ChatGPT

跨平台影音處理與播放 SDK 設計報告

功能需求與範疇

- 本 SDK 需提供豐富的影音處理功能，包括影片與音訊的轉檔（如格式編碼轉換）、剪輯截取、壓縮、合併，以及音訊抽取與混音等。具體功能可參考 FFmpeg 能力，如影片格式轉換（MP4↔Mov/H.264）[1]、影片裁剪（使用 `-ss`/`-to` 參數截取片段）[2]、多影片拼接（concat）或圖片序列合成影片[3][2]；音訊方面可支援從影片抽取音訊[4]、混合多軌音訊、轉換音訊編碼（如 AAC/MP3）[5][4]。此外，需支援字幕處理（嵌入或提取 SRT/ASS 等字幕）、圖片處理功能，如圖片格式轉換（PNG↔JPG/WEBP）、縮放與壓縮，以及從影片截取靜態圖片（生成縮圖）[6][7]。這些功能幾乎都可透過內嵌的 FFmpeg 指令達成，並以 Dart API 封裝調用。

- **影片處理**：轉檔/編碼（調整解析度、畫質等）[1][8]；裁剪截取（使用 `-ss`、`-t` 等參數）[2]；合併或串接影片。

- **音訊處理**：從影片抽取音訊[4]；多段音訊混音、編碼轉換（如 AAC、MP3 等）[5][4]。
- **字幕處理**：支援將外部字幕檔合併入影片或從影片中提取軟字幕。
- **圖片處理**：支援圖片格式轉換（PNG→JPG、WebP 等）[9][7]；縮放壓縮圖片；從影片截取幀作為縮圖[6]。

播放支援與整合

- 除了處理功能外，SDK 可考慮納入播放 API，提供統一的播放介面。現有 Flutter 跨平台播放套件如 media_kit 可直接播放多種影片格式。media_kit 已具備「跨平台播放」能力，支援 Android、iOS、macOS、Windows、Linux 及 Web 平台[10]；而 video_player_media_kit 則將標準 `video_player` 套件改以 media_kit 為後端，也可在所有平台使用[11]。因此，SDK 在整合播放功能時，可採用 media_kit 或 video_player_media_kit 作為底層播放引擎，確保不同平台均能順暢播放常見格式[10][11]。播放 API 可提供如開啟影片、播放/暫停、跳轉、調整音量等方法，同時暴露播放進度和狀態事件。需要注意的是，不同平台原生播放器對格式的支援有差異（例如 iOS 使用 AVPlayer，Android 使用 ExoPlayer[12]），SDK 可在介面層進行封裝或格式判斷，以最大化兼容性。

各平台後端策略

- 為了實現跨平台功能，需依不同平台選擇適當的後端實現：
- **行動裝置 (Android/iOS/macOS)**：可使用 FFmpegKit Flutter 套件，該套件已封裝 FFmpeg 與 FFprobe，可直接在 App 中執行 FFmpeg 指令[13]。FFmpegKit Flutter 新版支援 Android、iOS 和 macOS 平台[13]，其中 iOS/macOS 還支持硬體加速（Videotoolbox）。在這些平台上，建議使用 ffmpeg_kit_flutter_new 處理影音任務。
- **桌面端 (Windows/Linux)**：目前 ffmpeg_kit_flutter 暫不支援 Windows/Linux 平台[14]。因此，可採用兩種策略：其一是在程式中捆綁 FFmpeg 二進制檔，透過 Dart 的 `Process` 啟動系統 FFmpeg 執行命令；其二是採用 FFI 方式，直接呼叫原生 FFmpeg 動態庫（或靜態庫），如使用 ffipeg 等自動生成的 FFI 綁定。既有經驗指出「在跨平台 Flutter 桌面應用中沒有簡單的方法捆綁 FFmpeg 二進制檔，ffmpeg_kit_flutter 尚未提供 Windows/Linux 版本」[14]，因此需要自行集成。
- **Web 平台 (瀏覽器)**：瀏覽器環境無法直接執行本地 FFmpeg，因此可考慮使用 WebAssembly 版的 ffmpeg.wasm 進行原地轉檔，但有顯著限制：如檔案大小上限約為 2GB[15]、執行效能較低、且若需使用執行緒（SharedArrayBuffer）須設定 COOP/COEP 以達到跨域隔離[16]。綜合考量，大型或長片的

