。如果没有安排 yasm，则执行默认配置的时候，会提示错误：

```
    ffmpeg version n3.3.2 Copyright (c) 2000-2017 the FFmpeg developers
        built with Apple LLVM version 8.1.0 (clang-802.0.42)
        configuration: --disable-yasm
        libavutil      55. 58.100 / 55. 58.100
        libavcodec     57. 89.100 / 57. 89.100
        libavformat    57. 71.100 / 57. 71.100
        libavdevice    57.  6.100 / 57.  6.100
        libavfilter      6. 82.100 /  6. 82.100
        libswscale      4.  6.100 /  4.  6.100
        libswresample   2.  7.100 /  2.  7.100
    Hyper fast Audio and Video encoder
    usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options]
outfile}...

    Use -h to get full help or, even better, run 'man ffmpeg'
    liuqideMBP:n3.3.2 liuqi$ ../configure
    yasm/nasm not found or too old. Use --disable-yasm for a crippled build.

    If you think configure made a mistake, make sure you are using the latest
    version from Git.  If the latest version fails, report the problem to the
    ffmpeg-user@ffmpeg.org mailing list or IRC #ffmpeg on irc.freenode.net.
```

```
    Include the log file "ffbuild/config.log" produced by configure as this will
help
    solve the problem.
```

根据以上的错误提示，可以使用 --disable-yasm 来取消 yasm 编译配置，不过这么做的话就不会编译 FFmpeg 的汇编代码部分，相关的优化也会少一些。如果需要支持汇编优化，那么可以通过安装 yasm 汇编器来解决：

```
wget http://www.tortall.net/projects/yasm/releases/yasm-1.3.0.tar.gz
```

命令行执行后将会下载 yasm 源代码包：

```
--2017-07-17 17:04:09--
http://www.tortall.net/projects/yasm/releases/yasm-1.3.0.tar.gz
Resolving www.tortall.net... 69.55.226.36
Connecting to www.tortall.net|69.55.226.36|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1492156 (1.4M) [application/octet-stream]
Saving to: 'yasm-1.3.0.tar.gz'

yasm-1.3.0.tar.gz  100%[=====================>]1.42M  48.4KB/s    in 30s
2017-07-17 17:04:39 (48.9 KB/s) - 'yasm-1.3.0.tar.gz' saved [1492156/1492156]
```

下载 yasm 汇编器后，先进行 configure 操作，然后通过 make 编译，再执行 make install 安装即可。最后再回到 FFmpeg 源代码目录中进行之前的 configure 操作，之前的错误提示就会消失，代码如下：

```
install prefix            /usr/local
source path               /home/git/FFmpeg_down
C compiler                gcc
C library                 glibc
ARCH                      x86 (generic)
big-endian                no
runtime cpu detection     yes
standalone assembly       yes
x86 assembler             yasm
MMX enabled               yes
MMXEXT enabled            yes
3DNow! enabled            yes
3DNow! extended enabled   yes
SSE enabled               yes
SSSE3 enabled             yes
AESNI enabled             yes
AVX enabled               yes
XOP enabled               yes
FMA3 enabled              yes
FMA4 enabled              yes
i686 features enabled     yes
CMOV is fast              yes
EBX available             yes
EBP available             yes
debug symbols             yes
strip symbols             yes
```

```
    optimize for size         no
    optimizations             yes
    static                    yes
    shared                    no
    postprocessing support    no
    network support           yes
    threading support         pthreads
    safe bitstream reader     yes
    texi2html enabled         no
    perl enabled              yes
    pod2man enabled           yes
    makeinfo enabled          yes
    makeinfo supports HTML    no
```

1.7.3 FFmpeg 之 OS X 平台编译

有些开发者在 OS X 平台上使用 FFmpeg 进行一些音视频编转码或流媒体处理等工作，因此需要生成 OS X 平台相关的 FFmpeg 的可执行程序，在 OS X 平台上编译 FFmpeg 之前，首先需要安装所需要的编译环境，在 OS X 平台上使用的编译工具链为 LLVM：

```
    Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr
--with-gxx-include-dir=/usr/include/c++/4.2.1
    Apple LLVM version 8.1.0 (clang-802.0.42)
    Target: x86_64-apple-darwin16.6.0
    Thread model: posix
    InstalledDir:
    /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/
usr/bin
```

另外，还需要安装 yasm 汇编编译工具，否则在生成 Makefile 时会报错提示未安装 yasm 工具。

在 LLVM 下利用源码安装 FFmpeg 与其他平台基本相同，尤其是与 Linux 相同，FFmpeg 可从 git://source.ffmpeg.org/ffmpeg.git 将源代码克隆到本地：

```
    credential.helper=osxkeychain
    user.email=lingjiujianke@gmail.com
    user.name=Steven Liu
    core.repositoryformatversion=0
    core.filemode=true
    core.bare=false
    core.logallrefupdates=true
    core.ignorecase=true
    core.precomposeunicode=true
    remote.origin.url=git://source.ffmpeg.org/ffmpeg.git
    remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
    branch.master.remote=origin
    branch.master.merge=refs/heads/master
    branch.cmts.remote=origin
    branch.cmts.merge=refs/heads/master
```

