ChatGPT

Dart Web3生態系統功能缺口調查與開發優先級

基於目前 Dart Web3 生態的現況，我們發現核心功能主要由早期的 **web3dart** 套件提供，但該套件已於 2022 年停止維護 [1]。近期出現了一些替代方案（如受 ethers.js 啟發的新 **ethers.dart**，以及社群 fork 的 **webthree** 等）來填補空白。然而，許多領域仍存在功能缺口或成熟度不足的問題。以下我們依據不同模組逐一分析現有解決方案、缺陷與坑點，並說明其嚴重性。同時在報告末段，根據實用性和缺口嚴重性提出**優先實作順序**建議。

1. 基礎密碼學（secp256k1、SHA3/Keccak256、BIP-32/39/44 等）

現有方案： Dart 生態中已經有相當多的加密基礎模組，可供參考或直接使用。其中包括：

- **橢圓曲線與簽章：** `sec` 套件實現了 SEC (Standards for Efficient Cryptography) 演算法族，包含 secp256k1 曲線演算以及公鑰恢復等功能 [2]。此外，亦有 `secp256k1` 原生移植（如 blockchain_utils 套件內含的 Bitcoin Core secp256k1 移植 [3]），可用於以太坊簽章計算。
- **雜湊函數：** Dart 官方 `crypto` 套件涵蓋 SHA-2 等雜湊，但不含 Keccak。社群提供了 `sha3` / `keccak` 套件，可生成 Keccak-256 (以太坊常用的 SHA3 變體) 雜湊 [4]。這類套件通常由 JavaScript 實現翻譯而來，功能相對完整。
- **HD 錢包與助記詞：** 有多個 BIP 標準實現，例如 `bip39` 套件（從 bitcoinjs/bip39 轉譯）提供 BIP-39 助記詞到種子轉換 [5]；`bip39_plus` 對其進行維護改進，支援更新的 Dart 版本 [6]。針對 HD 金鑰推導 (BIP-32/44)，社群也有套件（如 blockchain_utils、或專門的 `wallet` 套件）支援。**Wallet** 套件就是一個涵蓋 Bitcoin/ETH/Tron HD 錢包功能的純 Dart 實現 [7]；它內部使用 `pointycastle` 提供低階加解密，並結合 `sec` 套件來處理 secp256k1 橢圓曲線演算法 [8]。

成熟度與坑點： 整體而言，基礎密碼學在 Dart 上有相當基礎，可避免從零實作。例如 `sec` 套件專注於安全實作 secp256k1，考慮了側信道攻擊風險 [2]。`pointycastle` 作為 BouncyCastle 的 Dart 移植，提供對稱加密、雜湊等全套算法，雖然完整但體積大、性能一般（相對 C/C++ 實現稍慢）。BIP-39/32 等助記詞與 HD 推導套件基本可用，但需要注意**隨機數生成**要使用安全隨機源（例如 `Random.secure()`）。由於以太坊私鑰長度較短（256-bit），使用 Dart BigInt 等類別處理需留意溢位和符號（通常問題不大）。總體來說，此領域現有實作較完善，**缺口嚴重性低**：開發者多可直接使用現有套件。然而，為了減少依賴，一個整合並優化的自有實現可能仍有價值（例如移除不必要的演算法，專注以太坊需要的部分以提升性能）。

優先性考量： 基礎加密是 SDK 的底層，雖然現有方案成熟度尚可，但我們仍需確保有可靠的 secp256k1 簽章、Keccak 雜湊和助記詞支援。由於可先行使用現成套件滿足需求，短期內不是最迫切的開發項目，但長遠看可考慮自行優化或整合，確保 SDK 不受制於外部套件更新。

2. ABI 編碼/解碼（含動態類型與 EIP-712 TypedData）

現有方案： ABI 編解碼是智能合約交互的關鍵。目前 Dart 的 Ethereum 庫已具備一定的 ABI 編碼功能：

- **web3dart/webthree 編碼器：** web3dart 內建 ABI 處理，可根據函數簽名封裝資料、解析回傳值。根據 web3dart 的 fork 版本 webthree 文件，其 ABI 支援絕大多數 Solidity 類型，僅有固定小數 (fixed 和 ufixed) 尚未實作，且這類型並不常用 [9]。這表示基本的數值、地址、字串、bytes、array、struct 等型別皆已覆蓋。

- **TypedData/EIP-712：** 早期 web3dart 並無內建 EIP-712 簽名資料的編碼支持，需要自行拼裝哈希或使用外部工具。現在社群有了獨立的 `eth_sig_util` 和 `eip712` 套件可協助。`eth_sig_util` 為 JS **eth-sig-util** 的 Dart 移植，提供對個人簽名 (personal_sign) 以及 EIP-712 Typed Data 簽名的支援 [10]。而 `eip712` 套件更進一步，完整實現了 EIP-712 結構化資料哈希與簽名的流程，支援 EIP-712 v3 和 v4，涵蓋各種以太坊類型（address、bool、bytesN、string、uint/int、array、自訂 struct 等）並提供型別安全檢查 [11]。它以嚴格類型實現，可方便地將結構化資料轉換為待簽名的 digest [12] [13]。

缺口與問題： **ABI 編碼/解碼**方面，傳統函數和日誌解析 web3dart 已能處理，但**動態巢狀結構**最初支援有限（如 struct 包含陣列等）。社群 fork（webthree 等）已修正了一些問題，例如支援結構體作為函式輸入、重載函式名稱區分等 [14] [15]。然而，目前 Dart 生態中缺少類似 ethers.js 那樣**統一且經過充分測試**的 ABI 編碼器。開發者常需要自己注意資料型別對齊、padding 等細節。一些額外的 ABI 功能（如 `abi.encodePacked` 用於計算哈希、或解析 event 日誌時處理 indexed topic）在現有套件中較為原始，需要手動處理。

