# Portage de FFmpeg vers Android

Lien de l'article original (CSDN)

---

## Origine

Pour ceux qui ne connaissent pas encore FFmpeg, je vous recommande de consulter FFmpeg Common Basic Commands pour explorer ses fonctionnalités en détail. Vous y découvrirez des outils puissants et intéressants pour de nombreux scénarios d'application, tels que la synthèse audio-vidéo, la lecture de divers encodages, la capture vidéo, la création de vidéos à partir de plusieurs images, le mixage audio, la conversion de formats, et bien plus encore.

Prenons l'exemple des applications de type "doublage", où le doublage est non seulement amusant, mais rend souvent le produit plus attrayant. Nous prévoyons de développer un module de doublage et avons donc jeté notre dévolu sur FFmpeg, en essayant de le porter sur la plateforme Android. Cela nous a pris environ 3 à 4 jours, avec plusieurs versions testées, et de nombreux articles en ligne n'ont pas abouti, jusqu'à ce que nous tombions sur un tutoriel intitulé "Utilisation de FFmpeg sur Android et appel d'interfaces", qui a finalement permis de résoudre le problème. Les tests pratiques ont montré que seule la combinaison de FFmpeg 1.2 avec ndk-r9 permettait un portage réussi.

---

## Réflexion

FFmpeg est un projet écrit en langage C qui contient une fonction `main()`. Notre objectif est :

1. Utilisez Android NDK pour compiler `libffmpeg.so`.
2. Compilez et modifiez le fichier `ffmpeg.c` en utilisant cette bibliothèque, en renommant la fonction `main()` en `video_merge(int argc, char **argv)`. Cela permettra de l'appeler directement depuis JNI pour effectuer des opérations telles que la fusion de vidéos.

Par exemple, la synthèse vidéo peut être réalisée de manière similaire à ce qui suit (correspondant à la commande `ffmpeg -i src1 -i src2 -y output`) :

```
video_merge(5, argv); // où argv simule les arguments de la ligne de commande
```

---

## Environnement

- Système d'exploitation : Ubuntu 12.04
- Version de FFmpeg : 1.2
- Version de NDK : ndk-r9

Avant de vous lancer, il est recommandé de consulter quelques tutoriels pertinents. Si vous rencontrez des problèmes, revenez comparer avec cet article pour éviter de prendre trop de détours.

---

## Modifier l'interface et Android.mk

Lors de l'écriture d'une interface JNI pour FFmpeg, il est nécessaire de rédiger un fichier `Android.mk` pour lier les bibliothèques et ainsi générer un fichier `.so` utilisable. Certains exemples de fichiers `Android.mk` peuvent ne pas fonctionner directement dans différents environnements. Leur rôle est d'indiquer au NDK quels fichiers sources doivent être compilés, à quelle bibliothèque ils doivent être liés, et d'autres informations similaires.

J'ai adopté une méthode de "compilation en deux étapes, puis édition de liens" :

1. Compilez d'abord pour obtenir une bibliothèque partagée nommée `myffmpeg`.
2. Dans un autre module `ffmpeg-jni`, liez ensuite `myffmpeg` pour finalement générer le fichier `.so` requis.

De plus, il est nécessaire de placer le fichier compilé `libffmpeg.so` dans le répertoire `jni` pour s'assurer qu'il puisse être trouvé lors de l'étape de liaison.

---

## Débogage de FFmpeg

Après avoir porté FFmpeg, il est souvent nécessaire de déboguer au niveau C pour appeler des fonctions via JNI. Si l'on pouvait voir les journaux détaillés comme dans la ligne de commande, il serait plus facile de localiser les problèmes.

Dans Eclipse, en maintenant la touche Ctrl enfoncée et en cliquant sur un appel tel que `av_log`, vous pouvez suivre l'implémentation de la fonction `av_log_default_callback` dans `ffmpeg/libavutil/log.c`. Cette fonction appelle `__android_log_print` d'Android pour imprimer dans Logcat. En examinant ces sorties, vous pouvez connaître l'état interne de FFmpeg, ce qui est utile pour diagnostiquer des problèmes tels que des échecs de synthèse ou le non-support de certains codecs.

