four-flap-meme-sdk 中文文档

本 SDK 面向 **four.meme** 与 **Flap Protocol** 两大平台，提供完整的「登录/上传/创建/交易/报价/读取状态」全链路集成。本文档专为**完全零基础**用户设计，无需任何平台使用经验，我们将详细讲解每个概念、方法和参数。

为什么使用这个 SDK?

这个 SDK 能帮你：

- **发行代币**：只需几行代码就能在 four.meme 或 Flap Protocol 上发币
- **交易代币**：使用联合曲线机制（bonding curve）进行买卖
- **查询状态**：实时获取代币价格和市场数据
- **理解平台**：全面掌握两大 DeFi 发币平台的运作原理

运行要求

- **Node.js**: 18 或更高版本（使用原生 fetch API）
- **模块类型**: ESM (NodeNext)
- **依赖库**:
  - `ethers@^6` - 以太坊交互库

安装

```
npm i four-flap-meme-sdk ethers
```

可选（使用 Pinata 上传 IPFS）

```
npm i pinata
```

目录

🚀 Flap Protocol

🔐 48.club 私有交易

📚 附录

🔎 获取代币 LP 信息（内盘/外盘一键识别）

SDK 提供 `inspectTokenLP(token, opts)` 工具：输入代币地址，自动识别该代币属于 four 内盘、Flap 内盘，或 Pancake V2/V3 外盘；若为外盘则返回与 WBNB/USDT 的交易对与储备量；若为内盘则返回曲线储备/供应量等可展示数据。

最小用法（BSC）

```
import { inspectTokenLP } from 'four-flap-meme-sdk';

const info = await inspectTokenLP('0xToken', {
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org',
  flapChain: 'BSC',                 // 需要识别 Flap 时传入
});
console.log(info);
```

可选：通过路由自动解析工厂地址

```
const info = await inspectTokenLP('0xToken', {
  chain: 'BSC',
  rpcUrl,
  routerV2: '0x10ED43C718714eb63d5aA57B78B54704E256024E', // Pancake V2
Router
  routerV3: '0x13f4EA83D0bd40E75C8222255bc855a974568Dd4', // Pancake V3
Router
  // 或直接 factoryV2/factoryV3 覆盖
});
```

返回结构示例

```
// four 内盘
{ platform: 'FOUR', four: { helper: '0x...', reserveNative: '123456...',
offerTokens: '7890...', lastPrice: '1234...' } }

// flap 内盘
{ platform: 'FLAP', flap: { quoteToken: '0x...', reserveNative: '...',
circulatingSupply: '...', price: '...' } }

// 外盘 V2
{ platform: 'PANCAKE_V2', v2: {
  wbnbPair: { address: '0x...', reserveToken: '...', reserveWBNB: '...' },
  usdtPair: { address: '0x...', reserveToken: '...', reserveUSDT: '...' }
}}

// 外盘 V3（多费档）
{ platform: 'PANCAKE_V3', v3: [
  { base: 'WBNB', fee: 500, pool: '0x...', tokenBalance: '...',
baseBalance: '...' },
  { base: 'USDT', fee: 2500, pool: '0x...', tokenBalance: '...',
baseBalance: '...' }
]}
```

平台概览

four.meme 是什么?

four.meme 是一个代币发行平台，使用**联合曲线**（bonding curve）机制在 DEX 上市前进行价格发现。

核心概念：

1. **联合曲线阶段**:
    - 代币刚创建时，不在 DEX 上
    - 价格由数学公式（联合曲线）决定
    - 随着购买增加，价格自动上涨
    - 随着卖出增加，价格自动下降
2. **迁移到 DEX**:
    - 当筹集足够资金（达到阈值）时，代币"毕业"
    - 自动在 PancakeSwap（BSC）上添加流动性
    - 之后像普通 DEX 代币一样交易
3. **版本说明**:
    - **V1 (TokenManager)**: 原始版本，仍在使用
    - **V2 (TokenManager2)**: 改进版本，功能更多
    - **Helper3**: 辅助合约，用于价格估算和交易路由

为什么使用 four.meme?

- **公平发行**: 所有人以相同的联合曲线价格购买
- **防止 Rug Pull**: 流动性自动添加到 DEX
- **早期参与**: 在 DEX 上市前以较低价格买入

Flap Protocol 是什么?

Flap Protocol 是另一个代币发行平台，具有高级联合曲线功能（CDPV2 曲线）。

核心特性：

1. **高级联合曲线 (CDPV2)**:
    - 更灵活的曲线参数：`r`、`h`、`k`
    - 更好的价格发现机制
    - 可自定义迁移阈值
2. **多链支持**:
    - BSC（币安智能链）
    - Base
    - X Layer
    - Morph

3. **报价代币选项**:
    - 可使用原生代币（BNB、ETH）发行
    - 可使用 ERC20 代币（USDT 等）发行
4. **税费与迁移器选项**:
    - 设置自定义税率
    - 选择 V2 或 V3 迁移器进入 DEX

理解联合曲线

什么是联合曲线?

联合曲线是一个数学公式，根据供应量决定代币价格。

简单例子：

想象一个代币有这样的规则：

- 前 1000 个代币：每个 $0.01
- 接下来 1000 个：每个 $0.02
- 再接下来 1000 个：每个 $0.03
- 依此类推...

联合曲线用平滑的公式实现这一点，而不是分段。

公式 (CDPV2):

```
价格 = k / (1,000,000,000 + h − 供应量)²
```

其中：

- k: 常数，影响整体价格水平
- h: 高度调整（曲线偏移）
- 供应量: 当前流通供应量

为什么重要：

- **早期买家**: 获得更低价格
- **价格上涨**: 随供应增长自动上涨
- **可预测**: 任何人都可以计算价格
- **防操纵**: 不会像 DEX 订单那样被抢跑

内盘与外盘

在代币发行平台中，有两个重要的交易阶段概念：

🔵 内盘（Bonding Curve 阶段）

定义：代币在联合曲线上交易的阶段，尚未迁移到 DEX。

特点：

- ✅ **价格由公式决定**：使用数学曲线（如 CDPV2）计算价格
- ✅ **流动性锁定**：所有资金锁定在平台合约中
- ✅ **防止 Rug Pull**：开发者无法提取流动性
- ✅ **公平价格发现**：所有人以相同曲线价格买卖
- ✅ **即时成交**：不需要订单簿或对手方
- ✅ **可预测性**：可以精确计算买入/卖出价格

在 four.meme 和 Flap Protocol 上：

- 代币刚创建时处于内盘阶段
- 使用 `FourClient` 或 `FlapPortalWriter` 进行交易
- 价格随着买卖自动调整
- 示例：`status = 1`（可交易状态）

内盘交易示例：

```
// 在内盘（联合曲线）上买入代币
const txHash = await writer.swapExactInput({
  inputToken: ZERO_ADDRESS,
  outputToken: tokenAddress,
  inputAmount: parseEther('1.0'),
  minOutputAmount: minAmount
});
```

🟢 外盘（DEX 阶段）

定义：代币已"毕业"并迁移到去中心化交易所（DEX）的阶段。

特点：

- ✅ **市场定价**：价格由供需关系决定，不再使用曲线
- ✅ **DEX 流动性池**：在 PancakeSwap、Uniswap 等 DEX 上交易
- ✅ **自由交易**：可以在任何支持的 DEX 和聚合器上交易
- ✅ **更大流动性**：通常流动性更深，滑点更小
- ✅ **标准 ERC20**：像普通代币一样交易

迁移条件：

- four.meme：当筹集的资金达到特定阈值（如 24 BNB）
- Flap Protocol：当储备金达到 `dexThresh`（可配置）

迁移后：

- 平台合约自动将流动性添加到 DEX

- 联合曲线关闭，无法再通过平台交易
- 代币状态变为 `status = 4`（已迁移到 DEX）

外盘交易示例：

```
// 代币已迁移到 DEX，使用 DEX 路由交易
// 例如使用 Uniswap SDK 或 PancakeSwap SDK
import { SwapRouter } from '@uniswap/v3-sdk';
// ... 使用标准 DEX 交易方法
```

📊 内盘 vs 外盘对比

特性	内盘（Bonding Curve）	外盘（DEX）
价格机制	数学公式（联合曲线）	市场供需
流动性来源	平台合约	DEX 流动性池
交易方式	通过平台合约	通过 DEX（PancakeSwap/Uniswap）
价格可预测性	高（可精确计算）	低（市场波动）
滑点	取决于曲线参数	取决于流动性深度
阶段	早期阶段	成熟阶段
SDK 使用	`FourClient` / `FlapPortalWriter`	DEX SDK（Uniswap/PancakeSwap）
代币状态	`status = 1`	`status = 4`

🔄 从内盘到外盘的迁移过程

1. **内盘阶段**：代币在联合曲线上交易
2. **达到阈值**：储备金达到迁移阈值
3. **自动迁移**：平台合约自动触发迁移
4. **添加流动性**：将储备金和代币添加到 DEX
5. **外盘阶段**：代币在 DEX 上自由交易

如何检查代币在哪个阶段?

```
import { FlapPortal } from 'four-flap-meme-sdk';

const portal = new FlapPortal({ chain: 'BSC', rpcUrl });
const state = await portal.getTokenV5(tokenAddress);

if (state.status === 1) {
  console.log('✅ 内盘：代币在联合曲线上交易');
  console.log('储备金:', state.reserve);
  console.log('迁移阈值:', state.dexSupplyThresh);
} else if (state.status === 4) {
  console.log('✅ 外盘：代币已迁移到 DEX');
```

```
  console.log('请在 PancakeSwap/Uniswap 上交易');
}
```

重要提示：

- ⚠ 代币迁移到 DEX 后，无法再通过平台合约交易
- ⚠ 迁移是单向的，不可逆转
- ⚠ 迁移后需要使用 DEX 交易工具（如 PancakeSwap、Uniswap）

---

four.meme 平台指南

four.meme 平台架构

four.meme 有**三个主要组件**：

1. **REST API**（后端）

- 用户认证（nonce + 签名）
- 图片上传（代币图标）
- 代币元数据管理
- 创建参数生成

2. **智能合约**（链上）

- **TokenManagerV1**: 原始代币管理器
- **TokenManagerV2**: 改进的代币管理器
- **TokenManagerHelper3**: 辅助合约（价格估算和路由）

3. **本 SDK**（你的接口）

- 简化所有 REST API 调用
- 封装智能合约交互
- 提供 V1/V2 统一接口

代币发行流程

在 four.meme 上发行代币需要 **5个步骤**：

```
1. 生成 Nonce      → 从服务器获取随机字符串
2. 签名消息        → 用钱包签名证明所有权
3. 登录            → 获取访问令牌
4. 上传图片        → 上传代币 logo/图标
5. 链上创建        → 部署代币合约
```

为什么需要这个流程?

- **认证**: 证明你拥有钱包
- **安全**: 访问令牌防止未授权操作
- **元数据**: 服务器存储代币链下信息
- **去中心化**: 实际代币在链上，不受服务器控制

一键发行代币

最简单的方式

使用 `createTokenFlow()` 自动处理全部 5 个步骤：

```
import { createTokenFlow } from 'four-flap-meme-sdk';

const result = await createTokenFlow({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...你的私钥',
  networkCode: 'BSC',

  // 可选：提供已上传的图片 URL
  imgUrl: 'https://...',

  // 或者：上传新图片
  image: imageBlob,
  // 或者：使用本地文件路径由 SDK 读取上传（二选一）
  // imagePath: '/absolute/path/to/logo.png',

  // 可选：覆盖默认初始金额 b0（默认 '8' BNB）
  // b0Amount: '8',

  payload: {
    name: '我的超棒代币',              // 全名
    shortName: 'MAT',                  // 符号（代码）
    desc: '一个革命性的代币',          // 描述
    label: 'Meme',                     // 类别
    preSale: '0',                      // 预购金额：'0' = 不购买，'0.5' = 购买
0.5 BNB
    onlyMPC: false,                    // MPC 专属模式（后面解释）

    // 可选的社交链接
    webUrl: 'https://example.com',
    twitterUrl: 'https://twitter.com/...',
    telegramUrl: 'https://t.me/...',
  }
});

console.log('代币已创建！');
console.log('交易哈希:', result.txReceipt.transactionHash);
console.log('你的地址:', result.accountAddress);
```

参数说明：

参数	类型	必填	说明
`rpcUrl`	string	✅	区块链 RPC 端点（BSC、Arbitrum 等）
`privateKey`	string	✅	你的钱包私钥（请保密！）
`networkCode`	'BSC'	✅	目前仅支持 BSC
`baseUrl`	string	❌	four.meme API 地址（默认：https://four.meme/meme-api）
`image`	Blob	❌ *	要上传的图片文件（*如无 imgUrl 则必填）
`imgUrl`	string	❌ *	已上传的图片 URL（*如无 image 则必填）
`imagePath`	string	❌ *	本地图片文件路径（与 image/imgUrl 互斥，三选一）
`b0Amount`	string	❌	初始金额（BNB），默认 '8'，会与 `preSale` 一起发送

Payload 字段：

字段	类型	必填	说明
`name`	string	✅	代币全名（如"比特币"）
`shortName`	string	✅	代币符号（如"BTC"）
`desc`	string	✅	代币描述
`label`	string	✅	类别：'Meme'、'AI'、'Defi'、'Games'、'Infra'、'De-Sci'、'Social'、'Depin'、'Charity'、'Others'
`preSale`	string	✅	BNB 预购金额（发币时自己先购买）。例：'0' = 不购买，'0.1' = 购买 0.1 BNB，'1' = 购买 1 BNB
`onlyMPC`	boolean	❌	如为 true，仅 MPC 钱包可购买（防机器人）
`launchTime`	number	❌	发行时间戳（毫秒，默认：当前时间）
`webUrl`	string	❌	项目网站
`twitterUrl`	string	❌	Twitter/X 账号
`telegramUrl`	string	❌	Telegram 群组/频道

返回值：

```
{
  accountAddress: string,         // 你的钱包地址
  accessToken: string,            // JWT 令牌（用于后续 API 调用）
  imgUrl: string,                 // 上传的图片 URL
  api: {
    createArg: string,            // 发送到合约的编码参数
    signature: string,            // 服务器验证签名
  },
  txReceipt: any,                 // 区块链交易回执
```

```
  tokenAddress?: string          // 可选：新创建代币地址（若可从事件解析）
}
```

分步发行

高级用户

如需更多控制，可使用单独的方法：

步骤 1: 生成 Nonce

```
import { FourClient } from 'four-flap-meme-sdk';

const four = new FourClient();
const nonce = await four.generateNonce({
  accountAddress: '0x...你的地址',
  verifyType: 'LOGIN',
  networkCode: 'BSC'
});
```

作用: 从服务器获取随机 nonce（防止重放攻击）。

步骤 2: 签名登录消息

```
import { buildLoginMessage } from 'four-flap-meme-sdk';
import { Wallet } from 'ethers';

const wallet = new Wallet(privateKey);
const message = buildLoginMessage(nonce); // "You are sign in Meme
{nonce}"
const signature = await wallet.signMessage(message);
```

作用: 创建签名消息，证明你拥有该钱包。

步骤 3: 登录获取访问令牌

```
const accessToken = await four.loginDex({
  region: 'WEB',
  langType: 'ZH',
  walletName: 'MetaMask',
  verifyInfo: {
    address: '0x...你的地址',
    networkCode: 'BSC',
    signature: signature,
    verifyType: 'LOGIN'
```

```
  }
});
```

作用: 用签名换取访问令牌（类似 session cookie）。

步骤 4: 上传图片

```
const imgUrl = await four.uploadImage(accessToken, imageBlob);
```