對 **EIP-712** 而言，以前是明顯缺口：沒有官方支援，需要自己組裝 Domain Separator 和資料哈希。`eip712` 套件的出現大大彌補了這個空白，提供了高階介面構造 TypedData 並計算簽名 [16] [17]。然而這也帶來依賴性問題——該套件目前依賴 web3dart 和 wallet 套件 [18]。若我們計劃獨立實作 SDK，可能希望內建這部分功能，避免引入不必要的相依。

成熟度： 整體而言，ABI 編碼功能**部分成熟**但仍有改進空間。常見類型基本沒問題，但罕見類型或新 EIP型別（例如日後 Solidity 新增的型別）需持續追蹤實作。型別安全與錯誤處理在現有方案中也相對基礎，一些錯誤可能只有執行期才暴露。

優先性考量： ABI 編碼/解碼直接影響合約調用和事件解析，是 SDK **核心功能之一**，應優先確保完善。尤其 EIP-712 現今被廣泛使用於訊息簽名（如 permit、meta-tx），這塊過去缺口嚴重，如今有方案但我們最好將其整合進 SDK。建議將 **ABI編解碼** 作為高優先級項目：可以參考 web3dart/webthree 的現有實作，加上完善的單元測試來驗證對各種型別的支援，同時整合 TypedData 簽名功能。

3. 簽名器種類與交易簽名邏輯（Legacy & EIP-1559 交易、訊息簽名、TypedData 簽名）

現有方案： 以太坊交易與訊息簽名在 Dart 中主要通過**Credentials**或**Wallet**介面提供。web3dart 定義了 `Credentials` 抽象類別代表簽名憑證，包括私鑰及可能的簽署行為。常見實現有：

- **私鑰簽名:** `EthPrivateKey`（web3dart）可從Hex私鑰建立，提供 `signTransaction` 等方法來簽署交易。web3dart 2.x 時期只支援 Legacy (預 EIP-1559) 交易的 R/S/V 簽名；在 EIP-155 實施後有增添 chainId 保護機制。後續社群 fork 已補充 **EIP-1559** 格式交易：如計算 `maxFeePerGas` / `maxPriorityFeePerGas` 等欄位，並在交易簽名前自動填入鏈ID [19]（對應 EIP-1559 的結構）。
- **HD Wallet / 助記詞:** `wallet` 套件與 web3dart 都支援從 BIP-39 助記詞推導 ETH 私鑰。web3dart 可以讀取 v3 錢包JSON並解密出私鑰 [20]。這些都可進一步轉為 Credentials 以簽名。
- **外部簽名（Remote Signer）:** 現有 Dart 庫並沒有直接提供 JSON-RPC 提供的 `eth_sign` 或 `eth_sendTransaction` (unlock account) 方式的**遠端簽名**，因為在前端應用中通常不會將私鑰託管於節點。不過，web3dart 的 `Credentials` 抽象讓我們可以自定義，例如為 **MetaMask** 或 **WalletConnect** 編寫 Credentials 實現。事實上，web3dart 的社群擴充已考慮 MetaMask：webthree 提供了一些 MetaMask 錢包事件（如 accountsChanged、chainChanged）的響應，以及**將 MetaMask 視為一種 Credentials** 的思路 [21] [22]。透過這種設計，我們可以把**外部錢包**封裝成 Credentials，使其簽名請求轉發給外部（例如 MetaMask 瀏覽器插件或手機錢包）。目前具體的 MetaMask/WC 實現需開發者自行處理，但架構上是可行的。

- **訊息/TypedData 簽名:** 如上節所述，`eth_sig_util` 提供了簽署 Ethereum 個人訊息（`\x19Ethereum Signed Message:\n` 前綴）以及 EIP-712 結構化資料的工具 [23]。這表示即使核心 web3dart 沒有封裝這些，高階上仍能完成。但是這屬於額外工具，而非統一的 SDK 介面。

坑點與不足： 首先是**EIP-1559交易**支持問題：老版本 web3dart 發送 EIP-1559 交易時需要手動指定參數，且早期版本可能缺少對 `maxFeePerGas` 等欄位的處理。如今 webthree 等已經加入對 EIP-1559 的支援 [14]，但我們要確保新 SDK 完整涵蓋 Legacy、EIP-155 與 EIP-1559 三種簽名方案，包括正確計算 chainId、防止重放。另外，以太坊交易類型還有 EIP-2930（Access List交易），這也需要考慮。現有 Dart 庫對 EIP-2930 的支持情況不明顯，可能需要另外實作。

簽名器種類方面，目前 Dart 生態缺少類似 ethers.js **Signer** 抽象的完善架構。web3dart 的 Credentials 接口實際上扮演類Signer的角色，但類型較單一（大多是私鑰或 wallet file）。在更豐富的 SDK 中，我們可能需要支持：軟體錢包 (PrivateKey/MnemonicSigner)、硬體錢包 (需要透過 HID/藍牙介面，這在Flutter中可能需要plugin)、遠端簽名服務、WalletConnect簽名等等。這些**目前並無現成 Dart 實現**，需我們自行設計。幸運的是，有 ethers.js / viem 作為參考模型：可以將**Account**（帳戶資料和簽名方法）與**Provider**（節點連接）分離，類似 viem 的 Public Client / Wallet Client 架構 [24]。目前 Dart 的新 ethers.dart 已初步採用這種思想：提供 `ethers.providers.JsonRpcProvider` 以及 `Wallet` 類別等 [25] [26]。但據其說明版本僅 0.0.1+3，功能還很有限。

成熟度： **交易與訊息簽名**算是較成熟的部分——基本私鑰簽名都能正常運作，EIP-1559 等新標準也有方案跟進。但**Signer 抽象層**仍不成熟，開發者需要手動管理不同簽名來源。另外，錯誤處理（例如不正確的簽章長度、隨機數重複等）在現有實現中鮮少提及，需要特別注意安全。