源代码下载成功后，开始进入编译阶段，通过如下几步操作即可完成基本的编译工作：

```
install prefix            /usr/local
source path               /Users/liuqi/multimedia/ffmpeg
C compiler                gcc
C library
ARCH                      x86 (generic)
big-endian                no
runtime cpu detection     yes
yasm                      yes
MMX enabled               yes
MMXEXT enabled            yes
3DNow! enabled            yes
3DNow! extended enabled   yes
SSE enabled               yes
SSSE3 enabled             yes
AESNI enabled             yes
AVX enabled               yes
XOP enabled               yes
FMA3 enabled              yes
FMA4 enabled              yes
i686 features enabled     yes
CMOV is fast              yes
EBX available             yes
EBP available             yes
debug symbols             yes
strip symbols             yes
optimize for size         no
optimizations             yes
static                    yes
shared                    no
postprocessing support    no
network support           yes
threading support         pthreads
safe bitstream reader     yes
texi2html enabled         no
perl enabled              yes
pod2man enabled           yes
makeinfo enabled          yes
makeinfo supports HTML    no
```

接下来，只需要执行 make 进行编译与执行 make install 进行安装即可。

1.8 FFmpeg 编码支持与定制

FFmpeg 本身支持一些音视频编码格式、文件封装格式与流媒体传输协议，但是支持的数量依然有限，FFmpeg 所做的只是提供一套基础的框架，所有的编码格式、文件封装格式与流媒体协议均可以作为 FFmpeg 的一个模块挂载在 FFmpeg 框架中。这些模块以第三方的外部库的方式提供支持，可以通过 FFmpeg 源码的 configure 命令查看 FFmpeg 所支持的音视频编码格式、文件封装格式与流媒体传输协议，对于 FFmpeg 不支持的格式，可以通过 configure --help 查看所需要的第三方外部库，然后通过增加对应的编译参数选项进行支持。帮助信息内容输出如下：

```
    External library support:
        Using any of the following switches will allow FFmpeg to link to the
        corresponding external library. All the components depending on that library
         will become enabled, if all their other dependencies are met and they are
not
        explicitly disabled. E.g. --enable-libwavpack will enable linking to
        libwavpack and allow the libwavpack encoder to be built, unless it is
        specifically disabled with --disable-encoder=libwavpack.
        Note that only the system libraries are auto-detected. All the other
external
        libraries must be explicitly enabled.
        Also note that the following help text describes the purpose of the
libraries
        themselves, not all their features will necessarily be usable by FFmpeg.

        --enable-avisynth        enable reading of AviSynth script files [no]
        --disable-bzlib          disable bzlib [autodetect]
        --enable-chromaprint     enable audio fingerprinting with chromaprint [no]
        --enable-frei0r          enable frei0r video filtering [no]
        --enable-gcrypt          enable gcrypt, needed for rtmp(t)e support
                                 if openssl, librtmp or gmp is not used [no]
        --enable-gmp             enable gmp, needed for rtmp(t)e support
                                 if openssl or librtmp is not used [no]
        --enable-gnutls          enable gnutls, needed for https support
                                 if openssl is not used [no]
        --disable-iconv          disable iconv [autodetect]
        --enable-jni             enable JNI support [no]
        --enable-ladspa          enable LADSPA audio filtering [no]
        --enable-libass          enable libass subtitles rendering,
                                 needed for subtitles and ass filter [no]
        --enable-libbluray       enable BluRay reading using libbluray [no]
        --enable-libbs2b         enable bs2b DSP library [no]
        --enable-libcaca         enable textual display using libcaca [no]
        --enable-libcelt         enable CELT decoding via libcelt [no]
        --enable-libcdio         enable audio CD grabbing with libcdio [no]
        --enable-libdc1394       enable IIDC-1394 grabbing using libdc1394
                                 and libraw1394 [no]
        --enable-libfdk-aac      enable AAC de/encoding via libfdk-aac [no]
        --enable-libflite        enable flite (voice synthesis) support via libflite
[no]
        --enable-libfontconfig   enable libfontconfig, useful for drawtext filter
[no]
        --enable-libfreetype     enable libfreetype, needed for drawtext filter [no]
        --enable-libfribidi      enable libfribidi, improves drawtext filter [no]
        --enable-libgme          enable Game Music Emu via libgme [no]
        --enable-libgsm          enable GSM de/encoding via libgsm [no]
        --enable-libiec61883     enable iec61883 via libiec61883 [no]
        --enable-libilbc         enable iLBC de/encoding via libilbc [no]
        --enable-libkvazaar      enable HEVC encoding via libkvazaar [no]
        --enable-libmodplug      enable ModPlug via libmodplug [no]
        --enable-libmp3lame      enable MP3 encoding via libmp3lame [no]
        --enable-libopencore-amrnb  enable AMR-NB de/encoding via libopencore-
amrnb [no]
        --enable-libopencore-amrwb   enable AMR-WB decoding via libopencore-amrwb
[no]
```

```
        --enable-libopencv      enable video filtering via libopencv [no]
        --enable-libopenh264    enable H.264 encoding via OpenH264 [no]
        --enable-libopenjpeg    enable JPEG 2000 de/encoding via OpenJPEG [no]
        --enable-libopenmpt     enable decoding tracked files via libopenmpt [no]
        --enable-libopus        enable Opus de/encoding via libopus [no]
        --enable-libpulse       enable Pulseaudio input via libpulse [no]
        --enable-librsvg        enable SVG rasterization via librsvg [no]
        --enable-librubberband  enable rubberband needed for rubberband filter [no]
        --enable-librtmp        enable RTMP[E] support via librtmp [no]
        --enable-libshine       enable fixed-point MP3 encoding via libshine [no]
        --enable-libsmbclient   enable Samba protocol via libsmbclient [no]
        --enable-libsnappy      enable Snappy compression, needed for hap encoding
[no]
        --enable-libsoxr        enable Include libsoxr resampling [no]
        --enable-libspeex       enable Speex de/encoding via libspeex [no]
        --enable-libssh         enable SFTP protocol via libssh [no]
        --enable-libtesseract   enable Tesseract, needed for ocr filter [no]
        --enable-libtheora      enable Theora encoding via libtheora [no]
        --enable-libtwolame     enable MP2 encoding via libtwolame [no]
        --enable-libv4l2        enable libv4l2/v4l-utils [no]
        --enable-libvidstab     enable video stabilization using vid.stab [no]
        --enable-libvo-amrwbenc enable AMR-WB encoding via libvo-amrwbenc [no]
        --enable-libvorbis      enable Vorbis en/decoding via libvorbis,
                                native implementation exists [no]
        --enable-libvpx         enable VP8 and VP9 de/encoding via libvpx [no]
        --enable-libwavpack     enable wavpack encoding via libwavpack [no]
        --enable-libwebp        enable WebP encoding via libwebp [no]
        --enable-libx264        enable H.264 encoding via x264 [no]
        --enable-libx265        enable HEVC encoding via x265 [no]
        --enable-libxavs        enable AVS encoding via xavs [no]
        --enable-libxcb         enable X11 grabbing using XCB [autodetect]
        --enable-libxcb-shm     enable X11 grabbing shm communication [autodetect]
        --enable-libxcb-xfixes  enable X11 grabbing mouse rendering [autodetect]
        --enable-libxcb-shape   enable X11 grabbing shape rendering [autodetect]
        --enable-libxvid        enable Xvid encoding via xvidcore,
                                native MPEG-4/Xvid encoder exists [no]
        --enable-libxml2        enable XML parsing using the C library libxml2 [no]
        --enable-libzimg        enable z.lib, needed for zscale filter [no]
        --enable-libzmq         enable message passing via libzmq [no]
        --enable-libzvbi        enable teletext support via libzvbi [no]
        --disable-lzma          disable lzma [autodetect]
        --enable-decklink       enable Blackmagic DeckLink I/O support [no]
        --enable-mediacodec     enable Android MediaCodec support [no]
        --enable-netcdf         enable NetCDF, needed for sofalizer filter [no]
        --enable-openal         enable OpenAL 1.1 capture support [no]
        --enable-opencl         enable OpenCL code
        --enable-opengl         enable OpenGL rendering [no]
        --enable-openssl        enable openssl, needed for https support
                                if gnutls is not used [no]
        --disable-schannel      disable SChannel SSP, needed for TLS support on
                                Windows if openssl and gnutls are not used
[autodetect]
        --disable-sdl2            disable sdl2 [autodetect]
        --disable-securetransport disable Secure Transport, needed for TLS support
                                  on OS X if openssl and gnutls are not used
```

```
[autodetect]
        --disable-xlib          disable xlib [autodetect]
        --disable-zlib          disable zlib [autodetect]
```