作用: 上传代币 logo 到 four.meme 的 CDN，返回 URL。

步骤 5: 获取创建参数

```
const { createArg, signature } = await four.createToken(accessToken, {
  name: '我的代币',
  shortName: 'MTK',
  desc: '描述',
  imgUrl: imgUrl,
  launchTime: Date.now(),
  label: 'Meme',
  lpTradingFee: 0.0025,        // 固定为 0.25%
  preSale: '0',
  onlyMPC: false,
  webUrl: 'https://...',
  twitterUrl: 'https://...',
  telegramUrl: 'https://...',
});
```

作用: 服务器生成并签名链上代币创建参数。

步骤 6: 链上创建代币

```
import { createTokenOnChain } from 'four-flap-meme-sdk';

const receipt = await createTokenOnChain({
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org',
  signerPrivateKey: privateKey,
  args: createArg,
  signature: signature,
});

console.log('代币已在交易中创建:', receipt.transactionHash);
```

作用: 向 TokenManager2 合约发送交易，部署你的代币。

在four.meme上交易

代币创建后，任何人都可以使用联合曲线买卖它。

交易流程概览

```
1. 估算价格（可选）      → 查看能获得多少
2. 检查授权（仅卖出）    → 允许 TokenManager 花费你的代币
3. 执行交易              → 买入或卖出
```

价格估算

交易前，应先估算成本/收益：

估算买入价格

```
import { tryBuy } from 'four-flap-meme-sdk';

// 选项 1: 用特定 BNB 金额买入（按资金）
const estimate = await tryBuy(
  'BSC',
  rpcUrl,
  tokenAddress,
  0n,                       // amount = 0 表示"按资金买入"
  1n * 10n ** 18n           // 1 BNB
);

console.log('你将大约获得:', estimate.estimatedAmount);
console.log('成本:', estimate.estimatedCost);
console.log('手续费:', estimate.estimatedFee);

// 选项 2: 买入特定代币数量
const estimate2 = await tryBuy(
  'BSC',
  rpcUrl,
  tokenAddress,
  10_000n * 10n ** 18n,     // 买入 10,000 个代币
  0n                        // funds = 0 表示"按数量买入"
);

console.log('将花费:', estimate2.estimatedCost, 'wei');
```

返回值说明:

```
{
  tokenManager: string,      // 使用哪个 TokenManager 合约（V1 或 V2）
```

```
  quote: string,              // 报价代币地址（通常是 WBNB）
  estimatedAmount: bigint,    // 你将获得的代币数
  estimatedCost: bigint,      // 总成本（wei，含手续费）
  estimatedFee: bigint,       // 交易手续费（wei）
  amountMsgValue: bigint,     // 作为 msg.value 发送的 BNB
  amountApproval: bigint,     // 需要授权的金额（如使用报价代币）
  amountFunds: bigint         // 合约调用中使用的实际资金
}
```

估算卖出价格

```
import { trySell } from 'four-flap-meme-sdk';

const estimate = await trySell(
  'BSC',
  rpcUrl,
  tokenAddress,
  1_000n * 10n ** 18n      // 卖出 1,000 个代币
);

console.log('你将获得:', estimate.funds, 'wei');
console.log('手续费:', estimate.fee, 'wei');
```

返回值:

```
{
  tokenManager: string,       // 使用哪个 TokenManager
  quote: string,              // 报价代币地址
  funds: bigint,              // 你将获得的 BNB（扣除手续费后）
  fee: bigint                 // 交易手续费
}
```

买入代币

有**三种方式**买入代币：

方法 1: 简单买入（推荐）

使用 `buyTokenWithFunds()` 直接在 V2 上按资金买入：

```
import { buyTokenWithFunds } from 'four-flap-meme-sdk';

await buyTokenWithFunds(
  'BSC',
  rpcUrl,
```

```
  privateKey,
  tokenAddress,
  1n * 10n ** 18n,  // 1 BNB
  0n,               // minAmount（滑点保护，可设 >0）
  yourAddress       // 可选接收地址
);
```

参数说明:

- `type: 'funds'`: 指定花费多少 BNB
- `type: 'amount'`: 指定买入多少代币
- `minAmount`/`maxFunds`: 滑点保护（防止价格变动导致不利交易）
- `to`: 可选接收地址（默认为你的地址）
- `origin`: 可选来源代码（用于推荐跟踪，默认：0n）

方法 2: 直接买入（高级）

如果已知是 V2，使用 `buyTokenWithFunds()`:

```
import { tryBuy, buyTokenWithFunds } from 'four-flap-meme-sdk';

// 首先，估算
const estimate = await tryBuy('BSC', rpcUrl, tokenAddress, 0n, 1n * 10n **
18n);

// 然后买入
await buyTokenWithFunds(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  estimate.amountFunds,    // 使用估算的资金
  0n,                      // minAmount（滑点）
  yourAddress              // 接收者
);
```

方法 3: 使用 TM1/TM2 类（专家）

最大控制:

```
import { TM2 } from 'four-flap-meme-sdk';

// 使用链枚举（SDK 自动处理合约地址）
const tm2 = TM2.connectByChain('BSC', rpcUrl);

// 按数量买入
await tm2.buyToken(
  tokenAddress,
```

```
  1_000n * 10n ** 18n,      // 买入 1,000 个代币
  2n * 10n ** 18n           // 最多 2 BNB
);

// 按资金买入（AMAP = As Much As Possible 尽可能多）
await tm2.buyTokenAMAP(
  tokenAddress,
  1n * 10n ** 18n,          // 花费 1 BNB
  0n                        // 最少 0 个代币（无滑点保护）
);
```

卖出代币

卖出需要先**授权代币**（让 TokenManager 花费你的代币）。

完整卖出流程

```
import { ensureSellApproval, tradeSell } from 'four-flap-meme-sdk';

const tokenAddress = '0x...';
const amountToSell = 1_000n * 10n ** 18n;  // 1,000 个代币

// 步骤 1: 授权 TokenManager 花费你的代币
await ensureSellApproval(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  yourAddress,
  amountToSell
);

// 步骤 2: 卖出
await tradeSell(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  {
    type: 'direct',
    amount: amountToSell,
    minFunds: 0n              // 最少获得 BNB
  }
);
```

为什么需要授权?

ERC20 代币卖出需要**两笔交易**:

1. **授权**: 给予 TokenManager 权限
2. **卖出**: 实际转移并获得 BNB

这是 ERC20 代币的安全特性。

路由卖出（第三方应用）

如果你在构建路由器/聚合器：

```
await tradeSell(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  {
    type: 'router',
    from: userAddress,          // 从用户钱包卖出
    amount: amountToSell,
    minFunds: 0n,
    feeRate: 100n,              // 1% 手续费（基点：100 = 1%）
    feeRecipient: routerAddress // 手续费接收地址
  }
);
```

自动路由

什么是自动路由?

four.meme 有两个版本（V1 和 V2），**接口不同**。SDK 提供统一方法，能：

1. 检测代币使用哪个版本
2. 调用正确的合约
3. 自动处理参数差异

使用自动路由

```
import { tradeBuy, tradeSell } from 'four-flap-meme-sdk';

// 对 V1 和 V2 代币都有效（当前签名不包含私钥）
await tradeBuy('BSC', rpcUrl, token, params);
await tradeSell('BSC', rpcUrl, token, params);
```

幕后工作:

1. 调用 `Helper3.getTokenInfo(token)`
2. 检查 `version` 字段（1 或 2）
3. 路由到 TM1 或 TM2

4. 自动适配参数

错误处理

four.meme 交易可能因各种原因失败。SDK 提供错误代码解析：

```
import { parseFourError } from 'four-flap-meme-sdk';

try {
  await tradeBuy('BSC', rpcUrl, token, params);
} catch (error) {
  const { code, message } = parseFourError(error);

  if (code === 'Slippage') {
    console.error('价格变动太大！尝试增加滑点容忍度');
  } else if (code === 'More BNB') {
    console.error('发送的 BNB 不足:', message);
  } else {
    console.error('错误:', code, message);
  }
}
```

错误代码参考

代码	描述	解决方案
GW	金额未对齐到 GWEI	四舍五入到 GWEI（除以 10^9）
ZA	目标地址是零地址	提供有效接收地址
TO	不能发送到 PancakePair	不要使用 LP 地址作为接收者
Slippage	价格变动太大	增加 minAmount/maxFunds 容忍度
More BNB	msg.value 中 BNB 不足	发送更多 BNB 或检查估算
FR	手续费率 > 5%	路由器手续费率必须 ≤ 5%
SO	订单太小	增加交易金额

MPC 专属代币

什么是 MPC 专属?

four.meme 上的某些代币标记为 **"MPC-only"**，意味着：

- 只有 **MPC 钱包**（多方计算钱包）可以购买
- 这是一个**防机器人**功能
- 普通 EOA（外部拥有账户）钱包被阻止

为什么使用 MPC 专属?

- **防止机器人**: 机器人通常使用简单的 EOA 钱包
- **公平发行**: 给真人用户更好的机会
- **减少操纵**: 更难被狙击或抢跑

如何检测 MPC 专属代币

链上检测（推荐）

```
import { isExclusiveOnChain } from 'four-flap-meme-sdk';

const isMPCOnly = await isExclusiveOnChain({
  chain: 'BSC',              // SDK 自动使用正确的代理合约地址
  tokenAddress: '0x...',
  rpcUrl: rpcUrl
});

if (isMPCOnly) {
  console.log('此代币是 MPC 专属! ');
  console.log('你需要 MPC 钱包来交易');
}
```

工作原理: 检查代币的 `template` 字段。如果 `template & 0x10000 > 0`，则为 MPC 专属。

链下检测（使用 API）

```
import { isExclusiveOffChain, FourClient } from 'four-flap-meme-sdk';

const four = new FourClient();
const isMPCOnly = await isExclusiveOffChain(
  (addr) => four.getTokenByAddress(addr),
  tokenAddress
);
```

工作原理: 检查 API 响应中 `version === 'V8'`。

---

four.meme API 参考

FourClient 类

构造函数

```
const four = new FourClient({
  baseUrl?: string  // 默认: 'https://four.meme/meme-api'
});
```

认证方法

generateNonce()

获取登录签名的随机 nonce。

```
const nonce = await four.generateNonce({
  accountAddress: string,      // 你的钱包地址
  verifyType: 'LOGIN',         // 固定值
  networkCode: 'BSC'           // 固定值（目前）
});
```

返回: `Promise<string>` - nonce 字符串

---

buildLoginMessage()

构建要签名的登录消息。

```
import { buildLoginMessage } from 'four-flap-meme-sdk';

const message = buildLoginMessage(nonce);  // 返回: "You are sign in Meme {nonce}"
```

---

loginDex()

用签名换取访问令牌。

```
const accessToken = await four.loginDex({
  region: 'WEB',                    // 固定值
  langType: 'EN' | 'ZH',            // 语言偏好
  walletName: string,               // 如 'MetaMask'、'WalletConnect'
  verifyInfo: {
    address: string,                // 你的钱包地址
    networkCode: 'BSC',             // 固定值
    signature: string,              // 签名 nonce 消息的签名
    verifyType: 'LOGIN'             // 固定值
  }
});
```

返回: `Promise<string>` - JWT 访问令牌

代币管理方法

uploadImage()

上传代币图片到 CDN。

```
const imgUrl = await four.uploadImage(
  accessToken: string,  // 来自 loginDex()
  file: Blob            // 图片文件（JPEG、PNG 等）
);
```

返回: `Promise<string>` - 图片 URL

---

createToken()

获取链上代币创建的签名参数。

```
const { createArg, signature } = await four.createToken(
  accessToken: string,  // 来自 loginDex()
  {
    name: string,                 // 代币全名
    shortName: string,            // 代币符号
    desc: string,                 // 描述
    imgUrl: string,               // 来自 uploadImage() 的图片 URL
    launchTime: number,           // 毫秒时间戳
    label: string,                // 类别（Meme、AI 等）
    lpTradingFee: 0.0025,         // 固定为 0.25%
    preSale: string,              // 预售金额（如 '0'、'0.1'）
    onlyMPC?: boolean,            // MPC 专属标志
    webUrl?: string,              // 可选网站
    twitterUrl?: string,          // 可选 Twitter
    telegramUrl?: string          // 可选 Telegram
  }
);
```

返回: `Promise<{ createArg: string, signature: string }>`

---

getTokenByAddress()

通过合约地址获取代币元数据。

```
const tokenInfo = await four.getTokenByAddress(
  address: string,          // 代币合约地址
  accessToken?: string      // 可选访问令牌
);
```

返回: 代币元数据对象，包含 `name`、`symbol`、`imgUrl`、`version` 等字段。

---

getTokensByAddresses()（批量）

并发按地址数组批量获取 four.meme 代币元数据（任一失败不会中断，会在对应项返回 `{ address, error }`）。

```
import { FourClient } from 'four-flap-meme-sdk';

const four = new FourClient();
const list = await four.getTokensByAddresses(
  ['0xA...', '0xB...', '0xC...'], // 代币地址数组
  /* 可选 */ accessToken          // four.meme 的访问令牌（无则传 undefined）
);
// 返回: any[]，每一项为 tokenInfo 或 { address, error }
```

---

getTokenById()

通过代币 ID 获取代币元数据。

```
const tokenInfo = await four.getTokenById(
  id: string | number,      // four.meme 的代币 ID
  accessToken?: string      // 可选访问令牌
);
```

---

getPublicConfig()

获取 four.meme 平台配置。

```
const config = await four.getPublicConfig();
```

链上方法

createTokenOnChain()

在链上部署代币合约。

```
import { createTokenOnChain } from 'four-flap-meme-sdk';

const { receipt, tokenAddress } = await createTokenOnChain({
  chain: 'BSC',                        // 链名称
```

```
  rpcUrl: string,                    // 区块链 RPC 端点
  signerPrivateKey: string,          // 你的私钥
  args: BytesLike,                   // 来自 four.createToken()
  signature: BytesLike,              // 来自 four.createToken()
  valueWei?: bigint                  // 可选发送的 BNB（用于预售）
});

console.log('已提交交易:', receipt.hash);
if (tokenAddress) console.log('代币地址:', tokenAddress);
```

返回: `{ receipt: TransactionReceipt, tokenAddress?: string }`

交易方法

tryBuy()

估算买入成本而不执行。

```
import { tryBuy } from 'four-flap-meme-sdk';

const estimate = await tryBuy(
  chain: 'BSC' | 'BASE' | 'ARBITRUM_ONE',  // 链名称
  rpcUrl: string,
  token: string,           // 代币地址
  amount: bigint,          // 代币数量（0 = 按资金买入）
  funds: bigint            // BNB 金额（0 = 按数量买入）
);
```

返回: `TryBuyResult` 对象，包含估算详情

---

trySell()

估算卖出收益而不执行。

```
import { trySell } from 'four-flap-meme-sdk';

const estimate = await trySell(
  chain: 'BSC' | 'BASE' | 'ARBITRUM_ONE',  // 链名称
  rpcUrl: string,
  token: string,
  amount: bigint           // 要卖出的代币数量
);
```

返回: `TrySellResult`，包含 funds 和 fee

---

buyTokenWithFunds()

用特定 BNB 金额买入代币（V2 AMAP 方法）。

```
import { buyTokenWithFunds } from 'four-flap-meme-sdk';

await buyTokenWithFunds(
  chain: 'BSC',                    // 链名称
  rpcUrl: string,
  signerPrivateKey: string,
  token: string,
  funds: bigint,                   // 要花费的 BNB 金额
  minAmount: bigint,               // 最少接收代币数
  to?: string,                     // 可选接收者
  origin?: bigint                  // 可选来源代码（默认：0n）
);
```