Parfois, FFmpeg peut générer des exceptions qui provoquent un crash de l'application. Vous pouvez utiliser la commande suivante pour localiser le problème :

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

Si la fonction `main()` originale de FFmpeg se termine par un `exit(0)`, assurez-vous de le commenter, sinon cela entraînera la fermeture de l'application.

#### Fuites de mémoire et solutions avec les Services

Une fois la synthèse terminée, si l'erreur `"INVALID HEAP ADDRESS IN dlfree ffmpeg"` se produit lors d'un nouvel appel, cela est probablement dû à une libération incomplète de la mémoire par FFmpeg. Une solution de contournement consiste à placer le processus de synthèse dans un `Service` distinct, puis à tuer ce Service une fois la synthèse terminée afin de libérer les ressources.

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

En enregistrant un `Receiver` ou par d'autres moyens, vous pouvez terminer le Service vous-même après la fin de la synthèse, évitant ainsi les problèmes de mémoire lors d'appels répétés.

---

## Problèmes potentiels

- **Problème de lecture des fichiers AAC**
  Certains modèles de téléphones (comme le Xiaomi 2s) peuvent ne pas être en mesure de lire des fichiers audio encodés en AAC via le `MediaPlayer` par défaut.

- **Support insuffisant des codecs**
  Pour prendre en charge des formats comme AMR-NB ou MP3, il est nécessaire d'activer manuellement les options correspondantes lors de la compilation de FFmpeg. Si le script de compilation ne parvient pas à trouver les bibliothèques ou fichiers d'en-tête nécessaires, il s'arrêtera avec une erreur.

- **Vitesse de synthèse**
  La synthèse d'une vidéo de 10 secondes en 1280×720 avec un mixage audio peut prendre de quelques dizaines de secondes à une minute. D'un point de vue expérience utilisateur, il serait peut-être préférable de permettre à l'utilisateur de pré-écouter avant de décider de finaliser la synthèse.

Dans la mise en œuvre concrète de doublage, les pratiques courantes sont les suivantes : 1. Télécharger à l'avance la vidéo originale, les sous-titres, ainsi que le fichier audio "dont les segments nécessitant un doublage ont été supprimés". 2. Enregistrer la voix de l'utilisateur, lors de la synthèse, il suffit de fusionner l'enregistrement avec les segments silencieux. 3. Si le temps de synthèse local ne vous satisfait pas, vous pouvez choisir de télécharger les données audio et vidéo sur le serveur, la synthèse sera effectuée côté serveur, puis téléchargée une fois terminée.

---

## Utilisation de NDK dans Eclipse

Il n'est pas nécessaire de taper `ndk-build` dans la ligne de commande. Il suffit de faire un clic droit sur le projet dans Eclipse, de sélectionner **Android Tools → Add Native Support**, et ensuite, chaque fois que vous cliquerez sur "Run", `ndk-build` sera exécuté automatiquement.

### Génération automatique des fichiers d'en-tête JNI en un clic

La rédaction des fichiers d'en-tête pour les fonctions JNI peut être fastidieuse, mais elle peut être automatisée grâce à la commande `javah`.
Dans Eclipse, il est possible de configurer cela comme un outil externe, en utilisant une commande similaire à celle-ci pour générer les fichiers d'en-tête :

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

Après exécution, un fichier similaire à `com_example_ffmpeg_MyFFmpeg.h` sera généré dans le répertoire `jni`. Ensuite, il vous suffit d'inclure ce fichier avec `#include` dans votre code C et d'implémenter les fonctions correspondantes.

---

## Résumé

Le portage de FFmpeg sur Android implique une multitude de connaissances, notamment la configuration de l'environnement NDK, la compilation et le lien en C/C++, les appels JNI, ainsi que le codage et le décodage audio et vidéo. Si vous rencontrez des problèmes tels que l'impossibilité de synthétiser, le non-support de certains formats ou des erreurs de liaison, il est nécessaire d'examiner attentivement la configuration et les sorties de logs. J'espère que cet article vous aidera à éviter certains pièges. Si vous utilisez également FFmpeg, n'hésitez pas à partager vos expériences ou problèmes dans les commentaires, pour un échange et un apprentissage mutuels.