優先性考量： **交易簽名能力**是整個 SDK 的關鍵，必須優先確保可靠。建議在早期階段就建立良好的 Signer 架構：例如定義 `Signer` / `Credentials` 介面，包含 `signTransaction`、`signMessage`、`signTypedData` 等方法，並實作常見子類（私鑰錢包、助記詞錢包）。同時，務必完善對 EIP-1559 等所有交易類型的處理。相對而言，**擴充多樣Signer類型**（如 WalletConnectSigner 等）可以隨著需求漸進實作。在優先順序上，**本地私鑰簽名**肯定是第一位的，其次是提供介面允許未來插入外部簽名方案（保障架構開放性）。整體而言，簽名相關功能屬於**高優先級**，因為它直接關係到發送交易和驗證消息這些核心用途。

4. RPC Provider 實現（HTTP/WebSocket、批次請求、Middleware 機制）

現有方案： RPC Provider 用於與區塊鏈節點通訊。Dart 生態中，web3dart 將此功能集中在 `Web3Client` 類別中。使用者可以提供一個 HTTP 客戶端（如 `package:http` 的 `Client`）或 WebSocket 通道，建立 Web3Client 實例來發送 JSON-RPC 請求 [27] [28]。Web3Client 支持通過 HTTP 或 WebSocket 與節點互動，也封裝了常見的 RPC 方法（如 `getBalance`、`sendTransaction` 等）。新出現的 webthree 與 ethers.dart 在這方面大同小異：webthree 沿用 Web3Client，但加入了一些**訂閱與篩選**優化（如允許自訂日誌輪詢間隔等 [29]）；ethers.dart 則仿效 ethers.js，引入 `JsonRpcProvider` 類別供開發者使用 [25]。

批次請求 (Batching)： 目前 web3dart 似乎**未內建** JSON-RPC 批次請求功能。也就是說，無法在一個 HTTP 請求中同時發多個方法調用。批次對某些情境很有用（例如同時請求多個區塊資訊以減少往返延遲）。此功能在 Dart SDK 中屬明顯缺漏，開發者只能自行並發多請求，無法利用 JSON-RPC batch 的原生優勢。

中介軟體 (Middleware)： ethers.js 有概念類似 Middleware 的機制嗎？嚴格說 ethers.js Provider 沒有插件管線，但 viem 採用 functional composition，可以在 Client 外包一層行為。Dart 現有實現中並沒有正式的 middleware 機制。例如我們無法方便地在每次 RPC 調用前後插入攔截器去記錄日誌、修改參數或執行快取。開發者只能透過包裝 Web3Client 來達成這些效果。**缺乏 middleware 機制**意味著靈活性不足。

坑點與改進： Web3Client 基本功能可用，但有一些限制和潛在問題： - **錯誤處理：** web3dart 對 RPC 錯誤的處理較簡單，很多時候直接拋出 Exception，錯誤細節（如錯誤碼、資料）可能需要自行解析。webthree 近期版本改善了錯誤訊息傳遞，將 error code/message/data 包進 Exception [30] 。 - **WebSocket 重連：** 若使用 WebSocket 訂閱事件，斷線重連的邏輯需要由應用層處理。SDK 層目前沒有自動重連或心跳機制。 - **並發請求：** Web3Client 本身是單線程順序處理還是能併發？大部分 Dart I/O 是非同步的，Web3Client 應可同時發多個 RPC。但要小心的是，以太坊節點對同一連線的併發請求有無限制。沒有批次的情況下，快速大量請求可能導致瓶頸。 - **自訂 RPC:** 如果要支援像 Infura 那種帶專案ID的 RPC、或需附加驗證 header，目前只能在建立 HTTP client 時注入對應配置，沒有高階API。

成熟度： RPC Client 的基礎還算成熟，因為 JSON-RPC 本身很穩定，web3dart 也歷經多次迭代。但與現代需求相比仍顯不足，特別是缺少**進階功能（批次、快取、攔截）**。相比 ethers.js 等的 Provider 抽象，Dart 目前的 Provider 更像是硬編碼的封裝，不夠模組化。

優先性考量： 強健的 RPC Provider 是整個 SDK 的**命脈**。我們應該將**開發新 Provider**作為最高優先級之一。建議的改進方向有： - 實現**批次請求**功能，以便批量獲取資料時效率更高。 - 設計**Middleware機制**，允許開發者注入攔截器（例如請求簽名、日誌、錯誤重試等）。 - 確保 HTTP 與 WebSocket 皆有良好支援，並加強 WebSocket 的斷線重連與錯誤處理。 - 提供**自訂 Transport**接口，使得未來可以對接其他協議（如 IPC，雖然移植到行動端可能較少用IPC）。

由於目前雖有 Web3Client 可暫用，但我們追求的是媲美 ethers.js 的彈性與功能，所以 **RPC Provider 重構**屬於**最優先級**的項目之一。沒有可靠的 Provider，其它高層功能都無從建立。

5. Public / Wallet Client 架構（對齊 viem/ethers 的 read/write 模型）

概念解釋： 現代以太坊開發庫（如 ethers.js v5、viem）強調將「讀操作」與「寫操作」的對象分離。**Public Client**（公共客戶端）提供純區塊鏈讀取功能，不涉及私鑰，用於查詢鏈上資訊。**Wallet Client**（錢包客戶端）則綁定了一個帳戶，可以發送交易、簽署訊息等。這種模型提高了清晰度和安全性：開發者可以明確知道某對象是否有簽署交易權限 [24] 。

現有方案： Dart 舊有的 web3dart 並未嚴格區分這兩種角色。Web3Client 既可查詢又可發交易（透過傳入 Credentials）。也就是說，每次要發交易時都需要傳入私鑰憑證；若不傳入則代表只讀操作。這種介面設計相對靈活但也容易混淆。新興的 ethers.dart 已開始引入類似 ethers.js 的模式，例如 `JsonRpcProvider` 用於讀操作，而 `Wallet` 類結合 Provider 用於寫操作 [31] [32] 。不過據我們了解，ethers.dart 仍處於早期階段，很多細節（如多帳戶管理、Provider 和 Signer 的互動）可能尚未完善。