---

sellToken()

卖出代币换 BNB（V2 方法）。

```
import { sellToken } from 'four-flap-meme-sdk';

await sellToken(
  chain: 'BSC',                    // 链名称
  rpcUrl: string,
  signerPrivateKey: string,
  token: string,
  amount: bigint,                  // 要卖出的代币数量
  minFunds: bigint,                // 最少接收 BNB
  origin?: bigint                  // 可选来源代码（默认：0n）
);
```

注意: 需要先授权代币（先使用 `ensureSellApproval`）

---

tradeBuy()

自动路由买入方法（适用于 V1 和 V2）。

```
import { tradeBuy } from 'four-flap-meme-sdk';

await tradeBuy(
  chain: 'BSC' | 'BASE' | 'ARBITRUM_ONE',  // 链名称
  rpcUrl: string,                           // 区块链 RPC 节点
  token: string,                            // 代币地址
```

```
  params: {
    type: 'funds' | 'amount',                    // 交易类型：按资金/按数量
    funds?: bigint,                              // 当 type==='funds'：花费的
BNB（wei）
    minAmount?: bigint,                          // 当 type==='funds'：最少接收代
币（滑点保护）
    amount?: bigint,                             // 当 type==='amount'：买入代币数
量（最小单位）
    maxFunds?: bigint,                           // 当 type==='amount'：最多花费
BNB（滑点保护）
    to?: string,                                 // 可选接收地址
    origin?: bigint                              // 可选来源代码（默认 0n）
  }
);
```

---

tradeSell()

自动路由卖出方法（适用于 V1 和 V2）。

```
import { tradeSell } from 'four-flap-meme-sdk';

await tradeSell(
  chain: 'BSC' | 'BASE' | 'ARBITRUM_ONE',  // 链名称
  rpcUrl: string,                           // 区块链 RPC 节点
  token: string,                            // 代币地址
  params: {
    type: 'direct' | 'router',              // 直接卖出 或 路由代卖
    amount: bigint,                         // 卖出数量（最小单位）
    minFunds?: bigint,                      // 最少接收 BNB（滑点保护）
    from?: string,                          // 当 type==='router'：用户钱包地
址
    feeRate?: bigint,                       // 当 type==='router'：手续费基点
（100=1%）
    feeRecipient?: string                   // 当 type==='router'：手续费接收
地址
  }
);
```

实用方法

批量校验私钥

批量校验一组私钥是否合规（格式 + 椭圆曲线有效性）。

```
import { validatePrivateKeys } from 'four-flap-meme-sdk';

const result = validatePrivateKeys([
```

```
  // 支持多种输入形式：
  '0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa', //
标准 0x + 64 hex
  'bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb',     //
无 0x 前缀
  'c'.repeat(63)
// 奇数/不足 64 位会自动左侧补零
]);

// 返回: Array<{ privateKey: string; valid: boolean; address?: string;
normalized?: string; error?: string }>
// - valid 为 true：包含解析出的 address 与规范化后的 normalized（0x+64hex）
// - valid 为 false：给出 error（如 'INVALID_FORMAT' 或具体错误信息）
```

校验与规范化规则：

- 自动去掉可选的 0x 前缀；
- 验证是否为纯十六进制，长度不超过 64；
- 若为奇数或不足 64 位，将在左侧补 0 直至 64 位；
- 最终使用 `ethers.Wallet` 构造验证曲线有效性；
- 通过时返回规范化私钥 `normalized` 与地址 `address`。

批量生成钱包地址与私钥

用于开发/测试环境快速生成指定数量的钱包。

```
import { generateWallets } from 'four-flap-meme-sdk';

const wallets = generateWallets(5);
// 返回示例：
// [
//   { address: '0xabc...', privateKey: '0x...' },
//   { address: '0xdef...', privateKey: '0x...' },
//   ...
// ]
```

注意：仅用于开发/测试。生产环境请安全保管私钥，不要明文存储或上传。

使用 Multicall3 批量查询代币余额

```
import { getTokenBalancesWithMulticall } from 'four-flap-meme-sdk';

const rpcUrl = 'https://bsc-dataseed.binance.org';        // 区块链 RPC 节点
const token = '0x...ERC20';                                // 要查询的 ERC20
代币地址
const holders = ['0x...A', '0x...B', '0x...C'];            // 持有人地址数组

// 简化签名：自动按链选择 Multicall3（未知链回退 ca11...ca11）
```

```
const results = await getTokenBalancesWithMulticall(
  rpcUrl,      // RPC 节点
  token,       // 代币地址
  holders      // 持有人地址数组
);
// results: [{ address, balance, success }, ...]
```

可选：如需显式覆盖 Multicall3 地址（向后兼容重载）：

```
const multicall3 = '0xYourMulticall3Address';              // 显式
Multicall3 合约地址
const results = await getTokenBalancesWithMulticall(
  rpcUrl,       // RPC 节点
  multicall3,   // Multicall3 合约地址
  token,        // 代币地址
  holders       // 持有人地址数组
);
```

说明：SDK 会根据网络 chainId 自动选择标准 Multicall3 地址（`0xca11bde05977b3631167028862be2a173976ca11`），无需手动传入。

新增：多代币 + 原生余额（推荐新用法）

```
import { getTokenBalancesWithMulticall, type MultiTokenBalancesResult }
from 'four-flap-meme-sdk';

const rpcUrl = 'https://bsc-dataseed.binance.org';              // 区块链
RPC 节点
const chain = 'BSC' as const;                                   // 链名
称: 'BSC' | 'BASE' | 'XLAYER' | 'MORPH' | 'ARBITRUM_ONE'
const tokenAddresses = [                                        // 要同时查
询的代币地址数组
  '0x...ERC20_A',
  '0x...ERC20_B'
];
const holders = ['0x...A', '0x...B', '0x...C'];                 // 持有人地
址数组

const results: MultiTokenBalancesResult[] = await
getTokenBalancesWithMulticall(
  rpcUrl,
  chain,            // 链名称（用于自动选择 Multicall3 标准地址）
  tokenAddresses,   // 多个代币地址
  holders           // 要查询的持有人地址数组
);

// 结果结构：每个地址包含原生币余额与各代币余额（数组形式，顺序与 tokenAddresses 一
致）
// [
```

```
//   {
//     address: '0x...A',
//     native: '1.23456789',                          // 原生币余额（ETH/BNB 单
位，字符串）
//     tokens: [
//       { token: '0x...ERC20_A', balance: '1.0' },
//       { token: '0x...ERC20_B', balance: '2.5' }
//     ],
//     success: true
//   },
//   ...
// ]
```

也可不传 chain（自动按 RPC 网络识别 Multicall3 地址）：

```
const results2 = await getTokenBalancesWithMulticall(
  rpcUrl,
  tokenAddresses,   // 多个代币地址
  holders           // 要查询的持有人地址数组
);
```

说明：

- 该新签名会同时返回每个地址的原生币余额（native）与传入数组中各代币的余额（tokens 映射）。
- Multicall3 地址将按 `chain` 自动选择；未知链将回退到标准地址 `0xca11...ca11`。
- 若某些解码失败，对应条目的 `success` 为 `false`，余额字段回退为 `0n`。

也可直接传 `chainId`（数字）：

```
const results3 = await getTokenBalancesWithMulticall(
  rpcUrl,
  56,               // chainId（示例：BSC 主网 56）
  tokenAddresses,
  holders
);
```

代币授权方法

SDK 提供了**多种授权方法**，以支持不同的 TokenManager 版本（V1/V2）和平台（four.meme/Flap Protocol）。

Four.meme 代币授权

V1 代币授权：

```
import { checkSellApprovalV1, ensureSellApprovalV1 } from 'four-flap-meme-
sdk';

// 检查 V1 授权状态（只读）
const status = await checkSellApprovalV1(
  'BSC',                    // 支持: 'BSC' | 'BASE' | 'ARBITRUM_ONE'
  rpcUrl,
  tokenAddress,
  ownerAddress,
  amount
);

// 确保 V1 授权（如需要则发送交易）
const result = await ensureSellApprovalV1(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  ownerAddress,
  amount
);
```

V2 代币授权：

```
import { checkSellApprovalV2, ensureSellApprovalV2 } from 'four-flap-meme-
sdk';

// 检查 V2 授权状态（只读）
const status = await checkSellApprovalV2(
  'BSC',                    // 支持: 'BSC' | 'BASE' | 'ARBITRUM_ONE'
  rpcUrl,
  tokenAddress,
  ownerAddress,
  amount
);

// 确保 V2 授权（如需要则发送交易）
const result = await ensureSellApprovalV2(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  ownerAddress,
  amount
);
```

通用方法（默认使用 V2）：

```
import { checkSellApproval, ensureSellApproval } from 'four-flap-meme-
sdk';

// 检查授权（默认使用 V2）
const status = await checkSellApproval(
  'BSC',
  rpcUrl,
  tokenAddress,
  ownerAddress,
  amount
);

// 确保授权（默认使用 V2）
const result = await ensureSellApproval(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  ownerAddress,
  amount
);
```

Flap Protocol 代币授权

```
import {
  checkFlapSellApproval,
  ensureFlapSellApproval,
  checkFlapSellApprovalBatch,
  ensureFlapSellApprovalBatch
} from 'four-flap-meme-sdk';

// 检查 Flap 授权状态（只读）
const status = await checkFlapSellApproval(
  'BSC',                    // 支持: 'BSC' | 'BASE' | 'XLAYER' | 'MORPH'
  rpcUrl,
  tokenAddress,
  ownerAddress,
  amount
);

// 确保 Flap 授权（如需要则发送交易）
const result = await ensureFlapSellApproval(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  ownerAddress,
  amount
);
```

批量检查/授权（默认按上限授权，无需传数额）：

```
// 批量检查（owners: 地址数组；默认 required = 2^256 - 1）
const checkList = await checkFlapSellApprovalBatch(
  'BSC',
  rpcUrl,
  tokenAddress,
  [ownerA, ownerB]
);
// checkList: Array<{ owner, isApproved, currentAllowance,
requiredAllowance }>

// 批量授权（privateKeys: 每个地址对应的私钥；默认授权上限）
const approveResults = await ensureFlapSellApprovalBatch(
  'BSC',
  rpcUrl,
  [pkA, pkB],           // 拥有者私钥数组
  tokenAddress          // 目标代币地址
);
// approveResults: Array<{ owner, alreadyApproved, currentAllowance,
requiredAllowance, txReceipt? }>
```

返回值：

所有 `check*` 方法返回：

```
{
  isApproved: boolean,       // 如果已授权则为 true
  currentAllowance: bigint,  // 当前授权额度
  requiredAllowance: bigint  // 所需授权额度
}
```

所有 `ensure*` 方法返回：

```
{
  alreadyApproved: boolean,  // 如果不需要发送交易则为 true
  currentAllowance: bigint,
  requiredAllowance: bigint,
  txReceipt?: TransactionReceipt  // 仅在发送了授权交易时存在
}
```

完整工作流示例：

Four.meme V2 代币卖出：

```
// 1. 检查授权状态（快速，只读）
const status = await checkSellApprovalV2('BSC', rpcUrl, token, wallet,
amount);

if (!status.isApproved) {
  // 2. 如需要则发送授权交易
  console.log('正在授权...');
  await ensureSellApprovalV2('BSC', rpcUrl, privateKey, token, wallet,
amount);
}

// 3. 执行卖出
await tradeSell('BSC', rpcUrl, token, { type: 'direct', amount, minFunds:
0n });
```

Flap Protocol 代币卖出：

```
// 1. 检查授权状态
const status = await checkFlapSellApproval('BSC', rpcUrl, token, wallet,
amount);

if (!status.isApproved) {
  // 2. 授权给 Flap Portal
  await ensureFlapSellApproval('BSC', rpcUrl, privateKey, token, wallet,
amount);
}

// 3. 执行卖出
const writer = new FlapPortalWriter({ chain: 'BSC', rpcUrl }, privateKey);
await writer.swapExactInput({
  inputToken: token,
  outputToken: ZERO_ADDRESS,
  inputAmount: amount,
  minOutputAmount: 0n,
  permitData: '0x'
});
```

UI 集成示例：

```
// 根据授权状态更新 UI
const status = await checkSellApprovalV2('BSC', rpcUrl, tokenAddr,
walletAddr, sellAmount);

if (status.isApproved) {
  showButton('立即卖出', 'green');
} else {
  showButton('需要授权', 'orange');
  showInfo(`当前额度: ${status.currentAllowance}, 需要:
```

```
${status.requiredAllowance}`);
}
```

重要提示:

- ⚠ **V1 和 V2 使用不同的代理合约** - 必须授权给正确的版本
- ⚠ **Four.meme 和 Flap 使用不同的合约** - 不能混用授权方法
- ✅ **通用方法默认使用 V2** 以保持向后兼容
- ✅ **建议新代码使用明确版本的方法**（V1/V2）

---

parseFourError()

解析 four.meme 错误代码。

```
import { parseFourError } from 'four-flap-meme-sdk';

const { code, message } = parseFourError(error);
```

返回: `{ code?: FourErrorCode, message: string }`

---

isExclusiveOnChain()

检查代币是否为 MPC 专属（链上）。

```
import { isExclusiveOnChain } from 'four-flap-meme-sdk';

const isExclusive = await isExclusiveOnChain({
  chain: 'BSC',  // 仅支持 BSC（TokenManagerV2）
  tokenAddress: string,
  rpcUrl: string
});
```

---

isExclusiveOffChain()

检查代币是否为 MPC 专属（通过 API）。

```
import { isExclusiveOffChain } from 'four-flap-meme-sdk';

const isExclusive = await isExclusiveOffChain(
  getTokenInfo: (addr: string) => Promise<{ version?: string }>,
  tokenAddress: string
);
```

Flap Protocol 指南

Flap 平台架构

Flap Protocol 是新一代代币发行平台，具有以下核心特性：

核心组件：

1. **Portal 合约**: 所有操作的主入口点
2. **CDPV2 联合曲线**: 高级数学曲线，具有三个参数（r、h、k）
3. **多网络支持**: BSC、Base、X Layer、Morph
4. **灵活配置**: 自定义税费、报价代币、迁移设置
5. **可配置费率**: 不同链可设置不同的买入/卖出手续费率

Flap 上的代币生命周期：

```
1. 创建代币        → 使用 CDPV2 曲线部署
2. 交易阶段        → 在联合曲线上买卖
3. 进度跟踪        → 监控储备与阈值
4. 迁移            → 达到阈值时自动迁移到 DEX
```

读取代币状态

Flap 有**四个版本**的代币状态读取方法：

版本对比

方法	返回	用途
`getTokenV2()`	基本状态（7 个字段）	旧版兼容
`getTokenV3()`	+ 报价代币信息（9 个字段）	报价代币支持
`getTokenV4()`	+ 扩展ID（10 个字段）	扩展功能支持
`getTokenV5()`	+ 完整 CDPV2 参数（12 个字段）	完整信息（推荐）

读取代币状态（V5 - 推荐）

```
import { FlapPortal } from 'four-flap-meme-sdk';

// 使用链枚举（SDK 自动处理合约地址和默认费率）
const portal = new FlapPortal({
  chain: 'BSC',  // 支持: 'BSC' | 'BASE' | 'XLAYER' | 'MORPH'
  rpcUrl: 'https://bsc-dataseed.binance.org'
});
```

```
const state = await portal.getTokenV5('0x...代币地址');

console.log('代币状态:', state.status);           // 0=无效, 1=可交易, 4=DEX
console.log('储备(wei):', state.reserve);         // 曲线中的当前 BNB
console.log('供应量:', state.circulatingSupply);     // 流通中的代币
console.log('价格:', state.price);                  // 当前价格（wei）
console.log('曲线参数:', state.r, state.h, state.k);  // CDPV2 参数
console.log('DEX 阈值:', state.dexSupplyThresh);     // 迁移所需供应量
console.log('报价代币:', state.quoteTokenAddress);     // 0x0 = 原生代币
```