通过以上帮助信息的输出内容可以看到，FFmpeg 所支持的外部库相对来说比较多，主要包含如下列表：

```
    bzip2 1.0.6 <http://bzip.org/>
    Fontconfig 2.11.94 <http://freedesktop.org/wiki/Software/fontconfig>
    Frei0r 20130909-git-10d8360 <http://frei0r.dyne.org/>
    GnuTLS 3.3.15 <http://gnutls.org/>
    libiconv 1.14 <http://gnu.org/software/libiconv/>
    libass 0.12.2 <http://code.google.com/p/libass/>
    libbluray 0.8.1 <http://videolan.org/developers/libbluray.html>
    libbs2b 3.1.0 <http://bs2b.sourceforge.net/>
    libcaca 0.99.beta18 <http://caca.zoy.org/wiki/libcaca>
    dcadec 20150506-git-98fb3b6 <https://github.com/foo86/dcadec>
    FreeType 2.5.5 <http://freetype.sourceforge.net/>
    Game Music Emu 0.6.0 <http://code.google.com/p/game-music-emu/>
    GSM 1.0.13-4 <http://packages.debian.org/source/squeeze/libgsm>
    iLBC 20141214-git-ef04ebe <https://github.com/dekkers/libilbc/>
    Modplug-XMMS 0.8.8.5 <http://modplug-xmms.sourceforge.net/>
    LAME 3.99.5 <http://lame.sourceforge.net/>
    OpenCORE AMR 0.1.3 <http://sourceforge.net/projects/opencore-amr/>
    OpenJPEG 1.5.2 <http://www.openjpeg.org/>
    Opus 1.1 <http://opus-codec.org/>
    RTMPDump 20140707-git-a1900c3 <http://rtmpdump.mplayerhq.hu/>
    Schroedinger 1.0.11 <http://diracvideo.org/>
    libsoxr 0.1.1 <http://sourceforge.net/projects/soxr/>
    Speex 1.2rc2 <http://speex.org/>
    Theora 1.1.1 <http://theora.org/>
    TwoLAME 0.3.13 <http://twolame.org/>
    vid.stab 0.98 <http://public.hronopik.de/vid.stab/>
    VisualOn AAC 0.1.3 <https://github.com/mstorsjo/vo-aacenc>
    VisualOn AMR-WB 0.1.2 <https://github.com/mstorsjo/vo-amrwbenc>
    Vorbis 1.3.5 <http://vorbis.com/>
    vpx 1.4.0 <http://webmproject.org/>
    WavPack 4.75.0 <http://wavpack.com/>
    WebP 0.4.3 <https://developers.google.com/speed/webp/>
    x264 20150223-git-121396c <http://videolan.org/developers/x264.html>
    x265 1.7 <http://x265.org/>
    XAVS svn-r55 <http://xavs.sourceforge.net/>
    Xvid 1.3.3 <http://xvid.org/>
    XZ Utils 5.2.1 <http://tukaani.org/xz>
    zlib 1.2.8 <http://zlib.net/>
```