轉檔應主要使用遠端伺服器後端處理，僅在小檔案或輕量任務下才使用 ffmpeg.wasm 作為可選方案。換言之，可在 Web 端實作遠端 API，將影片上傳至伺服器（或雲端函式）執行 FFmpeg，再返回結果，作為主要 fallback 策略。

SDK 架構與模組建議

- **Dart API 層**：在 Dart 層定義一組易用的介面方法，如 `transcode()`、`trim()`、`thumbnail()`、`probe()`、`concat()`、`mixAudio()` 等。每個操作可對應一個「Job Model」（任務模型）類別，封裝輸入/輸出檔案路徑、編碼參數（影片/音訊編碼器、位元率、解析度等）及其他設定。舉例，`TranscodeJob` 可包含 `inputPath`、`outputPath`、`videoCodec`、`audioCodec`、`resolution` 等欄位；API 方法接受此模型，並返回操作結果或狀態。
- **策略引擎 (Policy Engine)**：建立規則引擎自動決定使用的格式和參數。例如若要求「跨平台播放優先」，則強制使用兼容性高的編碼（MP4/H.264/AAC）；若目標平台支援新格式，可自動切換至 WebM/VP9 或 HEVC/H.265。此機制可依據平台、使用者偏好、或任務場景自動套用最佳格式規範。
- **進度與錯誤觀測**：影音處理通常是耗時任務，需提供進度回調和錯誤通知。可為每個任務返回一個 `Stream` 或註冊 callback，以實時上報目前處理進度（如轉換幀數、完成百分比）及階段性狀態。異常情況下，需拋出或回傳清晰的錯誤資訊，以便上層 UI 顯示或重試。建議統一定義錯誤碼（例如 `JobError` 類型），並包含底層 FFmpeg 返回的錯誤訊息以利調試。
- **平台路由層 (Backend Router)**：在 SDK 內部實作一個平台路由器，動態決定採用哪種後端（ffmpeg_kit、Process、WASM 或遠端服務）。例如在 Android/iOS 環境下自動呼叫 ffmpeg_kit，桌面下呼叫系統 FFmpeg，Web 下則轉為遠端 API 或 ffmpeg.wasm。透過這層抽象，上層 API 可保持一致，使用者無需關心底層平台差異。
- **多執行緒處理**：影音轉檔對 CPU 負載高，應放在 Dart isolate 或平台背景線程中執行，避免阻塞主線程與 UI。轉檔完成後再回傳結果或更新狀態給 Dart 端。

格式支援與策略

- **視訊格式**：預設可採用 MP4 容器搭配 H.264 影片編碼和 AAC 音訊編碼，此組合兼容性高，在 Web、Android、iOS 等平台均具備硬體或軟體支援 [8]。若使用 H.264/AAC 仍不足以覆蓋需求，可考慮 WebM（VP8/VP9，適用於支持 WebM 的瀏覽器與 Android）或 HEVC/H.265（可獲得更高壓縮率，但可能需支付授權費用）。在格式策略引擎中，可設定「跨平台兼容」條件時強制輸出 H.264/AAC；若平台專用或進階需求才切換至其他格式。
- **音訊格式**：常用 AAC（如 .m4a）或 MP3，均廣泛支持。對話或語音類可考慮 OGG/Opus；若需無損品質則 FLAC。在無須兼容舊設備情況下，可彈性選擇。
- **圖片格式**：支援 JPEG、PNG 及 WebP。JPEG/PNG 幾乎所有平台均支持；WebP 則在現代瀏覽器和 Android 上支援度高，有更佳壓縮效果。策略引擎可根據目標平台或用戶設定選擇輸出格式。
- **格式決策流程**：SDK 應內建格式決策邏輯，如輸出裝置或播放器僅支持特定編碼，則強制轉檔為相容格式；若不需要播放，只需節省空間，則選擇壓縮率更高的格式。Policy 引擎可根據平台能力、網路條件（如可用帶寬）和使用者偏好自動選擇最佳格式。

