FFmpeg を Android に移植する

原文リンク（CSDN）

縁起

FFmpeg についてまだ詳しくない方は、まずFFmpeg の基本的なコマンドを参照してみてください。その機能を詳しく研究することで、例えば音声と動画の合成、さまざまなコーデックの再生、動画の切り取り、複数の画像から動画を作成、音声と動画のミキシング、フォーマット変換など、多くの応用シーンで強力で興味深い機能を提供することができます。

「配音秀」のようなアプリを例に挙げると、ボイスオーバーは非常に楽しいだけでなく、製品の魅力を高めることがよくあります。私たちはボイスオーバーモジュールを開発する計画を立て、FFmpeg に目をつけ、それを Android プラットフォームに移植しようと試みました。3〜4 日かけて、いくつかのバージョンを試しましたが、ネット上の多くの記事では成功しませんでした。最終的に、「android 使用 ffmpeg 并调用接口」というチュートリアルを見つけて、ようやく成功しました。実際のテストでは、FFmpeg 1.2 と ndk-r9 の組み合わせのみが移植に成功することが確認されました。

考え方

FFmpeg は C 言語で書かれたプロジェクトで、その中には `main()` 関数が含まれています。私たちの目標は：

1. Android NDK を使用して `libffmpeg.so` をコンパイルします。
2. このライブラリを利用して `ffmpeg.c` ファイルをコンパイルし、元の `main()` 関数を `video_merge(int argc, char **argv)` に変更します。これにより、JNI から直接呼び出して動画の合成などの操作を行うことができます。

例えば、以下のような方法でビデオ合成を実現できます（対応するコマンドラインは `ffmpeg -i src1 -i src2 -y output` です）：

```
video_merge(5, argv); // ここで argv はコマンドライン引数をシミュレートします
```

環境

- オペレーティングシステム：Ubuntu 12.04
- FFmpeg バージョン：1.2
- NDK バージョン：ndk-r9

手を動かす前に、関連するチュートリアルをいくつか参照することをお勧めします。問題に遭遇したら、この記事に戻って比較し、遠回りを避けるようにしましょう。

インターフェースと Android.mk の修正

FFmpeg の JNI インターフェースを開発する際には、`.so` ファイルを生成するために `Android.mk` ファイルを記述し、ライブラリをリンクする必要があります。いくつかのサンプルの `Android.mk` は、異なる環境では直接実行できない場合があります。このファイルの役割は、NDK に対してどのソースファイルをコンパイルし、どのライブラリにリンクするかなどの情報を伝えることです。

私は「二回のコンパイル、その後リンク」という方法を採用しました：

1. まず、`myffmpeg` という名前の共有ライブラリをコンパイルします。
2. 次に、別のモジュール `ffmpeg-jni` の中で、`myffmpeg` をリンクし、最終的に必要な `.so` ファイルを生成します。

さらに、コンパイルされた `libffmpeg.so` を `jni` ディレクトリに配置する必要があります。これにより、リンク時に見つけることができるようになります。

FFmpeg のデバッグ

FFmpeg を移植した後、JNI を介して関数を呼び出す際、C 層でのデバッグが必要になることがよくあります。コマンドラインと同じように詳細なログ出力が見られれば、問題の特定がより簡単になります。

Eclipse で、Ctrl を押しながら `av_log` のような呼び出し位置をクリックすると、`ffmpeg/libavutil/log.c` 内の `av_log_default_callback` 関数の実装にたどり着きます。この関数は、Android の

__android_log_print を呼び出して Logcat に出力します。これらの出力を確認することで、FFmpeg の内部状態を把握し、合成の失敗や特定のコーデックがサポートされていないといった問題を調査することができます。

時々、FFmpeg が例外をスローしてアプリがクラッシュすることがあります。以下のコマンドを使用して問題を特定できます：

```
adb shell logcat | ndk-stack -sym obj/local/armeabi
```

このコマンドは、Android デバイスからログを取得し、ndk-stack ツールを使用してネイティブコードのスタックトレースをシンボル化するものです。obj/local/armeabi ディレクトリには、デバッグシンボルが含まれており、これを使用してクラッシュやエラーの詳細な情報を解析できます。