这些外部库可以通过 configure 进行定制，在编译好的 FFmpeg 可执行程序中也可以看到编译时定制的 FFmpeg 的外部库：

```
    ffmpeg version n3.3.2 Copyright (c) 2000-2017 the FFmpeg developers
        built with Apple LLVM version 8.1.0 (clang-802.0.42)
        configuration: --enable-fontconfig --enable-gpl --enable-libass --enable-
libbluray --enable-libfreetype --enable-libmp3lame --enable-libspeex --enable-
libx264 --enable-libx265 --enable-libfdk-aac --enable-version3 --cc='ccache gcc'
```

```
--enable-nonfree --enable-videotoolbox --enable-audiotoolbox
        libavutil      55. 58.100 / 55. 58.100
        libavcodec     57. 89.100 / 57. 89.100
        libavformat    57. 71.100 / 57. 71.100
        libavdevice    57.  6.100 / 57.  6.100
        libavfilter      6. 82.100 /  6. 82.100
        libswscale       4.  6.100 /  4.  6.100
        libswresample    2.  7.100 /  2.  7.100
        libpostproc    54.  5.100 / 54.  5.100
```

例如需要自己配置 FFmpeg 支持哪些格式，比如仅支持 H.264 视频与 AAC 音频编码，可以调整配置项将其简化如下：

```
../configure --enable-libx264 --enable-libfdk-aac --enable-gpl --enable-nonfree
```

命令行执行后的输出内容如下：

```
install prefix            /usr/local
source path               /Users/liuqi/multimedia/ffmpeg
C compiler                gcc
C library
ARCH                      x86 (generic)
big-endian                no
runtime cpu detection     yes
yasm                      yes
MMX enabled               yes
MMXEXT enabled            yes
3DNow! enabled            yes
3DNow! extended enabled   yes
SSE enabled               yes
SSSE3 enabled             yes
AESNI enabled             yes
AVX enabled               yes
XOP enabled               yes
FMA3 enabled              yes
FMA4 enabled              yes
i686 features enabled     yes
CMOV is fast              yes
EBX available             yes
EBP available             yes
debug symbols             yes
strip symbols             yes
optimize for size         no
optimizations             yes
static                    yes
shared                    no
postprocessing support    yes
network support           yes
threading support         pthreads
safe bitstream reader     yes
texi2html enabled         no
perl enabled              yes
pod2man enabled           yes
makeinfo enabled          yes
makeinfo supports HTML    no
```

如配置后输出的基本信息所示，如果要支持 H.264 与 AAC，则需要系统中包括 libx264 与 fdkaac 的第三方库进行支持，否则会出现错误提示，libfdk 未安装时的错误提示如下：

```
ERROR: libfdk_aac not found

If you think configure made a mistake, make sure you are using the latest
version from Git.  If the latest version fails, report the problem to the
ffmpeg-user@ffmpeg.org mailing list or IRC #ffmpeg on irc.freenode.net.
Include the log file "config.log" produced by configure as this will help
solve the problem.
```

如果没有安装 libx264，则可以看到如下的错误提示：

```
ERROR: libx264 not found

If you think configure made a mistake, make sure you are using the latest
version from Git.  If the latest version fails, report the problem to the
ffmpeg-user@ffmpeg.org mailing list or IRC #ffmpeg on irc.freenode.net.
Include the log file "config.log" produced by configure as this will help
solve the problem.
```

如果需要支持 H.265 编码，则只需要增加 --enable-libx265 即可，其与支持 H.264 基本类似，从前面的 help 信息中可以看到，其他对应的编码与此类似。

注意：

从 2016 年年初开始，FFmpeg 资深的 AAC 编码器质量逐步好转，至 2016 年年底，libfaac 已经从 FFmpeg 源代码中剔除。

FFmpeg 默认支持的音视频编码格式、文件封装格式和流媒体传输协议相对来说比较多，因此编译出来的 FFmpeg 体积比较大，在有些应用场景中，并不需要 FFmpeg 所支持的一些编码、封装或者协议，可以通过 configure --help 查看一些有用的裁剪操作，输出如下：

```
Individual component options:
    --disable-everything          disable all components listed below
    --disable-encoder=NAME        disable encoder NAME
    --enable-encoder=NAME         enable encoder NAME
    --disable-encoders            disable all encoders
    --disable-decoder=NAME        disable decoder NAME
    --enable-decoder=NAME         enable decoder NAME
    --disable-decoders            disable all decoders
    --disable-hwaccel=NAME        disable hwaccel NAME
    --enable-hwaccel=NAME         enable hwaccel NAME
    --disable-hwaccels            disable all hwaccels
    --disable-muxer=NAME          disable muxer NAME
    --enable-muxer=NAME           enable muxer NAME
    --disable-muxers              disable all muxers
    --disable-demuxer=NAME        disable demuxer NAME
```

```
--enable-demuxer=NAME          enable demuxer NAME
--disable-demuxers             disable all demuxers
--enable-parser=NAME           enable parser NAME
--disable-parser=NAME          disable parser NAME
--disable-parsers              disable all parsers
--enable-bsf=NAME              enable bitstream filter NAME
--disable-bsf=NAME             disable bitstream filter NAME
--disable-bsfs                 disable all bitstream filters
--enable-protocol=NAME         enable protocol NAME
--disable-protocol=NAME        disable protocol NAME
--disable-protocols            disable all protocols
--enable-indev=NAME            enable input device NAME
--disable-indev=NAME           disable input device NAME
--disable-indevs               disable input devices
--enable-outdev=NAME           enable output device NAME
--disable-outdev=NAME          disable output device NAME
--disable-outdevs              disable output devices
--disable-devices              disable all devices
--enable-filter=NAME           enable filter NAME
--disable-filter=NAME          disable filter NAME
--disable-filters              disable all filters
```

