Integration Engineer @ CoW DAO — Interview Q&A

> 中文標題：CoW DAO 整合工程師
>
> Source JD: `Integration Engineer @ CoW DAO.pdf` Total questions: 105 Questions with diagrams: 14

Subtitle (EN): TS/JS · EVM/Ethereum · DEX aggregator · MEV protection · SDK & API · partner integration

Subtitle (中): TypeScript/JS、EVM、DEX 聚合、MEV 防護、SDK/API、合作夥伴整合

Q001 · `basic` · `COW` `OVERVIEW`

EN: What is CoW Protocol and why does intent-based trading matter?

中: CoW Protocol 是什麼？為什麼以「意圖（intent）」為核心的交易模式很重要？

Answer (English)

- CoW = **Coincidence of Wants**: a meta DEX where users sign an *intent* ("I want X for at least Y") instead of building a swap transaction.
- Off-chain solvers compete in batch auctions to find the best execution path (peer match, AMMs, RFQ, multi-hop).
- Users get MEV protection, gasless orders (relayer pays), and price improvement (surplus) returned to them.
- It powers CoW Swap (the aggregator front-end), MEV Blocker (RPC), and CoW AMM (LVR-resistant pool).

答案（中文）

- CoW = **Coincidence of Wants（需求巧合）**，使用者簽的是「意圖」（我要 X、至少換到 Y），而不是直接送 swap 交易。
- 鏈下 solver 在 batch auction 中競標，從點對點匹配、AMM、RFQ、多跳路徑中找最佳執行。
- 使用者得到 MEV 保護、無 gas 下單（由 relayer 代付）、以及把價差紅利（surplus）退回。
- 它支撐 CoW Swap（聚合器前端）、MEV Blocker（RPC）、CoW AMM（抗 LVR 流動池）。

Q002 · `basic` · `JS` `CLOSURES`

EN: Explain JavaScript closures with a small example.

中: 請用一個簡短範例解釋 JavaScript 的 closure（閉包）。

Answer (English)

- A closure is a function plus the lexical environment captured at the moment it was created.
- It lets inner functions read variables from an outer scope even after the outer function has returned.
- Common uses: data privacy, partial application, memoization, event handlers with bound state.
- Watch out: closing over loop variables with `var` (use `let` to get per-iteration bindings).

```
function counter(start = 0) {
  let n = start;
  return {
    inc: () => ++n,
    value: () => n,
  };
}
const c = counter(10);
c.inc(); c.inc();
console.log(c.value()); // 12
```

答案（中文）

- Closure 是「函式 + 它建立時所捕捉的詞法環境」。
- 即使外層函式已經 return，內層函式仍能讀取外層變數。
- 常見用途：資料封裝、部分套用、memoization、帶狀態的事件處理。
- 注意：用 `var` 在迴圈中閉包會共享同一變數，改用 `let` 才能取得每次迭代的綁定。

```
function counter(start = 0) {
  let n = start;
  return {
    inc: () => ++n,
    value: () => n,
  };
}
const c = counter(10);
c.inc(); c.inc();
console.log(c.value()); // 12
```

Q003 · `basic` · `EVM` `ACCOUNTS`

EN: What are the two account types on Ethereum and how do they differ?

中: 以太坊上有哪兩種帳戶？它們的差別是什麼？

Answer (English)

- **EOA (Externally Owned Account)** — controlled by a private key, has nonce + balance, can initiate transactions.
- **Contract Account** — controlled by code, has nonce + balance + storage + bytecode, only executes when called.
- Both share the same 20-byte address space; only EOAs can sign, contracts must be triggered.
- EIP-7702 (Pectra, 2024) lets EOAs temporarily run contract code per transaction, blurring the line.

答案（中文）

- **EOA（外部帳戶）**：由私鑰控制，有 nonce + 餘額，可主動發起交易。
- **合約帳戶**：由程式碼控制，有 nonce + 餘額 + storage + bytecode，被呼叫時才執行。
- 兩者共用相同 20-byte 位址空間；只有 EOA 能簽章，合約只能被觸發。
- EIP-7702（Pectra，2024）讓 EOA 可在單筆交易中暫時執行合約程式碼，兩者界線變模糊。

Q004 · `intermediate` · `COW` `ARCHITECTURE`

EN: Walk me through the lifecycle of a CoW Swap order from sign to settlement.

中: 請逐步說明一筆 CoW Swap 訂單從簽章到結算的完整生命週期。

Answer (English)

- User signs an EIP-712 `GPv2Order` payload in-wallet (no on-chain tx, no gas).
- Order is POSTed to the orderbook API (off-chain) and becomes visible to all whitelisted solvers.
- In each ~30s batch, solvers submit settlement proposals; the auction picks the one returning most surplus to traders.
- Winning solver submits one `settle()` tx to the GPv2Settlement contract, which atomically transfers tokens for every order in the batch.

```
// Pseudocode for solver settlement
GPv2Settlement.settle(
  tokens,        // address[]
  clearingPrices,// uint256[]
  trades,        // signed orders
  interactions   // pre/intra/post hooks
);
```

答案（中文）

- 使用者在錢包簽 EIP-712 的 `GPv2Order` payload（不上鏈、不付 gas）。
- 訂單 POST 到 orderbook API（鏈下），所有 whitelisted solver 都能看到。
- 每個 ~30 秒批次中，solver 提交結算方案；競標選出回饋最多 surplus 給交易者的方案。
- 獲勝 solver 送一筆 `settle()` 到 GPv2Settlement 合約，原子地完成整批訂單的代幣移轉。

```
// Solver 結算偽碼
GPv2Settlement.settle(
  tokens,        // address[]
  clearingPrices,// uint256[]
  trades,        // 已簽訂單
  interactions   // pre/intra/post hooks
);
```

Diagram

`cow-01-protocol-flow.svg`

Q005 · `basic` · `JS` `EVENT-LOOP`

EN: How does the JavaScript event loop schedule microtasks vs macrotasks?

中: JavaScript event loop 如何排程 microtask 與 macrotask？

Answer (English)

- After every script/callback the engine **drains the entire microtask queue** before picking the next macrotask.
- Microtasks: `Promise.then`, `queueMicrotask`, `MutationObserver`. Macrotasks: `setTimeout`, `setInterval`, I/O, UI events.
- A microtask added inside a microtask is still processed in the same drain — beware infinite microtask loops starving the UI.
- Order: sync code → microtasks → render (browser) → next macrotask.

```
console.log('A');
setTimeout(() => console.log('B'));
Promise.resolve().then(() => console.log('C'));
console.log('D');
// A D C B
```

答案（中文）

- 每個 script/callback 跑完後，引擎會**先清空整個 microtask 佇列**才挑下一個 macrotask。
- Microtask：`Promise.then`、`queueMicrotask`、`MutationObserver`；Macrotask：`setTimeout`、`setInterval`、I/O、UI 事件。
- 在 microtask 中再加入 microtask 仍會在同一輪清掉，要避免讓 UI 餓死的無限 microtask 迴圈。
- 順序：同步碼 → microtask → 瀏覽器 render → 下一個 macrotask。

```
console.log('A');
setTimeout(() => console.log('B'));
Promise.resolve().then(() => console.log('C'));
console.log('D');
// A D C B
```

Q006 · `intermediate` · `EVM` `GAS` `EIP-1559`

EN: Explain EIP-1559: baseFee, priority tip, and maxFeePerGas.

中: 解釋 EIP-1559 的 baseFee、priority tip 與 maxFeePerGas。

Answer (English)

- `baseFee` is set per block by the protocol; it is **burned**, not paid to validators.
- `maxPriorityFeePerGas` (tip) is what the user pays the validator for inclusion.
- `maxFeePerGas` is the **upper cap**: actual cost = min(maxFee, baseFee + tip); the rest is refunded.
- baseFee adjusts ±12.5% per block targeting 50% block fullness — so wallets should set maxFee ≥ 2× current baseFee to survive a spike.

```
const tx = await wallet.sendTransaction({
  to: '0x...',
  value: 0n,
  maxFeePerGas: parseGwei('40'),
  maxPriorityFeePerGas: parseGwei('1.5'),
});
```

答案（中文）

- `baseFee` 由協定每個區塊計算後**燒掉**，不發給驗證者。
- `maxPriorityFeePerGas`（tip）才是使用者付給驗證者的「插隊費」。
- `maxFeePerGas` 是**上限**：實付 = min(maxFee, baseFee + tip)，其餘退回。
- baseFee 每區塊以 ±12.5% 動態調整，目標 50% 區塊填充率，所以建議 maxFee ≥ 2× 當下 baseFee 以撐過尖峰。

```
const tx = await wallet.sendTransaction({
  to: '0x...',
  value: 0n,
  maxFeePerGas: parseGwei('40'),
  maxPriorityFeePerGas: parseGwei('1.5'),
});
```

Diagram

`cow-09-eip1559-gas.svg`

Q007 · `intermediate` · `COW` `BATCH-AUCTION`

EN: Why does CoW use batch auctions instead of immediate execution?

中: 為什麼 CoW 採用 batch auction 而不是即時執行？

Answer (English)

- Batching gives a **uniform clearing price** per token pair so adversaries cannot reorder users within a batch.
- It enables CoW matches: if Alice sells DAI for ETH and Bob sells ETH for DAI in the same batch, they trade peer-to-peer — no AMM, no slippage, no MEV.
- It separates **price discovery** (off-chain auction) from **settlement** (one on-chain tx) — cheaper and MEV-free.
- Solvers compete on full-batch optimality (max surplus across all orders), not per-order, raising overall efficiency.

答案（中文）

- Batching 讓同一交易對在批次內有**單一結算價**，外部攻擊者無法在批次中重排使用者順序。
- 批內可形成 CoW 匹配：Alice 賣 DAI 換 ETH、Bob 賣 ETH 換 DAI 直接配對，免 AMM、無滑價、無 MEV。
- 把**價格發現**（鏈下競標）與**結算**（單筆鏈上交易）分離，成本更低且抗 MEV。
- Solver 競標的是「整批最大化總 surplus」而非單筆，全域效率更好。

Diagram

`cow-02-batch-auction-timeline.svg`

Q008 · `intermediate` · `JS` `PROMISES`

EN: Compare Promise.all, Promise.allSettled, Promise.race, and Promise.any.

中: 比較 Promise.all、Promise.allSettled、Promise.race、Promise.any。

Answer (English)

- `Promise.all` resolves when **all** succeed, rejects on first failure (fast-fail).
- `Promise.allSettled` waits for **all** to finish and returns `{status, value|reason}[]` — never rejects.
- `Promise.race` settles with the **first to settle** (fulfill or reject).
- `Promise.any` resolves with **first fulfilled**, rejects with `AggregateError` only if all reject.

```
const [user, prices] = await Promise.all([fetchUser(), fetchPrices()]);

const results = await Promise.allSettled(rpcs.map(r => fetch(r)));
const alive = results.filter(r => r.status === 'fulfilled');
```

答案（中文）

- `Promise.all`：**全部成功**才 resolve，任何一個 reject 就立刻 fast-fail。
- `Promise.allSettled`：**等所有結束**才回傳 `{status, value|reason}[]`，永遠不會 reject。
- `Promise.race`：**第一個結束的**為準（成功或失敗）。
- `Promise.any`：**第一個成功**，全部失敗才回 `AggregateError`。

```
const [user, prices] = await Promise.all([fetchUser(), fetchPrices()]);

const results = await Promise.allSettled(rpcs.map(r => fetch(r)));
const alive = results.filter(r => r.status === 'fulfilled');
```

Q009 · `intermediate` · `EIP-712` `SIGNATURES`

EN: Why does CoW use EIP-712 typed data signatures?

中: CoW 為什麼採用 EIP-712 結構化簽章？

Answer (English)

- EIP-712 lets wallets render structured fields (sellToken, buyAmount, validTo…) instead of an opaque hex blob.
- Domain separator (chainId + verifyingContract) prevents replay across chains and protocols.
- Hash is deterministic: `keccak256(0x1901 || domainHash || structHash)`.
- CoW orders use EIP-712 so the orderbook can verify the same hash a contract would compute — and so EIP-1271 smart wallets can validate the same payload.

```
const domain = {
  name: 'Gnosis Protocol',
  version: 'v2',
  chainId: 1,
  verifyingContract: '0x9008D19f58AAbD9eD0D60971565AA8510560ab41',
};
const signature = await signer.signTypedData(domain, ORDER_TYPES, order);
```

答案（中文）

- EIP-712 讓錢包顯示結構化欄位（sellToken、buyAmount、validTo…），而非難讀的 hex 位元組。
- Domain separator（chainId + verifyingContract）避免跨鏈或跨協定重放。
- 雜湊規則固定：`keccak256(0x1901 || domainHash || structHash)`。
- CoW 訂單以 EIP-712 簽章，orderbook 才能算出與合約完全相同的 hash，並讓 EIP-1271 智能錢包用同一份 payload 驗證。

```
const domain = {
  name: 'Gnosis Protocol',
  version: 'v2',
  chainId: 1,
  verifyingContract: '0x9008D19f58AAbD9eD0D60971565AA8510560ab41',
};
const signature = await signer.signTypedData(domain, ORDER_TYPES, order);
```

Q010 · `intermediate` · `COW` `SOLVERS`

EN: How does CoW solver competition work and how are solvers ranked?

中: CoW 的 solver 競標如何運作？solver 如何被評分？

Answer (English)

- Each batch the orderbook publishes the open orders; whitelisted solvers submit their best settlement proposal off-chain.
- The autopilot scores proposals by **trader surplus** (improvement vs reference price) and picks the winner.
- Winner pays the on-chain settlement gas and earns COW rewards proportional to surplus delivered (minus a stake bond if it fails).
- Repeated under-performance or fraud (e.g., self-trading) leads to slashing or removal from the whitelist.

答案（中文）

- 每個批次 orderbook 公開待結算訂單，whitelisted solver 在鏈下提交最佳方案。
- Autopilot 以**使用者 surplus**（相對參考價格的改善幅度）評分，選出獲勝者。
- 獲勝者支付鏈上結算 gas，並依 surplus 大小領取 COW 獎勵（執行失敗會扣 stake）。
- 長期表現不佳或舞弊（例如自我交易）會被 slash 或移出白名單。

Diagram

`cow-04-solver-competition.svg`

Q011 · `basic` · `TS` `NARROWING`

EN: What is type narrowing in TypeScript?

中: TypeScript 中的 type narrowing 是什麼？

Answer (English)

- TS analyses runtime checks (`typeof`, `instanceof`, `in`, equality) to refine a value's type within a branch.
- Custom narrowing via **user-defined type guards**: `function isUser(x): x is User { ... }`.
- Discriminated unions narrow nicely on a literal `kind` field — exhaustiveness checked with `never`.
- Avoid `as` casts when a guard is possible; casts bypass narrowing and hide bugs.

```
type Result<T> = { ok: true; value: T } | { ok: false; error: string };
function unwrap<T>(r: Result<T>): T {
  if (r.ok) return r.value;
  throw new Error(r.error);
}
```

答案（中文）

- TS 會分析 `typeof`、`instanceof`、`in`、等號等檢查，在條件分支中收窄變數型別。
- 可寫**user-defined type guard**：`function isUser(x): x is User { ... }`。
- Discriminated union 透過 `kind` 字面量欄位收窄，搭配 `never` 可做 exhaustive 檢查。
- 能寫 guard 就不要用 `as` 強轉；強轉會跳過收窄並掩蓋 bug。

```
type Result<T> = { ok: true; value: T } | { ok: false; error: string };
function unwrap<T>(r: Result<T>): T {
  if (r.ok) return r.value;
  throw new Error(r.error);
}
```

Q012 · `intermediate` · `EVM` `JSON-RPC`

EN: Compare eth_call and eth_sendTransaction.