缺口與必要性： **Public/Wallet Client 分離**本質上是一種架構設計，而非額外功能，但它影響 SDK 易用性和可擴充性。Dart 生態目前除了上述初步嘗試外，沒有成熟的對齊實現。這造成兩個問題： 1. **缺乏範式：** 開發者不容易基於現有庫構建類似 ethers 的使用習慣，比如不能輕易地從一個只讀客戶端衍生出帶特定私鑰的簽名客戶端。每次都要手動提供私鑰或維護全域的 client+credentials 組合。 2. **安全性與測試：** 在沒有嚴格分離時，可能不小心就使用了帶私鑰的對象去做只讀操作，或反之。分離後，可以避免將私鑰暴露給不需要的組件，也方便在測試時替換掉 Signer 部分。

參考實踐： **viem** 庫中，PublicClient 和 WalletClient 是透過組合實現：WalletClient = PublicClient + Account。它還允許 Account 為僅公鑰（無私鑰）的情況，以處理硬體錢包等 [33] 。我們可參考這種設計，在 Dart 中定義 PublicClient 接口（只包含chain讀操作），以及 WalletClient 繼承或持有 PublicClient 並加入交易簽名方法。

成熟度： 由於這屬設計層面，目前 Dart 並無可直接用的成熟代碼。但其本身不涉及複雜演算法，關鍵在於定義合理的 API。ethers.dart 的出現表明社群已有此意識，「Thanks to web3dart which is no longer maintained」[1] 暗示新的架構會汲取 ethers.js 經驗。

優先性考量： **架構調整**通常不被認為是「功能缺口」，但對我們要打造的 SDK 至關重要。我們建議將 **Public/Wallet Client 模型** 在早期就確立下來。這不僅影響代碼組織，也關係到使用者上手難易度。如果目標是替代 ethers.js，那麼提供類似的分離模式會讓以往使用 ethers 的開發者感到親切。同時，這有助於後續擴充（例如一個 WalletClient 可以很容易地換成不同帳戶或不同Provider）。綜上，**在優先順序上**，架構設計應與 RPC/簽名基礎同步考量。在完成核心通信與簽名模組時，即可著手引入 PublicClient/WalletClient 的接口划分。總的來說，我們認為這是**中高優先**事項：需要及早決定，但實作難度相對較低（更多是API設計），可並行進行。

6. WalletConnect v2 客戶端（Flutter 平台與手機錢包連線）

現有方案： WalletConnect 是鏈接去中心化App與手機錢包的常用協議。早期 Dart 有一個 `walletconnect_dart` 套件，對應 WalletConnect v1 協議，但最後版本停留在0.0.11（約3年前）且未明確支援v2[34]。該套件基本功能可以用，包括創建會話、生成QR Code URI 等，可以連接錢包簽名交易[35][36]。但 WalletConnect v1 本身已漸被棄用。**WalletConnect v2** 引入了多鏈協議、Relay服務網絡和新版 handshake 機制，與v1不相容。

目前 Dart 生態對 WalletConnect v2 的支援主要來自 **WalletConnect 官方的 Web3Modal Flutter**： - **web3modal_flutter** 套件是 WalletConnect 團隊推出的 Flutter 組件，提供了 v2 協議的整合，以及配套的錢包連接UI[37]。開發者只需初始化專案ID，即可調用其組件掃描連線錢包[38][39]。這套工具在UI上較方便。 - 另有一些社群方案，例如 **flutter_web3** 等，據稱內部也引入了 WalletConnect provider。但具體落實較少文件。

缺口與限制： 雖然有 web3modal_flutter，但它傾向**高階封裝**且與 UI 綁定。如果我們想在 Dart SDK 層面實現 WalletConnect v2 支援（例如作為一種 Credentials/Provider，讓應用開發者自行處理UI），目前沒有現成的輕量套件。`walletconnect_dart` 舊套件不支援v2，而 web3modal_flutter 又太高階且偏重 Flutter UI（不利於在純 Dart 環境或後端使用）。因此，**缺口在於**：沒有一個純 Dart 的 WalletConnect v2 **核心客戶端**庫。我們可能需要參考 WalletConnect 官方的 TypeScript Core SDK，實現 Dart 版本的 Client，處理底層 relay 通訊、topic加密、協議訊息格式等等。

現有的坑點： WalletConnect v2 較 v1 **協議更複雜**，涉及到項目ID、Relay服務、命名空間（指定支援哪些鏈和方法）等。如果從零開始實作，工作量和複雜度都不低。另外，如果沒有官方支援，將來協議升級時需要跟進。值得注意的是，一些第三方已經在自行實踐：例如 **Fuse** 團隊的 `wallet_core` 庫[40] 據稱包含錢包功能，不確定是否包括WalletConnect。

成熟度： 目前**WalletConnect v2 的 Dart 實作成熟度很低**。web3modal_flutter算是唯一完整方案，但把它納入SDK不切實際（太重且不符合「純 Dart」原則）。因此這方面我們幾乎得從頭開始或fork官方實作。

優先性考量： 如果我們的 SDK 目標涵蓋 Flutter DApp 開發場景，那 WalletConnect v2 支援是非常重要的——這直接決定了應用能否方便地讓用戶用行動錢包簽名交易。就**缺口嚴重性**而言，這塊明顯**非常缺**（沒有輕量方案）。然而，就**實用性**而言，其優先順序取決於我們SDK的主要使用場景： - 若我們專注提供類似ethers.js的核心功能，而應用層面如何連錢包留給開發者，或暫時接受v1方案，那 WalletConnect v2 可以稍後實作。 - 但從完整性角度，支持WalletConnect v2將使我們的SDK在Flutter生態更有吸引力。