可以通过这些选项关闭不需要用到的编码、封装与协议等模块，验证方法如下：

```
./configure --disable-encoders --disable-decoders --disable-hwaccels --disable-muxers --disable-demuxers --disable-parsers --disable-bsfs --disable-protocols --disable-indevs --disable-devices --disable-filters
```

关闭所有的模块之后，可以看到 FFmpeg 的编译配置项输出信息几乎为空，输出信息具体如下：

```
External libraries:
iconv          sdl2                videotoolbox        zlib
sdl            securetransport     xlib

External libraries providing hardware acceleration:
audiotoolbox  vda                  videotoolbox_hwaccel

Libraries:
avcodec        avfilter            avutil              swscale
avdevice       avformat            swresample

Programs:
ffmpeg         ffplay              ffprobe             ffserver

Enabled decoders:
Enabled encoders:
Enabled hwaccels:
Enabled parsers:
Enabled demuxers:
asf            mov                 mpegts              rm              rtsp

Enabled muxers:
ffm
```

```
Enabled protocols:
http          rtp               tcp               udp

Enabled filters:
aformat       crop              null              trim
anull         format            rotate            vflip
atrim         hflip             transpose

Enabled bsfs:
Enabled indevs:
Enabled outdevs:
License: LGPL version 2.1 or later
Creating configuration files ...
```

而且在关闭所有的模块之后，可以根据定制支持自己所需要的模块，例如希望支持 H.264 视频编码、AAC 音频编码、封装为 MP4，可以通过如下方式进行支持：

```
./configure --disable-filters --disable-encoders --disable-decoders --disable-
hwaccels --disable-muxers --disable-demuxers --disable-parsers --disable-bsfs
--disable-protocols --disable-indevs --disable-devices  --enable-libx264 --enable-
libfdk-aac --enable-gpl --enable-nonfree --enable-muxer=mp4
```

配置后输出的编译配置信息如下：

```
External libraries:
iconv         libx264         sdl2              videotoolbox        zlib
libfdk_aac    sdl             securetransport   xlib

External libraries providing hardware acceleration:
audiotoolbox  vda             videotoolbox_hwaccel

Libraries:
avcodec       avfilter        avutil            swresample
avdevice      avformat        postproc          swscale

Programs:
ffmpeg        ffplay          ffprobe           ffserver

Enabled decoders:
Enabled encoders:
Enabled hwaccels:
Enabled parsers:
Enabled demuxers:
asf           mov             mpegts            rm                  rtsp

Enabled muxers:
ffm           mov             mp4

Enabled protocols:
http          rtp             tcp               udp

Enabled filters:
aformat       crop            null              trim
anull         format          rotate            vflip
atrim         hflip           transpose
```

```
Enabled bsfs:
Enabled indevs:
Enabled outdevs:
License: nonfree and unredistributable
Creating configuration files ...
```

从以上的输出内容可以看到，FFmpeg 已经支持了 H.264 编码、AAC 编码与 MP4 封装格式。这样通过编译之后生成的 FFmpeg 即是配置裁剪过的 FFmpeg，体积会比默认编译的 FFmpeg 小很多。

1.8.1 FFmpeg 的编码器支持

FFmpeg 源代码中可以包含的编码非常多，常见的和不常见的都可以在编译配置列表中见到，可以通过使用编译配置命令 ./configure --list-encoders 参数来查看：

```
a64multi        flashsv         libwavpack      pcm_s16be         s302m
a64multi5       flashsv2        libwebp         pcm_s16be_planar  sgi
aac             flv             libwebp_anim    pcm_s16le         snow
aac_at          g723_1          libx262         pcm_s16le_planar  sonic
ac3             gif             libx264         pcm_s24be         sonic_ls
ac3_fixed       h261            libx264rgb      pcm_s24daud       srt
adpcm_adx       h263            libx265         pcm_s24le         ssa
adpcm_g722      h263p           libxavs         pcm_s24le_planar  subrip
adpcm_g726      h264_nvenc      libxvid         pcm_s32be         sunrast
adpcm_ima_qt    h264_omx        ljpeg           pcm_s32le         svq1
adpcm_ima_wav   h264_qsv        mjpeg           pcm_s32le_planar  targa
adpcm_ms        h264_vaapi      mjpeg_vaapi     pcm_s64be         text
adpcm_swf       h264_videotoolbox mlp           pcm_s64le         tiff
adpcm_yamaha    hap             movtext         pcm_s8            truehd
alac            hevc_nvenc      mp2             pcm_s8_planar     tta
alac_at         hevc_qsv        mp2fixed        pcm_u16be         utvideo
alias_pix       hevc_vaapi      mpeg1video      pcm_u16le         v210
amv             huffyuv         mpeg2_qsv       pcm_u24be         v308
apng            ilbc_at         mpeg2_vaapi     pcm_u24le         v408
ass             jpeg2000        mpeg2video      pcm_u32be         v410
asv1            jpegls          mpeg4           pcm_u32le         vc2
asv2            libfdk_aac      msmpeg4v2       pcm_u8            vorbis
avrp            libgsm          msmpeg4v3       pcx               vp8_vaapi
avui            libgsm_ms       msvideo1        pgm               wavpack
ayuv            libilbc         nellymoser      pgmyuv            webvtt
bmp             libkvazaar      nvenc           png               wmav1
cinepak         libmp3lame      nvenc_h264      ppm               wmav2
cljr            libopencore_amrnb nvenc_hevc    prores            wmv1
comfortnoise    libopenh264     opus            prores_aw         wmv2
dca             libopenjpeg     pam             prores_ks         wrapped_avframe
dnxhd           libopus         pbm             qtrle             xbm
dpx             libshine        pcm_alaw        r10k              xface
dvbsub          libspeex        pcm_alaw_at     r210              xsub
dvdsub          libtheora       pcm_f32be       ra_144            xwd
dvvideo         libtwolame      pcm_f32le       rawvideo          y41p
eac3            libvo_amrwbenc  pcm_f64be       roq               yuv4
ffv1            libvorbis       pcm_f64le       roq_dpcm          zlib
```

```
    ffvhuff         libvpx_vp8          pcm_mulaw      rv10              zmbv
    flac            libvpx_vp9          pcm_mulaw_at   rv20
```