中: 比較 eth_call 與 eth_sendTransaction。

Answer (English)

- `eth_call` is a **read-only** simulation against a recent block — no state change, no gas paid, no signature needed.
- `eth_sendTransaction` (or `eth_sendRawTransaction`) submits a **signed tx** to the mempool, mutating chain state and costing gas.
- Useful pattern: `eth_call` first to predict revert/return value, then send transaction.
- `eth_call` accepts `state overrides` (e.g., simulate with extra balance) — handy for partner debugging.

```
// 1. simulate
const out = await client.call({ to, data, blockTag: 'latest' });
// 2. estimate
const gas = await client.estimateGas({ account, to, data });
// 3. send
const hash = await wallet.sendTransaction({ to, data, gas });
```

答案（中文）

- `eth_call` 是針對某個區塊的**唯讀模擬**——不改狀態、不付 gas、不必簽章。
- `eth_sendTransaction`（或 `eth_sendRawTransaction`）把**已簽交易**送入 mempool，會改變鏈狀態並付 gas。
- 常見流程：先 `eth_call` 預測 revert / 回傳值，再送交易。
- `eth_call` 支援 `state overrides`（例如模擬餘額更多的情境），對合作夥伴除錯很方便。

```
// 1. 模擬
const out = await client.call({ to, data, blockTag: 'latest' });
// 2. 估 gas
const gas = await client.estimateGas({ account, to, data });
// 3. 送交易
const hash = await wallet.sendTransaction({ to, data, gas });
```

Q013 · `intermediate` · `COW` `ORDERS`

EN: What order types does CoW Swap support?

中: CoW Swap 支援哪些訂單類型？

Answer (English)

- **Market**: sell now, accept any price ≥ slippage tolerance — typical UX for swaps.
- **Limit**: sell only if price hits a target; can sit in the book for days/weeks until filled or expired.
- **TWAP** (Time-Weighted Average Price): split a large order into N child orders over a window, scheduled via programmatic orders.
- **Programmatic Orders (CoW Hooks)**: arbitrary on-chain conditions and pre/post hooks (e.g., yield bearing token swaps).

答案（中文）

- **市價單**：立刻成交，只要價格達到 slippage 容忍範圍——一般 swap 的標準 UX。
- **限價單**：到價才成交，可在 orderbook 中掛數天至數週。
- **TWAP**：把大單依時間切成多個子單，透過 programmatic order 排程執行。
- **Programmatic Order（CoW Hooks）**：可帶任意鏈上條件與 pre/post hook（如 yield bearing token swap）。

Q014 · `basic` · `SDK` `API`

EN: When should a partner use the CoW SDK vs the raw REST API?

中: 合作夥伴何時該用 CoW SDK，何時該用 raw REST API？

Answer (English)

- SDK = typed wrappers, signature helpers, error normalization, retries, network constants — fastest path for JS/TS partners.

- Raw REST = use when you are on a non-JS stack (Go/Rust/Python) or need the smallest possible bundle.
- Either way the wire protocol is the same OpenAPI spec; the SDK just removes boilerplate.
- Tip: even if a partner uses raw REST, recommend they read the SDK source to learn the canonical sign + post pattern.

```
import { OrderBookApi, SupportedChainId } from '@cowprotocol/cow-sdk';
const api = new OrderBookApi({ chainId: SupportedChainId.MAINNET });
const order = await api.getOrder(uid);
```

答案（中文）

- SDK 提供 typed wrapper、簽章 helper、錯誤正規化、重試、網路常數——對 JS/TS 夥伴是最快上手路徑。
- 當技術棧不是 JS（Go/Rust/Python）或需要極小 bundle，再考慮直接打 REST。
- 兩者底層都是同一份 OpenAPI 規範，SDK 只是省掉樣板碼。
- 小撇步：就算用 raw REST，仍建議夥伴看 SDK 原始碼學「簽 + post」的標準寫法。


```
import { OrderBookApi, SupportedChainId } from '@cowprotocol/cow-sdk';
const api = new OrderBookApi({ chainId: SupportedChainId.MAINNET });
const order = await api.getOrder(uid);
```

Q015 · `intermediate` · `INTEGRATION` `ONBOARDING`

EN: Describe a partner onboarding flow from first call to production.

中: 描述一個合作夥伴從首次接觸到上線生產的整合流程。

Answer (English)

- **Discovery call** — capture use case, chains, volume, custodian/wallet stack, regulatory constraints.
- **Sandbox** — give them Goerli/Sepolia keys, sample app, doc bundle; goal: a successful test order in <1 day.
- **Staging** — mainnet read-only via small order sizes, observability hooks for both sides, weekly sync.
- **Production** — go-live runbook, joint Slack channel, incident escalation matrix, success metrics review at 30/60/90 days.

答案（中文）

- **Discovery call**：盤點使用案例、上的鏈、預估量、託管/錢包技術棧、法規限制。
- **Sandbox**：發 Goerli/Sepolia 金鑰、範例 App、文件包，目標 1 天內完成第一筆測試單。
- **Staging**：主網小額試跑，雙方接 observability，每週同步。
- **Production**：上線 runbook、共用 Slack channel、事故升級矩陣、30/60/90 天成果回顧。

Diagram

`cow-08-onboarding-pipeline.svg`

Q016 · `intermediate` · `JS` `PROTOTYPES`

EN: How does prototypal inheritance work in JavaScript?

中: JavaScript 的原型繼承如何運作？

Answer (English)

- Every object has an internal `[[Prototype]]` (accessible via `Object.getPrototypeOf` / `__proto__`).
- Property lookup walks the prototype chain until found or `null` is reached.
- `class` syntax is sugar over functions + `prototype` — methods are stored on the prototype object, not per-instance.
- Mutating a built-in prototype is risky (monkey-patching); prefer composition or mixins for shared behaviour.

```
class Animal { speak() { return 'noise'; } }
class Dog extends Animal { speak() { return 'woof'; } }
const d = new Dog();
Object.getPrototypeOf(d) === Dog.prototype; // true
```

答案（中文）

- 每個物件都有內部 `[[Prototype]]`（可由 `Object.getPrototypeOf` / `__proto__` 取得）。
- 屬性查找會沿原型鏈往上找，直到命中或 `null`。
- `class` 是語法糖：方法存在 prototype 物件上，而非每個實例自己一份。
- 別改內建型別的 prototype（monkey-patch），用組合或 mixin 共享行為較安全。

```
class Animal { speak() { return 'noise'; } }
class Dog extends Animal { speak() { return 'woof'; } }
const d = new Dog();
Object.getPrototypeOf(d) === Dog.prototype; // true
```

Q017 · `intermediate` · `TS` `UNIONS`

EN: Why are discriminated unions powerful for modeling order states?

中: 為什麼 discriminated union 很適合用來建模訂單狀態？

Answer (English)

- A literal `kind` field gives TS a discriminator to narrow the union on each branch.
- Combined with `never` exhaustiveness checks, adding a new state forces every consumer to handle it.
- You avoid optional-property soup (`amountFilled?: bigint`) — each variant carries only the fields it needs.
- JSON parsing + `zod` schemas mirror the same pattern, so wire contracts and TS types stay in sync.

```
type Order =
  | { kind: 'open'; uid: string }
  | { kind: 'partiallyFilled'; uid: string; filled: bigint }
  | { kind: 'filled'; uid: string; tx: `0x${string}` }
  | { kind: 'expired'; uid: string };
function render(o: Order) {
  switch (o.kind) {
    case 'open': return '⏳';
    case 'partiallyFilled': return `${o.filled}…`;
    case 'filled': return '✅';
    case 'expired': return '⌛';
  }
}
```

答案（中文）

- 字面量 `kind` 欄位讓 TS 在每個分支自動收窄型別。
- 搭配 `never` 做 exhaustive 檢查，新增狀態就會強制所有消費端處理。
- 免去「一堆 optional 欄位」的混亂，每個 variant 只帶它需要的欄位。
- 搭 `zod` 解析 JSON 可同樣建模，wire 合約與 TS 型別保持一致。

```
type Order =
  | { kind: 'open'; uid: string }
  | { kind: 'partiallyFilled'; uid: string; filled: bigint }
  | { kind: 'filled'; uid: string; tx: `0x${string}` }
  | { kind: 'expired'; uid: string };
function render(o: Order) {
  switch (o.kind) {
    case 'open': return '⏳';
    case 'partiallyFilled': return `${o.filled}…`;
    case 'filled': return '✅';
    case 'expired': return '⌛';
  }
}
```

Q018 · `intermediate` · `ERC-20` `APPROVE`

EN: Walk me through the ERC-20 approve + transferFrom flow.

中: 說明 ERC-20 approve + transferFrom 的完整流程。

Answer (English)

- User calls `token.approve(spender, amount)` — sets allowance mapping `[owner][spender] = amount`.
- Spender (e.g., GPv2VaultRelayer) later calls `token.transferFrom(owner, to, amount)` — checks + decrements allowance, moves tokens.
- Two-tx UX cost: every new spender needs a separate approve before any swap.
- Pitfalls: zero-then-set pattern for legacy tokens (USDT) and front-running risk on changing allowance — Permit/Permit2 fixes both.

```
// approve once
await token.write.approve([SPENDER, MAX_UINT256]);
// then any number of swaps
await spender.write.swap([...]);
```

答案（中文）

- 使用者呼叫 `token.approve(spender, amount)`，設定授權映射 `[owner][spender] = amount`。
- Spender（如 GPv2VaultRelayer）日後呼叫 `token.transferFrom(owner, to, amount)`，檢查 + 扣除 allowance、轉移代幣。
- 兩筆交易成本：每個新 spender 都得先 approve 才能 swap。
- 常見陷阱：USDT 等舊代幣需先歸零再設值，且更改 allowance 有 race-condition 風險——Permit/Permit2 解決這兩問題。

```
// 先 approve 一次
await token.write.approve([SPENDER, MAX_UINT256]);
// 之後可重複 swap
await spender.write.swap([...]);
```

Diagram

`cow-07-approve-vs-permit2.svg`

Q019 · `basic` · `MEV`

EN: What is MEV and what are common MEV strategies?

中: 什麼是 MEV？常見的 MEV 策略有哪些？

Answer (English)

- **MEV (Maximal Extractable Value)** = profit a block builder can extract by reordering, inserting, or censoring transactions.
- **Sandwich**: front-run a victim swap, back-run after price moves.
- **Arbitrage**: rebalance prices across DEXes within one block.
- **Liquidations**: race to seize undercollateralized positions for the liquidation bonus.

答案（中文）

- **MEV（Maximal Extractable Value）**：區塊建構者透過重排、插入或審查交易所能榨取的額外價值。
- **Sandwich**：在受害者 swap 前後夾單獲利。
- **Arbitrage**：同一區塊內把不同 DEX 之間的價差吃掉。
- **清算**：搶在他人之前清算抵押不足的部位賺取清算獎金。

Q020 · `advanced` · `MEV` `BLOCKER`

EN: How does MEV Blocker route transactions to protect users?

中: MEV Blocker 如何路由交易以保護使用者？

Answer (English)

- Wallet swaps RPC URL to MEV Blocker; users see no UX change.

- Each tx is sent to a private orderflow auction where searchers bid to *back-run* (not front-run) for arbitrage profit.
- A share of the searcher's profit is rebated to the user (~90%) as ETH refund — turning would-be MEV loss into yield.
- Bundles are submitted via Flashbots / private relays; tx never appears in the public mempool.

答案（中文）

- 錢包改用 MEV Blocker RPC，使用者體驗完全無感。
- 每筆交易進入私有 orderflow auction，搜尋者只能**back-run**（不能 front-run）以套利。
- 搜尋者利潤約 90% 退回給使用者，把原本要被 MEV 吃掉的損失轉為收益。
- 透過 Flashbots / 私有 relay 出 bundle，交易不會出現在公開 mempool。

Diagram

`cow-06-mev-blocker.svg`

Q021 · `intermediate` · `COW` `HOOKS`

EN: What are CoW Hooks and what use cases do they unlock?

中: CoW Hooks 是什麼？它解鎖了哪些使用情境？

Answer (English)

- Hooks are arbitrary contract calls executed before (`pre`) or after (`post`) the swap inside the same settlement tx.
- Pre-hook example: unwrap stETH → wstETH so the order can sell wstETH that the user does not yet hold.
- Post-hook example: deposit received DAI into Aave / Maker DSR for instant yield.
- All atomic — if any hook fails the whole batch reverts; gas is bundled with the swap so users get one tx.

答案（中文）

- Hook 是在 settlement 同筆交易內、swap 前（`pre`）或後（`post`）執行的任意合約呼叫。
- Pre-hook 範例：把 stETH 解包成 wstETH，使用者就能賣自己尚未持有的 wstETH。
- Post-hook 範例：把收到的 DAI 直接存入 Aave / Maker DSR 立即賺利息。
- 全程 atomic——任何 hook 失敗整批 revert；gas 與 swap 合併，使用者只送一筆交易。

Q022 · `intermediate` · `COW` `AMM` `LVR`

EN: How does CoW AMM solve LVR (loss-versus-rebalancing)?

中: CoW AMM 如何解決 LVR（loss-versus-rebalancing）？

Answer (English)

- **LVR** = the structural loss LPs suffer because arbitrageurs always rebalance the pool against stale prices for free.
- Traditional AMM (Uniswap v2/v3): anyone can hit the pool the moment a CEX price moves — LP loses the spread.
- CoW AMM: only solvers in a batch auction may rebalance, and they must pay the LP the *winning surplus* — LVR is internalized as fee revenue.
- Net result: passive LPs in CoW AMM keep what previously leaked to MEV searchers.

答案（中文）

- **LVR**：套利者總能用過時的鏈上價格免費「再平衡」流動池，這對 LP 是結構性虧損。
- 傳統 AMM（Uniswap v2/v3）：CEX 價格一動，任何人都能立刻吃池子，LP 損失點差。
- CoW AMM：只有 batch auction 的 solver 才能再平衡，且必須把 surplus 還給 LP，LVR 變成池子的手續費收入。
- 結果：被動 LP 留下原本流向 MEV 搜尋者的價值。

Diagram

`cow-05-amm-vs-cowamm.svg`

Q023 · `intermediate` · `EIP-1271` `SIGNATURES`

EN: What are EIP-1271, presign, and EIP-712 — when do you use each?

中: EIP-1271、presign、EIP-712 各是什麼？分別何時使用？

Answer (English)

- **EIP-712**: standard typed-data signature for EOAs; default for end-user wallets.
- **EIP-1271**: smart contract wallets (Safe, Argent) implement `isValidSignature(hash, sig)` so they can sign without a private key.
- **Presign**: user calls `setPreSignature(orderUid, true)` on the settlement contract — useful for contracts/multisigs that cannot produce a 65-byte signature on the fly.

- Pick: EOA → EIP-712. Smart wallet that can sign hashes → EIP-1271. Multisig that approves orders one-by-one → presign.

答案（中文）

- **EIP-712**：EOA 標準化結構化簽章，一般使用者錢包預設用法。
- **EIP-1271**：智能合約錢包（Safe、Argent）實作 `isValidSignature(hash, sig)`，不需私鑰也能簽。
- **Presign**：使用者呼叫 settlement 合約的 `setPreSignature(orderUid, true)`，對沒辦法即時產生 65-byte 簽章的合約/多簽很有用。
- 選擇：EOA → EIP-712；可簽 hash 的智能錢包 → EIP-1271；逐單批准的多簽 → presign。

Diagram

`cow-03-signature-schemes.svg`

Q024 · `basic` · `COW` `ORDER-STATE`

EN: List the lifecycle states of a CoW order.