建議的路徑是：**中期優先**。也就是說，先把鏈上交互的核心功能完善，再著手WCv2。開發時可以先調研官方 WalletConnect-Kotlin/Swift SDK 或其訊息格式規範，設計出 Dart 版本。或考慮與 web3modal_flutter 協作，提取其核心邏輯。總之，WalletConnect v2 客戶端屬於**明顯功能缺口**，優先度中等偏高：不會是最先實作（需先有交易/合約等支援），但應該在核心完成後盡早著手，以補全我們SDK在錢包連接方面的能力。

7. 鏈網路/Chain Config 管理（chainId、RPC URL、區塊瀏覽器等配置）

現有方案： 在ethers.js中，會內建一些常用網路的設定（如 chainId 對應的網路名稱、ETH符號、區塊瀏覽器URL模板等），方便開發者切換。Dart 生態目前沒有看到明確獨立的“Chain config”套件，但零散的出現在一些工具中： - **ethereum_lists/chains 資料**：社群有一個 Ethereum-Lists/chains JSON 資料庫，列出了各條 EVM 鏈的參數（chainId、networkId、rpc URLs、瀏覽器、符號等）。這些資料沒有直接的 Dart 封裝，但我們可以手動引入作靜態資料使用。 - **web3dart/webthree** 並未提供網路常量，通常 chainId 需要開發者手動設定。比如 Web3Client.sendTransaction 時常需要傳入 chainId 或設 fetchChainIdFromNetworkId=true 去RPC抓取。 - **其他專案**：如 Truffle/Hardhat 等的鏈配置（但那是JS領域）。Dart中鮮少針對多鏈支援做文章，大多數應用只使用主網或自行輸入測試網RPC。

缺口： 缺少一個集中管理鏈資訊的模組。這對**多鏈應用**來說很實用：例如 walletconnect 需要宣告支援的鏈列表、DApp需要顯示當前連接鏈的名稱和瀏覽器。沒有統一模組的話，開發者可能各自寫一份對照表，容易出錯且重複勞動。

建議解決： 可以考慮建立一個 `ChainConfig` 模組，內含常見網路（Ethereum Mainnet、Goerli、Sepolia、Polygon、BSC、Arbitrum等）的資訊。這些資料可以從社群維護的清單獲取。例如 chainlist.org 提供了大部分EVM鏈的參數。我們可將有用的信息摘錄到SDK，如： - chainId -> 網路名稱，符號（ETH、MATIC等） - RPC URL 範例或建議（如 Infura/Alchemy 節點地址模板） - 區塊瀏覽器網址格式（用於生成交易或地址連結）

成熟度： 因為本質只是靜態資料容器，此模組實現起來難度不高，維護重點在於資料更新。當有新測試網、新L2出現時，需要更新清單。因此可以考慮提供**擴充介面**，允許使用者自行新增自定義鏈。

優先性考量： 從缺口嚴重性看，這雖然不影響核心功能，但屬於**開發體驗**的大提升——尤其對替代 ethers.js 而言，用戶期待開箱即用常見網路配置。因此我們建議將 Chain Config 作為**中等優先**項目實作。它不如交易簽名等緊迫，但相對容易完成，可以在核心功能告一段落後很快加上。具體順序上，可在第一版SDK完成基本鏈上交互後，把這作為一個增值功能插入。這會讓我們的SDK更貼近「一站式」：使用者不用到處查 chainId 或 RPC，一行代碼即可取得所需資訊 [41]（例如 `ChainConfig.ethereumMainnet.rpcUrls` 等）。

總之，Chain config 模組的實作難度低但實用性強，應該在資源允許的情況下盡早補全。

8. 合約抽象工具（合約類別代碼產生與介面封裝）

現有方案： 為了方便開發者調用智慧合約，ethers.js 提供了 `Contract` 類，透過 ABI 自動生成方法介面。Dart 生態也有類似思路的嘗試： - **web3dart 的 DeployedContract：** web3dart允許加載 ABI JSON，創建 `DeployedContract` 對象，再透過 `contract.function("methodName")` 獲取 `ContractFunction` 對象調用。這種方式需要手動處理輸入輸出，不直觀也容易出錯。 - **代碼產生器：** 為了解決上述問題，web3dart 開發者提供了 Dart build_runner Builder，可根據 ABI 自動生成對應的 Dart 類別。早期有個套件名為 **web3_contract_generator**（由 secure.vote 釋出） [42]。使用時將 ABI 檔案命名為 `.abi.json`，執行 `pub run build_runner build` 後，生成的 Dart 類包含每個合約方法作為強型別函式，可直接呼叫 [43] [44]。然而，這套 generator 發布於5年前且已不再更新，在 Dart 3 下不兼容 [45]。 - **社群維護的生成器：** 最近社群有一個 **web3dart_builders** 套件（由 xclud 發布）更新了合約生成邏輯 [46]。它其實是 web3dart 的一部分，被抽出方便更新。目前 3.0.1 版本支持 Dart 2.17+，需要依賴 web3dart 2.5.1 [47]。該生成器應該能產生類似先前 secure.vote 示例的合約類別，並修復了一些舊版問題（例如支援struct輸入、函數重載等 [48] [15]）。

缺口與問題： **合約抽象**在 Dart 上的明顯問題是缺少**官方或統一**的方案。web3dart 原作者的 generator 沒有隨套件長期維護，導致使用者可能不知道有這功能或碰到不相容問題。而新的 web3dart_builders 又依賴特定 fork 的 web3dart，整合起來較麻煩。此外，和 ethers.js 相比，我們缺少**執行期**的 Contract 類型：ethers 的 `Contract` 可在執行期根據 ABI 構造代理對象（透過 Proxy/Reflection 調用），而 Dart 沒有直接的語言機制實現動態代理調用（可以用 `reflectable` 但會導致繁瑣的編譯配置）。因此我們多半還是走**生成器**路線，用靜態代碼幫助開發者。

在目前情況下，**沒有現成的完美方案**可用。我們可能需要自己撰寫或fork現有 generator，使其： - 支援最新 Solidity ABI 特性（如 receive函數、fallback函數、自訂錯誤類型ABI等）。 - 輸出更現代的 Dart 代碼風格，並考慮 Null safety。 - 與我們SDK中的類型整合，例如使用我們自有的 BigInt封裝或Address類別等。