もし FFmpeg のオリジナルの main() の最後に exit(0) がある場合、それをコメントアウトすることを忘れないでください。そうしないと、アプリケーションが終了してしまいます。

メモリリークと Service ソリューション

合成が完了した後、再度呼び出した際に "INVALID HEAP ADDRESS IN dlfree ffmpeg" エラーが発生する場合、その多くは FFmpeg のメモリ解放が不完全であることが原因です。一つの妥協策として、合成プロセスを別の Service に分離し、合成が完了した後にその Service を終了させてリソースをクリーンアップする方法があります。

```
<!-- AndroidManifest.xml -->
<service android:name=".FFmpegService" />
```

このコードは、Android アプリケーションのマニフェストファイル（AndroidManifest.xml）に記述されるサービス宣言です。.FFmpegService という名前のサービスをアプリケーションに登録しています。このサービスは、FFmpeg を使用してメディア処理を行うためのバックグラウンドタスクを実行するために使用される可能性があります。

Receiver を登録するなどして、合成が完了した後に Service を終了させることで、繰り返し呼び出し時のメモリ問題を回避できます。

可能性のある問題

- **AAC ファイル再生の異常**
  一部の機種（例：Xiaomi 2s）では、デフォルトの `MediaPlayer` を使用して AAC エンコードのオーディオを再生できない場合があります。

- **エンコーダーのサポート不足**
  AMR-NB や MP3 などをサポートする場合、FFmpeg のコンパイル時に手動で対応するオプションを有効にする必要があります。コンパイルスクリプトが関連するライブラリやヘッダーファイルを見つけられない場合、エラーが発生して終了します。

- **合成速度**
  10 秒の 1280×720 ビデオを合成し、オーディオをミックスするのに数十秒から 1 分程度かかることがあります。ユーザー体験の観点から、まずユーザーに試聴させてから最終的な合成を決定する方が良いかもしれません。

配音アプリの具体的な実装において、一般的な手法は以下の通りです：

1. 事前にオリジナルの動画、字幕、および「吹き替えが必要な部分を除去した」音声ファイルをダウンロードしておく。
2. ユーザーの声を録音し、合成時には録音と無音部分を結合するだけで済むようにする。
3. ローカルでの合成時間に満足できない場合、音声と動画データをサーバーにアップロードし、サーバー側で合成を行い、完了後にダウンロードする選択肢を提供する。

Eclipse で NDK を使用する

必ずしもコマンドラインで `ndk-build` を入力する必要はありません。Eclipse でプロジェクトを右クリックし、**Android Tools → Add Native Support** を選択するだけで、その後「Run」をクリックするたびに自動的に `ndk-build` が実行されます。

JNI ヘッダーファイルをワンクリックで生成

JNI 関数のヘッダーファイルを手動で書くのは面倒ですが、`javah` コマンドを使って自動生成することができます。

Eclipse では、外部ツールとして設定し、以下のようなコマンドでヘッダーファイルを生成できます：

```
javah -jni -classpath bin/classes -d jni com.example.ffmpeg.MyFFmpeg
```

このコマンドは、`com.example.ffmpeg.MyFFmpeg` クラスの JNI（Java Native Interface）ヘッダーファイルを生成するためのものです。`-jni` オプションは JNI 形式のヘッダーファイルを生成することを指定し、`-classpath bin/classes` はクラスファイルが配置されているディレクトリを指定します。`-d jni` は生成されたヘッダーファイルを `jni` ディレクトリに出力することを意味します。

実行後、`jni` ディレクトリに `com_example_ffmpeg_MyFFmpeg.h` のようなファイルが生成されます。その後、あなたの C コードで `#include` して対応する関数を実装するだけです。

まとめ

FFmpeg を Android に移植するには、NDK 環境の設定、C/C++ のコンパイルとリンク、JNI 呼び出し、音声・動画のエンコード・デコードなど、多岐にわたる知識が必要です。合成ができない、特定のフォーマットがサポートされていない、リンクエラーが発生するなどの問題に遭遇した場合、設定やログ出力を慎重に確認する必要があります。この記事が、いくつかの落とし穴を避けるのに役立つことを願っています。もしあなたも FFmpeg を使用しているなら、コメント欄で経験や問題を共有し、互いに交流と学習を深めましょう。