中: 請列出一筆 CoW 訂單的生命週期狀態。

Answer (English)

- `open` — accepted by orderbook, visible to solvers.
- `partially filled` — only for `partiallyFillable=true` orders.
- `fulfilled` — fully matched and settled on-chain.
- `cancelled` (off-chain) or `expired` (validTo passed without fill).

答案（中文）

- `open`：orderbook 已收單、開放給 solver。
- `partially filled`：僅 `partiallyFillable=true` 的訂單才會出現。
- `fulfilled`：完全配對並已上鏈結算。
- `cancelled`（鏈下取消）或 `expired`（過了 validTo 仍未成交）。

Diagram

`cow-12-order-state-machine.svg`

Q025 · `basic` · `JS` `HOISTING`

EN: Explain hoisting: var vs let/const vs function declarations.

中: 解釋 hoisting：var、let/const、function declaration 的差別。

Answer (English)

- `function` declarations are fully hoisted (name + body) — callable before their textual position.
- `var` is hoisted but initialised to `undefined`; reading before assignment yields `undefined`.
- `let` / `const` are hoisted but in the **temporal dead zone** — touching them before declaration throws `ReferenceError`.
- Best practice: declare close to use, prefer `const`, never rely on `var` hoisting.

答案（中文）

- `function` 宣告會完整 hoist（含名稱與函式體），程式碼前面也能呼叫。
- `var` 會被 hoist 但初始化為 `undefined`，賦值前讀到的是 `undefined`。
- `let` / `const` 也會被 hoist，但處於 **temporal dead zone**，宣告前存取會丟 `ReferenceError`。
- 最佳實務：盡量靠近使用處宣告、優先 `const`、不要依賴 `var` 的 hoisting 行為。

Q026 · `intermediate` · `EVM` `LIBRARY`

EN: viem vs ethers v6 — when would you choose each?

中: viem 與 ethers v6 — 你會在什麼情況選哪一個？

Answer (English)

- **viem**: tree-shakeable, type-safe ABI inference, `bigint` primitives, smaller bundle, modern API. Default for new TS projects.
- **ethers v6**: longer history, more docs/examples, broader plugin ecosystem (Web3Modal, Hardhat).
- API differences: viem uses `client.readContract({...})` style; ethers uses `contract.fooMethod(...)` proxies.
- For an SDK we publish, viem gives partners better DX; for legacy code that already imports ethers, stay consistent.

```
import { createPublicClient, http } from 'viem';
import { mainnet } from 'viem/chains';
const client = createPublicClient({ chain: mainnet, transport: http() });
const block = await client.getBlock();
```

答案（中文）

- viem：可 tree-shake、ABI 型別推導、`bigint` 原生、bundle 較小、API 較現代——新 TS 專案首選。
- ethers v6：歷史悠久、文件範例多、外掛生態廣（Web3Modal、Hardhat）。
- API 風格：viem 用 `client.readContract({...})`；ethers 用 `contract.fooMethod(...)` 代理。
- 我們發 SDK 給夥伴用，viem 對他們的 DX 更好；既有專案已用 ethers 就保持一致。


```
import { createPublicClient, http } from 'viem';
import { mainnet } from 'viem/chains';
const client = createPublicClient({ chain: mainnet, transport: http() });
const block = await client.getBlock();
```

Q027 · `intermediate` · `EVM` `NONCE`

EN: What pitfalls does nonce management create when sending many txs?

中: 高頻送交易時，nonce 管理有哪些常見陷阱？

Answer (English)

- Each EOA has a strictly increasing nonce — gaps stall every later tx until filled or replaced.
- Always fetch nonce with `pending` block tag, not `latest`, when chaining txs.
- For parallel sending, maintain a local counter and reconcile on error; never let two senders race the same EOA.
- Replace stuck txs with the same nonce + ≥10% higher gas (replace-by-fee).

```
let nonce = await client.getTransactionCount({ address: signer.address, blockTag: 'pending' });
for (const tx of batch) {
  await wallet.sendTransaction({ ...tx, nonce: nonce++ });
}
```

答案（中文）

- 每個 EOA 的 nonce 嚴格遞增，中間留洞會卡住所有後續交易，直到該洞被填或被取代。
- 連續送交易時 nonce 要用 `pending` 區塊取得，不要用 `latest`。

- 若多進程平行送，要本地計數並在錯誤時對齊；不要讓兩個發送端搶同一 EOA。
- 卡住的交易可用相同 nonce + 至少高 10% 的 gas 取代（replace-by-fee）。

```
let nonce = await client.getTransactionCount({ address: signer.address, blockTag: 'pending' });
for (const tx of batch) {
  await wallet.sendTransaction({ ...tx, nonce: nonce++ });
}
```

Q028 · `intermediate` · `API` `IDEMPOTENCY`

EN: How do you design an idempotent order placement endpoint?

中: 如何設計一個 idempotent 的下單 API？

Answer (English)

- Require an `Idempotency-Key` header (UUID generated by the client).
- Server stores `(key → orderUid, response, expiry)` in Redis for ≥24h.
- Same key + same body → return cached response. Same key + different body → 409.
- For CoW orders the natural key is the deterministic `orderUid` (hash of sig + signer + validTo) — replays just hit the same uid.

```
POST /api/v1/orders
Headers: { 'Idempotency-Key': 'b8e6...' }
// retried by network → server returns same orderUid
```

答案（中文）

- 要求 `Idempotency-Key` header（client 產生的 UUID）。
- Server 把 `(key → orderUid, response, expiry)` 存 Redis 至少 24 小時。
- 同 key + 同 body → 直接回快取；同 key + 不同 body → 回 409。
- CoW 訂單天然有 `orderUid`（簽章 + signer + validTo 的雜湊），重送會自然 dedupe。

```
POST /api/v1/orders
Headers: { 'Idempotency-Key': 'b8e6...' }
// 網路重送 → server 回相同 orderUid
```

Q029 · `intermediate` · `API` `RATE-LIMIT`

EN: How would you design rate limiting for a public DeFi API?

中: 你會如何替公開 DeFi API 設計速率限制？

Answer (English)

- Tiered quotas: anonymous (low) → API key (mid) → partner contract (high).
- Token-bucket per key + per IP; refill at fixed rate — handles bursts gracefully.
- Return `429` with `Retry-After`, `X-RateLimit-Remaining`, `X-RateLimit-Reset` headers so clients can self-throttle.
- Heavy endpoints (full orderbook, trades dump) get tighter limits than lookups; protect with caching CDN.

答案（中文）

- 分級配額：匿名（低）→ API Key（中）→ 合作合約（高）。
- Per-key + per-IP 的 token bucket，定速回填，可優雅吸收瞬時流量。
- 回 `429` 要附 `Retry-After`、`X-RateLimit-Remaining`、`X-RateLimit-Reset`，讓 client 自己節流。
- 重型端點（完整 orderbook、交易 dump）限制要更嚴，並擺一層 CDN 快取。

Q030 · `intermediate` · `API` `RETRY`

EN: Describe a robust retry/backoff strategy for HTTP calls.

中: 描述穩健的 HTTP 呼叫 retry/backoff 策略。

Answer (English)

- Retry only **idempotent or explicitly safe** calls (GET, PUT with idempotency key).
- Exponential backoff with jitter: `delay = base * 2^attempt + random(0, base)`.
- Honour `Retry-After` header; cap attempts (≤5); cap total wait (≤60s).
- Distinguish error classes: 5xx + timeout → retry; 4xx (other than 408/429) → fail fast.

```
async function withRetry<T>(fn: () => Promise<T>, max = 4) {
  for (let i = 0; i < max; i++) {
    try { return await fn(); }
    catch (e) {
      if (i === max - 1) throw e;
      const wait = 200 * 2 ** i + Math.random() * 200;
      await new Promise(r => setTimeout(r, wait));
    }
  }
  throw new Error('unreachable');
}
```

答案（中文）

- 只對**冪等或明確安全**的呼叫重試（GET、附 idempotency key 的 PUT）。
- 指數退避加 jitter：`delay = base * 2^attempt + random(0, base)`。
- 尊重 `Retry-After` header；最多重試 5 次；總等待 ≤ 60 秒。
- 錯誤分類：5xx + timeout → 重試；4xx（除了 408/429）→ 直接失敗。

```
async function withRetry<T>(fn: () => Promise<T>, max = 4) {
  for (let i = 0; i < max; i++) {
    try { return await fn(); }
    catch (e) {
      if (i === max - 1) throw e;
      const wait = 200 * 2 ** i + Math.random() * 200;
      await new Promise(r => setTimeout(r, wait));
    }
  }
  throw new Error('unreachable');
}
```

Q031 · `intermediate` · `WEBHOOK` `API`

EN: How do you build reliable webhook delivery with retries and signing?

中: 如何打造可靠且帶簽章的 webhook delivery + retry？

Answer (English)

- Sign every payload with HMAC-SHA256 of the body using a per-tenant secret; include `X-CoW-Signature` and `X-CoW-Timestamp` headers.
- Receivers verify signature **and** reject timestamps older than 5 min (replay protection).
- Persist outbound events; retry 5xx + timeouts with exponential backoff (e.g., 30s, 2min, 10min, 1h, 6h).
- Move to dead-letter queue after N attempts; expose a partner UI to replay manually.

答案（中文）

- 每筆 payload 用 per-tenant secret 做 HMAC-SHA256 簽章，附 `X-CoW-Signature`、`X-CoW-Timestamp` header。
- 接收端驗章後**還要**檢查 timestamp 是否在 5 分鐘內，避免重放攻擊。
- Outbound event 落地，遇 5xx/timeout 指數退避重試（例如 30s, 2min, 10min, 1h, 6h）。
- 超過 N 次進 DLQ，並提供 partner UI 手動重送。

Diagram

`cow-11-webhook-retry.svg`

Q032 · `intermediate` · `SDK` `PACKAGING`

EN: How do you ship an SDK that works in both ESM and CommonJS?

中: 怎麼讓 SDK 同時支援 ESM 與 CommonJS？

Answer (English)

- Use the `exports` field with conditional `import` / `require` paths and explicit `types` entry.
- Build twice: tsup/rollup outputting `dist/index.mjs` (ESM) and `dist/index.cjs` (CJS) with matching `.d.ts` / `.d.cts`.
- Avoid `__dirname` and dynamic `require` in source; both blow up in ESM.
- Run a smoke test in `node --input-type=module` and a separate `--input-type=commonjs` to catch regressions in CI.

```
{
  "exports": {
    ".": {
      "import": { "types": "./dist/index.d.mts", "default": "./dist/index.mjs" },
      "require": { "types": "./dist/index.d.cts", "default": "./dist/index.cjs" }
    }
  }
}
```

答案（中文）

- 在 `package.json` 的 `exports` 設條件式 `import` / `require` 路徑，並明確指定 `types`。
- Build 兩次：tsup/rollup 輸出 `dist/index.mjs`（ESM）與 `dist/index.cjs`（CJS），各自配對 `.d.ts` / `.d.cts`。
- 原始碼不要使用 `__dirname` 或動態 `require`，兩者在 ESM 都會炸。
- CI 跑兩種 smoke test：`node --input-type=module` 與 `--input-type=commonjs`，避免 regression。

```
{
  "exports": {
    ".": {
      "import": { "types": "./dist/index.d.mts", "default": "./dist/index.mjs" },
      "require": { "types": "./dist/index.d.cts", "default": "./dist/index.cjs" }
    }
  }
}
```

Q033 · `intermediate` · `API` `PAGINATION`

EN: Cursor vs offset pagination — which would you pick for a trades API?

中: Cursor 與 offset 分頁，你會替交易 API 選哪一個？

Answer (English)

- Cursor (keyset) pagination: stable when rows are appended, O(log n) lookup, no `LIMIT … OFFSET` table scan pain.
- Offset is fine for static datasets but breaks under concurrent inserts (skipped/duplicated rows) and is slow for deep pages.
- For trades (append-only, growing) → cursor on `(block_number, log_index)` with `next_cursor` returned.
- Always cap `limit` server-side and document it; reject `limit > 200`.

答案（中文）

- Cursor（keyset）分頁：新資料插入後仍穩定，查詢 O(log n)，避免 `LIMIT … OFFSET` 全表掃描。
- Offset 適合靜態資料集；併發新增時會跳行或重複，深分頁也很慢。
- 交易資料（append-only 且持續成長）→ 用 `(block_number, log_index)` 當 cursor，回傳 `next_cursor`。
- 一律在 server 端封頂 `limit` 並寫進文件，例如 `limit > 200` 直接拒絕。

Q034 · `intermediate` · `API` `VERSIONING`

EN: How do you version a public API without breaking partners?

中: 如何在不影響合作夥伴的情況下做 API 版本管理？

Answer (English)

- Version via URL prefix (`/api/v1/…`) — simple, easy to route, easy to deprecate.
- Additive changes never bump version (new optional fields, new endpoints).
- Breaking change → ship `/v2` in parallel, keep v1 read-only for ≥6 months with a deprecation header.
- Publish a changelog and email partners ≥30 days before any sunset; track active v1 callers via API keys.

答案（中文）

- 版本走 URL prefix（`/api/v1/…`）——簡單好路由、易於 deprecate。
- 新增式變更不需升版（新增可選欄位、新增端點）。
- Breaking change → 並行推 `/v2`，v1 至少維持唯讀 6 個月並回 deprecation header。
- 一定要先公告 changelog，並至少提前 30 天通知夥伴；以 API key 追蹤誰還在用 v1。

Q035 · `intermediate` · `API` `OPENAPI`

EN: What benefits do you get from a single OpenAPI spec?

中: 維護一份 OpenAPI 規格能帶來什麼好處？

Answer (English)

- Single source of truth → auto-generated SDKs (TS/Go/Python), Postman collection, and docs site.
- Server can validate requests/responses against the schema in tests, catching drift early.
- Partners can mock the API locally with Prism/Mockoon before integrating.
- Schema diffs in PRs reveal accidental breaking changes during review.

答案（中文）

- 單一真實來源 → 自動產 SDK（TS/Go/Python）、Postman collection、文件站。
- Server 端在測試中驗證 req/res，提早抓出漂移。
- 夥伴可用 Prism/Mockoon 在本地端 mock，整合前先打通。
- PR 中的 schema diff 能在 review 階段就抓到誤改的 breaking change。

Q036 · `intermediate` · `INTEGRATION` `SANDBOX`

EN: Why is a partner sandbox important and what should it include?

中: 為什麼合作夥伴需要 sandbox？sandbox 應包含什麼？

Answer (English)

- Lets partners iterate without spending real funds and without flooding production observability.
- Should mirror prod behaviour: same OpenAPI, same error codes, same rate limits, same webhook signing.

- Provide pre-funded testnet wallets, faucet, and seeded orderbook so demo flows work day-one.
- Auto-reset (e.g., daily) so leftover state from one partner does not affect another.

答案（中文）

- 夥伴可在不花真錢、也不會干擾正式環境的條件下迭代。
- 行為要對齊正式環境：相同 OpenAPI、錯誤碼、rate limit、webhook 簽章方式。
- 預先資助測試錢包、提供 faucet、種一些 orderbook 資料，第一天就能跑 demo。
- 定期自動 reset（例如每日）避免不同夥伴的殘留狀態互相干擾。

Q037 · `intermediate` · `DEBUGGING` `INTEGRATION`

EN: A partner says "my orders never get filled" — how do you debug?

中: 合作夥伴回報「我的訂單都不會成交」——你會怎麼除錯？

Answer (English)

- Step 1: ask for one **orderUid**; pull it from the orderbook to confirm it was accepted, status, validTo, sell/buy amounts.
- Step 2: check signer balance + allowance for `GPv2VaultRelayer` — common cause is missing approve.
- Step 3: check competition data (`/api/v1/solver_competition/{auctionId}`) — was the order included? Why was no solver willing?
- Step 4: simulate via Tenderly with the winning solver tx; reproduce locally.

