Providers / Signers / ABI / Token Approval Flow

以太坊开发四大常客 · 小学生也能懂（超完整讲义）

总览：你一直看到的四个名词到底在干嘛？

一句话总表（先背起来）

- **Provider**：**只能看**区块链资料（读）
- **Signer**：**能看也能改**区块链资料（读 + 写，因为会签名）
- **ABI**：**翻译说明书**，教电脑怎么「叫到合约函数」
- **Approval Flow**：**ERC20 付款要两步**：先允许（approve），再扣钱（transferFrom）

为什么要学这四个？

因为你写 dApp 时，永远绕不开：

- 你要连到哪一个节点（Provider）
- 你要用谁的钱发交易（Signer）
- 你要怎么知道合约有哪些函数（ABI）
- 你要怎么让合约扣走用户的 ERC20（Approval Flow）

Provider（提供者）：区块链的「查询入口」

一句话给小学生

Provider 就像「看账本的窗口」：**你可以查，但不能改**。

Provider 到底是什么？

要读或写区块链，你必须跟一个「以太坊节点」沟通。节点里面有：

- 当前最新区块
- 每个地址余额
- 合约储存的状态（state）
- 交易记录与回执（receipt）

Provider 就是：**你的程式用来连接节点的那条线**，让你可以「读取」链上资料。

Provider 常见来源

- **钱包当中间人（MetaMask）**：网页透过钱包连节点
- **直接用 RPC URL**：你的程式直接连 Infura/Alchemy/自架节点

Provider 能做什么？（典型读操作）

- 查余额：某地址有多少 ETH
- 读合约的 `view/pure` 函数（不会改状态）
- 查交易详情、区块号、区块内容
- 查 NFT 的 owner（拥有者）

重要提醒

只有 Provider 不够用来发交易。
因为发交易一定要「签名」，而 Provider 本身没有钱包、没有私钥。

Signer（签名者）：带着钱包的 Provider

一句话给小学生

Signer 就像「能盖章的管理员」：**能查，也能改，因为他有印章（私钥）。**

Signer 到底是什么？

Signer = **Provider + 钱包（能签名的身份）**

- Provider 负责：连到节点、读取状态
- 钱包负责：用私钥帮交易签名

当你要「写入链上」（改变状态）时，就必须用 Signer。

为什么写入一定需要 Signer？

因为写入链上的动作其实是「交易」：

- 交易要付 Gas（要有钱）
- 交易要签名（证明是你本人授权）
- 交易送出去后，全网节点才会执行并更新状态

只有 Signer 才能做这件事。

选择题：Provider vs Signer

Provider 和 Signer 的差别是?

- Providers 只能读，Signers 只能写
- **Providers 只能读，Signers 能读也能写（正确）**
- Providers 只能写，Signers 只能读
- Providers 只能写，Signers 能读也能写

一句话总结

读数据：用 Provider；发交易：用 Signer。

ABI：合约的「翻译说明书」

一句话给小学生

ABI 就像「菜单」：告诉你这家店**有哪些菜（函数）、怎么点（参数）、会端出什么（回传值）**。

为什么需要 ABI？

Solidity 写完会被编译成字节码（bytecode）：

- 里面是 EVM 能懂的 OPCODE
- **函数名字、参数名字，在字节码里都看不到了**

但你在网页/前端想呼叫：`mint(address to)` 这种函数。
所以你需要一种规则，把「人类的函数名」变成「机器能找到的那段字节码入口」。这就是 ABI 在做的事。

ABI 负责的两件事（超关键）

- **Encode（编码）**：把 `transfer(to, amount)` 这种呼叫，变成 EVM 看得懂的资料格式
- **Decode（解码）**：把回传的 bytes 变回人类看得懂的数值/地址/字串

通常这一步由 **Ethers.js / Viem** 帮你做，你不用手写编码，但你必须提供 ABI。

常见踩雷

ABI 少一项，你就叫不到那一项函数。
例如 ABI 里没有 `balanceOf`，你就算合约真的有该函数，库也不知道怎么编码/解码，自然无法呼叫。

选择题：ABI 的用途

ABI 的主要用途是?

