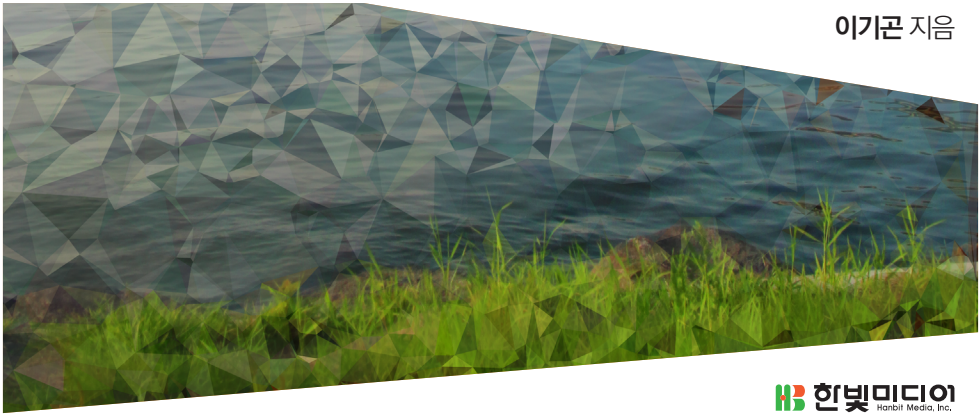
Hanbit RealTime 111

FFmpeg 라이브러리

코덱과 영상 변환을 중심으로

이기곤 지음



한빛미디어 Hanbit Media, Inc.

FFmpeg 라이브러리

코덱과 영상 변환을 중심으로

이기곤 지음

한빛미디어
Hanbit Media, Inc.

표지 사진 **유형진**

이 책의 표지는 유형진 님이 보내 주신 풍경 사진을 담았습니다.
리얼타임은 독자의 시선을 담은 풍경사진을 책 표지로 보여주고자 합니다.

사진 보내기 **ebookwriter@hanbit.co.kr**

FFmpeg 라이브러리 코덱과 영상 변환을 중심으로

초판발행 2015년 8월 13일

지은이 이기곤 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-772-9 15000 / **정가** 9,900원

총괄 배용석 / **책임편집** 김창수 / **기획·편집** 정지연 / **교정** 이미연
디자인 표지/내지 여동일, 조판 최송실
마케팅 박상용 / **영업** 김형진, 김진불, 조유미

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

지은이_ **이기곤**

아이렌소프트(AirenSoft)에서 풀스택(full-stack) 개발자로 일하고 있다. Github (https://github.com/sorrowhill)을 운영하고 있으며 멀티미디어와 관련된 라이브러리를 주로 다룬다. 사용하는 언어로는 C/C++를 가장 선호하며 현재는 대부분의 시간을 Rust와 함께 보내고 있다.

FFmpeg은 멀티미디어 분야에서 가장 많이 사용되는 오픈소스 프로젝트입니다. 멀티미디어는 매우 방대하고 쉽게 접근하기 어려운 분야지만, 멀티미디어를 쉽게 다룰 수 있도록 FFmpeg은 영상의 변환, 재생, 스트리밍 등의 강력한 기능을 제공합니다.

그런데 현재 FFmpeg 라이브러리를 자세하게 다루는 문서는 거의 없다시피 합니다. 인터넷에 게재된 소스 코드들도 너무 오래되어 최신 라이브러리에 적용하기에는 문제가 많으며 이해하기도 힘듭니다.

모든 멀티미디어 지식을 이 책에 담지는 못하지만, FFmpeg을 사용하면서 겪었던 경험을 되살려 반드시 알아야만 하는 요소 위주로 정리하였습니다. 이 책에서 소개하는 내용이 잘 이해되지 않는다고 해서 FFmpeg을 사용하지 못하는 것은 아닙니다. 오히려 FFmpeg을 사용하면서 이해가 되는 경우가 더 많기 때문에 책을 보며 이해가 되지 않는 부분은 가볍게 넘어가셔도 괜찮습니다.

이 책이 나오기까지 많은 분의 도움이 있었습니다. 늦은 원고 마감에도 출판을 위해 힘써주신 한빛미디어 관계자분들께 감사의 말씀을 드립니다. 많은 조언과 응원을 주신 부모님, 친구, 동료들께도 감사드립니다.