答案（中文）

- 步驟 1：要一筆 **orderUid**，到 orderbook 查狀態、validTo、賣/買數量是否被收。
- 步驟 2：檢查 signer 餘額與對 `GPv2VaultRelayer` 的 allowance——最常見就是漏 approve。
- 步驟 3：看 `/api/v1/solver_competition/{auctionId}` 的競標資料：訂單有被涵蓋嗎？為何沒人願意成交？
- 步驟 4：在 Tenderly 用獲勝 solver 的交易做模擬，本地重現。

Q038 · `intermediate` · `OBSERVABILITY` `LOGS`

EN: How do you correlate logs across partner client → API → solver → chain?

中: 如何把 partner client → API → solver → 鏈上 的 log 串起來？

Answer (English)

- Generate a `request_id` (ULID) at the API edge; propagate via headers + structured logs.
- Persist `request_id ↔ orderUid ↔ tx_hash` in a join table — one query covers the full journey.
- Use OpenTelemetry traces so spans across HTTP / RPC / queue link automatically.
- Expose a `/internal/trace?orderUid=...` tool for support; saves hours when partners ping #integration.

答案（中文）

- 在 API 邊界產生 `request_id`（ULID），用 header + 結構化 log 一路傳下去。
- 存 `request_id ↔ orderUid ↔ tx_hash` 對照表，一條 query 就能拉出完整旅程。
- 採用 OpenTelemetry trace，HTTP / RPC / queue 跨服務 span 自動串接。
- 對內提供 `/internal/trace?orderUid=...` 工具，#integration 頻道收到問題能秒查。

Q039 · `intermediate` · `INTEGRATION` `PITFALLS`

EN: What are the most common partner mistakes when computing token amounts?

中: 合作夥伴在算代幣金額時最常犯哪些錯？

Answer (English)

- Treating amounts as floats — DAI has 18 decimals, USDC has 6, WBTC has 8; use `bigint` everywhere.
- Mixing units: passing whole-token instead of base units (1 vs 1_000_000 for USDC).
- Rounding errors when converting between sell/buy amounts at quote time — round in user's favour.
- Forgetting EIP-2612 `value` is in base units too, and that the deadline is a Unix timestamp not ms.

```
import { parseUnits, formatUnits } from 'viem';
const sellAmount = parseUnits('1000', 6); // 1000 USDC -> 1_000_000_000n
```

答案（中文）

- 把金額當 float 算——DAI 18 位、USDC 6 位、WBTC 8 位，全部用 `bigint`。
- 單位搞混：把整顆代幣傳成 base unit（USDC 1 vs 1_000_000）。
- Quote 時 sell/buy 換算的取整捨入錯誤——一律往使用者有利的方向取整。
- EIP-2612 的 `value` 也是 base unit，且 deadline 是 Unix 秒不是毫秒。

```
import { parseUnits, formatUnits } from 'viem';
const sellAmount = parseUnits('1000', 6); // 1000 USDC -> 1_000_000_000n
```

Q040 · `intermediate` · `INCIDENT` `TRIAGE`

EN: A partner pings #integration: "settlement failing for 30 min". How do you triage?

中: 夥伴在 #integration 喊「settlement 失敗 30 分鐘了」——你如何 triage？

Answer (English)

- Acknowledge in <2 min, ask for orderUid + chain + tx hash; confirm severity (some orders vs all).
- Check status page + Grafana: is it just this partner, this chain, or systemic? Pull recent solver competition data.
- If systemic, page on-call SRE; if partner-specific, pair on Tenderly to reproduce.
- Keep partner updated every 15 min even with "still investigating"; write post-mortem within 48h.

答案（中文）

- 2 分鐘內回應，要 orderUid、鏈別、tx hash，確認影響面（少數還是全部）。
- 看 status page + Grafana：只有這夥伴、這條鏈，還是系統性問題？拉最近 solver competition 資料。
- 系統性 → 叫 on-call SRE；夥伴特有 → 一起在 Tenderly 重現。
- 即使「仍在排查」也每 15 分鐘更新夥伴一次；48 小時內出 post-mortem。

Q041 · `basic` · `RUNBOOK` `OPS`

EN: What does a good runbook look like for an integration team?

中: 一份好的整合團隊 runbook 該長什麼樣？

Answer (English)

- Top: incident summary template, severity matrix, on-call rota link.
- Per-symptom playbook ("orders pending >10 min", "API 5xx spike"): exact commands, dashboards, decision tree.
- Communication templates (status page, partner email, internal Slack).
- Post-mortem checklist: timeline, impact, root cause, action items with owners + dates.

答案（中文）

- 頁首：事故摘要範本、嚴重度矩陣、on-call 輪值連結。
- 依症狀分節（「訂單 pending 超過 10 分」、「API 5xx 爆量」），含明確指令、儀表板連結、決策樹。
- 溝通範本（status page、夥伴郵件、內部 Slack）。
- Post-mortem 檢核清單：時間軸、影響、根因、含負責人與期限的 action items。

Q042 · `intermediate` · `SECURITY` `APPROVAL`

EN: What are the risks of infinite ERC-20 approvals?

中: 無限額度的 ERC-20 approve 有哪些風險？

Answer (English)

- If the spender contract is compromised or upgradeable to malicious code, all approved funds drain instantly.
- Phishing UX: looks identical to a one-shot approve, users rarely revoke afterward.
- Mitigations: per-trade approve, EIP-2612 permit (signed approval), Permit2 (transient + expiring approvals).
- Tools to revoke: `revoke.cash`, Etherscan token approvals page; advise partners to expose this in their UI.

答案（中文）

- Spender 合約被攻陷或可升級為惡意程式碼，所有授權資金瞬間被搬空。
- 釣魚 UX：看起來跟單次 approve 一樣，使用者事後幾乎不會主動 revoke。
- 緩解：單次 approve、EIP-2612 permit（簽章授權）、Permit2（短暫 + 可過期授權）。
- 撤銷工具：`revoke.cash`、Etherscan 授權頁；建議夥伴在自家 UI 也提供這功能。

Q043 · `advanced` · `PERMIT2` `SECURITY`

EN: How does Uniswap Permit2 differ from EIP-2612 permit?

中: Uniswap Permit2 與 EIP-2612 permit 有什麼不同？

Answer (English)

- EIP-2612 needs the **token contract itself** to implement `permit(...)` — many older tokens (USDC pre-bridge, WBTC) don't.

- Permit2 is a **shared singleton contract** holding allowances for any ERC-20; one-time approval to Permit2, then per-spender signed approvals.
- Supports batched, expiring, and signature-transfer flows — gasless approvals across all tokens.
- Risk: trust shifts to Permit2 contract — single point of failure if exploited (audited heavily).

答案（中文）

- EIP-2612 需要**代幣合約本身**實作 `permit(...)`，許多舊代幣（如舊版 USDC、WBTC）並不支援。
- Permit2 是**共用單例合約**，替任何 ERC-20 持有 allowance；對 Permit2 一次性 approve，再用簽章對個別 spender 授權。
- 支援批次、過期、signature-transfer 流程——所有代幣都能無 gas 授權。
- 風險：信任集中到 Permit2 合約，一旦被攻破就是 single point of failure（已經經過大量 audit）。

Diagram

`cow-07-approve-vs-permit2.svg`

Q044 · `intermediate` · `MEV` `ATTACKS`

EN: Walk me through a sandwich attack on a Uniswap swap.

中: 示範一次 Uniswap swap 上的 sandwich 攻擊。

Answer (English)

- Searcher sees victim's pending swap (sell ETH for DAI, large size, high slippage tolerance) in the public mempool.
- 1) Front-run: searcher swaps ETH→DAI first, pushing price up.
- 2) Victim swap executes at the worse price, paying the manipulated price within their slippage tolerance.
- 3) Back-run: searcher swaps DAI→ETH, pocketing the spread; victim subsidises the profit via worse execution.

答案（中文）

- 搜尋者在公開 mempool 看到受害者待打包的大額 ETH→DAI swap，且滑價容忍度高。
- 1）Front-run：搜尋者搶先做 ETH→DAI，把價格推高。
- 2）受害者 swap 以更差價格執行，仍落在自己的滑價容忍內。

- 3）Back-run：搜尋者再做 DAI→ETH 把價差吃回，受害者實質補貼了這筆利潤。

Q045 · `advanced` · `DEFI` `ORACLES`

EN: What is oracle manipulation and how do good protocols defend against it?

中: Oracle manipulation 是什麼？穩健的協定如何防範？

Answer (English)

- If a protocol prices assets via a single AMM spot price, an attacker can flash-loan-push the price for one block to liquidate or drain.
- Defences: use **TWAP** (time-weighted average over N blocks) — costs attacker many blocks of capital lock-up.
- Use multiple independent oracles (Chainlink + UniV3 TWAP + Pyth) and reject outliers.
- Cap update slope (e.g., max 10% price move per update) and enforce minimum liquidity thresholds.

答案（中文）

- 若協定僅用單一 AMM 即時價，攻擊者可閃電貸推價一個區塊就清算或抽乾池子。
- 防禦：用 **TWAP**（N 區塊時間加權平均價），攻擊者必須鎖住資本多個區塊。
- 混用多個獨立 oracle（Chainlink + UniV3 TWAP + Pyth），剔除離群值。
- 限制每次更新的最大斜率（如 10%）並設定最低流動性門檻。

Q046 · `intermediate` · `SECURITY` `SIGNATURES`

EN: What is signature malleability and how does Ethereum mitigate it?

中: Signature malleability 是什麼？Ethereum 如何緩解？

Answer (English)

- ECDSA signatures `(r, s, v)` have two valid `s` values: `s` and `n-s` — both verify the same message.
- Pre-2017 contracts using sig as a uniqueness key could be replayed with the alternate form.
- EIP-2 (Homestead) requires `s ≤ n/2` (low-s); higher `s` is rejected by clients and `ecrecover`-using contracts.
- OpenZeppelin's `ECDSA.recover` enforces this; libraries like viem produce low-s by default.

答案（中文）

- ECDSA 簽章 `(r, s, v)` 有兩個合法的 `s`：`s` 與 `n-s`，兩者都能驗章成功。
- 2017 年前若有合約把簽章本身當唯一鍵，攻擊者可改用另一型重放。
- EIP-2（Homestead）強制 `s ≤ n/2`（low-s），用 `ecrecover` 的合約會拒收高 `s`。
- OpenZeppelin 的 `ECDSA.recover` 強制此規則；viem 等函式庫預設產生 low-s 簽章。

Q047 · `basic` · `SECURITY` `REPLAY`

EN: What replay protection mechanisms does Ethereum provide?

中: Ethereum 內建哪些 replay protection 機制？

Answer (English)

- Tx nonce — each EOA tx has a strictly increasing nonce; same tx cannot be re-mined.
- EIP-155 chainId — included in the signed tx hash, prevents cross-chain replay (mainnet vs Goerli).
- EIP-712 domain separator — binds signed message to (name, version, chainId, verifyingContract).
- CoW orders include `validTo` + signer nonce inside the order struct — orderbook rejects duplicates.

答案（中文）

- Tx nonce：每個 EOA 的 nonce 嚴格遞增，同一筆交易不能再被打包。
- EIP-155 chainId：寫進簽章 hash，防止跨鏈重放（主網 vs Goerli）。
- EIP-712 domain separator：把簽章與 (name, version, chainId, verifyingContract) 綁定。
- CoW 訂單在結構內含 `validTo` + signer nonce，orderbook 會拒絕重複。

Q048 · `basic` · `SOLIDITY` `SECURITY`

EN: Explain reentrancy and the checks-effects-interactions pattern.

中: 解釋 reentrancy 與 checks-effects-interactions pattern。

Answer (English)

- Reentrancy: external call (e.g., `.call{value}`) lets the callee re-enter your function before state updates finish — the DAO hack pattern.
- CEI: 1) **check** preconditions, 2) **update state** (effects), 3) **call** external contracts last.

- Defense in depth: also use OpenZeppelin's `nonReentrant` modifier (mutex flag).
- Modern Solidity 0.8+ has overflow checks but reentrancy still requires careful ordering.

```
function withdraw() external nonReentrant {
    uint256 bal = balances[msg.sender];
    require(bal > 0);
    balances[msg.sender] = 0;          // effects FIRST
    (bool ok,) = msg.sender.call{value: bal}('');
    require(ok);
}
```

答案（中文）

- Reentrancy：對外呼叫（如 `.call{value}`）會讓被呼叫者在你函式狀態尚未更新前再次進入——這就是 DAO Hack。
- CEI 原則：1）**check** 前置條件，2）先**更新狀態**（effects），3）最後再**呼叫**外部合約。
- 雙保險：再加上 OpenZeppelin 的 `nonReentrant` modifier（mutex 旗標）。
- Solidity 0.8+ 已有 overflow 檢查，但 reentrancy 仍要靠順序與保護。

```
function withdraw() external nonReentrant {
    uint256 bal = balances[msg.sender];
    require(bal > 0);
    balances[msg.sender] = 0;          // effects 先
    (bool ok,) = msg.sender.call{value: bal}('');
    require(ok);
}
```

Q049 · `intermediate` · `MEV` `FLASHBOTS`

EN: How do private mempools / Flashbots bundles work?

中: 私有 mempool / Flashbots bundle 怎麼運作？

Answer (English)

- Searcher submits a *bundle* (ordered list of txs) directly to a builder via `eth_sendBundle` — never enters public mempool.
- Builder includes the whole bundle atomically or not at all; searcher pays for inclusion through priority tip / coinbase.transfer.
- Used for arbitrage, liquidations, and now MEV protection (MEV Blocker uses bundles to isolate users from front-runners).
- Trade-off: privacy vs censorship resistance; needs trust in the builder set.

答案（中文）

- 搜尋者把 *bundle*（有序的交易清單）以 `eth_sendBundle` 直接送給 builder，不進公開 mempool。
- Builder 要嘛整批包進區塊、要嘛不打包；搜尋者透過 priority tip 或 `coinbase.transfer` 付費。
- 用途：套利、清算，現在還用於 MEV 保護（MEV Blocker 就靠 bundle 把使用者和 front-runner 隔開）。
- 權衡：隱私換 censorship resistance，必須信任 builder 集合。

Q050 · `basic` · `COMMS` `MEV`

EN: How would you explain MEV to a non-technical PM in 60 seconds?

中: 你會如何用 60 秒對非技術 PM 解釋 MEV？

Answer (English)

- Analogy: imagine ordering a stock trade and the broker secretly buying first, selling to you at a markup, then exiting. That hidden tax is MEV.
- On Ethereum it happens because validators choose tx order; bots scan pending tx and exploit any predictable slippage.
- CoW Protocol's batch auctions and MEV Blocker's private orderflow remove that predictability — users keep the value.
- Business framing: this is *invisible cost* shaved off every public swap; we sell predictable best-execution.

答案（中文）

- 比喻：你下股票買單，券商偷偷先買進、加價賣給你、再賣掉——這個隱形稅就是 MEV。
- Ethereum 上會發生是因為驗證者決定交易順序，機器人掃 pending tx 利用可預期的滑價。
- CoW 的 batch auction 與 MEV Blocker 的私有 orderflow 把可預期性消掉——價值留給使用者。
- 商業說法：這是每筆公開 swap 上被偷走的「隱形成本」，我們賣的是可預期的最佳執行。

Q051 · `basic` · `COMMS` `BUG-REPORT`

EN: What makes a great bug report from a partner?

