Portar FFmpeg a Android

Enlace al artículo original (CSDN)

Origen

Para aquellos que aún no están familiarizados con FFmpeg, pueden comenzar por revisar los Comandos básicos comunes de FFmpeg y estudiar detenidamente sus funciones, como la combinación de audio y video, la reproducción de varios tipos de codificación, la captura de video, la combinación de múltiples imágenes en un video y la mezcla de audio, la conversión de formatos, entre otros. FFmpeg puede ofrecer funcionalidades potentes e interesantes para una variedad de escenarios de aplicación.

Tomando como ejemplo aplicaciones del tipo “Dubbing Show”, el doblaje no solo es extremadamente divertido, sino que también suele hacer que el producto sea más atractivo. Planeamos desarrollar un módulo de doblaje y, por lo tanto, nos interesamos en FFmpeg, intentando portarlo a la plataforma Android. Nos tomó entre 3 y 4 días, probamos varias versiones, y muchos artículos en línea no lograron ayudarnos, hasta que encontramos un tutorial titulado “Uso de FFmpeg en Android y llamada a interfaces”, que finalmente nos permitió resolver el problema. Las pruebas prácticas mostraron que solo la combinación de FFmpeg 1.2 con ndk-r9 logró una portabilidad exitosa.

Ideas

FFmpeg es un proyecto escrito en lenguaje C que contiene una función `main()`. Nuestro objetivo es:

1. Compilar `libffmpeg.so` utilizando Android NDK.
2. Utilizar esta biblioteca para compilar y modificar el archivo `ffmpeg.c`, cambiando la función original `main()` a `video_merge(int argc, char **argv)`, lo que permitirá llamarla directamente desde JNI para realizar operaciones como la combinación de videos.

Por ejemplo, se puede lograr la composición de videos de manera similar a la siguiente (correspondiente al comando `ffmpeg -i src1 -i src2 -y output`):

```
video_merge(5, argv); // donde argv simula los parámetros de la línea de comandos
```

Entorno

- Sistema operativo: Ubuntu 12.04
- Versión de FFmpeg: 1.2
- Versión de NDK: ndk-r9

Antes de comenzar, te recomiendo consultar algunos tutoriales relacionados. Si encuentras algún problema, puedes volver a este artículo para comparar y evitar perder demasiado tiempo en el camino.

Modificación de la Interfaz y Android.mk

Al escribir una interfaz JNI para FFmpeg, es necesario crear un archivo `Android.mk` para enlazar las bibliotecas y generar un archivo `.so` utilizable. Algunos ejemplos de `Android.mk` pueden no funcionar directamente en diferentes entornos. Su propósito es indicar al NDK qué archivos fuente deben compilarse y a qué bibliotecas deben enlazarse, entre otra información.

Adopté un enfoque de “compilar dos veces y luego enlazar”:

1. Primero, compila para obtener una biblioteca compartida llamada `myffmpeg`.
2. Luego, en otro módulo llamado `ffmpeg-jni`, enlaza `myffmpeg` para finalmente generar el archivo `.so` requerido.

Además, es necesario colocar el archivo compilado `libffmpeg.so` en el directorio `jni` para asegurarse de que pueda ser encontrado durante el proceso de enlace.

Depuración de FFmpeg

Después de portar FFmpeg, para invocar funciones a través de JNI, a menudo es necesario depurar en la capa de C. Si pudieras ver registros detallados como en la línea de comandos, sería mucho más fácil identificar problemas.

En Eclipse, mantén presionada la tecla Ctrl y haz clic en una llamada como `av_log` para rastrear la implementación de la función `av_log_default_callback` en `ffmpeg/libavutil/log.c`. Esta función llama a `__android_log_print` de Android para imprimir en Logcat. Al revisar estas salidas, puedes obtener información sobre el estado interno de FFmpeg, lo cual es útil para diagnosticar problemas como fallos en la síntesis o la falta de soporte para ciertos códecs.