成熟度： 生成器本身技術上不複雜，但需要處理大量 ABI 格式細節。從 secure.vote 的實驗性版本 [49] 就提醒「尚不建議用於正式代碼」。經過社群改進的版本有所提升，但下載量和使用回饋很少 [50]（downloads僅數百次），說明還未廣泛驗證。整體而言，**合約抽象的成熟度偏低**，大部分 Flutter/Dart 開發者可能仍直接用低階呼叫或乾脆在後端用其他語言處理。

優先性考量： 合約抽象直接影響開發者體驗。如果沒有這層封裝，使用我們SDK將會很繁瑣，需要手動編碼方法ID和處理encode/decode。為了達到替代 ethers.js 的目標，**合約抽象工具應該是高優先**項目。然而，由於其依賴 ABI 編碼器完善以及 SDK 其他部分準備就緒，實際開始實作可能略晚於核心功能。建議步驟： 1. 先確定 ABI 編碼/解碼模組穩定，能支援輸入輸出的序列化。 2. 設計 Contract 抽象的 API。可以參考 ethers 的 `Contract` 類，讓使用者提供 ABI 或直接提供已生成的 contract interface 類。 3. 實作代碼產生器或者其他方案。為簡化初期工作，可以考慮**fork web3dart_builders** [46]，在其基礎上做修改，然後逐步融入我們SDK。 4. 提供用戶指引：例如整合 Dart build_runner，使開發者只需將 ABI 丟進專案，即可生成對應類別並import使用。

總之，**合約抽象工具的重要性高，但實作順序稍後**：待核心 RPC、簽名等就緒後著手。同時我們可列出可參考或fork的專案（web3dart_builders、secure.vote範例），以節省開發時間。

9. 事件訂閱與 Reorg 重組處理（合約 Logs 訂閱及鏈重組應對）

現有方案： 智慧合約 Event 是區塊鏈應用的重要組成，web3dart 提供了對事件 Logs 的監聽支持。例如開發者可利用 Web3Client 的 `events` 流，或自行透過 `Web3Client.call` 定義 Filter 來輪詢日誌 [51]。在 WebSocket 模式下，還可以訂閱 `eth_subscribe` 得到即時事件推播。基本的事件監聽流程在 Dart 上是可行的，官方說明也提到庫**可以監聽智能合約所發出的事件** [51]。例如： - 透過 `FilterOptions` 設定要監看的合約地址與事件topics。 - 調用 `Web3Client.events(filter)` 獲取一個 Dart Stream，持續收到契合filter的新事件。

鏈重組 (Reorg) 問題： 當區塊鏈短暫出現分叉時，已發出的事件可能被回溯取消（即「重組」導致某些 earlier logs 成為孤塊無效）。這在以太坊上時有發生，因此**健壯的事件監聽**需要處理重組：撤銷已通知的事件或重新抓取正確的事件集。很遺憾，目前 Dart 生態尚沒有現成的 reorg 處理工具。web3dart/Web3Client 並未內建對重組的自動處理，監聽 Stream 只是如實地把節點推送或輪詢到的結果發出。如果發生重組，節點（如 Geth）會透過 WebSocket 發送帶有 `removed: true` 的 Log 通知，但應用需自行判斷並處理。對於HTTP輪詢模式，需要對比區塊號或使用一些**確認機制**（如只在事件出現N個塊確認後再向應用匯報）。

社群討論： 在以太坊開發圈，新手常問如何處理 reorg [52]。常見方案是在接收到事件後暫存，等待幾個區塊確認再正式採用，或者監看最新區塊通知，一旦發現重組，手動比對日誌 [53]。顯然，這是一個複雜且容易出錯的環節。以太坊 Python生態（web3.py）或JS生態有一些工具處理 reorg，而 Dart 這方面是缺失的。

成熟度： **事件訂閱**本身 web3dart 已給出基礎實現，但**高級應用（如重組處理、事件去重等）**缺乏支持。目前沒有開源的 Dart 庫提供「可靠事件監聽」封裝。

坑點： 開發者如果直接使用 Web3Client.events： - 需要自己維護 last seen block，避免漏掉或重複事件。 - 重組情況下，可能收到重複事件甚至遺漏事件而不自知。 - WebSocket 模式下的 removed標誌要處理，但 web3dart 可能未暴露 removed flag（需確認節點返回）。

優先性考量： 對一個完善的 Web3 SDK 而言，提供健全的事件訂閱是加分項。在優先順序上，這可能不是初版必備（因為開發者可以接受先拿到原始事件Stream，自己簡單處理），但中後期應加上。建議： - **短期**：先提供基本的事件監聽接口，確保 WebSocket 和 HTTP 方式都能用，行為與 web3dart 保持一致。 - **中期**：實作一個**可選的強穩定事件監聽**器。例如提供參數讓使用者選擇等待N個確認再發出事件，或自動過濾 removed: true 的事件。可以參考一些開源實現（如 Python web3 middleware 或 EthDeFi 社群的 event reader [54]）。 - **長期**：與鏈上 indexing 解決方案集成（例如TheGraph），但這超出SDK範圍。

總的來說，**事件訂閱基礎**功能應盡早有，但**Reorg處理**屬進階需求，實作較複雜，可稍後投入。優先度上，事件訂閱屬**中等優先**：它對即時性DApp（如行情板、通知）很重要，但不影響發交易等核心能力。Reorg處理則視資源決定，可在SDK趨於成熟時增強，以提升可靠性。