从上面的输出信息中可以看出，FFmpeg 支持的编码器比较全面，比如 AAC、AC3、H.264、H.265、MPEG4、MPEG2VIDEO、PCM、FLV1 的编码器支持。

1.8.2 FFmpeg 的解码器支持

FFmpeg 源代码本身包含了很多的解码支持，解码主要是在输入的时候进行解码，也可以理解为将压缩过的编码进行解压缩，关于解码的支持，可以通过 ./configure –list-decoders 命令来进行查看：

```
    aac                 atrac1          eightbps          kmvc          mpl2
    aac_at              atrac3          eightsvx_exp      lagarith      msa1
    aac_fixed           atrac3al        eightsvx_fib      libcelt       mscc
    aac_latm            atrac3p         escape124         libfdk_aac    msmpeg4_
crystalhd
    aasc                atrac3pal       escape130         libgsm        msmpeg4v1
    ac3                 aura            evrc              libgsm_ms     msmpeg4v2
    ac3_at              aura2           exr               libilbc       msmpeg4v3
    ac3_fixed           avrn            ffv1              libopencore_amrnb   msrle
    adpcm_4xm           avrp            ffvhuff           libopencore_amrwb   mss1
    adpcm_adx           avs             ffwavesynth       libopenh264   mss2
    adpcm_afc           avui            fic               libopenjpeg   msvideo1
    adpcm_aica          ayuv            flac              libopus       mszh
    adpcm_ct            bethsoftvid     flashsv           librsvg       mts2
    adpcm_dtk           bfi             flashsv2          libspeex      mvc1
    adpcm_ea            bink            flic              libvorbis     mvc2
    adpcm_ea_maxis_xa   binkaudio_dct   flv               libvpx_vp8    mxpeg
    adpcm_ea_r1         binkaudio_rdft  fmvc              libvpx_vp9    nellymoser
    adpcm_ea_r2         bintext         fourxm            libzvbi_teletext    nuv
    adpcm_ea_r3         bitpacked       fraps             loco          on2avc
    adpcm_ea_xas        bmp             frwu              m101          opus
    adpcm_g722          bmv_audio       g2m               mace3         paf_audio
    adpcm_g726          bmv_video       g723_1            mace6         paf_video
    adpcm_g726le        brender_pix     g729              magicyuv      pam
    adpcm_ima_amv       c93             gif               mdec          pbm
    adpcm_ima_apc       cavs            gsm               metasound     pcm_alaw
    adpcm_ima_dat4      ccaption        gsm_ms            microdvd      pcm_alaw_at
    adpcm_ima_dk3       cdgraphics      gsm_ms_at         mimic         pcm_bluray
    adpcm_ima_dk4       cdxl            h261              mjpeg         pcm_dvd
    adpcm_ima_ea_eacs   cfhd            h263              mjpeg_cuvid   pcm_f16le
    adpcm_ima_ea_sead   cinepak         h263i             mjpegb        pcm_f24le
    adpcm_ima_iss       clearvideo      h263p             mlp           pcm_f32be
    adpcm_ima_oki       cljr            h264              mmvideo       pcm_f32le
    adpcm_ima_qt        cllc            h264_crystalhd    motionpixels  pcm_f64be
    adpcm_ima_qt_at     comfortnoise    h264_cuvid        movtext       pcm_f64le
    adpcm_ima_rad       cook            h264_mediacodec   mp1           pcm_lxf
    adpcm_ima_smjpeg    cpia            h264_mmal         mp1_at        pcm_mulaw
    adpcm_ima_wav       cscd            h264_qsv          mp1float      pcm_mulaw_at
    adpcm_ima_ws        cyuv            h264_vda          mp2           pcm_s16be
    adpcm_ms            dca             h264_vdpau        mp2_at        pcm_s16be_
planar
```

```
    adpcm_mtaf          dds             hap             mp2float        pcm_s16le
    adpcm_psx           dfa             hevc            mp3             pcm_s16le_
planar
    adpcm_sbpro_2       dirac           hevc_cuvid      mp3_at          pcm_s24be
    adpcm_sbpro_3       dnxhd           hevc_mediacodec mp3adu          pcm_s24daud
    adpcm_sbpro_4       dpx             hevc_qsv        mp3adufloat     pcm_s24le
    adpcm_swf           dsd_lsbf        hnm4_video      mp3float        pcm_s24le_
planar
    adpcm_thp           dsd_lsbf_planar hq_hqa          mp3on4          pcm_s32be
    adpcm_thp_le        dsd_msbf        hqx             mp3on4float     pcm_s32le
    adpcm_vima          dsd_msbf_planar huffyuv         mpc7            pcm_s32le_
planar
    adpcm_xa            dsicinaudio     iac             mpc8            pcm_s64be
    adpcm_yamaha        dsicinvideo     idcin           mpeg1_cuvid     pcm_s64le
    aic                 dss_sp          idf             mpeg1_vdpau     pcm_s8
    alac                dst             iff_ilbm        mpeg1video      pcm_s8_planar
    alac_at             dvaudio         ilbc_at         mpeg2_crystalhd     pcm_
u16be
    alias_pix           dvbsub          imc             mpeg2_cuvid     pcm_u16le
    als                 dvdsub          indeo2          mpeg2_mmal      pcm_u24be
    amr_nb_at           dvvideo         indeo3          mpeg2_qsv       pcm_u24le
    amrnb               dxa             indeo4          mpeg2video      pcm_u32be
    amrwb               dxtory          indeo5          mpeg4           pcm_u32le
    amv                 dxv             interplay_acm   mpeg4_crystalhd     pcm_u8
    anm                 eac3            interplay_dpcm  mpeg4_cuvid     pcm_zork
    ansi                eac3_at         interplay_video mpeg4_mediacodec    pcx
    ape                 eacmv           jacosub         mpeg4_mmal      pgm
    apng                eamad           jpeg2000        mpeg4_vdpau     pgmyuv
    ass                 eatgq           jpegls          mpeg_vdpau      pgssub
    asv1                eatgv           jv              mpeg_xvmc       pictor
    asv2                eatqi           kgv1            mpegvideo       pixlet
    pjs                 s302m           targa           vc1_mmal        wmavoice
    png                 sami            targa_y216      vc1_qsv         wmv1
    ppm                 sanm            tdsc            vc1_vdpau       wmv2
    prores              scpr            text            vc1image        wmv3
    prores_lgpl         screenpresso    theora          vcr1            wmv3_crystalhd
    psd                 sdx2_dpcm       thp             vmdaudio        wmv3_vdpau
    ptx                 sgi             tiertexseqvideo vmdvideo        wmv3image
    qcelp               sgirle          tiff            vmnc            wnv1
    qdm2                sheervideo      tmv             vorbis          ws_snd1
    qdm2_at             shorten         truehd          vp3             xan_dpcm
    qdmc                sipr            truemotion1     vp5             xan_wc3
    qdmc_at             smackaud        truemotion2     vp6             xan_wc4
    qdraw               smacker         truemotion2rt   vp6a            xbin
    qpeg                smc             truespeech      vp6f            xbm
    qtrle               smvjpeg         tscc            vp7             xface
    r10k                snow            tscc2           vp8             xl
    r210                sol_dpcm        tta             vp8_cuvid       xma1
    ra_144              sonic           twinvq          vp8_mediacodec      xma2
    ra_288              sp5x            txd             vp8_qsv         xpm
    ralf                speedhq         ulti            vp9             xsub
    rawvideo            srgc            utvideo         vp9_cuvid       xwd
    realtext            srt             v210            vp9_mediacodec      y41p
    rl2                 ssa             v210x           vplayer         ylc
    roq                 stl             v308            vqa             yop
```

```
roq_dpcm            subrip          v408            wavpack         yuv4
rpza                subviewer       v410            webp            zero12v
rscc                subviewer1      vb              webvtt          zerocodec
rv10                sunrast         vble            wmalossless     zlib
rv20                svq1            vc1             wmapro          zmbv
rv30                svq3            vc1_crystalhd   wmav1
rv40                tak             vc1_cuvid       wmav2
```