API 設計建議

- **轉檔 (`transcode`)**：方法參數可包含 `inputPath`、`outputPath`、`videoCodec`、`audioCodec`、`resolution`、`bitrate` 等；也可允許傳遞 FFmpeg 原生命令參數以擴充彈性。例：`transcode(job)` 返回一個操作結果對象。
- **縮圖 (`thumbnail`)**：從影片指定時間點截取幀圖。參數例：`videoPath`、`timePosition`、`outputImagePath`。底層可執行 `-ss time -i video -vframes 1`。
- **媒體信息探測 (`probe`)**：讀取影片/音訊檔案的元資料（分辨率、編解碼器、時長、封裝格式等），返回結構化資訊（可透過 FFprobe 實現）。

- **合併串接 (`concat`)**：接受多個影片或圖片序列路徑，生成單一輸出影片。可先構造 FFmpeg concat 文件列表或管道串流。
- **音訊混合 (`mixAudio`)**：將多個音訊檔混合為單一音訊檔，參數為輸入音訊陣列與輸出路徑。
- **視頻剪輯 (`trim`)**：從影片中截取指定時間區間，參數包含 `startTime`、`endTime`。
- **添加字幕 (`addSubtitle`)**：將字幕文件硬編碼入影片，或將字幕軟嵌在影片容器中；參數包括字幕檔路徑及輸出檔路徑。
- 每個方法應盡量簡潔易用，並將底層的 FFmpeg 工作（參數組裝、日誌解析）對使用者隱藏。

Web 端實現策略

- **WebAssembly 限制**：ffmpeg.wasm 在瀏覽器中執行時，受限於可用記憶體與單線程。當前版本檔案處理上限約 2GB [15]，若使用執行緒加速又須啟用跨域隔離 (COOP/COEP) [16]。因此，瀏覽器端適合處理小檔案或短片。對於大檔案或高耗時轉檔，應使用後端服務執行，並回傳結果給前端。
- **COOP/COEP 設定**：若決定使用 ffmpeg.wasm，應確保網頁或服務設置 `Cross-Origin-Opener-Policy: same-origin` 與 `Cross-Origin-Embedder-Policy: require-corp` 等標頭，以啟用 SharedArrayBuffer 和 WebAssembly 執行緒 [16]。需要自行部署 ffmpeg.wasm 所需的 JS 和 WASM 檔於同源，避免跨域資源阻擋。
- **後端替代方案**：可設計備援機制，先檢測瀏覽器是否支援 WebAssembly 和必要的 Web API；若不支援或文件過大，則自動將檔案上傳到後端伺服器（或雲端），在伺服器端執行 FFmpeg，最後將處理後的檔案或縮圖傳回客戶端。這樣可兼顧效能與可用性。
- **部署注意**：若使用靜態網站（如 GitHub Pages）部署 ffmpeg.wasm，需注意放置 `ffmpeg-core.wasm`、`ffmpeg-core.worker.js` 等檔案，並正確設定 MIME 類型與 CORS 標頭。最佳做法是透過 Node.js 或專用伺服器提供相關檔案，以確保 COOP/COEP 的正確應用 [16]。