10. 進階功能（ERC-4337 Account Abstraction、Permit2、MEV Bundle、Multicall 等）

最後這部分涵蓋一些最新或高階的以太坊功能。目前 Dart 生態對這些幾乎沒有內建支援，屬**明顯空白**區域。我們分別說明：

- **ERC-4337 帳戶抽象 (Account Abstraction):** 這是2023年起逐漸興起的新標準，透過名為「UserOperation」的交易實現智能合約錢包的去信任中繼。完整實現需要與 **Bundler** 節點交互。Dart 中此前有 **Fuse** 團隊推出的 `userop.dart` 套件來支持4337 [55]。它提供了構建 UserOperation、與 Bundler RPC 溝通的介面 [56] [57]。然而該套件僅發佈到0.3.1就停止，並聲明已被新的 `permissionless` 套件取代 [58]（可能Fuse將功能整合到自家應用，不再單獨維護）。總體而言，目前沒有活躍維護的4337 Dart SDK。我們若要支援，需要自行研發或追蹤以太坊社群的新進展（例如 Candide錢包據稱有在研究Dart SDK [59] [60]）。**缺口嚴重性**高，但**實用性**取決於我們目標用戶：4337 仍在早期採用階段，如果我們SDK短期主攻一般DApp開發，可將其優先度放低；但長期看，account abstraction 潛力巨大，值得提前布局。

- **Permit2:** 由 Uniswap 推出的代幣授權新機制，允許使用單一簽名批量授權多代幣，並提供更精細的授權控制。Permit2 本質上是在客戶端簽名一筆 EIP-712 訊息，合約合併驗證。目前 Dart 沒有專門的 Permit2 工具。但我們有 EIP-712 基礎，可依據 Uniswap Permit2 的消息結構實作簽名工具（即構造對應 TypedData 結構並簽名）。Permit2 主要涉及標準的TypedData簽名和 `permitTransferFrom` 之類的合約呼叫，技術上不難。**缺口**在於需要有人去整理 Permit2 規範並寫好範本代碼。由於 Permit2 已被越來越多 DApp 採用（改善用戶體驗，無需逐一代幣授權），支持它會讓我們SDK更有前瞻性。建議優先度：**中等**。可在 SDK 穩定後，作為 utils 類提供（例如 Utility函式將特定Permit資料轉換簽名，或一個 PermitHelper 類封裝請求與鏈上確認）。

- **MEV Bundles:** Flashbots 等提供將多個交易打包並私下提交給礦工的介面，用於擠出最大化收益（MEV）。這部分極為前沿，JS/TS界也主要透過直接調用 Flashbots 提供的 RPC（改裝的 `eth_sendBundle` 方法）來使用。Dart 目前沒有任何對應庫。若需支持，我們可能只需讓 RPC Provider 容易擴展，以便調用自定義的 RPC 方法。例如 Flashbots bundle 需要先簽名交易封裝，再調用 flashbots的中繼節點 RPC。這可以通過我們SDK讓用戶組裝好簽名交易列表後，用自定義 method

發送。**缺口嚴重性**對一般DApp不高（MEV主要專業交易者用），因此可以緩辦。未來若我們鎖定DeFi套利這些場景，再考慮深度支援（例如提供 bundle 簽名工具、與flashbots API對接）。

- **Multicall:** Multicall是一種在單一RPC請求中執行多個合約只讀調用的方案，通常由部署在鏈上的 Multicall 合約實現。ethers.js 有內建 Multicall 封裝。Dart 目前沒有專門類別，但開發者可以自行呼叫已有的 Multicall 合約（比如 MakerDAO 的 Multicall合約地址）。為方便使用，我們可考慮在 SDK 提供 Multicall helper：用戶傳入一系列合約方法調用，我們打包成 multicall 合約的輸入並一次讀取返回。這涉及 ABI 編碼和解析，目前技術基礎已具備，因此實作難度不高。**優先度**可定為**中等偏高**，因為它對提升多重查詢效率很有幫助，在前端環境尤其有價值（減少多次RPC延遲）。相比其它進階功能，Multicall 實作簡單且應用場景廣（例如一次查很多代幣餘額），值得較早提供。

總體而言，進階功能的實用性各異。依據一般性原則，**Permit2** 和 **Multicall** 對DApp開發者比較有直接幫助，優先度可以高於其他；**ERC-4337** 和 **MEV** 偏專業或未大規模普及，可作為長期規劃。在資源有限的情況下，我們初版SDK可以不涵蓋這些，或僅涵蓋 Multicall。隨著SDK基礎完善，再逐步加入 Permit2 工具、4337 支援等，保持對新標準的追蹤能力。

優先實作順序與結論

綜合以上各模組的缺口嚴重性與實用價值，我們建議的實作優先順序如下：

1. **核心通訊與簽章基礎** – **RPC Provider** 與 **交易簽名**相關功能應當首先落實。這包括設計新的 Provider 接口（支援 HTTP/WS、批次請求） [27] 、實現 Signer/Credentials 抽象以及 EIP-1559 等交易類型支援 [61] 。這些是整個 SDK 的根基，必須**優先且完善**。

2. **ABI 編碼/解碼與合約互動** – 在能發交易後，我們需要確保能**編碼呼叫資料並解析回傳結果**。完善 ABI 編碼器（含 EIP-712） [11] 屬高優先，其次盡快提供**合約抽象工具**方便調用。合約代碼生成功能可稍緩一步，先提供簡單易用的動態Contract接口，再著手實現 generator 自動化 [62] 。這部分優先度也很高，因為大部分DApp需求都是與合約互動。

3. **Public/Wallet Client 架構** – 在開發上述核心時，即應同步考慮**架構分離**。雖然這不直接體現為功能，但對日後擴充與用戶體驗至關重要。可在完成 Provider/Signer 基本功能後立即將其拆分為 PublicClient 與 WalletClient 兩套接口，對齊 ethers/viem 模型 [24] 。這屬中高優先度，因為調整架構越早越好（影響整體API設計）。

4. **錢包連接（WalletConnect）** – 在本地功能齊備後，應著手支援外部錢包。WalletConnect v2 客戶端的缺口很大，目前Flutter只能用官方UI套件 [37] 。因此我們建議將**WalletConnect v2 支援**作為中期目標，在 SDK 有基本能力後立即投入。這將包含開發一個可以處理WalletConnect Session、Relay、簽名請求的模組，可能需要較多工作，但回報是讓 SDK 真正覆蓋移動端 DApp 的主要場景。