状态字段说明

```
{
  status: number,              // 0=无效, 1=可交易, 2=决斗中, 3=已杀死, 4=DEX
  reserve: bigint,             // 曲线中的当前储备（wei）
  circulatingSupply: bigint,   // 流通中的代币（wei, 18 位小数）
  price: bigint,               // 当前价格（每个代币的 wei）
  tokenVersion: number,        // 代币实现版本
  r: bigint,                   // CDPV2 参数 r（储备偏移）
  h: bigint,                   // CDPV2 参数 h（高度调整）
  k: bigint,                   // CDPV2 参数 k（曲线常数）
  dexSupplyThresh: bigint,     // DEX 迁移的供应阈值
  quoteTokenAddress: Address,  // 原生代币为 0x0，或 ERC20 地址
  nativeToQuoteSwapEnabled: boolean,  // 可否将原生币兑换为报价币
  extensionID: bytes32         // 扩展标识符
}
```

CDPV2 联合曲线

CDPV2（恒定动态价格 V2）曲线是 Flap 价格机制的核心。

理解 CDPV2 参数

参数 `r`（储备偏移）

- **是什么**: 起始储备金额
- **效果**: 更高的 `r` = 更高的初始价格
- **典型值**: 0.1 ETH（0.1 × 10^18 wei）

参数 `h`（高度调整）

- **是什么**: 移动供应曲线
- **效果**: 调整曲线开始的位置
- **典型值**: 0（无偏移）

参数 `k`（曲线常数）

- **是什么**: 决定价格增长率
- **效果**: 更高的 k = 整体价格更高
- **典型值**: r × 10^9（10 亿 × r）

CDPV2 公式

```
供应量 (S) = 1,000,000,000 + h − k / (R + r)
储备 (R) = k / (1,000,000,000 + h − S) − r
价格 (P) = k / (1,000,000,000 + h − S)²
```

其中：

- S = 流通供应量
- R = 储备（曲线中的 BNB/ETH）
- P = 每个代币的价格

使用 CDPV2 类

```
import { CDPV2 } from 'four-flap-meme-sdk';

// 创建 r=0.1, h=0, k=1e8 的曲线
const curve = CDPV2.getCurve(0.1, 0, 1e8);

// 计算 10 ETH 储备所需的供应量
const supply = curve.estimateSupply('10');
console.log('10 ETH 时的供应量:', supply.toString());

// 计算 8 亿供应所需的储备
const reserve = curve.estimateReserve('800000000');
console.log('8 亿供应时的储备:', reserve.toString());

// 获取 8 亿供应时的当前价格
const price = curve.price('800000000');
console.log('8 亿供应时的价格:', price.toString());

// 获取 FDV（完全稀释估值）
const fdv = curve.fdv('800000000');
console.log('FDV:', fdv.toString());
```

计算进度

进度 = 代币距离 DEX 迁移有多近

```
import { FlapPortal } from 'four-flap-meme-sdk';

const portal = new FlapPortal({
  chain: 'BSC',
```

```
  rpcUrl: 'https://bsc-dataseed.binance.org'
});

const state = await portal.getTokenV5(tokenAddress);
const { price, progress } = portal.computePriceAndProgress(state);

console.log('当前价格:', price, '每个代币的 ETH');
console.log('进度:', progress);  // '0.0000' 到 '1.0000' (0% 到 100%)
```

进度含义:

- `0.0000`: 刚发行
- `0.5000`: 距 DEX 迁移一半
- `1.0000`: 准备 DEX 迁移
- 进度 = （当前储备）/（阈值所需储备）

Flap 价格报价

SDK 提供了 **两种报价方式**，各有优势：

特性	离线报价	链上报价
方法	`quoteBuy()` / `quoteSell()`	`quoteExactInput()`
速度	⚡ 极快（本地计算）	🦕 较慢（RPC 调用）
准确度	📊 高（99.9%+）	🎯 100% 精确
费用	💰 免费	💸 消耗 RPC 配额
适用场景	UI 实时显示、快速预览	交易前最终确认
依赖	需要代币状态	只需要代币地址

方式 1：离线报价（推荐用于 UI）⚡

优势:

- ⚡ **极快**：本地计算，0 延迟
- 💰 **免费**：不消耗 RPC 配额
- 📊 **准确**：使用与链上相同的公式和费率
- 🎨 **实时**：适合用户输入时的实时更新

方法：`quoteBuy()` / `quoteSell()`

原理:

- 基于 CDPV2 联合曲线公式进行本地计算
- 自动扣除平台手续费（可配置）
- 与链上实际结果 99.9%+ 一致

```
import { FlapPortal } from 'four-flap-meme-sdk';

// 🌟 使用链枚举，SDK 自动应用该链的默认费率
const portal = new FlapPortal({
  chain: 'BSC',  // SDK 自动使用 BSC 的默认费率：买入 1%，卖出 1%
  rpcUrl: 'https://bsc-dataseed.binance.org'
});

// 如果需要覆盖默认费率：
// const portal = new FlapPortal({
//   chain: 'BSC',
//   rpcUrl: 'https://bsc-dataseed.binance.org',
//   buyFeeRate: 0.015,   // 自定义买入费率 1.5%
//   sellFeeRate: 0.015   // 自定义卖出费率 1.5%
// });

// 1. 获取代币状态（只需一次）
const tokenAddress = '0x...' as any;
const state = await portal.getTokenV5(tokenAddress);

// 2. 离线买入报价：花 1 BNB 能得到多少代币
const tokenAmount = portal.quoteBuy(state, '1.0');
console.log('💰 买入报价:');
console.log('  支付: 1.0 BNB');
console.log('  手续费: 0.01 BNB (1%)');
console.log('  实际用于购买: 0.99 BNB');
console.log('  预计获得:', tokenAmount, '个代币');

// 3. 离线卖出报价：卖 1000000 个代币能得到多少 BNB
const ethAmount = portal.quoteSell(state, '1000000');
console.log('\n💸 卖出报价:');
console.log('  卖出: 1000000 个代币');
console.log('  按曲线计算后扣除 1% 手续费');
console.log('  实际收到:', ethAmount, 'BNB');
```

实际应用 - UI 实时更新：

```
// 场景：用户在输入框输入金额，实时显示报价
let cachedState: TokenStateV5;

async function init() {
  // 初始化时获取一次代币状态
  cachedState = await portal.getTokenV5(tokenAddress);
}

function onUserInputChange(inputValue: string) {
  // 用户输入时，立即计算并显示（无延迟）
  const outputAmount = portal.quoteBuy(cachedState, inputValue);
  updateUI(`您将获得约 ${formatNumber(outputAmount)} 个代币`);
```

```
  // 可选：定期刷新 cachedState（如每 10 秒）以保持准确性
}
```

方式 2：链上报价（精确）🎯

优势：

- 🎯 **100% 精确**：直接调用链上合约模拟
- ✅ **实时状态**：使用最新的链上数据
- 🔒 **可靠**：交易前的最终确认

劣势：

- 🐢 需要 RPC 调用（较慢）
- 💸 消耗 RPC 配额

方法：`quoteExactInput()` / `previewBuy()` / `previewSell()`

```
const portal = new FlapPortal({
  chain: 'BSC',                          // 链名称：'BSC' | 'BASE' |
'XLAYER' | 'MORPH'
  rpcUrl: 'https://bsc-dataseed.binance.org' // 区块链 RPC 节点
});

// 方法 A：通用报价（支持任意 input/output token 组合）
const outputAmount = await portal.quoteExactInput({
  inputToken: '0x0000000000000000000000000000000000000000' as any,  // 输入
代币地址：0x0 代表原生币（BNB/ETH）
  outputToken: tokenAddress,          // 输出代币地址：目标代币
  inputAmount: 1n * 10n ** 18n       // 输入金额（wei）：示例为 1 BNB
});
console.log('精确报价 - 你将获得:', formatEther(outputAmount), '个代币');

// 方法 B：专用买入报价（更简洁）
const tokenAmount = await portal.previewBuy(
  tokenAddress,     // 目标代币地址
  1n * 10n ** 18n   // 输入原生币金额（wei）：示例为 1 BNB
);
console.log('买入报价 - 你将获得:', formatEther(tokenAmount), '个代币');

// 方法 C：专用卖出报价（更简洁）
const ethAmount = await portal.previewSell(
  tokenAddress,          // 待卖出的代币地址
  1000n * 10n ** 18n     // 卖出数量（代币最小单位，18 位小数）：示例 1000 个
);
console.log('卖出报价 - 你将获得:', formatEther(ethAmount), 'BNB');
```

📋 使用建议

推荐流程：

```
import { FlapPortal, FlapPortalWriter, ZERO_ADDRESS } from 'four-flap-
meme-sdk';
import { parseEther, formatEther } from 'ethers';

const rpcUrl = 'https://bsc-dataseed.binance.org';
const tokenAddress = '0x...';  // 代币地址
const zeroAddress = ZERO_ADDRESS;

// 创建只读实例用于报价
const portal = new FlapPortal({ chain: 'BSC', rpcUrl });

// 1️⃣ UI 显示阶段：使用离线报价（快速）
const state = await portal.getTokenV5(tokenAddress);
const estimatedOutput = portal.quoteBuy(state, userInput);
showToUser(`预计获得: ${estimatedOutput} 个代币`);

// 2️⃣ 用户点击"确认交易"时：使用链上报价（精确）
const exactOutput = await portal.quoteExactInput({
  inputToken: zeroAddress,
  outputToken: tokenAddress,
  inputAmount: parseEther(userInput)
});
showConfirmDialog(`确切获得: ${formatEther(exactOutput)} 个代币`);

// 3️⃣ 执行交易 - 创建 Writer 实例
const writer = new FlapPortalWriter(
  { chain: 'BSC', rpcUrl },
  'YOUR_PRIVATE_KEY'
);

await writer.swapExactInput({
  inputToken: zeroAddress,
  outputToken: tokenAddress,
  inputAmount: parseEther(userInput),
  minOutputAmount: exactOutput * 99n / 100n  // 允许 1% 滑点
});
```

费率说明：

离线报价的手续费率取决于创建 `FlapPortal` 时的配置：

```
// 🌟 推荐：使用链枚举，自动应用该链的实际费率
const portal1 = new FlapPortal({
  chain: 'BSC',  // 自动使用 BSC 实际费率：买入 1%, 卖出 1%
  rpcUrl
});

const portal2 = new FlapPortal({
  chain: 'MORPH',  // 自动使用 MORPH 实际费率：买入 2.5%, 卖出 2.5%
  rpcUrl
```

```
});

// 如需自定义费率:
const portal3 = new FlapPortal({
  chain: 'BSC',
  rpcUrl,
  buyFeeRate: 0.015,   // 覆盖默认费率
  sellFeeRate: 0.015
});
```

代币兑换

Flap Protocol 提供了多种交易方法，适用于不同场景：

方法 1: 简化买卖方法（推荐新手）

```
import { FlapPortalWriter } from 'four-flap-meme-sdk';

// 使用链枚举（SDK 自动处理合约地址）
const writer = new FlapPortalWriter(
  {
    chain: 'BSC',
    rpcUrl: 'https://bsc-dataseed.binance.org'
  },
  '0x...你的私钥'
);

// 买入代币（返回 TransactionReceipt）
const buyReceipt = await writer.buy(
  tokenAddress,                // 代币地址
  yourAddress,                 // 接收地址
  950n * 10n ** 18n,           // 最少获得 950 个代币（5% 滑点）
  1n * 10n ** 18n              // 发送 1 BNB
);
console.log('买入交易:', buyReceipt.hash);

// 卖出代币（返回 TransactionReceipt）
const sellReceipt = await writer.sell(
  tokenAddress,                // 代币地址
  1000n * 10n ** 18n,          // 卖出 1000 个代币
  95n * 10n ** 16n             // 最少获得 0.95 BNB（5% 滑点）
);
console.log('卖出交易:', sellReceipt.hash);
```

方法 2: 通用兑换方法（高级用户）

```
import { FlapPortalWriter, ZERO_ADDRESS } from 'four-flap-meme-sdk';
```

```
// 创建 Writer 实例
const writer = new FlapPortalWriter(
  { chain: 'BSC', rpcUrl: 'https://bsc-dataseed.binance.org' },
  'YOUR_PRIVATE_KEY'
);

// 用 1 BNB 买入代币（返回 TransactionReceipt）
const receipt = await writer.swapExactInput(
  {
    inputToken: ZERO_ADDRESS,                // BNB
    outputToken: tokenAddress,               // 要买的代币
    inputAmount: 1n * 10n ** 18n,           // 1 BNB
    minOutputAmount: 950n * 10n ** 18n,     // 最少代币（滑点）
    permitData: '0x'                         // 无 permit（可选）
  },
  1n * 10n ** 18n                           // msg.value = 1 BNB
);
console.log('兑换交易:', receipt.hash);
```

方法 3: 创建后立即买入

当前版本未提供 buyOnCreation 接口。建议：创建完成后按常规买入流程执行一次买入交易（可结合私有交易/Bundle 保证原子性）。

使用 Permit 兑换（高级）

Permit 允许在无需事先授权交易的情况下花费 ERC20 代币。SDK 提供了自动签名方法：

```
import { buildPermitPiggybackAuto, FlapPortalWriter } from 'four-flap-
meme-sdk';

// SDK 自动签名 Permit 并使用正确的 Portal 代理地址
const permitData = await buildPermitPiggybackAuto(
  'BSC',                    // 链名称（SDK 自动使用该链的 Portal 地址）
  privateKey,               // 你的私钥
  tokenAddress,             // ERC20 代币地址
  amount,                   // 授权金额
  deadline,                 // 截止时间戳
  nonce                     // Permit nonce（从代币合约获取）
);

// 然后使用 permit 兑换（无需额外授权交易！）
const writer = new FlapPortalWriter({ chain: 'BSC', rpcUrl }, privateKey);
await writer.swapExactInput({
  inputToken: tokenAddress,
  outputToken: targetToken,
  inputAmount: amount,
  minOutputAmount: 0n,
  permitData                // ✅ 授权 + 兑换一笔交易完成
});
```

在Flap上创建代币

步骤 1：上传元数据到 IPFS

Flap Protocol 未公开 IPFS 上传端点，你需要使用第三方 IPFS 服务上传代币元数据。

推荐：使用 Pinata（最简单）

```
import { PinataSDK } from "pinata-web3";

// 初始化 Pinata
const pinata = new PinataSDK({
  pinataJwt: process.env.PINATA_JWT,  // 从 https://pinata.cloud 获取
});

// 1. 上传图片
const imageUpload = await pinata.upload.file(imageFile);
const imageCid = imageUpload.IpfsHash;

// 2. 构建并上传元数据 JSON
const metadata = {
  image: imageCid,                    // 图片的 CID
  description: "我的代币描述",
  creator: yourAddress,
  website: "https://example.com",
  twitter: "https://x.com/mytoken",
  telegram: "https://t.me/mytoken",
  buy: null,
  sell: null
};

const jsonUpload = await pinata.upload.json(metadata);
const cid = jsonUpload.IpfsHash;     // 这就是要传给合约的 CID

console.log('元数据 CID:', cid);
```