技術風險與避坑

- **iOS 靜態連結限制**：Apple 要求包含 FFmpeg 等原生函式庫的 App 使用靜態連結，導致應用體積大幅增加。FFmpegKit 的完整版包含大量編碼器庫（如 x264、x265） [13] [17]。建議按需選用 minimal 或 audio/video 版套件，或在後端先處理 GPL 編碼需求，避免在 App 中打包過多資源。
- **Web 性能瓶頸**：ffmpeg.wasm 無法使用 GPU 加速，全部運算在 CPU 上，對於大型影片處理速度遠不及原生環境。加上瀏覽器記憶體限制及執行緒開銷，使其不適合長時間轉檔。對策是避免在瀏覽器中使用 ffmpeg.wasm 做大型處理，必要時轉交後端；在前端則限制處理範圍（如壓縮小檔案、截取幀圖等）。
- **播放行為差異**：不同平台的原生播放器對格式和編碼的支援不一致 [12]。例如 iOS 的 AVPlayer 不原生支援 WebM/VP9、Android 的 ExoPlayer 在不同裝置支援度也有差異。設計時需優先使用兼容性高的格式，或在 App 層偵測平台後選擇適合的檔案；亦可在 UI 層提供錯誤處理流程（例如提示用戶轉檔格式）以避免播放失敗。
- **授權與法規**：使用 H.264、H.265 等編碼器可能涉及授權費用（HEVC 甚至需要額外授權），若不希望承擔授權成本，可使用開放格式（如 VP9、AV1）。此外，FFmpegKit 集成了 GPL 編碼器（x264、x265 等） [13] [17]，需確保滿足相關開源授權要求。
- **錯誤處理困難**：FFmpeg 執行時可能不斷輸出大量日誌，且中途錯誤時往往只顯示「conversion failed」等訊息。SDK 應儘量捕獲並解析錯誤日誌，並轉換為友善的錯誤訊息返回；同時支援中斷取消功能（透過 FFmpegKit 提供的 cancel API 或自行管理 Process）以避免進程無法結束。
- **檔案存取權限**：各平台對檔案系統存取方式不同（如 Android 要使用 SAF URI、iOS 可能需調整資源存取權限）。SDK 應處理好路徑轉換與權限申請的細節。FFmpegKit 已具備對 Android Storage Access Framework (SAF) URI 的支援 [13]；對桌面亦需提醒使用者安裝或將檔案置於允許讀寫的位置。

綜上所述，本 SDK 可透過封裝 FFmpeg 與播放引擎，利用策略引擎與路由機制實現跨 Android、iOS、Windows、macOS、Linux、Web 六大平台的影音處理與播放。關鍵在於選擇合適的後端（如行動裝置採用 FFmpegKit、桌面採用系統 FFmpeg、Web 採用遠端服務），並在 Dart 層定義統一的任務模型與 API。透過周

密的格式策略和健全的錯誤/進度機制，確保 SDK 能在不同環境下穩定運行，並彈性因應各平台限制和特性。各項功能設計均可參考上述 FFmpeg 常見用例與跨平台播放套件 [1] [10] [8] 所示範的實踐模式。

參考資料： Flutter FFmpegKit 相關文件 [13]、MediaKit/Video_Player 多平台播放 [10] [11]、ffmpeg.wasm 限制討論 [15] [16]、通用影音格式建議 [8]、Flutter video_player 支援格式 [12] 等。

---

[1] [2] [3] [4] [5] [6] [7] [9] The Ultimate Guide to FFmpeg in Flutter | by Rugved Apraj | Oct, 2025 | Medium
https://rugvedapraj.medium.com/the-ultimate-guide-to-ffmpeg-in-flutter-2d9c01478b5d

[8] An End To Confusion: The Ultimate Guide To Video Format Types - Designrr
https://designrr.io/video-formats-and-codecs/

[10] dart - Does Flutter have video players for all platforms (iOS, Android, Windows, Linux, MacOS)? - Stack Overflow
https://stackoverflow.com/questions/76422412/does-flutter-have-video-players-for-all-platforms-ios-android-windows-linux

[11] video_player_media_kit | Flutter package
https://pub.dev/packages/video_player_media_kit

[12] video_player | Flutter package
https://pub.dev/packages/video_player

[13] [17] ffmpeg_kit_flutter_new | Flutter package
https://pub.dev/packages/ffmpeg_kit_flutter_new

[14] ffipeg | Dart package
https://pub.dev/packages/ffipeg

[15] Larger video file inputs · Issue #623 · ffmpegwasm/ffmpeg.wasm · GitHub
https://github.com/ffmpegwasm/ffmpeg.wasm/issues/623

[16] javascript - FFmpeg.wasm stopped working after adding cross origin headers - Stack Overflow
https://stackoverflow.com/questions/69166217/ffmpeg-wasm-stopped-working-after-adding-cross-origin-headers