中: 一份好的合作夥伴 bug report 該包含什麼？

Answer (English)

- One sentence summary + expected vs actual behaviour.
- Reproduction steps (or a curl/script that triggers it deterministically).
- Identifiers: orderUid, tx hash, chainId, timestamp (UTC), API response body.
- Impact (single user vs systemic) and any workaround already in place.

答案（中文）

- 一句話摘要 + 期望 vs 實際行為。
- 重現步驟（或可確定性觸發的 curl / script）。
- 識別碼：orderUid、tx hash、chainId、時間戳（UTC）、API 回應 body。
- 影響範圍（單一使用者 vs 系統性）以及已有的暫時 workaround。

Q052 · `intermediate` · `COMMS` `SALES`

EN: How do you structure a technical sales call as the engineer in the room?

中: 在 technical sales call 中作為工程師代表，你怎麼安排流程？

Answer (English)

- 5 min — listen: capture their use case, current stack, success metric. Don't pitch yet.
- 15 min — show: live demo or whiteboard exactly the integration touch-points relevant to them.
- 10 min — co-design: sketch the data flow, identify gaps (e.g., they need partial-fill UX we don't expose yet).
- End with: written next steps owner+date, calendar invite for follow-up, sandbox keys in their inbox before they leave the call.

答案（中文）

- 前 5 分鐘——聆聽：弄清使用案例、現有技術棧、成功指標，先別推銷。
- 中段 15 分鐘——展示：用 live demo 或白板畫出與他們相關的整合接點。
- 再 10 分鐘——共同設計：把資料流畫出，找出落差（例如他們要 partial-fill UX 但我們未開放）。
- 結尾：寫下含負責人與期限的 next steps、約下次會議、夥伴離開會議前 sandbox 金鑰已寄到信箱。

Q053 · `intermediate` · `COMMS` `PRODUCT`

EN: How do you turn partner feedback into roadmap influence?

中: 你怎麼把合作夥伴的 feedback 轉化為對 roadmap 的影響力？

Answer (English)

- Tag every partner ask in a shared CRM/Linear with use case + dollar/volume impact + workaround cost.
- Aggregate monthly: top 3 themes by weighted impact go into the product team's planning.
- Bring receipts: not "Acme wants TWAP", but "5 partners covering $X volume need TWAP, current workaround costs Y hours/week".
- Close the loop: once shipped, tell partners by name and ask them to validate.

答案（中文）

- 在共用 CRM/Linear 標註每個夥伴需求：使用案例、金額/量影響、現行 workaround 成本。
- 每月彙整：依加權影響挑前 3 大主題進產品團隊規劃。
- 帶證據：不是「Acme 要 TWAP」，而是「5 個夥伴占 $X 量需要 TWAP，現行 workaround 每週耗 Y 小時」。
- 回饋閉環：上線後具名通知夥伴並請他們驗證。

Q054 · `intermediate` · `DOCS` `TAXONOMY`

EN: How do you organise developer documentation for an SDK + protocol?

中: 面向 SDK + 協定的開發者文件，你會如何組織？

Answer (English)

- Diátaxis four quadrants: **Tutorials** (learning), **How-to guides** (task-oriented), **Reference** (look-up), **Explanations** (background).
- Tutorials: a 15-min "first order" walkthrough with copy-pasteable code that always runs.
- How-tos: discrete recipes ("integrate Permit2", "handle order cancellation") indexed by partner question patterns.
- Reference: auto-generated from OpenAPI + TypeDoc — never hand-write what a tool can produce.

答案（中文）

- Diátaxis 四象限：**Tutorials**（學習）、**How-to guides**（任務導向）、**Reference**（查表）、**Explanations**（背景知識）。
- Tutorials：一份 15 分鐘「第一筆訂單」走完，code 直接貼到能跑。
- How-tos：依夥伴常問問題分類的單一 recipe（如「整合 Permit2」、「處理訂單取消」）。
- Reference：從 OpenAPI + TypeDoc 自動產，能自動就不要手寫。

Q055 · `intermediate` · `DOCS` `PROCESS`

EN: How do you prevent documentation rot?

中: 怎麼預防文件腐朽（doc rot）？

Answer (English)

- Treat docs as code: live in the same repo, reviewed in PRs, broken on intentional API changes.
- Run docs code samples in CI — if `pnpm test:docs` fails, the PR fails.
- Add `last_verified` dates and a quarterly review rotation; archive (don't delete) outdated pages with a banner.
- Track usage: if a page has 0 views in 6 months, audit; if it has >1k views and high search-exit rate, fix.

答案（中文）

- 把文件當程式碼：與原始碼同 repo、PR review、API 改動會 break。
- 在 CI 跑文件中的範例 code，`pnpm test:docs` 失敗就擋 PR。
- 加 `last_verified` 日期，每季輪流人工 review；過時頁面**封存**並加 banner，不要直接刪。
- 追蹤流量：6 個月 0 view 要審視；高流量但搜尋後立刻離開要修。

Q056 · `basic` · `REMOTE` `PRACTICES`

EN: What async-first remote work practices keep an integration team productive?

中: 哪些 async-first 的遠距工作實務能讓整合團隊保持高效？

Answer (English)

- Decisions captured in writing (ADRs / Notion / GitHub issues) — not in Zoom calls only.
- Default to public Slack channels; DMs are for confidential or social.

- Working hours overlap window of ~3h across timezones; outside that, sync is opt-in not expected.
- Loom/recorded demos for partner walk-throughs — they can replay without booking time.

答案（中文）

- 決策寫下來（ADR / Notion / GitHub issue），不要只存在 Zoom 通話裡。
- 預設使用公開 Slack 頻道，DM 留給機密或社交。
- 跨時區安排約 3 小時的核心重疊時段；其餘時間同步是選擇題不是必修。
- 夥伴 walk-through 用 Loom/錄影 demo——他們可隨時重播不必預約。

Q057 · `basic` · `SOLIDITY` `BASICS`

EN: Explain Solidity function visibility: public, external, internal, private.

中: 解釋 Solidity 函式可見性：public、external、internal、private。

Answer (English)

- `public` — callable from anywhere (other contracts, EOAs, internal); auto-generates getters for state vars.
- `external` — only callable from outside via tx/ `this.`. Cheaper for large calldata (no copy to memory).
- `internal` — callable from inside the contract and derived contracts; no external access.
- `private` — only the same contract; subclasses cannot call (still visible on-chain, not real privacy).

答案（中文）

- `public`：任何地方都能呼叫（其他合約、EOA、內部），state 變數還會自動產 getter。
- `external`：只能從外部以交易或 `this.` 呼叫；calldata 大時較省 gas（不複製到 memory）。
- `internal`：自身合約與繼承合約可呼叫，外部不可。
- `private`：只有自身合約可呼叫，子合約也不行（仍寫在鏈上，不算真正隱私）。

Q058 · `intermediate` · `SOLIDITY` `STORAGE`

EN: How is Solidity storage laid out and why does packing matter?

中: Solidity storage 如何配置？為什麼打包（packing）重要？

Answer (English)

- Storage is a 256-bit-word keyed map; each state variable occupies one or more slots in declaration order.
- Variables ≤32 bytes can be **packed** into the same slot if declared consecutively — saves SSTORE gas dramatically.
- Mappings/dynamic arrays compute slot via `keccak256(key . slot)`; not packed.
- Bad ordering (`uint256, uint128, uint128`) = 3 slots; good ordering (`uint128, uint128, uint256`) = 2 slots.

答案（中文）

- Storage 是 256-bit 字長的 map，state 變數依宣告順序占一個或多個 slot。
- 小於等於 32 bytes 的變數連續宣告可**打包**進同一 slot，大幅省 SSTORE gas。
- Mapping/動態陣列以 `keccak256(key . slot)` 算位置，不打包。
- 錯誤順序（`uint256, uint128, uint128`）= 3 slot；正確順序（`uint128, uint128, uint256`）= 2 slot。

Q059 · `intermediate` · `RUST` `BASICS`

EN: Explain Rust ownership and borrowing in two minutes.

中: 用兩分鐘解釋 Rust 的 ownership 與 borrowing。

Answer (English)

- Every value has exactly one **owner**; when the owner goes out of scope, the value is dropped.
- You can **move** ownership (default for non-`Copy` types) or **borrow** (`&T` shared, `&mut T` exclusive).
- Rule: at any time, either many `&T` *or* one `&mut T` — never both. Enforced at compile time.
- Eliminates whole classes of bugs (use-after-free, data races) without GC overhead.

```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() { x } else { y }
}
```

答案（中文）

- 每個值只有一個**擁有者**，擁有者離開作用域就會被 drop。
- 可以**移轉所有權**（非 `Copy` 型別預設行為），或**借用**（`&T` 共享、`&mut T` 獨占）。

- 規則：同時要嘛多個 `&T`、要嘛一個 `&mut T`，二者擇一，編譯期就檢查。
- 不靠 GC 就能消除一整類 bug（use-after-free、data race）。

```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() { x } else { y }
}
```

Q060 · `basic` · `PYTHON` `WEB3`

EN: Show a minimal web3.py snippet to read an ERC-20 balance.

中: 示範用 web3.py 最小程式碼讀 ERC-20 餘額。

Answer (English)

- Install: `pip install web3`. Use a JSON-RPC URL (Alchemy/Infura/public).
- Load token via `w3.eth.contract(address, abi=ERC20_ABI)`.
- Call `balanceOf` (read-only, free).
- Convert from base units using `decimals()` for display.

```
from web3 import Web3
w3 = Web3(Web3.HTTPProvider('https://eth.llamarpc.com'))
token = w3.eth.contract(address='0xA0b8...eB48', abi=ERC20_ABI)
bal = token.functions.balanceOf('0xUSER...').call()
print(bal / 10 ** token.functions.decimals().call())
```

答案（中文）

- 安裝：`pip install web3`，用 JSON-RPC URL（Alchemy/Infura/公共節點）。
- `w3.eth.contract(address, abi=ERC20_ABI)` 載入代幣合約。
- 呼叫 `balanceOf`（唯讀、免費）。
- 取得 `decimals()` 後再除以做人類可讀格式。

```
from web3 import Web3
w3 = Web3(Web3.HTTPProvider('https://eth.llamarpc.com'))
token = w3.eth.contract(address='0xA0b8...eB48', abi=ERC20_ABI)
bal = token.functions.balanceOf('0xUSER...').call()
print(bal / 10 ** token.functions.decimals().call())
```

Q061 · `intermediate` · `DUNE` `ANALYTICS`

EN: How would you measure CoW Swap volume share on Ethereum using Dune?

中: 你會如何用 Dune 測量 CoW Swap 在 Ethereum 上的成交量市占？

Answer (English)

- Use the spellbook table `dex.trades` — it normalises swaps across DEXes and aggregators.
- Filter `project = 'cow_protocol'` for CoW volume; aggregate by `date_trunc('day', block_time)`.
- Compare against total DEX volume for the same period to get share.
- Watch for double-counting: aggregator trades may also appear under the underlying AMM — use `aggregator_name IS NULL` for raw AMM.

```
select date_trunc('day', block_time) as d,
       sum(amount_usd) as vol
from dex.trades
where blockchain = 'ethereum'
  and project = 'cow_protocol'
  and block_time >= now() - interval '30' day
group by 1
order by 1;
```

答案（中文）

- 用 spellbook 的 `dex.trades` 表，它已經把各 DEX 與 aggregator 的 swap 正規化。
- `project = 'cow_protocol'` 篩出 CoW 成交，依 `date_trunc('day', block_time)` 彙總。
- 同期間總 DEX volume 相除得到市占。
- 注意重複計算：aggregator 交易可能也記在底層 AMM，計算純 AMM 量時加 `aggregator_name IS NULL`。

```
select date_trunc('day', block_time) as d,
       sum(amount_usd) as vol
from dex.trades
where blockchain = 'ethereum'
  and project = 'cow_protocol'
  and block_time >= now() - interval '30' day
group by 1
order by 1;
```

Q062 · `intermediate` · `PYTHON` `PANDAS`

EN: Sketch a pandas pipeline to bucket DEX trades by hour and compute median size.

中: 描述一個 pandas 管線：把 DEX 成交按小時分桶並計算中位數。

Answer (English)

- Read CSV/Parquet → ensure `block_time` is `datetime64[ns, UTC]`.
- Set as index, then `df.resample('1H')` (or `groupby(pd.Grouper(freq='1H'))`).
- Aggregate with `.agg(median_size=('amount_usd', 'median'), n=('tx_hash', 'count'))`.
- Cache intermediate results in Parquet — avoid re-pulling raw RPC for every notebook iteration.

```
import pandas as pd
df = pd.read_parquet('trades.parquet').sort_values('block_time')
df['block_time'] = pd.to_datetime(df['block_time'], utc=True)
out = (df.set_index('block_time')
         .resample('1H')
         .agg(median_size=('amount_usd', 'median'),
              trade_count=('tx_hash', 'count')))
print(out.tail())
```

答案（中文）

- 讀 CSV/Parquet，確認 `block_time` 是 `datetime64[ns, UTC]`。
- 設為 index 後 `df.resample('1H')`（或 `groupby(pd.Grouper(freq='1H'))`）。
- `.agg(median_size=('amount_usd', 'median'), n=('tx_hash', 'count'))`。
- 中間結果存 Parquet 快取，避免每次跑 notebook 都重抓 RPC。

```
import pandas as pd
df = pd.read_parquet('trades.parquet').sort_values('block_time')
df['block_time'] = pd.to_datetime(df['block_time'], utc=True)
out = (df.set_index('block_time')
         .resample('1H')
         .agg(median_size=('amount_usd', 'median'),
              trade_count=('tx_hash', 'count')))
print(out.tail())
```

Q063 · `intermediate` · `EVM` `LOGS`

EN: How do event logs work — topics vs data?

中: Event log 怎麼運作？topic 與 data 有什麼差別？

Answer (English)

- A log has up to 4 `topics` (32-byte each) + a `data` blob.
- `topics[0]` is the event signature hash `keccak256("Transfer(address,address,uint256)")`.
- Indexed parameters → topics (cheap to filter via `eth_getLogs`); non-indexed → ABI-encoded in `data`.
- Use `viem` / `ethers` to decode logs back to typed args; don't hand-roll.

```
const logs = await client.getLogs({
  address: USDC,
  event: parseAbiItem('event Transfer(address indexed from, address indexed to, uint256 value)'),
  fromBlock: 18_000_000n,
});
```

答案（中文）

- 一筆 log 最多 4 個 `topic`（每個 32 byte）+ 一個 `data` blob。
- `topics[0]` 是事件簽章 hash `keccak256("Transfer(address,address,uint256)")`。
- `indexed` 參數放 topic（方便 `eth_getLogs` 過濾）；非 indexed 以 ABI 編碼放在 `data`。
- 用 `viem` / `ethers` decode 回 typed 參數，不要自己手刻。

```
const logs = await client.getLogs({
  address: USDC,
  event: parseAbiItem('event Transfer(address indexed from, address indexed to, uint256 value)'),
  fromBlock: 18_000_000n,
});
```

Q064 · `intermediate` · `EVM` `ABI`

EN: What is ABI encoding and how is calldata constructed?