其他 IPFS 服务

- **Web3.Storage**（免费）：https://web3.storage/
- **Infura IPFS**：https://infura.io/
- **NFT.Storage**（免费）：https://nft.storage/
- **自己的 IPFS 节点**：运行 `ipfs daemon`

步骤 2：创建代币

使用步骤 1 获得的 CID 创建代币。

newTokenV2 - 标准创建

```
import { FlapPortalWriter, ZERO_ADDRESS } from 'four-flap-meme-sdk';

const writer = new FlapPortalWriter(
  {
    chain: 'BSC',
    rpcUrl: 'https://bsc-dataseed.binance.org'
  },
  privateKey
);

const receipt = await writer.newTokenV2({
  // 基本信息
  name: '我的代币',
  symbol: 'MTK',
  meta: cid,                          // 步骤 1 中获得的 IPFS CID

  // 代币设置
  dexThresh: 1,                       // 1 = FOUR_FIFTHS (80%)
  migratorType: 0,                    // 0 = V3_MIGRATOR
  salt: '0x0000...',                  // CREATE2 的随机盐（使用
predictVanityTokenAddressByChain 计算）
  taxRate: 0,                         // 基点税率（0 = 无税）

  // 💰 创建时购买（可选）
  quoteToken: ZERO_ADDRESS,           // 0x0 = 使用原生代币（BNB/ETH）购买
  quoteAmt: 1n * 10n ** 18n,          // 创建时购买 1 BNB/ETH 的代币
  beneficiary: yourAddress,           // 购买的代币发送到这个地址
  msgValue: 1n * 10n ** 18n,          // 发送 1 BNB/ETH（与 quoteAmt 相同）

  // 可选
  permitData: '0x',                   // Permit 数据（如使用 ERC20 代币购买）
});

console.log('代币已创建:', receipt.transactionHash);
console.log('已自动购买代币发送到:', yourAddress);
```

💡 **创建时购买说明**：

`newTokenV2` 支持在创建代币的同时购买代币，这是一个原子操作：

1. **不购买**（只创建）：

```
quoteToken: ZERO_ADDRESS,
quoteAmt: 0n,               // 设置为 0 表示不购买
beneficiary: yourAddress,
msgValue: 0n                // 不发送 BNB/ETH
```

2. **创建 + 购买**（使用原生币）：

```
  quoteToken: ZERO_ADDRESS,  // 0x0 = 使用 BNB/ETH
  quoteAmt: 1n * 10n ** 18n, // 购买 1 BNB/ETH 的代币
  beneficiary: yourAddress,  // 购买的代币发送到这里
  msgValue: 1n * 10n ** 18n  // 必须与 quoteAmt 相同
```

3. **创建 + 购买**（使用 ERC20 代币）：

```
  quoteToken: '0x...USDT',   // 使用 USDT 购买
  quoteAmt: 100n * 10n ** 18n, // 购买 100 USDT 的代币
  beneficiary: yourAddress,
  permitData: '0x...',       // 必须提供 Permit 签名
  msgValue: 0n               // 使用 ERC20 时不需要发送原生币
```

优势：

- ✅ **原子操作**：创建和购买在同一笔交易完成
- ✅ **节省 gas**：比分两笔交易便宜
- ✅ **防抢跑**：确保创建者能获得初始代币
- ✅ **灵活**：可以使用原生币或 ERC20 代币购买

---

newTokenV3 - 支持扩展功能

V3 版本支持扩展功能，可用于未来的协议升级：

```
const receipt = await writer.newTokenV3({
  // 基本信息
  name: '我的代币 V3',
  symbol: 'MTK3',
  meta: cid,                          // 步骤 1 中获得的 IPFS CID

  // 代币设置
  dexThresh: 1,
  salt: '0x0000...',
  taxRate: 0,
  migratorType: 0,
  quoteToken: ZERO_ADDRESS,
  quoteAmt: 1n * 10n ** 18n,
  beneficiary: yourAddress,
  permitData: '0x',
  msgValue: 1n * 10n ** 18n,

  // V3 新增参数
  extensionID: '0x0000...',           // 扩展功能 ID
  extensionData: '0x'                 // 扩展功能数据（可选）
});

console.log('V3 代币已创建:', receipt.transactionHash);
```

完整示例：创建代币 + 初始购买（解决靓号地址错误）

```
import {
  FlapPortalWriter,
  FlapPortal,
  predictVanityTokenAddressByChain,
  ZERO_ADDRESS
} from 'four-flap-meme-sdk';

// 1. 获取当前 nonce（可选，仅做信息展示）
const portal = new FlapPortal({
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org'
});
const nonce = await portal.nonce();

// 2. 预测符合靓号要求的地址和 salt（新版 API 仅需 salt）
// 方式A：自行提供 salt（bytes32 或任意字符串）
const salt =
'0x000000000000000000000000000000000000000000000000000000000000abcd';
const predictedAddress = predictVanityTokenAddressByChain('BSC', salt,
false);

console.log('✅ 预测地址:', predictedAddress);  // 例如以 8888 结尾
console.log('✅ 使用 salt:', salt);

// 3. 创建代币 + 初始购买
const writer = new FlapPortalWriter(
  { chain: 'BSC', rpcUrl: 'https://bsc-dataseed.binance.org' },
  privateKey
);

const receipt = await writer.newTokenV2({
  name: 'My Token',
  symbol: 'MTK',
  meta: metaCid,                    // IPFS CID
  dexThresh: 1,
  salt: salt,                       // ✅ 使用预测的 salt，避免
VanityAddressRequirementNotMet 错误
  taxRate: 0,
  migratorType: 0,

  // 创建时购买 0.5 BNB 的代币
  quoteToken: ZERO_ADDRESS,
  quoteAmt: 5n * 10n ** 17n,        // 0.5 BNB
  beneficiary: yourAddress,
  msgValue: 5n * 10n ** 17n         // 发送 0.5 BNB
});

console.log('✅ 代币创建成功:', receipt.transactionHash);
```

```
  console.log('✅ 代币地址:', predictedAddress);
  console.log('✅ 已购买 0.5 BNB 的代币到:', yourAddress);
```

参数详情

dexThresh（迁移阈值）

决定在什么供应量时代币迁移到 DEX：

值	名称	阈值	描述
0	TWO_THIRDS	66.67%	在最大供应量的 2/3 时迁移
1	FOUR_FIFTHS	80%	在最大供应量的 4/5 时迁移（推荐）
2	HALF	50%	在最大供应量的 1/2 时迁移
3	_95_PERCENT	95%	在最大供应量的 95% 时迁移
4	_81_PERCENT	81%	在最大供应量的 81% 时迁移
5	_1_PERCENT	1%	在最大供应量的 1% 时迁移（测试）

migratorType（DEX 版本）

值	名称	DEX 版本
0	V3_MIGRATOR	Uniswap V3（推荐）
1	V2_MIGRATOR	Uniswap V2

taxRate（转账税）

- **单位**: 基点（1 bp = 0.01%）
- **范围**: 0 到 10000（0% 到 100%）
- **示例**: 250 = 转账时 2.5% 税
- **推荐**: 大多数情况下使用 0（无税）

salt（CREATE2 盐）

- **用途**: 确定性地址生成
- **类型**: 32 字节十六进制字符串
- **用例**: 靓号地址（见下一节）

靓号地址

靓号地址是具有自定义模式的代币地址，使代币更容易识别和记忆。

什么是靓号地址?

靓号地址是使用 CREATE2 机制，通过调整 `salt` 参数，使得生成的代币合约地址具有特定模式（通常是特定的结尾）。

Flap 平台的默认靓号后缀：

- `8888`: 普通代币的默认靓号后缀（NORMAL）
- `7777`: 有税代币的默认靓号后缀（TAXED）

这两个后缀只是 **Flap 平台的约定俗成**，用于快速识别代币类型：

- 看到 `8888` 结尾 → 可能是普通无税代币
- 看到 `7777` 结尾 → 可能是有交易税的代币

你可以生成任何自定义的靓号地址：

- ✅ `888` - 更短，更容易找到
- ✅ `8888` - Flap 平台默认
- ✅ `88888` - 更长，更稀有（需要更多计算）
- ✅ `666`, `999`, `abc`, `def` - 任何你想要的十六进制模式
- ✅ `yourname` - 甚至可以是英文单词（如果能找到）

计算复杂度：

- 每增加一位后缀，难度增加 16 倍（十六进制）
- `888` (3位) ≈ 几千次迭代
- `8888` (4位) ≈ 几万次迭代
- `88888` (5位) ≈ 上百万次迭代
- `888888` (6位) ≈ 上千万次迭代

预测代币地址

```
import { predictVanityTokenAddressByChain } from 'four-flap-meme-sdk';

// 使用链名称（SDK 自动处理 Portal 和 TokenImpl 地址）
const predictedAddress = predictVanityTokenAddressByChain(
  'BSC',                    // 链名称
  salt                      // 32 字节盐
);

console.log('代币将部署在:', predictedAddress);

// 对于 BSC，如果创建有税代币，可以指定 taxed 参数
const predictedTaxedAddress = predictVanityTokenAddressByChain(
  'BSC',
  salt,
  true                      // taxed = true
);
```

寻找靓号盐

```
import { findSaltEndingByChain } from 'four-flap-meme-sdk';

// 示例 1: 寻找 Flap 平台默认的 8888 结尾（普通代币）
const result1 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: '8888',             // Flap 平台默认（普通代币）
  maxIterations: 1_000_000    // 通常几万次就能找到
});
console.log('找到地址:', result1.address); // 如: 0x...8888
console.log('使用盐:', result1.salt);
console.log('迭代次数:', result1.iterations);

// 示例 2: 寻找更短的靓号（更快）
const result2 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: '888',              // 只需 3 位，更容易找到
  maxIterations: 100_000      // 通常几千次就能找到
});

// 示例 3: 寻找自定义靓号
const result3 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: 'abc',              // 自定义后缀
  maxIterations: 100_000
});

// 示例 4: 寻找更长的靓号（需要更多时间）
const result4 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: '88888',            // 5 位，更稀有
  maxIterations: 10_000_000,  // 可能需要数百万次迭代
  seed:
'0x0000000000000000000000000000000000000000000000000000000000000001'  // 可
选的起始盐
});

// 示例 5: 寻找有税代币的默认靓号（BSC 链）
const result5 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: '7777',             // Flap 平台默认（有税代币）
  taxed: true,                // 指定为有税代币
  maxIterations: 1_000_000
});

// 创建代币时使用找到的盐
import { FlapPortalWriter } from 'four-flap-meme-sdk';

const writer = new FlapPortalWriter(
  { chain: 'BSC', rpcUrl: 'https://bsc-dataseed.binance.org' },
  'YOUR_PRIVATE_KEY'
);

await writer.newTokenV2({
```

```
  name: 'My Token',
  symbol: 'MTK',
  imageFile: imageFile,          // 图片文件（SDK 自动上传到 IPFS）
  description: 'My awesome token',
  website: 'https://example.com',
  dexThresh: 0,
  salt: result1.salt,            // 使用找到的盐
  taxRate: 0,
  migratorType: 0,
  quoteToken: '0x0000000000000000000000000000000000000000',
  quoteAmt: 0n,
  beneficiary: '0x1234...'
});
```

重要提示：

- ⚠ 靓号地址生成是 **CPU 密集型**操作
- ⚠ 在生产环境中，建议 **提前离线计算** 好盐值
- ⚠ 更长的后缀需要 **指数级增长** 的计算时间
- ⚠ 建议使用 **3-4 位** 后缀，既有辨识度又不会太慢
- ✅ 8888 和 7777 只是 Flap 平台的约定，你可以使用 **任何后缀**
- ✅ 后缀是 **十六进制** (0-9, a-f)，大小写不敏感

上传代币元数据

SDK 提供 `uploadTokenMeta(file, meta, apiUrl?)` 工具方法，或可使用第三方 IPFS 服务（Pinata、Infura、Web3.Storage 等）。注意：`FLAP_IPFS_API_URL` 是占位符端点，可能不可用，推荐传入可用的自定义 GraphQL 端点。

```
import { uploadTokenMeta } from 'four-flap-meme-sdk';

const cid = await uploadTokenMeta(imageFile as File, {
  description: '我的神奇代币',
  creator: yourAddress,
  website: 'https://example.com',
  twitter: 'https://x.com/mytoken',
  telegram: 'https://t.me/mytoken'
}, 'https://your-ipfs-graphql.example');
```

使用 Pinata（可选）

你也可以直接使用 Pinata SDK 通过本 SDK 的 `PinataClient` 完成文件/JSON 上传，快速获取 CID。参考 Pinata Quickstart

```
import { PinataClient, ZERO_ADDRESS, pinFileToIPFSWithJWT, pinImageByPath,
pinFileToIPFSWithJWTWeb, pinDataURLWithJWTWeb } from 'four-flap-meme-sdk';
```

```
// 1) 初始化（推荐在服务端使用环境变量存 JWT）
const pinata = new PinataClient({
  jwt: process.env.PINATA_JWT as string,
  gateway: 'example-gateway.mypinata.cloud' // 可选
});

// 2) 直接上传 JSON 获取 CID（也可用 uploadFile 上传图片 File）
const { cid: metaCid } = await pinata.uploadJSON({
  image: 'ipfs://<image_cid>',
  description: '我的代币描述',
  creator: '0xYourAddress',
  website: 'https://example.com'
});

// 3) 在创建代币时传入 meta: cid
const receipt = await writer.newTokenV2({
  name: 'My Token',
  symbol: 'MTK',
  meta: metaCid,
  dexThresh: 1,
  salt: '0x...',
  taxRate: 0,
  migratorType: 0,
  quoteToken: ZERO_ADDRESS,
  quoteAmt: 1n * 10n ** 18n,
  beneficiary: '0xYourAddress',
  permitData: '0x',
  msgValue: 1n * 10n ** 18n
});
```

直接使用 Pinata REST（Node 与 Web）

```
// Node：本地路径或“流/Buffer”上传
const { cid } = await pinFileToIPFSWithJWT(PINATA_JWT,
'/abs/path/img.png');
// 或：
const stream = fs.createReadStream('/abs/path/img.png');
const { cid } = await pinFileToIPFSWithJWT(PINATA_JWT, undefined, stream,
'img.png');

// Node：便捷（路径）
const cid2 = await pinImageByPath(PINATA_JWT, '/abs/path/img.png');

// Web：File/Blob 上传
const input = document.querySelector('input[type=file]') as
HTMLInputElement;
const file = input.files![0];
const { cid: webCid } = await pinFileToIPFSWithJWTWeb(PINATA_JWT, file,
file.name);

// Web：dataURL 上传（如 Canvas.toDataURL()）
```

```
const { cid: dataUrlCid } = await pinDataURLWithJWTWeb(PINATA_JWT,
dataUrl, 'image.png');
```

手动上传元数据（高级用户）

如果需要在创建代币前单独上传元数据：

```
import { uploadTokenMeta } from 'four-flap-meme-sdk';

// 手动上传（使用 Flap 官方免费端点）
const cid = await uploadTokenMeta(
  imageFile,
  {
    description: '我的神奇代币，做 XYZ',
    creator: '0x1234...',
    website: 'https://example.com',
    twitter: 'https://x.com/mytoken',
    telegram: 'https://t.me/mytoken'
  }
);

console.log('元数据 CID:', cid);  // 返回 IPFS CID 如: 'bafkreicwlkp...'

// 使用自定义 IPFS 端点（可选）
const customCid = await uploadTokenMeta(
  imageFile,
  { description: '...', creator: '...' },
  'https://your-custom-ipfs-endpoint.com/graphql'
);
```