5. **鏈網路配置** – 待上述主要功能完成，我們可以加入**ChainConfig**模組作為增強。優先級中等，但實作容易。透過內建常見鏈資料，方便開發者切換網路而無需自行查詢 [41] 。這一步驟可與 WalletConnect 並行進行，因為 WalletConnect 也需要宣告支援鏈列表，兩者相輔相成。

6. **事件監聽與可靠性** – 基本事件監聽功能應該與合約調用一起提供（優先度中等）。但**reorg處理**等高級特性可稍後實作。一旦SDK進入較穩定階段，我們可推出**增強版事件訂閱**：例如提供選項等待 12 確認才通知事件，或自動處理重組的事件移除。這將是 SDK 的一大賣點，但考量複雜度，優先度相對靠後。

7. **進階功能（4337/Permit2/MEV/Multicall）** – 這些屬**最後階段**的目標。尤其 ERC-4337 實現複雜，可等待標準更成熟或有合作機會再投入。Permit2 和 Multicall 倒是可以較早納入計畫：Multicall 工具因

簡單可提到中優先，Permit2 屬應用增強可在SDK穩定後加入。MEV bundle 則極為特殊，可留到用戶有需求時再支持。

最後，參考上述順序，我們建議首先專注於**不可或缺的核心模組**（加密、交易、RPC、ABI）和**架構打底**，確保 SDK 有與 ethers.js 相媲美的基礎能力 [1]。隨後逐步填補目前 Dart 生態缺乏的關鍵功能（如 WalletConnect、合約代碼產生）。透過這樣的漸進開發，我們有望打造出一個**功能完備的 Dart 原生 Web3 SDK**，彌補現存生態的諸多短板，在未來取代 ethers.js / viem 成為 Dart/Flutter 領域的首選方案。

參考資料：

- web3dart 停止維護公告與 ethers.dart 簡介 [1]
- Dart 密碼學庫（secp256k1, SHA3 等） [2] [4]
- BIP-39 助記詞 Dart 套件 [5]
- EIP-712 Dart 完整實現套件說明 [11]
- ethers.js vs viem 架構比較（PublicClient/WalletClient 定義） [24]
- Flutter 專案常用的 web3 套件清單 [63] [64]
- web3dart 合約代碼產生器（舊版）警告 [49]
- web3dart 支援事件監聽功能說明 [51]
- Fuse `userop.dart` 套件簡介（ERC-4337） [56] (已停更)
- WalletConnect v2 Flutter 模組使用示例 [37] [38]

---

[1] [25] [26] [31] [32] ethers - Dart API docs
https://pub.dev/documentation/ethers/latest/

[2] sec | Dart package - Pub.dev
https://pub.dev/packages/sec

[3] secp256k1 library - Dart API
https://pub.dev/documentation/blockchain_utils/latest/crypto_crypto_cdsa_secp256k1_secp256k1/

[4] keccak256 library - Dart API - Pub.dev
https://pub.dev/documentation/lunaris_engine/latest/Cryptographic_keccak256

[5] bip39 - Dart API docs - Pub.dev
https://pub.dev/documentation/bip39/latest/

[6] bip39_plus | Dart package - Pub.dev
https://pub.dev/packages/bip39_plus

[7] [8] wallet | Dart package
https://pub.dev/packages/wallet

[9] [20] webthree | Dart package
https://pub.dev/packages/webthree

[10] [23] [27] [28] [35] [36] [37] [38] [39] [63] [64] Best Flutter Packages for Web3 and Blockchain That Help Build Web3 and Blockchain-Based Applications in Flutter. | by Ranjan Kumar | Medium
https://medium.com/@rk0936626/best-flutter-packages-for-web3-and-blockchain-that-help-build-web3-and-blockchain-based-6d6661719316

[11] [12] [13] [16] [17] [18] eip712 | Dart package
https://pub.dev/packages/eip712

14 15 19 21 22 29 30 48 61 webthree changelog | Dart package
https://pub.dev/packages/webthree/changelog

24 Viem vs. Ethers.js: A Comparison for Web3 Developers
https://metamask.io/news/viem-vs-ethers-js-a-detailed-comparison-for-web3-developers

33 Frequently Asked Questions - Viem
https://viem.sh/docs/faq

34 walletconnect_dart package - All Versions
https://pub.dev/packages/walletconnect_dart/versions

40 GitHub - Zfinix/awesome-dart-web3: ⚡ A list of Web3 Tools for building in Dart/Flutter
https://github.com/Zfinix/awesome-dart-web3

41 ChainList
https://chainlist.org/

42 43 44 45 49 62 web3_contract_generator | Dart package
https://pub.dev/packages/web3_contract_generator

46 47 50 web3dart_builders | Dart package
https://pub.dev/packages/web3dart_builders

51 simolus3/web3dart: Ethereum library, written in Dart. - GitHub
https://github.com/simolus3/web3dart

52 What is the best way to maintain a complete list of events using ...
https://ethereum.stackexchange.com/questions/141080/what-is-the-best-way-to-maintain-a-complete-list-of-events-using-web3-py

53 new to ethers/web3 : How to handle reorgs? : r/ethereum - Reddit
https://www.reddit.com/r/ethereum/comments/xmnnam/new_to_ethersweb3_how_to_handle_reorgs/

54 eth_defi.event_reader.reorganisation_monitor
https://web3-ethereum-defi.readthedocs.io/_modules/eth_defi/event_reader/reorganisation_monitor.html

55 56 57 58 userop | Dart package
https://pub.dev/packages/userop

59 The Stack behind Candide Mobile Wallet
https://docs.candide.dev/blog/mobile-app-tech-stack/

60 ERC-4337 Development Blog - HackMD
https://hackmd.io/@erc4337