从上面的输出信息中可以看到 FFmpeg 所支持的解码器模块 decoders 支持了 MPEG4、H.264、H.265（HEVC）、MP3 等格式。

1.8.3 FFmpeg 的封装支持

FFmpeg 的封装（Muxing）是指将压缩后的编码封装到一个容器格式中，如果要查看 FFmpeg 源代码中都可以支持哪些容器格式，可以通过命令 ./configure --list-muxers 来查看：

```
a64             filmstrip       matroska_audio  pcm_f32be       smjpeg
ac3             flac            md5             pcm_f32le       smoothstreaming
adts            flv             microdvd        pcm_f64be       sox
adx             framecrc        mjpeg           pcm_f64le       spdif
aiff            framehash       mkvtimestamp_v2 pcm_mulaw       spx
amr             framemd5        mlp             pcm_s16be       srt
apng            g722            mmf             pcm_s16le       stream_segment
asf             g723_1          mov             pcm_s24be       swf
asf_stream      gif             mp2             pcm_s24le       tee
ass             gsm             mp3             pcm_s32be       tg2
ast             gxf             mp4             pcm_s32le       tgp
au              h261            mpeg1system     pcm_s8          truehd
avi             h263            mpeg1vcd        pcm_u16be       tta
avm2            h264            mpeg1video      pcm_u16le       uncodedframecrc
bit             hash            mpeg2dvd        pcm_u24be       vc1
caf             hds             mpeg2svcd       pcm_u24le       vc1t
cavsvideo       hevc            mpeg2video      pcm_u32be       voc
chromaprint     hls             mpeg2vob        pcm_u32le       w64
crc             ico             mpegts          pcm_u8          wav
dash            ilbc            mpjpeg          psp             webm
data            image2          mxf             rawvideo        webm_chunk
daud            image2pipe      mxf_d10         rm              webm_dash_manifest
dirac           ipod            mxf_opatom      roq             webp
dnxhd           ircam           null            rso             webvtt
dts             ismv            nut             rtp             wtv
dv              ivf             oga             rtp_mpegts      wv
eac3            jacosub         ogg             rtsp            yuv4mpegpipe
f4v             latm            ogv             sap
ffm             lrc             oma             scc
ffmetadata      m4v             opus            segment
fifo            matroska        pcm_alaw        singlejpeg
```

从封装（又称复用）格式所支持的信息中可以看到，FFmpeg 支持生成裸流文件，如 H.264、AAC、PCM，也支持一些常见的格式，如 MP3、MP4、FLV、M3U8、WEBM 等。