대상 독자 및 참고사항

이 책은 리눅스 기반의 개발 환경에서 FFmpeg 라이브러리를 사용하거나 사용하려는 독자를 대상으로 합니다. 따라서 C 언어를 깊게 이해하고 있어야 하며 GNU Make와 GNU GCC에 대한 전반적인 이해가 필요합니다. 또한, 이 책에서는 안드로이드 NDK와 관련된 내용은 다루지 않습니다.

이 책을 쓰는 시점에서 사용한 FFmpeg 버전은 2.7.1이며 메이저 버전이 낮거나 높은 FFmpeg 버전에서는 일부 코드가 동작하지 않을 수 있습니다.

이 책에서 사용한 코드는 Github(https://github.com/sorrowhill)에서 다운로드할 수 있습니다.

한빛 리얼타임

한빛 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾기도 쉽지 않습니다. 또한, 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1 eBook First – 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다

500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 좀 더 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한, 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2 무료로 업데이트되는 전자책 전용 서비스입니다

종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3 독자의 편의를 위해 DRM-Free로 제공합니다

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 어려운 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

차례

Part 2 FFmpeg 라이브러리 활용 — 039

Part 1

동영상에 대한 이해

FFmpeg 라이브러리는 멀티미디어를 쉽게 다룰 수 있게 도와주는 도구입니다. 하지만 FFmpeg 라이브러리는 다양한 동영상 규격을 통합하여 동일한 인터페이스로 사용할 수 있게 도와줄 뿐 동영상을 쉽게 만들어 주지는 않습니다. 따라서 FFmpeg 라이브러리를 사용하려면 먼저 동영상의 특징과 구조에 대해 알아야 합니다.

1장에서는 동영상에 대한 전반적인 지식을 살펴봅니다. 동영상을 이루는 요소에 무엇이 있는지와 함께 각각의 요소가 왜 필요한지에 대해서 알아봅니다.

2장에서는 동영상의 요소 중 하나인 컨테이너와 코덱의 특징을 집중적으로 살펴봅니다. 그리고 현재 이슈로 떠오르는 4K 해상도가 등장한 이유와 차세대 코덱의 특징을 살펴보겠습니다.

chapter 1

동영상의 구성

1.1 컨테이너

동영상 파일은 재생과 편집을 원활하게 하기 위한 일련의 규격을 담고 있습니다. 이러한 규격을 가진 파일을 컨테이너Container라고 합니다. 동영상 파일이 어떤 규격의 컨테이너를 가졌는지는 파일의 확장자로 알 수 있습니다.

[그림 1-1]과 같이 컨테이너는 하나 이상의 스트림Stream을 가지고 있습니다. 여기서 스트림이란 비디오, 오디오와 같이 시간에 따라 변하는 일련의 데이터를 의미합니다.

그림 1-1 컨테이너의 구성

컨테이너는 오직 스트림을 제어하기 위한 정보만을 가지고 있을 뿐 스트림이 어떤 방식으로 압축되었는지는 알 수 없습니다. 또한, 규격에 맞는 스트림만을 담을 수 있기 때문에 모든 형식의 스트림을 담을 수는 없습니다.

캡처된 비디오와 녹음된 오디오를 저장하기 위해서는 반드시 이를 제어할 수 있는 컨테이너에 담아야 합니다. 이때 컨테이너에 스트림을 담는 일련의 과정을 멀티플

렉싱Multiplexing, 줄여서 먹싱Muxing이라 합니다. 반대로 컨테이너에 있는 스트림을 컨테이너에서 분리하는 일련의 과정을 디멀티플렉싱Demultiplexing이라 합니다.

동영상 컨테이너가 스트림을 제어하는 데 사용하는 정보 중 대표적인 것은 다음과 같습니다.