- **包含合约函数资讯，用来编码/解码呼叫与回传（正确）**
- 只列出函数名字，用于链上数据分析

一句话总结

ABI = 让前端 + 库（Ethers/Viem）知道怎么跟合约对话的「规则书」。

Token Approval Flow：为什么 ERC20 付款要两笔交易？

一句话给小学生

ERC20 不是原生钱，合约不能「直接伸手拿」，所以要两步：
先允许（approve），再扣款（transferFrom）。

先搞懂：为什么 ETH 可以一笔，ERC20 不行？

- **ETH 是以太坊原生币**：交易可以直接带 `msg.value` 给合约（payable）
- **ERC20 是一份合约**：你要转 ERC20，本质上是在呼叫该代币合约的 `transfer(...)`

所以 ERC20 不能像 ETH 一样「把代币塞进 function call」。

关键危险点（为什么必须要 approve）

如果允许合约「随便从你的钱包拉走代币」，那我可以写个恶意合约把大家的钱全偷走。
所以 ERC20 标准规定：**别人要从你这里扣代币，必须先获得你的允许（allowance）**。

Allowance（额度）是啥？

一句话

Allowance 就是：**你允许某个合约/地址，最多可以帮你花多少 ERC20**。

ERC20 里两条很重要的规则

- `approve(spender, amount)`：我允许 `spender` 最多花我 `amount` 代币
- `transferFrom(from, to, amount)`：如果我有 allowance，就能从 `from` 把钱转到 `to`

用故事讲：Bob 买 Alice 的 NFT（用 AliceCoin 付款）

角色设定（故事版）

- Alice 发了一个 ERC20：**AliceCoin**
- Alice 也写了一个 NFT 合约（卖 NFT）
- NFT 售价：**10 AliceCoin**
- Bob 手上有 AliceCoin，想买 NFT

重点：Bob 必须做两笔交易

- **交易 1（Approve）**：先允许 NFT 合约可以花我 10 AliceCoin
- **交易 2（Purchase）**：呼叫 NFT 合约的购买函数，NFT 合约内部再用 `transferFrom` 扣款

为什么一定要两笔？

- 第一笔：只是「授权」，**没有转钱**
- 第二笔：购买时合约才真的去 **transferFrom 扣钱**

如果你跳过第一笔授权，第二笔会失败（因为 allowance 不够）。

把流程画成步骤（考试会考的那种）

ERC20 付款的标准流程（你要背）

- Step 0：用户钱包里要有足够的 ERC20
- Step 1：用户对「代币合约」呼叫 `approve(spender= 你的合约, amount= 价格)`
- Step 2：用户对「你的合约」呼叫 `buy()/mint()/purchase()`（随你命名）
- Step 3：你的合约在内部呼叫代币合约：`transferFrom(user, seller/contract, amount)`

开发者提示（很实用）

- 用户可能已经 approve 过：你可以先查 `allowance(owner, spender)`
- 不够再提示用户做 approve（否则浪费 gas 失败）
- 有些代币要先 approve(0) 再 approve(new) 才安全（旧代币习惯问题）

小测验（把这章学会）

题目 1：如果 Alice 想让某合约从她这里扣 AliceCoin，她要怎么做？

- Call approve(...) on AliceCoin contract
- Call transferFrom(...) on AliceCoin contract
- **先 approve(...) 再呼叫另一份合约的函数（正确）**

题目 2：为什么 ERC20 不能像 ETH 一样直接 payable？

- ETH 是原生币，交易可以带 `msg.value`
- ERC20 是合约，要转账等于在呼叫它的函数
- 没有 allowance 机制就会出现合约偷钱的问题

本章小学生总结（最后一页背完）

一页背完

- Provider：只能读链上资料
- Signer：有钱包能签名，能读也能写
- ABI：合约的说明书，负责 encode/decode
- ERC20 付款：先 approve 授权，再 transferFrom 扣款（两笔交易）

教学提示：以后你写 DEX、NFT 市集、订阅制、USDC 付款，几乎都会用到 Approval Flow。