A veces, FFmpeg puede lanzar excepciones que hacen que la aplicación se bloquee. Puedes utilizar el siguiente comando para identificar el problema:

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

Nota: El comando anterior no necesita traducción, ya que es una instrucción técnica en el entorno de desarrollo de Android. Sin embargo, si necesitas una explicación en español, aquí está:

Este comando se utiliza para capturar los registros de `logcat` en un dispositivo Android y luego pasarlos a través de `ndk-stack` para convertir las direcciones de memoria en símbolos legibles, utilizando los archivos de símbolos ubicados en `obj/local/armeabi`. Esto es útil para depurar aplicaciones nativas en Android.

Si el `main()` original de FFmpeg tiene un `exit(0)` al final, asegúrate de comentarlo, de lo contrario, hará que la aplicación se cierre.

Fugas de Memoria y Soluciones con Service

Después de completar la síntesis, si aparece el error `"INVALID HEAP ADDRESS IN dlfree ffmpeg"` al llamar nuevamente, es probable que se deba a una liberación incompleta de la memoria de FFmpeg. Una solución intermedia es colocar el proceso de síntesis en un `Service` separado y, una vez finalizada la síntesis, eliminar dicho Service para liberar los recursos.

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

Al registrar un `Receiver` y finalizar el Service manualmente después de completar la síntesis, se pueden evitar problemas de memoria en llamadas repetidas.

Posibles problemas

- **Problemas de reproducción de archivos AAC**
  Algunos modelos de dispositivos (como el Xiaomi 2s) pueden no ser capaces de reproducir audio codificado en AAC a través del `MediaPlayer` predeterminado.

- **Soporte insuficiente de codificadores**
  Si se desea admitir formatos como AMR-NB, MP3, etc., es necesario habilitar manualmente las opciones correspondientes al compilar FFmpeg. Si el script de compilación no puede encontrar las bibliotecas o archivos de cabecera necesarios, se detendrá y mostrará un error.

- **Velocidad de síntesis**
  La síntesis de un video de 10 segundos en 1280×720 con mezcla de audio puede tardar desde varias decenas de segundos hasta un minuto. Desde la perspectiva de la experiencia del usuario, podría ser mejor permitir que el usuario escuche una vista previa antes de decidir si realizar la síntesis final.

En la implementación específica de una aplicación de doblaje, las prácticas comunes incluyen:

1. Descargar previamente el video original, los subtítulos y el archivo de audio con “los fragmentos que requieren doblaje eliminados”.
2. Grabar la voz del usuario, y durante la síntesis, simplemente combinar la grabación con los fragmentos de silencio.
3. Si no estás satisfecho con el tiempo de síntesis local, puedes optar por subir los datos de audio y video al servidor, donde se realizará la síntesis en el servidor, y luego descargar el resultado final.

Usar NDK en Eclipse

No es necesario ingresar `ndk-build` en la línea de comandos. Simplemente haz clic derecho en el proyecto en Eclipse, selecciona **Android Tools → Add Native Support**, y cada vez que hagas clic en “Run”, se ejecutará automáticamente `ndk-build`.

Generación automática de archivos de cabecera JNI con un solo clic

Escribir archivos de cabecera de funciones JNI puede ser tedioso, pero se puede automatizar utilizando el comando `javah`.
En Eclipse, puedes configurarlo como una herramienta externa y usar un comando similar al siguiente para generar el archivo de cabecera:

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

Nota: El comando no se traduce ya que es un comando específico de la línea de comandos y debe mantenerse en su forma original para su correcto funcionamiento.

Después de la ejecución, se generará un archivo similar a `com_example_ffmpeg_MyFFmpeg.h` en el directorio `jni`. Luego, solo necesitas incluir (`#include`) este archivo en tu código C e implementar las funciones correspondientes.

Resumen

La portabilidad de FFmpeg en Android involucra múltiples áreas de conocimiento, incluyendo la configuración del entorno NDK, la compilación y enlace de C/C++, la invocación de JNI, y la codificación y decodificación de audio y video. Si te encuentras con problemas como la imposibilidad de sintetizar, la falta de soporte para ciertos formatos o errores de enlace, es necesario revisar cuidadosamente la configuración y los registros de salida. Espero que este artículo te ayude a evitar algunos obstáculos. Si también estás utilizando FFmpeg, no dudes en compartir tus experiencias o problemas en la sección de comentarios, para que podamos intercambiar conocimientos y aprender juntos.