中: ABI encoding 是什麼？calldata 怎麼組？

Answer (English)

- ABI = Application Binary Interface: deterministic rules for serialising function calls + return values into bytes.
- Calldata layout: 4-byte selector (`keccak256("foo(uint256,address)")[:4]`) + 32-byte aligned arguments.
- Dynamic types (string, bytes, arrays) use offsets pointing to lengths + data tail.
- Use viem `encodeFunctionData` / ethers `Interface.encodeFunctionData`; manual encoding is bug-prone.

```
import { encodeFunctionData, parseAbi } from 'viem';
const data = encodeFunctionData({
  abi: parseAbi(['function transfer(address,uint256)']),
  functionName: 'transfer',
  args: ['0xRecipient', 1_000_000n],
});
```

答案（中文）

- ABI = Application Binary Interface：把函式呼叫與回傳序列化為位元組的固定規則。
- Calldata 結構：4-byte selector（`keccak256("foo(uint256,address)")[:4]`）+ 32-byte 對齊的參數。
- 動態型別（string、bytes、array）用偏移量指向 length + data 尾段。
- 用 viem `encodeFunctionData` 或 ethers `Interface.encodeFunctionData`，手刻容易出錯。


```
import { encodeFunctionData, parseAbi } from 'viem';
const data = encodeFunctionData({
  abi: parseAbi(['function transfer(address,uint256)']),
  functionName: 'transfer',
  args: ['0xRecipient', 1_000_000n],
});
```

Q065 · `intermediate` · `TS` `GENERICS`

EN: Why are generics useful — show a typed fetch helper.

中: Generic 的好處在哪？示範一個 typed fetch helper。

Answer (English)

- Generics let one function/type work for many shapes while preserving type information at the call site.
- Inference does the heavy lifting — callers rarely need explicit type args.
- Constraints (`<T extends Foo>`) ensure operations stay valid for the parameter.
- Common in API clients: response type is inferred from the URL or schema.

```
async function getJSON<T>(url: string): Promise<T> {
  const r = await fetch(url);
  if (!r.ok) throw new Error(`${r.status}`);
  return r.json() as Promise<T>;
}
const order = await getJSON<Order>(`/orders/${uid}`);
```

答案（中文）

- Generic 讓同一個函式/型別能適用多種形狀，又能在呼叫端保留型別資訊。
- 型別推導大多自動，呼叫者很少需要寫 explicit 型別。
- 可加 `<T extends Foo>` constraint 限制操作仍合法。
- API client 常用：回傳型別由 URL 或 schema 推導。

```
async function getJSON<T>(url: string): Promise<T> {
  const r = await fetch(url);
  if (!r.ok) throw new Error(`${r.status}`);
  return r.json() as Promise<T>;
}
const order = await getJSON<Order>(`/orders/${uid}`);
```

Q066 · `intermediate` · `TS` `DECORATORS`

EN: When are TypeScript decorators useful and what changed in TS 5.0?

中: TypeScript decorator 何時派上用場？TS 5.0 改了什麼？

Answer (English)

- Decorators wrap classes, methods, fields, and accessors with reusable behaviour (logging, validation, DI).
- TS 5.0 ships **stage-3 decorators** matching TC39 spec — different signatures from old experimental ones.
- No more `experimentalDecorators` flag for stage-3; old NestJS/TypeORM code still uses the legacy form for now.
- Avoid heavy decorator usage in libs you ship — many bundlers still struggle with TC39 decorators downstream.

```
function logged(target: any, ctx: ClassMethodDecoratorContext) {
  return function (this: any, ...args: any[]) {
    console.log(`call ${String(ctx.name)}`, args);
    return target.apply(this, args);
  };
}
class Api { @logged getOrder(uid: string) { /* ... */ } }
```

答案（中文）

- Decorator 可包裝 class、method、field、accessor，加入可重用行為（log、驗證、DI）。
- TS 5.0 推出**stage-3 decorator**，與 TC39 規範一致；簽章與舊實驗版不同。

- Stage-3 不需要 `experimentalDecorators`；舊 NestJS/TypeORM 目前仍用 legacy 形式。
- 對外發 library 時別重度使用，許多打包器對 TC39 decorator 支援還不穩。

```
function logged(target: any, ctx: ClassMethodDecoratorContext) {
  return function (this: any, ...args: any[]) {
    console.log(`call ${String(ctx.name)}`, args);
    return target.apply(this, args);
  };
}
class Api { @logged getOrder(uid: string) { /* ... */ } }
```

Q067 · `basic` · `JS MODULES`

EN: ESM vs CommonJS — what should I know in 2026?

中: ESM vs CommonJS — 2026 年我該知道哪些？

Answer (English)

- ESM is the standard: `import` / `export`, async by default, tree-shakeable, supported in all modern Node + browsers.
- CJS (`require` / `module.exports`) is sync, still common in legacy npm packages and tooling configs (`postcss.config.js`).
- Interop pain: ESM can `import` CJS, but CJS cannot easily `require` ESM (use dynamic `import()` or top-level await).
- For new SDKs publish dual ESM/CJS via `exports` field; for apps go ESM-only.

答案（中文）

- ESM 是標準：`import` / `export`、預設 async、可 tree-shake，現代 Node 與瀏覽器都支援。
- CJS（`require` / `module.exports`）同步，仍常見於舊 npm 套件與工具設定檔（如 `postcss.config.js`）。
- Interop 痛點：ESM 可以 import CJS，但 CJS 很難直接 require ESM（要用動態 `import()` 或 top-level await）。
- 新發 SDK 用 `exports` 欄位提供 dual ESM/CJS；應用程式建議 ESM-only。

Q068 · `basic` · `TS CONFIG`

EN: Which tsconfig flags would you turn on for a new TypeScript project?

中: 新 TypeScript 專案你會打開哪些 tsconfig flag？

Answer (English)

- `strict: true` — bundles `noImplicitAny`, `strictNullChecks`, `strictFunctionTypes`, etc. Non-negotiable.
- `noUncheckedIndexedAccess: true` — surfaces `T | undefined` for `arr[i]` / `obj[key]`, catches a class of nullable bugs.
- `exactOptionalPropertyTypes: true` — distinguishes `{x?: T}` from `{x: T | undefined}`.
- `moduleResolution: "Bundler"` for new projects with Vite/Next; `"NodeNext"` for Node libs.

答案（中文）

- `strict: true`：包含 `noImplicitAny`、`strictNullChecks`、`strictFunctionTypes` 等，沒得商量。
- `noUncheckedIndexedAccess: true`：讓 `arr[i]` / `obj[key]` 推導為 `T | undefined`，防一整類 bug。
- `exactOptionalPropertyTypes: true`：區分 `{x?: T}` 與 `{x: T | undefined}`。
- 搭 Vite/Next 用 `moduleResolution: "Bundler"`；發 Node 套件用 `"NodeNext"`。

Q069 · `intermediate` · `TS` `MONOREPO`

EN: How would you set up a TypeScript monorepo?

中: 你會如何設定一個 TypeScript monorepo？

Answer (English)

- pnpm workspaces (or yarn berry) — fast install + content-addressable store.
- Project references in `tsconfig.json` so each package builds independently and incrementally.
- Turbo or Nx for task orchestration (build, test, lint) with cache.
- Single shared `tsconfig.base.json` extended by every package; publish only `dist/`.

```
// pnpm-workspace.yaml
packages:
  - 'packages/*'
  - 'apps/*'

// tsconfig.base.json
{ "extends": "@tsconfig/node20", "compilerOptions": { "strict": true } }
```

答案（中文）

- pnpm workspace（或 yarn berry）—— 安裝快、儲存體共用。
- `tsconfig.json` 用 project references 讓每個 package 可獨立增量編譯。
- Turbo 或 Nx 做任務排程（build/test/lint）並啟用 cache。

- 共用一份 `tsconfig.base.json` 給所有 package extends；只發佈 `dist/`。

```
# pnpm-workspace.yaml
packages:
  - 'packages/*'
  - 'apps/*'

# tsconfig.base.json
# { "extends": "@tsconfig/node20", "compilerOptions": { "strict": true } }
```

Q070 · `intermediate` · `EVM` `MEMPOOL`

EN: What is the mempool and how does tx propagation work?

中: Mempool 是什麼？交易如何傳播？

Answer (English)

- Each node maintains a **local** pool of unconfirmed signed txs; nodes gossip new tx hashes to peers (devp2p / libp2p).
- There is no canonical mempool — different nodes/searchers see slightly different sets.
- Builders (post-merge) pick from their own view + private bundles to build a block.
- Public mempool exposes you to MEV; private orderflow (MEV Blocker, Flashbots Protect) hides intent until inclusion.

答案（中文）

- 每個節點維護**本地**未確認交易池，並透過 devp2p / libp2p 將新交易 hash gossip 給 peer。
- 沒有所謂「規範的」mempool——不同節點/搜尋者看到的集合略有差異。
- 合併後 builder 從自己的視角 + 私有 bundle 中挑單組區塊。
- 進公開 mempool 就會曝險 MEV；私有 orderflow（MEV Blocker、Flashbots Protect）能在打包前隱藏意圖。

Q071 · `advanced` · `MEV` `FLASHBOTS`

EN: How would you submit an arb bundle to Flashbots from Node.js?

中: 你會如何從 Node.js 把套利 bundle 送到 Flashbots？

Answer (English)

- Use `@flashbots/ethers-provider-bundle` (or call `eth_sendBundle` directly via Flashbots relay URL).
- Sign each tx with the searcher EOA, set `targetBlockNumber = currentBlock + 1`, simulate first with `callBundle`.
- Pay the builder via priority fee or `block.coinbase.transfer(...)` inside the bundle.
- Monitor inclusion via `getBundleStats`; on miss, escalate or switch to next block.

```
import { FlashbotsBundleProvider } from '@flashbots/ethers-provider-bundle';
const fb = await FlashbotsBundleProvider.create(provider, authSigner);
const signed = await fb.signBundle([{ signer, transaction }]);
const sim = await fb.simulate(signed, blockNumber + 1);
await fb.sendRawBundle(signed, blockNumber + 1);
```

答案（中文）

- 用 `@flashbots/ethers-provider-bundle`（或直接呼叫 Flashbots relay 的 `eth_sendBundle`）。
- 每筆 tx 用搜尋者 EOA 簽，設 `targetBlockNumber = currentBlock + 1`，先 `callBundle` 模擬。
- 付給 builder：用 priority fee 或在 bundle 內 `block.coinbase.transfer(...)`。
- 靠 `getBundleStats` 監看是否進塊；沒中就加價或往下個 block。

```
import { FlashbotsBundleProvider } from '@flashbots/ethers-provider-bundle';
const fb = await FlashbotsBundleProvider.create(provider, authSigner);
const signed = await fb.signBundle([{ signer, transaction }]);
const sim = await fb.simulate(signed, blockNumber + 1);
await fb.sendRawBundle(signed, blockNumber + 1);
```

Q072 · `intermediate` · `ERC-2612` `PERMIT`

EN: How does ERC-2612 permit save users gas?

中: ERC-2612 permit 怎麼幫使用者省 gas？

Answer (English)

- Replaces `approve()` tx (~46k gas) with an off-chain EIP-712 signature — free for the user.
- Signature is consumed inside the swap tx via `permit(owner, spender, value, deadline, v, r, s)`.
- Single-tx UX: instead of approve+swap = 2 txs, partner does swap-with-permit = 1 tx.
- Caveat: not all tokens implement permit; falls back to classic approve or Permit2.

```
// Off-chain
const sig = await signer.signTypedData(domain, PERMIT_TYPES, msg);
// On-chain (inside router)
IERC20Permit(token).permit(user, router, value, deadline, v, r, s);
IERC20(token).transferFrom(user, router, value);
```

答案（中文）

- 把 `approve()`（約 46k gas）換成鏈下 EIP-712 簽章，使用者免費。
- 簽章在 swap 交易內以 `permit(owner, spender, value, deadline, v, r, s)` 消費。
- UX 從 approve + swap 兩筆變成一筆 swap-with-permit。
- 注意：不是所有代幣都實作 permit，需 fallback 到傳統 approve 或 Permit2。

```
// 鏈下
const sig = await signer.signTypedData(domain, PERMIT_TYPES, msg);
// 鏈上（router 內）
IERC20Permit(token).permit(user, router, value, deadline, v, r, s);
IERC20(token).transferFrom(user, router, value);
```

Q073 · `intermediate` · `COW` `PARTIAL-FILL`

EN: When should an order be partially fillable?

中: 什麼時候應該設定訂單為 partially fillable？

Answer (English)

- Large orders (≥1% of pool depth) — solver may fill in chunks across multiple batches to minimise impact.
- TWAP-style execution where the user wants exposure to a time-averaged price.
- Liquidity-thin tokens where a single fill at quoted price is unlikely.
- Set `partiallyFillable: true` and a fee model that accepts proportional fills.

答案（中文）

- 大單（≥1% 池深）——solver 可在多個批次分塊成交以降低衝擊。
- TWAP 風格的執行：使用者想要時間平均價。
- 流動性稀薄的代幣，單筆按報價成交不太可能。
- 把 `partiallyFillable: true` 打開，並用接受比例成交的手續費模型。

Q074 · `intermediate` · `COW` `APPDATA`

EN: What is appData on a CoW order and why does it matter?

中: CoW 訂單上的 appData 是什麼？為何重要？

Answer (English)

- `appData` is a 32-byte hash committing to an off-chain JSON metadata document (referrer, hooks, slippageBps, version).
- It is part of the EIP-712 hash — once signed, the JSON cannot change without invalidating the signature.
- Used for revenue sharing (referrer/affiliate), partner attribution analytics, and CoW Hooks definitions.
- Best practice: pin the JSON to IPFS and put the CID hash in `appData`.

```
const appData = {
  appCode: 'AcmeWallet',
  metadata: { referrer: { address: '0xRef', version: '0.1.0' } },
  version: '1.1.0',
};
const hash = await api.uploadAppData(appData); // returns IPFS hash bytes32
```

答案（中文）

- `appData` 是 32-byte 雜湊，承諾一份鏈下 JSON metadata（referrer、hook、slippageBps、版本）。
- 它屬於 EIP-712 hash 的一部分，簽完後改 JSON 會讓簽章失效。
- 用於分潤（referrer/affiliate）、夥伴歸因分析、CoW Hook 定義。
- 建議：JSON 固定到 IPFS，把 CID 雜湊放進 `appData`。

```
const appData = {
  appCode: 'AcmeWallet',
  metadata: { referrer: { address: '0xRef', version: '0.1.0' } },
  version: '1.1.0',
};
const hash = await api.uploadAppData(appData); // 回傳 IPFS hash bytes32
```

Q075 · `basic` · `COW` `SLIPPAGE`

EN: How does slippage tolerance map onto a CoW order?

中: 滑價容忍度如何對應到 CoW 訂單？

Answer (English)

- For a `sell` order: `buyAmount` is set to `quotedBuy * (1 - slippageBps/10000)` — the minimum acceptable.
- For a `buy` order: `sellAmount` is set to `quotedSell * (1 + slippageBps/10000)` — the maximum to spend.
- A solver must respect these bounds; failing means the order won't be selected.
- Default 0.5% works for liquid pairs; raise for volatile or thin tokens.

答案（中文）

- `sell` 單：`buyAmount = quotedBuy * (1 - slippageBps/10000)`，最少能收到。
- `buy` 單：`sellAmount = quotedSell * (1 + slippageBps/10000)`，最多會付。
- Solver 必須符合這些上下限，否則該方案不會被選中。
- 流動好的對 0.5% 夠用；波動大或稀薄資產要拉高。

Q076 · `intermediate` · `COW` `ORDERBOOK`

EN: Why is the CoW orderbook off-chain — what trade-offs does that create?

中: CoW orderbook 為何放鏈下？這帶來哪些取捨？

Answer (English)

- Off-chain = free to place/cancel, fast iteration, no per-order gas; great UX.
- Trade-off: the orderbook server is a (replicated) trust point — must be highly available + open for solvers to read.
- Censorship vector mitigated by **anyone can run a solver** + on-chain settlement contract is permissionless.
- Open data: orderbook API and DB snapshots are public so partners can audit fairness.