- 동영상 촬영 당시, 또는 임의로 기록된 메타 정보(촬영 날짜, 위치 등)
- 컨테이너가 가지고 있는 스트림의 개수
- 동영상의 전체 길이
- DVD에서 제공하는 메뉴와 자막 정보
- 인터넷을 통한 동영상 재생(스트리밍) 시 빠른 탐색fast seek에 필요한 스트림 위치 정보

물론 모든 컨테이너가 이와 같은 정보를 가지고 있지는 않습니다. 스트리밍에 특화된 컨테이너가 있는가 하면, 확장성을 고려하여 설계한 컨테이너도 있습니다. 또한, 예전에는 하드디스크 고장이 잦았기 때문에 안전한 저장을 목적으로 설계한 컨테이너도 있었습니다. 이와 관련된 자세한 이야기는 2장에서 살펴보겠습니다.

1.2 코덱

아날로그 신호로 이루어진 영상을 디지털 영상으로 변환하는 작업에는 여러 어려움이 있습니다. 가장 큰 어려움은 아날로그 데이터를 압축하지 않고 저장하기에 영상의 크기가 너무 크다는 것입니다. 그래서 영상을 조금 더 효율적으로 압축하기 위한 연구가 끊임없이 이루어져 왔습니다.

[그림 1-2]처럼 코덱Codec은 아날로그 신호나 스트림 데이터로 이루어진 비디오와 오디오를 압축된 부호로 변환하기 위한 압축 규격을 제공하며, 이 과정을 인코딩Encoding이라 합니다. 또한, 코덱은 압축된 데이터를 본래의 아날로그 신호나 스트림 데이터로 복원하기 위한 규격도 제공하는데, 이 과정을 디코딩Decoding이라 합

니다. 대부분 코덱은 인코딩과 디코딩을 모두 지원하지만, 그렇지 않은 코덱도 있습니다. 디코딩에는 제한이 없지만 인코딩하기 위해서는 로열티를 내야 하는 코덱도 있습니다.

그림 1-2 코덱의 역할

코덱의 압축 방식에는 원본이 손상되는 손실 압축, 원본의 손실 없이 그대로 보전되는 무손실 압축이 있습니다. 압축 효율은 손실 압축 방식이 더 높지만 원본을 보전해야 하는 파일이나 편집과 같은 특수한 상황에서 사용할 때는 무손실 압축 방식을 선호하기도 합니다.

그림 1-3 압축률 인자 q(100=최대값, 1=최소값)에 따른 압축률 비교[01]

01 그림 출처: (c) copyright 2008, Blender Foundation, www.bigbuckbunny.org

Part 2

FFmpeg 라이브러리 활용

FFmpeg은 동영상의 재생, 녹화, 스트리밍과 변환을 위해 필요한 기능을 제공하는 멀티미디어 프레임워크입니다. 그중에서도 가장 강력한 기능인 영상 변환에 대하여 살펴보겠습니다.

먼저 3장에서는 라이브러리를 사용하기 전에 알아두어야 할 내용에 대해 살펴봅니다. 개발환경을 구성하는 방법부터 FFmpeg에서 제공하는 기능에 대해 간략하게 알아보겠습니다.

4장과 5장에서는 FFmpeg 라이브러리로 영상을 변환하는 데 필요한 기능을 단계별로 설명합니다. 컨테이너에서 스트림을 분해하는 과정부터 영상을 원하는 규격으로 인코딩하는 방법까지 살펴봅니다.

chapter 3

FFmpeg 살펴보기

FFmpeg은 라이브러리 규모가 큰 편인 데다가 사용되는 외부 라이브러리도 많아서 개발환경을 구성하기가 쉽지 않습니다. 이번 장에서는 개발환경을 구성하는 방법과 함께 FFmpeg이 가진 기능과 FFmpeg에서 사용하는 컨텍스트Context를 간단하게 살펴봅니다.

3.1 개발환경 구성

이 책에서는 FFmpeg 빌드를 위해 사용하는 특별한 설정이 몇 가지 있습니다.

첫 번째는 독립된 개발 환경을 위해 라이브러리 설치 경로를 변경합니다. 설치 경로는 '/opt/ffmpeg'입니다. 두 번째로 라이브러리는 정적 라이브러리를 사용하지 않고 동적 라이브러리를 사용합니다. 정적 라이브러리가 필요할 경우에는 `--enable-static` 옵션을 추가하면 됩니다. 마지막으로 H.264(x264)와 AAC(fdk_aac), MP3(mp3lame)를 외부 코덱으로 사용합니다.