1.8.4 FFmpeg 的解封装支持

FFmpeg 的解封装（Demuxing）是指将读入的容器格式拆解开，将里面压缩的音频流、视频流、字幕流、数据流等提取出来，如果要查看 FFmpeg 的源代码中都可以支持哪些输入的容器格式，可以通过命令 ./configure --list-demuxers 来查看：

```
    aa              dv              image_sgi_pipe      nsv         smjpeg
    aac             dvbsub          image_sunrast_pipe  nut         smush
    ac3             dvbtxt          image_svg_pipe      nuv         sol
    acm             dxa             image_tiff_pipe     ogg         sox
    act             ea              image_webp_pipe     oma         spdif
    adf             ea_cdata        image_xpm_pipe      paf         srt
    adp             eac3            ingenient           pcm_alaw    stl
    ads             epaf            ipmovie             pcm_f32be   str
    adx             ffm             ircam               pcm_f32le   subviewer
    aea             ffmetadata      iss                 pcm_f64be   subviewer1
    afc             filmstrip       iv8                 pcm_f64le   sup
    aiff            flac            ivf                 pcm_mulaw   svag
    aix             flic            ivr                 pcm_s16be   swf
    amr             flv             jacosub             pcm_s16le   tak
    anm             fourxm          jv                  pcm_s24be   tedcaptions
    apc             frm             libgme              pcm_s24le   thp
    ape             fsb             libmodplug          pcm_s32be   threedostr
    apng            g722            libopenmpt          pcm_s32le   tiertexseq
    aqtitle         g723_1          live_flv            pcm_s8      tmv
    asf             g729            lmlm4               pcm_u16be   truehd
    asf_o           genh            loas                pcm_u16le   tta
    ass             gif             lrc                 pcm_u24be   tty
    ast             gsm             lvf                 pcm_u24le   txd
    au              gxf             lxf                 pcm_u32be   v210
    avi             h261            m4v                 pcm_u32le   v210x
    avisynth        h263            matroska            pcm_u8      vag
    avr             h264            mgsts               pjs         vc1
    avs             hevc            microdvd            pmp         vc1t
    bethsoftvid     hls             mjpeg               pva         vivo
    bfi             hnm             mjpeg_2000          pvf         vmd
    bfstm           ico             mlp                 qcp         vobsub
    bink            idcin           mlv                 r3d         voc
    bintext         idf             mm                  rawvideo    vpk
    bit             iff             mmf                 realtext    vplayer
    bmv             ilbc            mov                 redspark    vqf
    boa             image2          mp3                 rl2         w64
    brstm           image2_alias_pix    mpc             rm          wav
    c93             image2_brender_pix  mpc8            roq         wc3
    caf             image2pipe      mpegps              rpl         webm_dash_
manifest
    cavsvideo       image_bmp_pipe  mpegts              rsd         webvtt
    cdg             image_dds_pipe  mpegtsraw           rso         wsaud
    cdxl            image_dpx_pipe  mpegvideo           rtp         wsd
    cine            image_exr_pipe  mpjpeg              rtsp        wsvqa
    concat          image_j2k_pipe  mpl2                sami        wtv
    dash            image_jpeg_pipe mpsub               sap         wv
    data            image_jpegls_pipe   msf             sbg         wve
    daud            image_pam_pipe  msnwc_tcp           scc         xa
```

```
dcstr          image_pbm_pipe      mtaf            sdp          xbin
dfa            image_pcx_pipe      mtv             sdr2         xmv
dirac          image_pgm_pipe      musx            sds          xvag
dnxhd          image_pgmyuv_pipe   mv              sdx          xwma
dsf            image_pictor_pipe   mvi             segafilm     yop
dsicin         image_png_pipe      mxf             shorten      yuv4mpegpipe
dss            image_ppm_pipe      mxg             siff
dts            image_psd_pipe      nc              sln
dtshd          image_qdraw_pipe    nistsphere      smacker
```

从解封装（Demuxer，又称解复用）格式支持信息中可以看到，FFmpeg 源代码中已经支持的 demuxer 非常多，包含图片（image）、MP3、FLV、MP4、MOV、AVI 等。

1.8.5 FFmpeg 的通信协议支持

FFmpeg 不仅仅支持本地的多媒体处理，而且还支持网络流媒体的处理，支持的网络流媒体协议相对来说也很全面，可以通过命令 ./configure --list-protocols 查看：

```
async          gopher      librtmpte       rtmps      tls_gnutls
bluray         hls         libsmbclient    rtmpt      tls_openssl
cache          http        libssh          rtmpte     tls_schannel
concat         httpproxy   md5             rtmpts     tls_securetransport
crypto         https       mmsh            rtp        udp
data           icecast     mmst            sctp       udplite
ffrtmpcrypt    librtmp     pipe            srtp       unix
ffrtmphttp     librtmpe    prompeg         subfile
file           librtmps    rtmp            tcp
ftp            librtmpt    rtmpe           tee
```

从协议的相关信息列表中可以看到，FFmpeg 支持的流媒体协议比较多，包括 MMS、HTTP、HTTPS、HLS（M3U8）、RTMP、RTP，甚至支持 TCP、UDP，其也支持使用 file 协议的本地文件操作和使用 concat 协议支持的多个文件串流操作，后面的章节中会有详细的介绍。

1.9 小结

本章重点介绍了 FFmpeg 的获取、安装、容器封装与解封装的格式支持、音视频编码与解码的格式支持，以及流媒体传输协议的支持。综合来说，FFmpeg 所支持的容器、编解码、协议相对来说比较全面，是一款功能强大的多媒体处理工具和开发套件。