其他 IPFS 服务：

- **Pinata**：https://pinata.cloud/
- **Infura IPFS**：https://infura.io/
- **Web3.Storage**：https://web3.storage/
- **NFT.Storage**：https://nft.storage/

注意：第三方服务需要注册和配置，并且可能使用不同的 API 格式。

按地址获取发布元数据（CID/JSON）

如果你只知道 Flap 代币地址，想要取回"发布时上传到 IPFS 的元数据（CID 与 JSON）"，SDK 直接读取代币合约的 `metaURI()`/`meta()`（不扫描区块）；批量查询使用 Multicall3 并发读取。针对 CID 内容类型：

- JSON：返回 `data`
- 图片等非 JSON：返回 `imageUrl`（网关直链）

```
import { getFlapMetaByAddress, getFlapMetasByAddresses } from 'four-flap-
meme-sdk';

// 单个地址：返回 CID；若 CID 指向 JSON 则附带 data
const single = await getFlapMetaByAddress(
  'BSC',                                    // 链：'BSC' | 'BASE' | 'XLAYER'
| 'MORPH'
  'https://bsc-dataseed.binance.org',       // RPC 节点
  '0x...TokenAddress',                      // 代币地址
  { ipfsGateway: 'https://ipfs.io/ipfs/' }  // 可选：自定义 IPFS 网关
);
// single?: { cid: string; data?: any }

// 批量地址：并发查询多个地址（内部使用 Multicall3）
const batch = await getFlapMetasByAddresses(
  'BSC',                                     // 链
  'https://bsc-dataseed.binance.org',        // RPC 节点
  ['0xA...', '0xB...'],                      // 代币地址数组
  {
    // 可选：自定义 Multicall3 地址；未提供时按链使用标准地址 ca11...ca11
    // multicall3: '0xca11bde05977b3631167028862be2a173976ca11',
    // 可选：自定义 IPFS 网关
    // ipfsGateway: 'https://gateway.pinata.cloud/ipfs/'
  }
);
// 返回: Array<{ token: string; cid?: string; data?: any; imageUrl?:
string; error?: string }>
```

说明：

- SDK 先读合约 `metaURI()`，失败再读 `meta()`；批量场景使用 Multicall3 分两轮完成（metaURI 优先、meta 兜底）。
- 单地址方法仅返回 `cid` 与可选 `data`；如需图片直链回退，请使用批量方法，它会在获取 JSON 失败时返回 `imageUrl` 回退；仅在网关获取失败或链上读取失败时返回 `error`。
- 如果返回值形如 `ipfs://...`，SDK 会自动去前缀；可自定义 `ipfsGateway` 以选择更快的网关。

仅获取 CID（底层：getFlapMetaUrisWithMulticall）

如需更底层的 CID 获取（同时也会尝试预取 JSON 或回退图片 URL），可使用：

```
import { getFlapMetaUrisWithMulticall } from 'four-flap-meme-sdk';
import { JsonRpcProvider } from 'ethers';

const provider = new JsonRpcProvider('https://bsc-dataseed.binance.org');
const list = await getFlapMetaUrisWithMulticall(
  provider,
  'BSC',
  ['0xA...', '0xB...'],
  /* 可选 */ undefined,                          // 覆盖 Multicall3 地址
  /* 可选 */ 'https://ipfs.io/ipfs/'             // 覆盖 IPFS 网关
```

```
);
// 返回: Array<{ token: string; cid?: string; data?: any; imageUrl?:
string; error?: string }>
```

Flap 错误处理

SDK 提供了专门的错误解析功能，帮助你理解交易失败的原因：

```
import {
  FlapPortalWriter,
  parseFlapError,
  getFlapErrorMessage,
  getFlapErrorMessageEn,
  FlapErrorCode
} from 'four-flap-meme-sdk';

// 创建 Writer 实例
const writer = new FlapPortalWriter(
  { chain: 'BSC', rpcUrl: 'https://bsc-dataseed.binance.org' },
  'YOUR_PRIVATE_KEY'
);

try {
  await writer.swapExactInput({
    inputToken: '0x0000000000000000000000000000000000000000',
    outputToken: '0x...',
    inputAmount: 1n * 10n ** 18n,
    minOutputAmount: 950n * 10n ** 18n
  }, 1n * 10n ** 18n);
} catch (error) {
  const errorCode = parseFlapError(error);
  const message = getFlapErrorMessage(errorCode);  // 中文错误消息
  console.error('交易失败:', message);

  // 或使用英文
  const messageEn = getFlapErrorMessageEn(errorCode);
  console.error('Transaction failed:', messageEn);
}
```

常见错误代码

错误代码	说明	解决方法
TOKEN_ALREADY_EXISTS	代币已存在	使用不同的盐值或参数
TOKEN_NOT_TRADABLE	代币不可交易	检查代币状态，可能已迁移
ALREADY_ON_DEX	代币已在 DEX 上	在 DEX 上交易而非 Portal
INSUFFICIENT_BALANCE	余额不足	增加钱包余额或减少交易金额

错误代码	说明	解决方法
SLIPPAGE_EXCEEDED	滑点过大	调整 minTokenAmount 或稍后重试
PERMIT_EXPIRED	Permit 签名已过期	重新生成 Permit 签名
INVALID_SIGNATURE	无效的签名	检查签名参数
INVALID_AMOUNT	无效的金额	检查输入金额是否正确
INVALID_QUOTE_TOKEN	无效的报价代币	使用平台支持的报价代币
INVALID_DEX_THRESH	无效的 DEX 阈值	使用 0-5 的有效阈值

Flap 常量定义

SDK 提供了预定义的常量，简化配置：

```
import {
  FlapPortal,
  FLAP_DEFAULT_FEE_RATES,
  FLAP_DEX_THRESHOLDS,
  FLAP_TOTAL_SUPPLY,
  FLAP_IPFS_API_URL,
  FLAP_VANITY_SUFFIX
} from 'four-flap-meme-sdk';

// 创建 FlapPortal（SDK 自动管理合约地址）
const portal = new FlapPortal({ chain: 'BSC', rpcUrl: 'https://...' });
// 支持的链: 'BSC', 'BASE', 'MORPH', 'XLAYER'

// 默认手续费率
const { buy, sell } = FLAP_DEFAULT_FEE_RATES.BSC;
console.log(`买入费率: ${buy * 100}%, 卖出费率: ${sell * 100}%`);
// 输出: 买入费率: 1%, 卖出费率: 1%

// DEX 迁移阈值（百分比形式）
const thresh = FLAP_DEX_THRESHOLDS.FOUR_FIFTHS; // 0.8 (80%)

// 代币总供应量（1 billion with 18 decimals）
console.log('总供应量:', FLAP_TOTAL_SUPPLY);
// 输出: 1000000000000000000000000000n

// IPFS API URL
const ipfsUrl = FLAP_IPFS_API_URL; // 'https://api.flap.sh/graphql'

// 靓号地址后缀
const normalSuffix = FLAP_VANITY_SUFFIX.NORMAL; // '8888'
const taxedSuffix = FLAP_VANITY_SUFFIX.TAXED;   // '7777'
```

支持的链

当前支持的链及其默认配置：

链	买入费率	卖出费率
BSC	1%	1%
BASE	2.5%	2.5%
MORPH	2.5%	2.5%
XLAYER	1.5%	1.5%

注意事项：

- **买入/卖出费率**：由 **Flap Protocol 平台**收取的交易手续费
- 这些费用会在每次交易时自动扣除：
  - **买入时**：从你支付的 BNB/ETH 中扣除手续费，剩余部分用于购买代币
  - **卖出时**：从你应得的 BNB/ETH 中扣除手续费
- 这与代币的 `beneficiary`（受益人）费用不同：
  - `beneficiary` 接收的是迁移到 DEX 后的 LP 收益分成（V3 迁移器）或代币转账税（有税代币）
  - 平台交易手续费由 Flap Protocol 收取，用于平台运营

💰 **示例计算**：

假设在 **BSC** 上交易（手续费率 1%）：

买入示例：

```
你支付:              1.0 BNB
扣除平台手续费:    0.01 BNB  → Flap Protocol 收取
实际用于购买:       0.99 BNB  → 按联合曲线价格购买代币
```

卖出示例：

```
卖出代币后应得:    1.0 BNB    (按联合曲线计算)
扣除平台手续费:    0.01 BNB  → Flap Protocol 收取
你实际收到:        0.99 BNB
```

假设在 **BASE** 或 **MORPH** 上交易（手续费率 2.5%）：

买入示例：

```
你支付:              1.0 ETH
扣除平台手续费:    0.025 ETH → Flap Protocol 收取
实际用于购买:       0.975 ETH → 按联合曲线价格购买代币
```

卖出示例：

```
卖出代币后应得:    1.0 ETH    (按联合曲线计算)
扣除平台手续费:    0.025 ETH → Flap Protocol 收取
你实际收到:        0.975 ETH
```

假设在 **XLAYER** 上交易（手续费率 1.5%）：

买入示例:

```
你支付:             1.0 ETH
扣除平台手续费:     0.015 ETH → Flap Protocol 收取
实际用于购买:       0.985 ETH → 按联合曲线价格购买代币
```

注意: 使用链枚举时会自动应用该链的默认费率，也可以覆盖：

```javascript
const portal = new FlapPortal({
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org',
  buyFeeRate: 0.015,   // 可选：覆盖买入费率为 1.5%
  sellFeeRate: 0.015   // 可选：覆盖卖出费率为 1.5%
});
```

Flap API 参考

FlapPortal 类

构造函数

```javascript
// 使用链枚举（自动处理地址和费率）
const portal = new FlapPortal({
  chain: 'BSC' | 'BASE' | 'XLAYER' | 'MORPH',  // 链名称
  rpcUrl: string,               // RPC 端点
  buyFeeRate?: number,          // 可选：覆盖默认买入费率
  sellFeeRate?: number          // 可选：覆盖默认卖出费率
});
```

读取方法

getTokenV2()

获取基本代币状态（7 个字段）。

```
const state = await portal.getTokenV2(token: Address);
// 返回: TokenStateV2
```

---

getTokenV3()

获取带报价代币信息的代币状态（9 个字段）。

```
const state = await portal.getTokenV3(token: Address);
// 返回: TokenStateV3
```

---

getTokenV4()

获取包含扩展ID的代币状态（10 个字段）。

```
const state = await portal.getTokenV4(token: Address);
// 返回: TokenStateV4
```

---

getTokenV5()

获取完整代币状态（12 个字段，推荐）。

```
const state = await portal.getTokenV5(token: Address);
// 返回: TokenStateV5
```

计算方法

computePriceAndProgress()

计算当前价格和迁移进度。

```
const { price, progress } = portal.computePriceAndProgress(state:
TokenStateV5);
// price: string（每个代币的 ETH）
// progress: string（'0.0000' 到 '1.0000'）
```

---

quoteBuy()

链下买入报价（快速但近似）。

```
const tokensOut = portal.quoteBuy(
  state: TokenStateV5,
  inputEth: string        // 可读格式的金额（如 '1'）
);
// 返回: string（输出代币）
```

---

quoteSell()

链下卖出报价（快速但近似）。

```
const ethOut = portal.quoteSell(
  state: TokenStateV5,
  inputToken: string      // 可读格式的代币金额
);
// 返回: string（输出 ETH）
```

---

quoteExactInput()

通过合约模拟的链上报价（精确）。

```
const outputAmount = await portal.quoteExactInput({
  inputToken: Address,
  outputToken: Address,
  inputAmount: bigint
});
// 返回: Promise<bigint>
```

---

previewBuy()

链上买入报价（专用方法）。

```
const tokenAmount = await portal.previewBuy(
  token: Address,
  ethAmount: bigint
);
// 返回: Promise<bigint> - 预计获得的代币数量
```

---

previewSell()

链上卖出报价（专用方法）。

```
const ethAmount = await portal.previewSell(
  token: Address,
  tokenAmount: bigint
);
// 返回: Promise<bigint> - 预计获得的 ETH 数量
```

---

getFeeRate()

获取平台手续费率。

```
const { buyFeeRate, sellFeeRate } = await portal.getFeeRate();
// 返回: Promise<{buyFeeRate: bigint, sellFeeRate: bigint}>
// 单位: basis points (bps), 例如 250 = 2.5%
```

---

nonce()

获取合约的 nonce 值（用于代币创建计数）。

```
const nonce = await portal.nonce();
// 返回: Promise<bigint>
```

---

version()

获取合约版本信息。

```
const version = await portal.version();
// 返回: Promise<string>
```

---

getLocks()

获取代币的锁仓信息。

```
const locks = await portal.getLocks(token: Address);
// 返回: Promise<bigint[]> - 锁仓 ID 数组
```

FlapPortalWriter 类

用于写操作（需要私钥）。

构造函数

```
// 使用链枚举（SDK 自动处理合约地址）
const writer = new FlapPortalWriter(
  {
    chain: 'BSC' | 'BASE' | 'XLAYER' | 'MORPH',  // 链名称
    rpcUrl: string                                 // RPC 端点
  },
  privateKey: Hex
);
```

---

buy()

买入代币（简化方法）。

```
const receipt = await writer.buy(
  token: Address,            // 目标代币地址
  to: Address,               // 接收地址
  minAmount: bigint,         // 最少获得代币数量（滑点保护）
  msgValue: bigint           // 发送的原生币（BNB/ETH），单位 wei
);
// 返回: Promise<TransactionReceipt>
```

// buyOnCreation 已移除

---

sell()

卖出代币（简化方法）。

```
const receipt = await writer.sell(
  token: Address,            // 待卖出代币地址
  amount: bigint,            // 卖出数量（最小单位，18 位小数）
  minEth: bigint             // 最少获得原生币（wei）
);
// 返回: Promise<TransactionReceipt>
```

---

swapExactInput()

执行代币兑换（通用方法）。

```
const receipt = await writer.swapExactInput(
  {
    inputToken: Address,      // 输入代币地址（ZERO_ADDRESS 表示原生币）
    outputToken: Address,     // 输出代币地址
    inputAmount: bigint,      // 输入数量（最小单位）
    minOutputAmount: bigint, // 最少输出数量（滑点保护）
    permitData?: Hex          // 可选 Permit 数据（ERC20）
  },
  msgValue?: bigint           // 当 inputToken 为原生币时需发送的数额（wei）
);
// 返回: Promise<TransactionReceipt>
```

---

swapExactInputV3()

执行 V3 代币兑换（支持扩展功能）。

```
const receipt = await writer.swapExactInputV3(
  {
    inputToken: Address,      // 输入代币地址
    outputToken: Address,     // 输出代币地址
    inputAmount: bigint,      // 输入数量（最小单位）
    minOutputAmount: bigint, // 最少输出数量（滑点保护）
    permitData?: Hex,         // 可选 Permit
    extensionData?: Hex       // V3 扩展数据
  },
  msgValue?: bigint           // 当 inputToken 为原生币时需要
);
// 返回: Promise<TransactionReceipt>
```

---

newTokenV2()

创建新代币（标准版本）。需传入 IPFS 元数据 CID（meta）。

```
const receipt = await writer.newTokenV2({
  name: string,
  symbol: string,
  meta: string,                // IPFS CID（例如 Pinata 返回的 IpfsHash）
  dexThresh: number,           // 0–5（见 DexThreshType 枚举）
  salt: Hex,                   // 32 字节盐
  taxRate: number,             // 基点（0–10000）
  migratorType: number,        // 0 或 1（见 MigratorType 枚举）
  quoteToken: Address,         // 原生币为 0x0
  quoteAmt: bigint,
  beneficiary: Address,
```

```
  permitData?: Hex,
  msgValue?: bigint
});
// 返回: Promise<TransactionReceipt>
```