答案（中文）

- 鏈下 = 下/撤單免費、迭代快、零 gas，UX 很好。
- 取捨：orderbook server 是（多副本）信任點，必須高可用且對 solver 開放。
- 審查風險靠**任何人可跑 solver** + 結算合約是 permissionless 來緩解。
- 資料公開：orderbook API 與資料庫快照公開，夥伴可稽核公平性。

Q077 · `intermediate` · `COW` `SURPLUS`

EN: What is "surplus" and how is it captured for the user?

中: "Surplus" 是什麼？如何把它還給使用者？

Answer (English)

- Surplus = the difference between what the user signed for (`buyAmount` minimum) and what they actually receive.
- When the solver finds a better path (peer match, AMM combo), the extra goes to the user, not the solver.
- On-chain: the settlement contract enforces `out >= signedMinimum` — surplus is just the excess transferred.
- CoW publishes per-trade surplus; competitive pressure keeps solvers passing it through.

答案（中文）

- Surplus = 使用者簽下的最低 `buyAmount` 與實際拿到金額的差額。
- Solver 找到更好的路徑（peer 配對、AMM 組合）時，多出的部分歸使用者而非 solver。
- 鏈上：結算合約強制 `out >= signedMinimum`，多出的就直接轉給使用者。
- CoW 公開每筆 surplus，競爭壓力會逼 solver 持續把 surplus 傳遞。

Q078 · `intermediate` · `COW` `SIGNATURES`

EN: When would a partner contract use presign instead of EIP-1271?

中: 什麼情況夥伴合約該用 presign 而不是 EIP-1271？

Answer (English)

- EIP-1271 requires implementing `isValidSignature(hash, sig)` — the contract decides validity per call.
- Presign requires the contract to send a tx calling `setPreSignature(orderUid, true)` once per order.
- Use presign when the contract cannot reason about EIP-712 hashes (older multisigs) but *can* approve concrete order UIDs after multi-sig vote.
- Use EIP-1271 when the contract can validate signatures programmatically (Safe modules, smart accounts).

答案（中文）

- EIP-1271 要實作 `isValidSignature(hash, sig)`，合約每次呼叫自行判斷是否合法。
- Presign 則要合約對每筆訂單送一次 `setPreSignature(orderUid, true)`。

- 當合約難以對 EIP-712 hash 做推論（舊多簽），但能在多簽通過後核准具體 orderUid，就用 presign。
- 當合約可程式化驗章（Safe module、智能帳戶）就用 EIP-1271。

Diagram

`cow-03-signature-schemes.svg`

Q079 · `advanced` · `COW` `SOLVERS`

EN: What is the solver's objective function — how do they decide which orders to include?

中: Solver 的目標函式是什麼？他們如何決定包含哪些訂單？

Answer (English)

- Maximise `Σ surplus_i` for orders included in the batch, subject to clearing prices and liquidity constraints.
- Cost side: gas to settle, slippage on AMM hops, capital risk if rebalancing across venues.
- Net reward = surplus delivered − gas − inventory risk; solver bids the proposed surplus to autopilot.
- NP-hard at scale; solvers use MILP, heuristics, or proprietary ML to find good solutions in seconds.

答案（中文）

- 在 clearing price 與流動性限制下，最大化批內訂單的 `Σ surplus_i`。
- 成本：結算 gas、AMM 跳價、跨場再平衡的資本風險。
- 淨報酬 = 交付 surplus − gas − 庫存風險；solver 把提案 surplus 投標給 autopilot。
- 規模上是 NP-hard，solver 會用 MILP、啟發式或自家 ML 在數秒內找到好解。

Q080 · `advanced` · `COW` `TWAP`

EN: How are TWAP orders implemented via programmatic orders / ComposableCoW?

中: TWAP 訂單如何透過 programmatic order / ComposableCoW 實作？

Answer (English)

- ComposableCoW is a contract that derives many child orders from one signed parent commitment.

- TWAP factory: parent specifies (sellToken, buyToken, totalAmount, n, t, t0, span); each `t`-second window emits a child order for `totalAmount/n`.
- Solver fetches the next valid child via `getTradeableOrder()`, builds it into the batch.
- Cancellable by the user via `removeOrder()` on the factory; survives indefinitely otherwise.

答案（中文）

- ComposableCoW 是一個合約，可從一筆簽過的父承諾衍生出多筆子訂單。
- TWAP factory：父單指定 (sellToken, buyToken, totalAmount, n, t, t0, span)，每 `t` 秒視窗釋出一筆 `totalAmount/n` 子單。
- Solver 透過 `getTradeableOrder()` 取下一筆可成交子單，包進批次。
- 使用者可呼叫 factory 的 `removeOrder()` 取消，否則會持續到結束。

Q081 · `intermediate` · `COW` `CHAINS`

EN: What chains does CoW deploy on and how do they differ?

中: CoW 部署在哪些鏈？彼此有什麼差異？

Answer (English)

- Ethereum mainnet, Gnosis Chain, Arbitrum, Base, plus testnets (Sepolia).
- Same `GPv2Settlement` contract bytecode + address (CREATE2 deterministic) on each chain.
- Solver set may differ per chain — partners should not assume identical liquidity routing.
- Gas economics differ wildly: mainnet batch worthwhile at $5-50 in fees; L2 batch can be sub-cent.

答案（中文）

- Ethereum 主網、Gnosis Chain、Arbitrum、Base，加上測試網（Sepolia）。
- 每條鏈上 `GPv2Settlement` 合約 bytecode 與位址相同（CREATE2 確定性部署）。
- 各鏈 solver 集合可能不同，夥伴不能假設流動性路由完全一致。
- Gas 經濟差異大：主網批次手續費 $5-50 才划算；L2 可低到不到一美分。

Diagram

`cow-10-multichain.svg`

Q082 · `intermediate` · `API` `AUTH`

EN: JWT vs API key — which would you offer partners?

中: JWT vs API key — 你會給夥伴用哪一種？

Answer (English)

- API key: simple shared secret, easy to rotate, fine for server-to-server traffic with TLS.
- JWT: signed claims with expiry; useful for short-lived access (e.g., browser sessions) or when you need scoped permissions.
- For CoW partners (mostly backend services calling the orderbook): API key with per-key rate limits + IP allowlist.
- Either way, never accept secrets in URL query strings — only in `Authorization` header.

答案（中文）

- API key：簡單的共享秘密，易於 rotate，適合 TLS 下的 server-to-server。
- JWT：附帶簽章 claim 與過期時間；適合短期存取（瀏覽器 session）或需要 scope 權限。
- CoW 夥伴大多是 backend 呼叫 orderbook：用 API key + per-key rate limit + IP 白名單即可。
- 無論哪種，秘密絕不可放 URL query，只放 `Authorization` header。

Q083 · `intermediate` · `JSON-RPC` `BATCH`

EN: How do JSON-RPC batch requests improve throughput and what limits exist?

中: JSON-RPC 批次請求如何提升吞吐？有哪些限制？

Answer (English)

- Send an array of requests in one HTTP body; provider returns an array of responses keyed by `id`.
- Saves TCP/TLS round-trips; great for "fan out 50 reads" patterns (multi-token balances).
- Most providers cap batch size (Alchemy: ~1000, Infura: ~100); enforce per-batch and per-window limits.
- Failures are per-call — partial errors return `error` for some `id`s; handle each result individually.

```
const body = blocks.map((n, i) => ({ jsonrpc: '2.0', id: i, method: 'eth_getBlockByNumber', params: [n, false] }));
const res = await fetch(RPC, { method: 'POST', headers: {'content-type':'application/json'}, body: JSON.stringify(body) });
```

答案（中文）

- 把一批請求放進同一個 HTTP body，provider 回傳依 `id` 對應的陣列。
- 省 TCP/TLS round-trip，適合「同時讀 50 種代幣餘額」這種扇出。
- 大多 provider 有批次上限（Alchemy ~1000、Infura ~100），自己也要限 per-batch 與 per-window。
- 錯誤是 per-call——部分失敗會在對應 `id` 回 `error`，需逐筆處理。


```
const body = blocks.map((n, i) => ({ jsonrpc: '2.0', id: i, method: 'eth_getBlockByNumber', params: [n, false] }));
const res = await fetch(RPC, { method: 'POST', headers: {'content-type':'application/json'}, body: JSON.stringify(body) });
```

Q084 · `basic` · `EVM` `TOOLS`

EN: When debugging on-chain, what tools do you reach for first?

中: 除錯鏈上交易時，你最先伸手拿哪些工具？

Answer (English)

- Etherscan / Blockscout: tx receipt, internal calls, decoded logs, contract source.
- Tenderly: time-travel debugger, state diffs, simulated re-execution with edits.
- `cast call --trace` (Foundry) for fast local CLI debugging without leaving the terminal.
- For quote/RPC issues: provider dashboards (Alchemy, QuickNode) for rate-limit + status.

答案（中文）

- Etherscan / Blockscout：tx receipt、internal call、decoded log、合約原始碼。
- Tenderly：時光機 debugger、state diff、可改參數重跑模擬。
- `cast call --trace`（Foundry）：本地 CLI 快速除錯不必離開終端。
- 報價/RPC 問題：provider dashboard（Alchemy、QuickNode）看限流與狀態。

Q085 · `intermediate` · `JS` `ERRORS`

EN: How do you handle unhandled promise rejections in Node?

中: 在 Node 中如何處理未捕捉的 Promise rejection ?

Answer (English)

- Default behaviour in Node 16+: unhandled rejections crash the process (`--unhandled-rejections=strict`).
- Add a global `process.on('unhandledRejection', err => log/alert)` for last-ditch logging *before* exit.
- Better: ensure every `await` is inside try/catch or has a `.catch(...)` chained.
- Pattern: `void someAsyncFn().catch(captureException)` for fire-and-forget background work.

```
process.on('unhandledRejection', (reason) => {
  logger.fatal({ err: reason }, 'unhandled');
  process.exit(1);
});
```

答案（中文）

- Node 16+ 預設行為：未處理的 rejection 會 crash 進程（ `--unhandled-rejections=strict` ）。
- 加全域 `process.on('unhandledRejection', err => log/alert)` 在退出前留紀錄。
- 更好做法：每個 `await` 都包 try/catch 或鏈上 `.catch(...)` 。
- 背景 fire-and-forget 用 `void someAsyncFn().catch(captureException)` 。

```
process.on('unhandledRejection', (reason) => {
  logger.fatal({ err: reason }, 'unhandled');
  process.exit(1);
});
```

Q086 · `basic` · `JS` `EVENT-LOOP`

EN: Why can a long synchronous loop freeze a Node server?

中: 為什麼一段同步迴圈會凍結 Node server ?

Answer (English)

- Node uses a single-threaded event loop; sync code blocks I/O callbacks, timers, and HTTP handlers.

- A 500ms hash loop holds the loop for 500ms, denying every concurrent request.
- Solution: chunk work with `setImmediate` / `queueMicrotask`, offload to `worker_threads`, or use a queue + worker process.
- Profile with `clinic.js` or `--prof` to find blocking hot spots.

答案（中文）

- Node 是單執行緒事件迴圈，同步程式碼會擋住 I/O callback、timer、HTTP handler。
- 500ms 的 hash 迴圈會把 loop 卡 500ms，期間所有請求都被拒。
- 解法：用 `setImmediate` / `queueMicrotask` 切片、丟到 `worker_threads`、或走佇列 + worker 進程。
- 用 `clinic.js` 或 `--prof` 找阻塞熱點。

Q087 · `intermediate` · `TS` `GUARDS`

EN: When should you write a custom type guard vs use `as`?

中: 什麼情況該寫自訂 type guard，而不是用 `as`？

Answer (English)

- `as` is a *trust me* cast — TS does not verify; bug reports get hidden until runtime.
- A type guard (`x is Foo`) does runtime validation and gives the compiler a refined type for free.
- Use guards at I/O boundaries (network, storage, user input); never trust unvalidated JSON.
- For complex shapes use `zod` / `valibot` and derive the TS type from the schema.

```
import { z } from 'zod';
const Order = z.object({ uid: z.string(), buyAmount: z.string() });
type Order = z.infer<typeof Order>;
const o = Order.parse(await res.json()); // throws if invalid
```

答案（中文）

- `as` 是 *相信我* 的強轉，TS 不會驗證；bug 會藏到 runtime 才爆。
- Type guard（`x is Foo`）在 runtime 驗證，順便給編譯器收窄後的型別。
- I/O 邊界（網路、儲存、使用者輸入）一律用 guard，不要信任未驗證的 JSON。
- 複雜結構用 `zod` / `valibot`，由 schema 推導 TS 型別。

```ts
import { z } from 'zod';
const Order = z.object({ uid: z.string(), buyAmount: z.string() });
type Order = z.infer<typeof Order>;
const o = Order.parse(await res.json()); // 不合法會 throw
```

Q088 · `advanced` · `RUST` `BORROW`

EN: Common Rust borrow checker errors and how to fix them.

中: Rust borrow checker 常見錯誤與修法。

Answer (English)

- "Cannot borrow as mutable while immutable borrow is active" — split scopes or copy/clone the read first.
- "Borrowed value does not live long enough" — change return type, use `'static`, or collect into an owned value.
- "Cannot move out of borrowed content" — use `std::mem::take`, `Option::take`, or restructure ownership.
- When fighting the checker, often the right answer is "your data model is wrong" — split the struct.

```rust
let mut v = vec![1, 2, 3];
let first = v[0];          // copy primitive
v.push(4);                 // ok now
println!("{first} {v:?}");
```

答案（中文）

- "Cannot borrow as mutable while immutable borrow is active"——切作用域，或先複製/clone 讀出來。
- "Borrowed value does not live long enough"——改回傳型別、加 `'static`、或 collect 成擁有值。
- "Cannot move out of borrowed content"——用 `std::mem::take`、`Option::take` 或重構所有權。
- 與 borrow checker 拉鋸久了，多半是「資料模型不對」——把 struct 拆開。

```rust
let mut v = vec![1, 2, 3];
let first = v[0];          // primitive 直接 copy
v.push(4);                 // 現在 ok
println!("{first} {v:?}");
```

Q089 · `advanced` · `DEFI` `LVR`

EN: Define LVR formally and explain why it persists in CFMMs.

中: 正式定義 LVR，並解釋為何在 CFMM 中持續存在。

Answer (English)

- LVR (Loss-Versus-Rebalancing) = the value the LP could have captured if they rebalanced their portfolio at the true CEX price *themselves*, vs letting arbitrageurs do it for free.
- Model: per-second LVR ≈ `(σ² / 8) * V` where `σ` is volatility and `V` is the value locked — independent of fees!
- Persists because CFMM prices update only when traded against; arbitrageurs always have a free option to extract the gap.
- Fix shapes: dynamic fees (Uni v4 hooks), batch auctions (CoW AMM), oracle-aware AMMs.

答案（中文）

- LVR（Loss-Versus-Rebalancing）= 如果 LP 自己用真實 CEX 價即時再平衡能拿到的價值，與任由套利者免費再平衡之間的差距。
- 模型：每秒 LVR ≈ `(σ² / 8) * V`，`σ` 是波動率、`V` 是鎖倉價值——與手續費無關！
- 會持續是因為 CFMM 價格只在被交易時才更新，套利者永遠有「免費的選擇權」吃這個價差。
- 修法方向：動態手續費（Uni v4 hook）、batch auction（CoW AMM）、oracle-aware AMM。

Q090 · `intermediate` · `UX` `WALLET`

EN: What wallet UX issues do CoW orders create vs classic AMM swaps?

中: CoW 訂單相比傳統 AMM swap，會帶來哪些錢包 UX 議題？

Answer (English)

- No tx hash at the moment of "swap" — users used to seeing instant Etherscan link can be confused.
- Approval is to `GPv2VaultRelayer` (not the router they expect); good UI must label this clearly.
- EIP-712 popup looks "scary" to users used to clicking simple Approve — invest in better signing prompts.