> NOTE
>
> 추가 코덱이 필요한 경우에는 FFmpeg Codec 문서(https://www.ffmpeg.org/ffmpeg-codecs.html)에서 사용할 수 있는 코덱 목록을 확인할 수 있습니다.

FFmpeg은 기본으로 소프트웨어 방식 인코딩을 사용합니다. 따라서 사용할 수 있는 CPU 자원이 많으면 많을수록 좋습니다. 가상 머신을 사용할 경우에는 가상

머신에 CPU 자원을 많이 할당하는 것이 큰 도움이 됩니다.

메모리는 그렇게 많이 필요하지 않습니다. 물론 많은 메모리를 사용하도록 프로그램을 개발할 수도 있습니다. 하지만 사용할 수 있는 CPU 자원이 제한되어 있어서 많은 메모리를 사용하더라도 인코딩 성능이 개선되지는 않습니다.

> NOTE
>
> 이 장에서는 CentOS 7을 기준으로 FFmpeg을 설치합니다. 데비안/우분투 계열 리눅스에서는 다음 링크를 참고하여 개발환경을 구성할 수 있습니다.
>
> https://trac.ffmpeg.org/wiki/CompilationGuide/Ubuntu

가장 먼저 FFmpeg을 빌드하는 데 필요한 컴파일러와 빌드 소프트웨어를 설치합니다.

```
# yum install git autoconf automake cmake gcc gcc-c++ libtool make nasm
pkgconfig zlib-devel
```

다음은 추가 라이브러리 경로를 위한 설정입니다. /etc/profile이나 해당 계정의 bashrc에 다음 설정을 추가합니다(여기서는 /etc/profile에 추가하였습니다).

```
FFMPEG_PREFIX=/opt/ffmpeg
FFMPEG_BIN_PREFIX=$FFMPEG_PREFIX/bin
export PATH=$PATH:$FFMPEG_BIN_PREFIX
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$FFMPEG_PREFIX/lib
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$FFMPEG_PREFIX/lib/pkgconfig
```

이후 환경 설정을 다음과 같이 적용합니다.

```
# source /etc/profile
```

yasm 설치

x264를 사용하기 위해서는 x264에서 사용하는 yasm 어셈블러가 필요합니다. 설치 경로는 앞서 설정한 $FFMPEG_PREFIX와 $FFMPEG_BIN_PREFIX를 사용합니다. 적당한 곳에 yasm을 받은 뒤 다음과 같이 설치를 진행합니다.

```
git clone --depth 1 git://github.com/yasm/yasm.git
cd yasm
./autogen.sh
./configure --prefix="$FFMPEG_PREFIX" --bindir="$FFMPEG_BIN_PREFIX"
make -j 4
make install
```

x264 설치

yasm을 설치한 후에는 H.264 인코더와 디코더를 사용할 수 있는 x264를 설치합니다.

```
git clone --depth 1 git://git.videolan.org/x264
cd x264
./configure --prefix="$FFMPEG_PREFIX" --bindir="$FFMPEG_BIN_PREFIX" --enable-
shared
make -j 4
make install
```

fdk_aac 설치

FFmpeg에서는 자체적으로 AAC 인코더를 지원하지만 fdk_aac로 더 높은 품질의 AAC를 얻을 수 있습니다. 여기서는 fdk_aac로 AAC를 인코딩하겠습니다.

```
git clone --depth 1 git://git.code.sf.net/p/opencore-amr/fdk-aac
cd fdk-aac
./autogen.sh
./configure --prefix="$FFMPEG_PREFIX" --enable-shared
make -j 4
make install
```

LAME 설치

MP3 인코더와 디코더로 가장 널리 알려진 LAME을 설치합니다. 설치 과정은 다음과 같습니다.

```
curl -L -O http://downloads.sourceforge.net/project/lame/lame/3.99/lame-3.99.5.tar.gz
tar -xvzf lame-3.99.5.tar.gz
cd lame-3.99.5
./configure --prefix="$FFMPEG_PREFIX" --enable-shared
make -j 4
make install
```