---

newTokenV3()

创建新代币（支持扩展功能）。需传入 IPFS 元数据 CID（meta）。

```
const receipt = await writer.newTokenV3({
  name: string,
  symbol: string,
  meta: string,                // IPFS CID（例如 Pinata 返回的 IpfsHash）
  dexThresh: number,
  salt: Hex,
  taxRate: number,
  migratorType: number,
  quoteToken: Address,
  quoteAmt: bigint,
  beneficiary: Address,
  permitData?: Hex,
  extensionID: Hex,           // V3 扩展 ID
  extensionData?: Hex,        // V3 扩展数据
  msgValue?: bigint
});
// 返回: Promise<TransactionReceipt>
```

---

claim()

受益人领取收益。

```
const result = await writer.claim(token: Address);
// 返回: Promise<{ txHash: Hex, tokenAmount: bigint, ethAmount: bigint }>
// 注意：需要从交易回执的事件中解析实际领取数量
```

---

delegateClaim()

委托领取受益人收益。

```
const result = await writer.delegateClaim(token: Address);
// 返回: Promise<{ txHash: Hex, tokenAmount: bigint, ethAmount: bigint }>
```

合约工厂方法

SDK 提供便捷的工厂方法来创建合约实例，无需手动输入地址。

Four.meme 合约

```
import { getTokenManagerV2, getTokenManagerHelper3 } from 'four-flap-meme-sdk';

// 只读合约（不需要私钥）
const tm2 = getTokenManagerV2('BSC', rpcUrl);
const info = await tm2._tokenInfos(tokenAddress);

// 可写合约（需要私钥）
const tm2Writer = getTokenManagerV2Writer('BSC', rpcUrl, privateKey);
await tm2Writer.buyToken(tokenAddress, minTokens, { value: funds });
```

可用方法:

- `getTokenManagerV1(chain, rpcUrl)` - V1 只读
- `getTokenManagerV2(chain, rpcUrl)` - V2 只读
- `getTokenManagerHelper3(chain, rpcUrl)` - Helper3 只读
- `getTokenManagerV1Writer(chain, rpcUrl, privateKey)` - V1 可写
- `getTokenManagerV2Writer(chain, rpcUrl, privateKey)` - V2 可写
- `getTokenManagerHelper3Writer(chain, rpcUrl, privateKey)` - Helper3 可写
- `getTokenManagerAddress(chain, version)` - 获取合约地址

CDPV2 类

getCurve()

创建曲线实例。

```
const curve = CDPV2.getCurve(
  r: number,              // 储备偏移（如 0.1）
  h?: number,             // 高度调整（可选，默认：0）
  k?: number              // 曲线常数（可选，默认：r * 1e9）
);
```

---

estimateSupply()

计算给定储备的供应量。

```
const supply = curve.estimateSupply(reserve: string);
// 返回: Decimal
```

---

estimateReserve()

计算给定供应量的储备。

```
const reserve = curve.estimateReserve(supply: string);
// 返回: Decimal
```

---

price()

计算给定供应量时的价格。

```
const price = curve.price(supply: string);
// 返回: Decimal
```

---

fdv()

计算完全稀释估值。

```
const fdv = curve.fdv(supply: string);
// 返回: Decimal
```

实用函数

uploadTokenMeta()

```
const cid = await uploadTokenMeta(
  apiUrl: string,
  file: File,
  meta: TokenMetaInput
);
// 返回: Promise<string> (IPFS CID)
```

---

buildPermitPiggybackAuto()

自动签名并使用正确的 Flap Portal 代理地址。

```
import { buildPermitPiggybackAuto } from 'four-flap-meme-sdk';

const tokenAddress = '0x...';  // 支持 EIP-2612 Permit 的 ERC20 代币
const value = 1_000n * 10n ** 18n;
const deadline = 2_000_000_000n;
const nonce = 0n;  // 从 token.nonces(owner) 获取

// 自动使用私钥签名并使用正确的 Portal 代理地址
const permitData = await buildPermitPiggybackAuto(
  'BSC',
  privateKey,
  tokenAddress,
  value,
  deadline,
  nonce
);
// 返回: string (hex encoded)
```

完整示例（配合代币兑换）：

```
import { FlapPortalWriter, buildPermitPiggybackAuto } from 'four-flap-
meme-sdk';
import { Contract, JsonRpcProvider } from 'ethers';

// 1. 从代币合约获取 nonce
const provider = new JsonRpcProvider(rpcUrl);
const token = new Contract(tokenAddress, ['function nonces(address) view
returns (uint256)'], provider);
const nonce = await token.nonces(walletAddress);

// 2. 构建 permit 数据（自动签名）
const deadline = BigInt(Math.floor(Date.now() / 1000) + 3600); // 1 小时后过
期
const permitData = await buildPermitPiggybackAuto(
  'BSC',
  privateKey,
  tokenAddress,
  amount,
  deadline,
  nonce
);

// 3. 在兑换中使用 permit（无需授权交易！）
const writer = new FlapPortalWriter(
  { chain: 'BSC', rpcUrl },
  privateKey
);
```

```
await writer.swapExactInput({
  inputToken: tokenAddress,
  outputToken: targetToken,
  inputAmount: amount,
  minOutputAmount: 0n,
  permitData  // ✅ 授权 + 兑换一笔交易完成
});
```

48.club 私有交易

48SP 模式与配置

SDK 同时支持"普通通道（无 48SP）"与"会员通道（48SP）"。

- 通道类型
  - 普通：不带 48SP 签名；调用 `sendPrivateTransaction` 或不附带 `48spSign` 的 `eth_sendBundle`。
  - 会员（48SP）：提供 48SP 签名，享受更高优先级与通过率。
- 如何配置
  - 大多数高阶 Bundle 助手都接受包含以下字段的配置：
    - `spMode?: 'none' | 'timestampPersonalSign' | 'concatTxHash'`（默认 `'timestampPersonalSign'`）
    - `spVMode?: '27_28' | '0_1'`（可选 v 值规范化）
    - `spPrivateKey?: string`（可选 48SP 私钥；未提供时回退为 `privateKeys[0]`）
  - 当 `spMode: 'none'` 时，SDK 不会生成/附带 48SP 签名。
- 底层用法
  - `Club48Client.sendBundle(params, opts)`
  - `Club48Client.sendPrivateTransactionWith48SP(tx, spKey, opts)`
  - `sendBatchPrivateTransactions(txs, spKey, endpoint, opts)`

```
import { Club48Client } from 'four-flap-meme-sdk';

const client = new Club48Client({ endpoint: 'https://puissant-bsc.48.club'
});

// 普通 Bundle（无 48SP）
await client.sendBundle({ txs, maxTimestamp: now + 300 }, { spMode: 'none'
});

// 会员 Bundle（推荐：时间戳 personal_sign）
await client.sendBundle(
  { txs, maxTimestamp: now + 300 },
  { spMode: 'timestampPersonalSign', spPrivateKey: '0x...spKey', spVMode:
```

```
  '27_28' }
  );

  // 单笔私有交易（48SP）
  await client.sendPrivateTransactionWith48SP(signedTx, '0x...spKey', {
    spMode: 'timestampPersonalSign',
    spVMode: '27_28'
  });
```

注意事项

- 默认使用 `'timestampPersonalSign'`。仅当你的服务端明确要求旧版 `concatTxHash` 时再切换。
- 若遇到"invalid signature recovery id"，尝试在 `'27_28'` 与 `'0_1'` 之间切换 `spVMode`。
- 避免同一地址并发提交多个 bundle（会触发替换/加价）。参考 48 文档：`https://docs.48.club/`。

48.club 是什么？

48.club 是一个 MEV（Maximal Extractable Value，最大可提取价值）保护服务，提供：

- 🔒 **私有内存池**：交易在公共内存池中不可见
- ⚡ **快速执行**：直连验证者，更快上链
- 🛡️ **防抢跑**：防止机器人抢跑你的交易
- 📦 **批量交易**：原子性提交多笔交易（全部成功或全部失败）
- 🎯 **Backrun**：在目标交易后立即执行你的交易

私有交易的优势

为什么使用私有交易？

公共内存池的问题：

- 🔴 机器人可以在交易上链前看到你的交易
- 🔴 抢跑者可以复制并先于你执行
- 🔴 你的交易细节对所有人可见
- 🔴 三明治攻击可能窃取你的利润

48.club 私有交易的优势：

- ✅ **隐藏到上链**：交易细节保持私密
- ✅ **无法抢跑**：机器人看不到也无法复制
- ✅ **MEV 保护**：验证者不会对你不利排序
- ✅ **批量原子性**：多钱包同时买卖，全成功或全失败
- ✅ **48 SoulPoint (48SP)**：高级服务，更高优先级

何时使用私有交易？

适用于：

- 🎯 **大额交易**：避免被抢跑造成价格影响

- 💰 **代币发行**：购买新币时不被狙击
- 📦 **协同买入**：多钱包原子性一起买入
- 🤖 **反机器人**：防止公平发行被机器人狙击
- 🔐 **隐私保护**：保持你的交易策略机密

Four.meme 私有交易

SDK 提供在 four.meme 上使用 48.club 私有交易的便捷方法。

单笔私有买入

私密买入代币，不暴露在公共内存池。

```
import { fourPrivateBuy } from 'four-flap-meme-sdk';

// 使用常规私有交易买入
const txHash = await fourPrivateBuy({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...',
  tokenAddress: '0x...',      // 代币合约地址
  funds: '1.0',               // BNB 数量（字符串，如 '1.0' = 1 BNB）
  to: '0x...',                // 可选：接收地址（默认为买家地址）
});

console.log('私有买入交易:', txHash);
```

使用 48 SoulPoint（高级服务）：

```
// 使用 48SP 获得更高优先级和更快执行
const txHash = await fourPrivateBuy({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...',
  tokenAddress: '0x...',
  funds: '1.0',
  spPrivateKey: '0x...',      // 48 SoulPoint 私钥，享受高级服务
});
```

参数说明：

- `rpcUrl`: BSC RPC 节点地址
- `privateKey`: 你的钱包私钥
- `tokenAddress`: 要购买的代币地址（four.meme 内部使用 0x0 代表 BNB）
- `funds`: BNB 数量字符串（人类可读，如 '0.5', '1.0', '10'）
- `to`: 可选接收地址
- `spPrivateKey`: 可选 48SP 密钥，享受高级服务
- `club48Endpoint`: 可选自定义 48.club 端点
- `club48ExplorerEndpoint`: 可选自定义 48.club 浏览器端点

单笔私有卖出

私密卖出代币。

```
import { fourPrivateSell } from 'four-flap-meme-sdk';

// 私有卖出代币
const txHash = await fourPrivateSell({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...',
  tokenAddress: '0x...',
  amount: '1000',            // 代币数量字符串（18 位小数）
  minFunds: 0n,              // 可选：最少收到的 BNB（滑点保护）
});

console.log('私有卖出交易:', txHash);
```

使用 48SP：

```
const txHash = await fourPrivateSell({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...',
  tokenAddress: '0x...',
  amount: '1000',
  spPrivateKey: '0x...',     // 使用 48SP 高级服务
});
```

参数说明：

- `amount`: 代币数量字符串（如 '1000', '50000'）
- `minFunds`: 可选最小 BNB 输出（bigint，默认 0n）

注意：卖出前确保已授权 TokenManager 合约。

批量私有买入（多钱包）

从多个钱包原子性买入代币 - 要么全部成功，要么全部失败。

```
import { fourBatchPrivateBuy } from 'four-flap-meme-sdk';

// 3 个钱包批量买入（可选 48SP；不传 spPrivateKey 时走 bundle 提交）
const success = await fourBatchPrivateBuy({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKeys: [
    '0x...wallet1_key',
    '0x...wallet2_key',
```

```
    '0x...wallet3_key',
  ],
  fundsList: [
    '0.5',    // 钱包 1 用 0.5 BNB 买入
    '1.0',    // 钱包 2 用 1.0 BNB 买入
    '0.3',    // 钱包 3 用 0.3 BNB 买入
  ],
  tokenAddress: '0x...',
  // spPrivateKey: '0x...48sp_key',  // 可选：使用 48SP 单通道
});

console.log('批量买入成功:', success);
```

使用场景：

- 🎯 **协同发行**：多个团队成员一起买入
- 💰 **公平分配**：原子性地将买入分散到多个钱包
- 🔐 **全有或全无**：如果一笔失败，全部回滚（不会部分执行）

重要提示：

- ✅ 不传 `spPrivateKey` 时默认走 bundle 通道；传入则走 48SP 单通道
- ⚠ 所有交易原子性执行（全成功或全失败）
- ⚠ `privateKeys.length` 必须等于 `fundsList.length`

批量私有卖出（多钱包）

从多个钱包原子性卖出代币。

```
import { fourBatchPrivateSell } from 'four-flap-meme-sdk';

// 3 个钱包批量卖出（可选使用 48SP）
const success = await fourBatchPrivateSell({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKeys: [
    '0x...wallet1_key',
    '0x...wallet2_key',
    '0x...wallet3_key',
  ],
  amounts: [
    '1000',   // 钱包 1 卖出 1000 代币
    '2000',   // 钱包 2 卖出 2000 代币
    '500',    // 钱包 3 卖出 500 代币
  ],
  tokenAddress: '0x...',
  // spPrivateKey: '0x...48sp_key',  // 可选：使用 48SP 提升优先级
  minFundsEach: 0n,                // 可选：每个钱包最少收到的 BNB
});

console.log('批量卖出成功:', success);
```

参数说明:

- `amounts`: 代币数量字符串数组
- `minFundsEach`: 可选统一的最小 BNB 输出（适用于所有卖家）
- `spPrivateKey`: 可选的 48SP 密钥（不传则走普通 bundle）

Flap Protocol 私有交易

Flap Protocol 的类似私有交易方法。

单笔私有买入（Flap）

```
import { flapPrivateBuy } from 'four-flap-meme-sdk';

// 在 Flap Protocol 上私有买入
const txHash = await flapPrivateBuy({
  chain: 'BSC',              // 支持: 'BSC', 'BASE', 'XLAYER', 'MORPH'
  privateKey: '0x...',
  tokenAddress: '0x...',
  amountIn: '1.0',           // 原生币数量（BNB/ETH）
  to: '0x...',               // 可选：接收者
  config: {
    rpcUrl: 'https://bsc-dataseed.binance.org',
    club48Endpoint: 'https://puissant-bsc.48.club',  // 可选
  },
});

console.log('Flap 私有买入交易:', txHash);
```

使用 48SP:

```
const txHash = await flapPrivateBuy({
  chain: 'BSC',
  privateKey: '0x...',
  tokenAddress: '0x...',
  amountIn: '1.0',
  spPrivateKey: '0x...',     // 使用 48SP 高级服务
  config: { rpcUrl },
});
```

单笔私有卖出（Flap）

```
import { flapPrivateSell } from 'four-flap-meme-sdk';
```

```
// 在 Flap Protocol 上私有卖出
const txHash = await flapPrivateSell({
  chain: 'BSC',
  privateKey: '0x...',
  tokenAddress: '0x...',
  amount: '1000',             // 代币数量（18 位小数）
  minEth: 0n,                 // 可选：最少收到的 ETH/BNB
  config: { rpcUrl },
});

console.log('Flap 私有卖出交易:', txHash);
```

批量私有买入（Flap）

```
import { flapBatchPrivateBuy } from 'four-flap-meme-sdk';

// Flap 批量买入（需要 48SP）
const success = await flapBatchPrivateBuy({
  chain: 'BSC',
  privateKeys: ['0x...key1', '0x...key2', '0x...key3'],
  amountsIn: ['0.5', '1.0', '0.3'],  // BNB 数量
  tokenAddress: '0x...',
  spPrivateKey: '0x...48sp_key',     // 必需
  config: { rpcUrl },
});

console.log('Flap 批量买入成功:', success);
```

批量私有卖出（Flap）

```
import { flapBatchPrivateSell } from 'four-flap-meme-sdk';

// Flap 批量卖出（可选使用 48SP）
const result = await flapBatchPrivateSell({
  chain: 'BSC',
  privateKeys: ['0x...key1', '0x...key2'],
  amounts: ['1000', '2000'],          // 代币数量
  tokenAddress: '0x...',
  // spPrivateKey: '0x...48sp_key',  // 可选：使用 48SP 提升优先级
  minEthEach: 0n,                    // 可选：每个钱包最少收到的 ETH
  config: { rpcUrl },
});
// 返回结构（根据是否使用 48SP 路径）：
// - 使用 48SP：{ submitted: boolean, sellTxs: string[] }
// - 使用 bundle：{ submitted: true, bundleUuid?: string, status?:
BundleStatus, sellTxs: string[] }
console.log('批量卖出结果:', result);
```

Flap Protocol Bundle 交易

Bundle 交易允许你将多笔交易打包成一个原子操作，要么全部成功，要么全部失败。这对于**创建代币 + 捆绑购买**场景非常有用。

为什么使用 Bundle?