- Order can sit pending; show clear status: "Submitted → Open in batch → Filled with surplus +X".

答案（中文）

- 按下 swap 當下沒有 tx hash——習慣看 Etherscan 即時連結的使用者會困惑。
- Approval 對象是 `GPv2VaultRelayer`（不是想像中的 router），UI 要清楚標示。
- EIP-712 簽章視窗對習慣按 Approve 的使用者很「可怕」，要投資在更好的簽章提示。
- 訂單會 pending 一段時間，狀態要清楚：「已送出 → 進入批次 → 成交（含 +X surplus）」。

Q091 · `intermediate` · `EVM` `GAS`

EN: How do you handle a stuck transaction on a partner integration?

中: 夥伴整合中遇到卡住的交易要怎麼處理？

Answer (English)

- Verify the tx is actually pending (not dropped); inspect with `cast tx <hash>`.
- Re-broadcast or replace: same nonce + ≥10% higher `maxFeePerGas` / `maxPriorityFeePerGas`.
- Or send a `0 ETH self-transfer` at the same nonce with high gas to "cancel" the original.
- If pattern repeats, instrument fee estimation: poll `eth_feeHistory` and pad maxFee 2× current base fee.

```
// Cancel: send self-tx at same nonce with bumped fee
await wallet.sendTransaction({
  to: wallet.address,
  value: 0n,
  nonce: stuckNonce,
  maxFeePerGas: stuckMaxFee * 12n / 10n,
  maxPriorityFeePerGas: stuckTip * 12n / 10n,
});
```

答案（中文）

- 先確認交易真的在 pending（沒被 drop），用 `cast tx <hash>` 看。
- 重廣播或取代：同 nonce + 至少高 10% 的 `maxFeePerGas` / `maxPriorityFeePerGas`。
- 或在同 nonce 送一筆「自轉 0 ETH」搭配高 gas 把原本的取消掉。
- 若反覆發生，改善 fee 估算：輪詢 `eth_feeHistory` 並把 maxFee 設為當前 base fee 的 2 倍。

```
// 取消：同 nonce + 加價的自轉
await wallet.sendTransaction({
  to: wallet.address,
  value: 0n,
  nonce: stuckNonce,
  maxFeePerGas: stuckMaxFee * 12n / 10n,
  maxPriorityFeePerGas: stuckTip * 12n / 10n,
});
```

Q092 · `intermediate` · `SOLIDITY` `TOOLING`

EN: Hardhat vs Foundry — when do you reach for which?

中: Hardhat vs Foundry — 你什麼情況選哪一個？

Answer (English)

- **Foundry** (`forge` / `cast`): native fast tests in Solidity, fuzzing/invariants, CLI ergonomics — preferred for new contract work.
- **Hardhat**: JS/TS scripting, vast plugin ecosystem (Etherscan verify, OpenZeppelin upgrades), fits well in TS monorepos.
- Often combined: Foundry for unit + invariant tests, Hardhat for deploy scripts and integration tests.
- For partner debugging, `cast` is hard to beat for one-off RPC calls.

答案（中文）

- **Foundry**（`forge` / `cast`）：原生 Solidity 高速測試、fuzzing/invariant、CLI 體驗佳——新合約專案首選。
- **Hardhat**：JS/TS 腳本、外掛生態龐大（Etherscan verify、OpenZeppelin upgrade），與 TS monorepo 相容好。
- 常見組合：Foundry 跑單元 + invariant 測試，Hardhat 寫部署腳本與整合測試。
- 夥伴除錯時 `cast` 做一次性 RPC 呼叫無人能敵。

Q093 · `basic` · `INTEGRATION` `REPRO`

EN: A partner cannot reproduce a bug — what do you ask for?

中: 夥伴重現不出某個 bug——你會跟他要什麼？

Answer (English)

- Exact endpoint, request body, response body, headers (especially `X-Request-ID`).
- Full `User-Agent` (SDK version!) and Node/runtime version.
- Network conditions (region, proxy, corporate VPN) — issues often correlate with geo.
- Time window (UTC) so you can pull server-side logs at the same instant.

答案（中文）

- 具體端點、request body、response body、header（特別是 `X-Request-ID`）。
- 完整 `User-Agent`（SDK 版本！）與 Node/runtime 版本。
- 網路狀況（地區、proxy、企業 VPN）——問題常與地理相關。
- UTC 時間區段，方便對到 server 端 log。

Q094 · `intermediate` · `PRIORITISATION` `COMMS`

EN: How do you prioritise across 5 partners all asking for "urgent" features?

中: 5 個夥伴都喊「急件」，你如何排序？

Answer (English)

- Score each ask: revenue impact, contract obligation, user count, blast radius, effort.
- Apply a simple framework like RICE (Reach × Impact × Confidence / Effort).
- Be transparent: share the scoring model with partners so denials feel principled, not political.
- Always reply within 24h — even "we hear you, ETA next quarter" beats silence.

答案（中文）

- 替每個需求打分：營收影響、合約義務、使用者數、爆炸半徑、所需工作量。
- 套簡單框架如 RICE（Reach × Impact × Confidence / Effort）。
- 把計分模型公開給夥伴，拒絕才會像規則而非人情。
- 24 小時內務必回覆，「收到，預計下季」也比沉默強。

Q095 · `intermediate` · `SLA` `COMMS`

EN: A partner wants a 99.99% SLA — what do you negotiate?

中: 夥伴想要 99.99% SLA — 你會怎麼談？

Answer (English)

- 99.99% = ~52 min/year downtime — extremely expensive for a multi-tenant DeFi service.
- Push back: ask which user-visible metric matters (order acceptance? settlement?), define SLI precisely.
- Offer tiered SLAs: 99.9% on orderbook accept, 99.5% on settlement (depends on chain), with clear exclusions (force majeure, chain reorg).
- Tie penalties to revenue share, not absolute dollars; align incentives so both sides invest in reliability.

答案（中文）

- 99.99% ≈ 每年 52 分鐘 downtime，對多租戶 DeFi 服務代價極高。
- 反問：使用者真正在乎哪個指標（訂單接收？結算？），把 SLI 定義清楚。
- 提分層 SLA：orderbook 接收 99.9%、結算 99.5%（受鏈影響），並明列例外（force majeure、reorg）。
- 罰則綁營收分潤而非固定金額，讓雙方都有動力投資可用性。

Q096 · `advanced` · `API` `BREAKING`

EN: You must change the response shape of /orders. How do you do it without breaking partners?

中: 你必須改 /orders 的回應結構——如何在不弄壞夥伴的前提下做到？

Answer (English)

- Add new fields *additively* first; never rename or remove in v1.
- If the change is incompatible, bump to `/v2`; serve both for 6+ months.
- Use `Accept-Version` header as a feature flag during rollout, defaulting to v1 for old clients.
- Log usage by API key — proactively reach out to top callers, don't just announce.

答案（中文）

- 先**新增**欄位，v1 內絕不改名或移除。
- 若無法相容，升 `/v2` 並讓兩版並行 6 個月以上。
- 滾動更新時用 `Accept-Version` header 當 flag，老 client 預設 v1。
- 依 API key 紀錄用量，主動聯絡前 N 大呼叫者，別只發公告。

Q097 · `intermediate` · `SDK` `RELEASE`

EN: How do you ship a clean SDK release?

中: 如何 release 一個乾淨的 SDK 版本？

Answer (English)

- Semver discipline: patch = bug, minor = additive, major = breaking — automate with changesets.
- Generate a CHANGELOG entry per PR (changesets/CLI), publish via CI on merge to main.
- Smoke test the published artifact in a fresh container before tagging "latest".
- Announce in #integration-partners + email; pin a "migration guide" for any breaking change.

答案（中文）

- 嚴守 semver：patch = bug、minor = 新增、major = breaking——用 changesets 自動化。
- 每個 PR 產一條 CHANGELOG（changesets/CLI），merge to main 觸發 CI 發版。
- 在乾淨容器中跑已發佈的 artifact smoke test，再打 `latest` tag。
- 在 #integration-partners + email 宣布；breaking change 必附「遷移指南」。

Q098 · `intermediate` · `COMMS` `EXEC`

EN: Explain CoW batch auctions to a busy executive in three bullets.

中: 請用三點向忙碌的高階主管解釋 CoW batch auction。

Answer (English)

- We collect orders for ~30 seconds, then let solvers compete to settle them all together.
- Same price for everyone in the batch → impossible to be sandwiched, and matching peers internally beats AMM rates.
- The user keeps every bit of price improvement (surplus); solvers get paid by the protocol if they win — not by hidden fees.

答案（中文）

- 我們收 ~30 秒訂單，再讓 solver 競標一次結算全部。
- 同批同價 → 不可能被夾擊，內部配對 peer 又比 AMM 划算。

- 價差紅利（surplus）100% 留給使用者；solver 獲勝由協定付費，沒有暗藏手續費。

Q099 · `intermediate` · `SDK` `TREESHAKE`

EN: How do you make sure your SDK is tree-shakeable?

中: 如何確保你的 SDK 是 tree-shakeable？

Answer (English)

- Pure ESM output, named exports only — no default-export blobs.
- Mark `"sideEffects": false` in `package.json` (or list specific files that have side effects).
- Avoid top-level `new SomeBigClient()` instantiation — defer construction to the user.
- Verify with `agadoo` or `publint`; check that `esbuild --bundle` shrinks the output of an import that uses one function.

答案（中文）

- 輸出純 ESM、只用 named export，不要 default-export 整包。
- 在 `package.json` 標 `"sideEffects": false`（或列出有副作用的檔案）。
- 避免在 top-level `new SomeBigClient()`，把建構延遲到使用者。
- 用 `agadoo` 或 `publint` 驗證；用 `esbuild --bundle` 看「只用一個 function」的輸出有沒有變小。

Q100 · `intermediate` · `CODEREVIEW` `PROCESS`

EN: What does your code review checklist look like for an integration repo?

中: 整合 repo 的 code review checklist 你怎麼設？

Answer (English)

- Correctness: read tests first; do they fail without the diff? Edge cases for BigInt, decimals, signature replay?
- Security: any new external call? approve scope minimal? secrets in logs?
- API contract: any breaking change to OpenAPI? version bumped? changelog updated?
- Operability: feature flag? backwards-compat read path? metrics + alerts updated?

答案（中文）

- 正確性：先看測試，沒有 diff 是否會 fail？BigInt、小數、簽章重放等邊界處理了嗎？

- 安全：新增外部呼叫？approve 範圍最小化？log 沒寫秘密？
- API 合約：OpenAPI 是否 breaking change？版本是否提升？changelog 是否更新？
- 可運維性：有 feature flag？讀取路徑向後相容？metric 與 alert 同步更新？

Q101 · `advanced` · `API` `RATE-LIMIT`

EN: Compare token bucket, leaky bucket, and sliding window rate limiters.

中: 比較 token bucket、leaky bucket、sliding window 限流器。

Answer (English)

- **Token bucket**: capacity + refill rate; allows bursts up to bucket size, smooths long-term to refill rate.
- **Leaky bucket**: outflow at constant rate; bursts queued, dropped if queue full — strict shaping.
- **Sliding window** (log or counter): true rolling N requests per window, more accurate but more memory.
- For public APIs token bucket per key + global concurrency cap is the pragmatic default.

答案（中文）

- **Token bucket**：有容量與回填速率，可吸收尖峰至 bucket 大小，長期穩定為回填速率。
- **Leaky bucket**：固定流出速率，超出排隊或丟掉——嚴格整形。
- **Sliding window**（log 或 counter）：真正滾動的「N requests / window」，較精準但較吃記憶體。
- 公開 API 多採 per-key token bucket + 全域併發上限，是務實預設。

Q102 · `intermediate` · `DEVREL` `CONTENT`

EN: How would you write a great DevRel blog post about a new SDK feature?

中: 你會如何寫一篇關於新 SDK 功能、優秀的 DevRel blog post？

Answer (English)

- Lead with the *user pain* the feature removes — not the feature name.
- A 30-line copy-paste runnable example before any prose explanation.
- Show the diff: "before / after" code, plus a chart if there is a measurable improvement.
- End with: link to docs, link to discord/forum, sample repo, and what's next.

答案（中文）

- 以使用者**痛點**開頭，而非功能名稱。
- 前 30 行直接放可貼可跑的範例 code，再開始講道理。
- 展示 diff：「之前 / 之後」程式碼，能量化就放圖表。
- 結尾：文件連結、Discord/論壇連結、範例 repo，以及 what's next。

Q103 · `intermediate` · `COW` `ORDER-KIND`

EN: What is the difference between sell and buy order kinds?

中: sell 與 buy 兩種 order kind 有什麼差別？

Answer (English)

- `sell`: user fixes `sellAmount`, accepts any `buyAmount ≥ minimum`. Common for "swap 100 USDC into ETH".
- `buy`: user fixes `buyAmount`, accepts any `sellAmount ≤ maximum`. Useful for "I need exactly 1 ETH, spend up to X USDC".
- Solvers handle them symmetrically; both produce surplus only on the unfixed side.
- Many partners default to `sell` for simpler UX; `buy` is essential for billing/treasury "exact payment" flows.

答案（中文）

- `sell`：使用者固定 `sellAmount`，接受 `buyAmount ≥ 最低`。常見於「賣 100 USDC 換 ETH」。
- `buy`：使用者固定 `buyAmount`，接受 `sellAmount ≤ 最高`。用於「我要剛好 1 ETH，最多花 X USDC」。
- Solver 對兩者對稱處理，surplus 只會出現在未固定的那一邊。
- 多數夥伴預設 `sell`（UX 較單純）；`buy` 對「付款金額固定」的帳務/treasury 流程不可或缺。

Q104 · `basic` · `UX` `WALLET-CONNECT`

EN: Walk through a WalletConnect v2 session for a partner dapp.

中: 描述夥伴 dapp 一次 WalletConnect v2 連線流程。

Answer (English)

- Dapp builds a `Sign` client, calls `client.connect({ requiredNamespaces })` → returns a URI.

- URI shown as QR or deep link; mobile wallet scans, both sides establish a relay-mediated session.
- Wallet exposes `eth_signTypedData_v4` and `eth_sendTransaction` over the session.
- Best practice: handle session events (`session_delete`, `session_update`) and reconnect gracefully.

答案（中文）

- Dapp 建立 `Sign` client，呼叫 `client.connect({ requiredNamespaces })` 取得 URI。
- URI 以 QR 或 deeplink 顯示，行動錢包掃描後雙方透過 relay 建立 session。
- 錢包透過 session 提供 `eth_signTypedData_v4`、`eth_sendTransaction`。
- 最佳實務：處理 `session_delete`、`session_update` 等事件，重連流程要順。

Q105 · `intermediate` · `GRANTS` `BOUNTY`

EN: How would you structure a CoW Grants technical bounty?

中: 你會如何規劃一個 CoW Grants 的技術懸賞？

Answer (English)

- Define a measurable acceptance criterion (e.g., "ship a Telegram bot that places 100 limit orders/day for ≥1 month").
- Split payout: 30% on milestone-1 demo, 50% on production launch, 20% after 30-day stability.
- Provide a reviewer (engineer + BD) and a Discord channel; cap response time to 48h.
- Open-source license requirement (MIT/Apache); deliverable lives in a public repo with CI.

答案（中文）

- 定可量化的驗收標準（例如「Telegram bot 每天送 100 筆限價單、持續一個月」）。
- 付款分段：30% 給 milestone-1 demo、50% 上線、20% 上線 30 天穩定後。
- 指派 reviewer（工程 + BD）+ Discord 頻道；回覆上限 48 小時。
- 強制 open-source license（MIT/Apache）；交付物放公開 repo 並有 CI。