FFmpeg 설치

외부 코덱 설치가 모두 완료되었다면 FFmpeg을 설치합니다. 앞서 설치한 x264와 fdk_aac 인코더를 사용하려면 FFmpeg의 라이선스를 GPLv3으로 변경해야 합니다. 필터링을 사용하는 데 필요한 avfilter도 추가해야 합니다. configure에서 사용할 수 있는 다양한 옵션과 이에 대한 설명은 부록을 참고하기 바랍니다.

```
git clone --depth 1 git://source.ffmpeg.org/ffmpeg
cd ffmpeg
./configure --prefix="$FFMPEG_PREFIX" --enable-shared --enable-gpl --enable-version3 --enable-nonfree --extra-cflags="-I$FFMPEG_PREFIX/include" --extra-ldflags="-L$FFMPEG_PREFIX/lib" --extra-libs=-ldl --enable-avfilter --enable-libx264 --enable-libfdk_aac --enable-libmp3lame
make -j 4
make install
```

모든 설치가 끝나면 정상적으로 FFmpeg을 실행할 수 있는지 확인합니다.

```
#ffmpeg
ffmpeg version git-2015-06-03-9614df4 Copyright (c) 2000-2015 the FFmpeg developers
built with gcc 4.8.3 (GCC) 20140911 (Red Hat 4.8.3-9)
```

```
configuration: --prefix=/opt/ffmpeg --enable-shared --enable-gpl --enable-
version3 --enable-nonfree --extra-cflags=-I/opt/ffmpeg/include --extra-
ldflags=-L/opt/ffmpeg/lib --extra-libs=-ldl --enable-avfilter --enable-libx264
--enable-libfdk_aac --enable-libmp3lame
...
```

이렇게 하여 FFmpeg을 사용할 수 있는 환경 구성이 완료되었습니다.

3.2 FFmpeg CLI 사용법

FFmpeg을 설치하였다면 ffmpeg과 ffprobe를 사용할 수 있습니다. 이번 절에서는 FFmpeg 라이브러리를 사용하기 전에 간단하게 ffmpeg과 ffprobe 프로그램을 사용해 보겠습니다.

NOTE

이 책에서 사용하는 샘플 동영상은 다음 주소에서 다운로드할 수 있습니다.

http://mirrorblender.top-ix.org/peach/bigbuckbunny_movies/big_buck_bunny_480p_h264.mov

다른 영상을 사용하여도 상관없습니다. 하지만 테스트에 사용할 영상인 만큼 다음과 같은 조건을 만족하면 좋습니다.

- 비디오 코덱은 H.264나 MPEG4와 같이 플랫폼에 종속되지 않는 것이 좋습니다.
- 오디오 코덱도 마찬가지로 AAC나 MP3가 가장 좋습니다.
- 영상의 길이가 길면 그만큼 CPU 자원을 쓰기 때문에 영상의 길이가 짧을수록 좋습니다.

3.2.1 ffmpeg

ffmpeg은 영상을 변환할 때 사용하는 프로그램으로, FFmpeg 라이브러리에서 사용할 수 있는 모든 기능이 있습니다. 다음은 ffmpeg 프로그램의 기본적인 명령어입니다.

```
# ffmpeg -i [INPUT] -c:v [video codec] -c:a [audio codec] [OUTPUT]
```

어떤 영상을 H.264와 AAC 영상으로 변환하고 싶다면 다음과 같이 입력하면 됩니다.

```
# ffmpeg -i big_buck_bunny_480p_h264.mov -c:v libx264 -c:a libfdk_aac ./out.mp4
```

테스트 영상은 라이브러리를 이용하여 영상을 사용하는 데 목적이 있기 때문에 영상이 길어야 할 이유가 없습니다. 따라서 앞으로 사용할 샘플 영상의 길이를 다음과 같이 조절합니다.

```
# ffmpeg -i big_buck_bunny_480p_h264.mov -ss 00:00:00 -t 00:01:00 -c:v libx264
-c:a libfdk_aac ./sample.mp4
```

3.2.2 ffprobe

ffprobe는 특정 영상에 대한 정보를 빠르게 파악할 수 있기 때문에 가장 많이 사용되는 스캐닝 툴입니다. 비슷한 툴로는 MediaInfo[01]가 있습니다.

다음은 ffprobe 프로그램의 기본적인 사용 방법입니다.

```
# ffprobe ./sample.mp4
...
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from './sample.mp4':
    Metadata:
        ...
    Duration: 00:01:00.04, start: 0.042667, bitrate: 1491 kb/s
        Stream #0:0(eng): Video: h264 (High) (avc1 / 0x31637661), yuv420p,
854x480, 997 kb/s, 24 fps, 24 tbr, 12288 tbn, 48 tbc (default)
        ...
        Stream #0:1(eng): Audio: aac (LC) (mp4a / 0x6134706D), 48000 Hz, 5.1,
fltp, 488 kb/s (default)
        ...
```

01 https://mediaarea.net

컨테이너와 코덱에 대한 정보를 한눈에 볼 수 있습니다. 더 자세한 정보가 필요한 경우에는 -show_streams 옵션을 추가하면 확인할 수 있습니다.

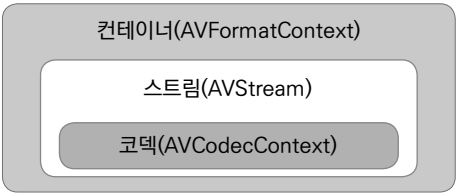
```
# ffprobe ./sample.mp4 -show_streams
[STREAM]
index=0
codec_name=h264
codec_long_name=H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10
profile=High
codec_type=video...
[/STREAM]
[STREAM]
index=1
codec_name=aac
codec_long_name=AAC (Advanced Audio Coding)
profile=LC
codec_type=audio...
[/STREAM]
```

Part 1에서 본 익숙한 용어가 보입니다. ffprobe는 영상을 분석할 때 가장 먼저 사용하는 툴이며 소스도 매우 간단하여 FFmpeg 라이브러리에 익숙해지기 위해서는 소스를 분석해 보는 것도 큰 도움이 됩니다.

3.3 FFmpeg 구조

FFmpeg은 컨테이너와 스트림, 코덱을 관리하기 위해 [그림 3-1]과 같이 각각의 컨텍스트라는 구조체로 관리합니다.

그림 3-1 FFmpeg 구조



AVFormatContext

AVFormatContext는 파일로부터 읽은 컨테이너의 내용을 저장하거나 새로 생성한 컨테이너를 파일에 쓰기 위한 용도로 사용합니다. AVFormatContext 안에는 상당히 많은 변수가 들어 있는데 읽은 파일에서 사용하는 변수와 파일을 쓰기 위해 사용하는 변수가 같은 컨텍스트 안에 있기 때문입니다.

AVStream

AVFormatContext 내부에는 적어도 하나 이상의 스트림이 있습니다. 허용 가능한 최대 스트림 개수는 컨테이너의 종류에 따라 다르며, 이 스트림 정보는 AVStream 컨텍스트 안에 있습니다. AVStream 컨텍스트에서는 시간과 관련된 정보를 가져올 수 있습니다. 가장 많이 사용하는 정보로는 프레임 레이트와 타임베이스Timebase가 있습니다.

NOTE

타임베이스는 컨테이너가 영상을 재생할 때 사용하는 시간 단위로, FFmpeg에서는 타임베이스를 변환하는 과정이 두 군데 있습니다.

첫 번째는 파일을 읽은 직후입니다. 원본 컨테이너에서 사용하는 타임베이스를 FFmpeg에서 사용하는 타임베이스로 변경하여 변환 작업에 유용하게 사용하기 위해서입니다.

두 번째는 인코딩 이후 파일을 쓰기 전입니다. FFmpeg에서 사용하는 타임베이스를 컨테이너에서 사용하는 타임베이스로 변환해야 컨테이너에서 정상적인 재생이 가능하기 때문입니다.

AVCodecContext

AVStream 안에는 코덱과 관련된 정보를 가진 AVCodecContext가 있습니다. 이 컨텍스트는 AVStream 안에 한 개만 있으며 코덱의 종류에 따라 가진 정보가 다를 수 있습니다. 비디오 코덱인 경우에는 해상도, 픽셀 포맷과 같은 정보가 있으며 오디오의 경우에는 샘플레이트, 채널 개수와 같은 정보를 담고 있습니다.