✅ **原子性**：所有交易同时成功或失败
✅ **防抢跑**：创建和购买在同一区块完成
✅ **多钱包协同**：支持多个钱包同时买入
✅ **公平发行**：防止机器人在创建后立即狙击

注意事项

⚠ **仅支持 BSC**：48.club Bundle 服务目前仅在 BSC 链上可用
⚠ **建议开启 48SP**：将 `spMode` 设为 `'timestampPersonalSign'` 并提供 `spPrivateKey`，可在失败时返回详细原因，便于排查
⚠ **gas 兜底**：买入交易在"创建未上链、但同一 Bundle 内紧随创建"时，链上 `estimateGas` 可能回滚；SDK 已在预估失败时为 `swapExactInputBuy` 使用保守兜底 gas。请勿将兜底设得过低（建议 600000 ~ 800000）
⚠ **滑点/余额/窗口**：适当增大 `slippageBps`（如 200）排除滑点问题；确保所有地址余额充足（gas + 买入金额）；可增大 `bundleBlockOffset`（如 200~300）与 `waitTimeoutMs` 以提升通过率
⚠ **参数一致性**：`tokenAddress` 必须与同一 `salt`、同一链配置预测结果一致；`taxed` 与 `taxRate` 需一致（无税用 `taxRate: 0`）
⚠ **同地址多笔**：不推荐从同一地址并发/连发多个 bundle。SDK 已做顺序 nonce 分配，但是否接受取决于验证者策略；建议改用多个地址，或使用"私有批量"接口。

示例（开启 48SP 与合理的保护参数）：

```
const result = await flapCreateTokenWithBundleBuy({
  chain: 'BSC',
  privateKeys: [devKey, ...buyerKeys],
  buyAmounts: ['0.5', '1.0'],
  tokenInfo: { name: 'My', symbol: 'MY', meta: metaCid },
  tokenAddress: predictedAddress,
  config: {
    rpcUrl,
    club48Endpoint: 'https://puissant-bsc.48.club',
    club48ExplorerEndpoint: 'https://puissant-bsc.48.club',
    spMode: 'timestampPersonalSign',
    spPrivateKey: devKey,
    slippageBps: 200,
    bundleBlockOffset: 200,
    waitTimeoutMs: 120000,
    gasLimitMultiplier: 1.2,
    txType: 0
  },
```

```
  dexThresh: 1,
  migratorType: 0,
  taxRate: 0,
  salt
});
```

---

1. 创建代币 + 捆绑购买

创建代币后立即多个钱包原子性买入。

步骤 1：准备 IPFS 元数据

```
import { PinataSDK } from "pinata-web3";

const pinata = new PinataSDK({
  pinataJwt: process.env.PINATA_JWT,
});

// 上传图片
const imageUpload = await pinata.upload.file(imageFile);
const imageCid = imageUpload.IpfsHash;

// 上传元数据
const metadata = {
  image: imageCid,
  description: "我的代币描述",
  creator: devAddress,
  website: "https://example.com",
  twitter: "https://x.com/mytoken",
  telegram: "https://t.me/mytoken",
  buy: null,
  sell: null
};

const jsonUpload = await pinata.upload.json(metadata);
const metaCid = jsonUpload.IpfsHash;
```

步骤 2：预测代币地址

```
import { predictVanityTokenAddressByChain } from 'four-flap-meme-sdk';

// 预测代币地址（新版 API 仅需 salt）
const salt =
'0x0000000000000000000000000000000000000000000000000000000000000abcd';
const predictedAddress = predictVanityTokenAddressByChain('BSC', salt,
false);
```

```
console.log('预测代币地址:', predictedAddress);
console.log('使用盐值:', salt);
```

步骤 3：创建 + 捆绑购买

```
import { flapCreateTokenWithBundleBuy } from 'four-flap-meme-sdk';

const result = await flapCreateTokenWithBundleBuy({
  chain: 'BSC',

  // 私钥数组：[创建者, 买家1, 买家2, ...]
  privateKeys: [
    '0x...dev_key',
    '0x...buyer1_key',
    '0x...buyer2_key',
    '0x...buyer3_key',
  ],

  // 购买金额（BNB）：对应买家1、买家2、买家3
  buyAmounts: ['0.5', '1.0', '0.3'],

  // 代币信息
  tokenInfo: {
    name: 'My Token',
    symbol: 'MTK',
    meta: metaCid,  // 步骤 1 的 IPFS CID
  },

  // 步骤 2 预测的地址
  tokenAddress: predictedAddress,

  // 配置
  config: {
    rpcUrl: 'https://bsc-dataseed.binance.org',
    club48Endpoint: 'https://puissant-bsc.48.club',
    club48ExplorerEndpoint: 'https://puissant-bsc-tx.48.club',
    // 48SP（会员）通道 —— 如需走普通通道可改为 'none'
    spMode: 'timestampPersonalSign',
    // spVMode: '27_28',              // 可选：v 值规范化
    // spPrivateKey: '0x...48sp_key', // 可选：明确指定 48SP 私钥
    // 交易保护参数（可选）
    slippageBps: 100,                 // 买入滑点（基点），默认 100 = 1%
    waitTimeoutMs: 120000             // 等待 bundle 超时时间（毫秒），默认
60000
  },

  // 使用 V2 创建（支持 salt 靓号），以下为必填/推荐参数
  dexThresh: 1,                       // 0~5（推荐 1=80%）
  migratorType: 0,                    // 0=V3, 1=V2
  taxRate: 0,                         // 基点，0 表示无税
  salt: salt                          // 32 字节盐（与 predictedAddress 对
应）
```

```
});

console.log('Bundle UUID:', result.bundleUuid);
console.log('状态:', result.status);
console.log('创建交易:', result.createTx);
console.log('购买交易:', result.buyTxs);
```

执行流程:

1. 创建者发送交易创建代币
2. 买家1、买家2、买家3 同时发送购买交易
3. 所有交易作为 Bundle 原子性执行
4. 如果任何一笔失败，全部回滚

使用场景:

- 🎯 **公平发行**：防止机器人在创建后立即狙击
- 💰 **团队买入**：多个团队成员同时获得初始代币
- 🔐 **协同启动**：确保代币一创建就有初始流动性

---

2. 批量购买（已创建的代币）

对已存在的代币执行多钱包原子性购买。

```
import { flapBatchBuyWithBundle } from 'four-flap-meme-sdk';

const result = await flapBatchBuyWithBundle({
  chain: 'BSC',

  // 私钥数组
  privateKeys: [
    '0x...buyer1_key',
    '0x...buyer2_key',
    '0x...buyer3_key',
  ],

  // 对应的购买金额（BNB）
  buyAmounts: ['0.5', '1.0', '0.3'],

  // 代币地址
  tokenAddress: '0x...',

  // 配置
  config: {
    rpcUrl: 'https://bsc-dataseed.binance.org',
    club48Endpoint: 'https://puissant-bsc.48.club',
    spMode: 'timestampPersonalSign',  // 或 'none'
    // spVMode: '27_28',              // 可选
    // spPrivateKey: '0x...48sp_key', // 可选
  },
```

```
});

console.log('Bundle UUID:', result.bundleUuid);
console.log('状态:', result.status);
console.log('购买交易:', result.buyTxs);
```

参数说明:

- `privateKeys.length` 必须等于 `buyAmounts.length`
- 所有交易原子性执行（全成功或全失败）

---

3. 批量卖出

对代币执行多钱包原子性卖出。

```
import { flapBatchSellWithBundle } from 'four-flap-meme-sdk';

const result = await flapBatchSellWithBundle({
  chain: 'BSC',

  // 私钥数组
  privateKeys: [
    '0x...seller1_key',
    '0x...seller2_key',
    '0x...seller3_key',
  ],

  // 对应的卖出数量（代币，18 位小数）
  sellAmounts: ['1000', '2000', '500'],

  // 代币地址
  tokenAddress: '0x...',

  // 配置
  config: {
    rpcUrl: 'https://bsc-dataseed.binance.org',
    club48Endpoint: 'https://puissant-bsc.48.club',
    spMode: 'timestampPersonalSign',  // 或 'none'
    // spVMode: '27_28',              // 可选
    // spPrivateKey: '0x...48sp_key', // 可选
  },
});

console.log('Bundle UUID:', result.bundleUuid);
console.log('状态:', result.status);
console.log('卖出交易:', result.sellTxs);
```

注意:

- 卖出前确保所有地址已授权 Portal 合约
- 所有交易原子性执行

完整示例：从零发币并防狙击

```
import {
  FlapPortal,
  predictVanityTokenAddressByChain,
  flapCreateTokenWithBundleBuy
} from 'four-flap-meme-sdk';
import { PinataSDK } from "pinata-web3";

// ========== 1. 上传元数据 ==========
const pinata = new PinataSDK({ pinataJwt: process.env.PINATA_JWT });
const imageUpload = await pinata.upload.file(imageFile);
const metadata = {
  image: imageUpload.IpfsHash,
  description: "革命性的代币",
  creator: devAddress,
};
const jsonUpload = await pinata.upload.json(metadata);
const metaCid = jsonUpload.IpfsHash;

// ========== 2. 预测地址 ==========
const portal = new FlapPortal({ chain: 'BSC', rpcUrl: 'https://bsc-
dataseed.binance.org' });
// 新版预测仅需 salt
const salt =
'0x0000000000000000000000000000000000000000000000000000000000abcd';
const predictedAddress = predictVanityTokenAddressByChain('BSC', salt,
false);

// ========== 3. Bundle 创建 + 购买 ==========
const result = await flapCreateTokenWithBundleBuy({
  chain: 'BSC',
  privateKeys: [
    process.env.DEV_KEY!,      // 创建者
    process.env.BUYER1_KEY!,   // 买家1
    process.env.BUYER2_KEY!,   // 买家2
  ],
  buyAmounts: ['0.5', '1.0'],  // 买家购买金额
  tokenInfo: {
    name: 'Revolution Token',
    symbol: 'REV',
    meta: metaCid,
  },
  tokenAddress: predictedAddress,
  config: {
    rpcUrl: 'https://bsc-dataseed.binance.org',
  },
  // 使用 V2 创建（支持 salt 靓号）
```

```
  dexThresh: 1,
  migratorType: 0,
  taxRate: 0,
  salt: salt
});

console.log('✅ 代币创建成功! ');
console.log('代币地址:', result.tokenAddress);
console.log('Bundle 状态:', result.status);
```

优势:

- ✅ 代币创建和购买在同一个区块完成
- ✅ 防止机器人在创建后立即狙击
- ✅ 团队成员公平获得初始代币
- ✅ 如果任何一笔失败，全部回滚（资金安全）

Bundle 状态说明

```
enum BundleStatus {
  PENDING = 'pending',      // 等待执行
  EXECUTED = 'executed',    // 已执行（成功）
  FAILED = 'failed',        // 执行失败
  TIMEOUT = 'timeout',      // 超时
  CANCELLED = 'cancelled',  // 已取消
}
```

查询 Bundle 状态:

```
import { Club48Client } from 'four-flap-meme-sdk';

const client = new Club48Client({
  endpoint: 'https://puissant-bsc.48.club'
});

// 等待 Bundle 完成
const status = await client.waitForBundle(bundleUuid);
console.log('最终状态:', status);
```

常见问题

Q: 为什么创建 + 购买需要预测地址?
A: 因为购买交易需要知道代币地址，但代币还未创建。通过预测地址，可以提前构建购买交易。

Q: 如果预测的地址不对怎么办?
A: 确保使用正确的 nonce 和参数。如果 nonce 在预测后发生变化（其他人创建了代币），地址会不同，Bundle 会失败。

Q: Bundle 失败后资金会退回吗?
A: 会！Bundle 是原子操作，如果任何一笔失败，所有交易都回滚，资金自动退回。

Q: 可以在其他链上使用 Bundle 吗?
A: 目前仅支持 BSC。48.club 未来可能支持更多链。

Q: Bundle 需要 48SP 吗?
A: 是的，Bundle 功能需要 48 SoulPoint 私钥。

底层 48.club API

为需要直接访问 48.club 服务的高级用户提供。

Club48Client 类

```
import { Club48Client } from 'four-flap-meme-sdk';

const client = new Club48Client({
  endpoint: 'https://puissant-bsc.48.club',           // 可选
  explorerEndpoint: 'https://puissant-bsc-tx.48.club' // 可选
});
```

发送单笔私有交易

```
// 不使用 48SP（常规私有交易）
const txHash = await client.sendPrivateTransaction(signedRawTx);

// 使用 48SP（高级服务，更高优先级）
const txHash = await client.sendPrivateTransactionWith48SP(
  signedRawTx,       // 十六进制编码的已签名交易
  spPrivateKey       // 48 SoulPoint 私钥
);

console.log('私有交易已提交:', txHash);
```

发送批量私有交易

```
import { sendBatchPrivateTransactions } from 'four-flap-meme-sdk';

// 批量提交需要 48SP
```

```
const success = await sendBatchPrivateTransactions(
  [signedTx1, signedTx2, signedTx3],  // 已签名交易数组
  spPrivateKey,                       // 48 SoulPoint 私钥
  'https://puissant-bsc.48.club'      // 可选端点
);

console.log('批量已提交:', success);
```

隐秘转账（多跳 Bundle）

通过 Bundle 将多笔转账打包原子执行，实现"隐秘、不可被公共内存池窥探"的多跳转账。支持两种模式：原生币与任意标准 ERC-20。

API（简化版）

```
import { stealthTransfer } from 'four-flap-meme-sdk';

type StealthTransferSimpleParams = {
  rpcUrl: string;                       // JSON-RPC 端点
  chainId: number;                      // 链 ID（BSC 主网=56，测试网=97 等）
  rootPrivateKey: string;               // 根地址私钥（为各 hop 注资）
  finalTo: string;                      // 最终收款地址
  hopCount: number;                     // 中转地址数量（>=1）
  amount: string;                       // 人类可读金额：原生 '0.001' / 代币
'123.45'
  tokenAddress?: string;                // 代币地址：提供则走 ERC-20，多跳同理
  tokenHolderPrivateKey?: string;       // 代币持有者私钥（默认 root）
  bundleEndpoint?: string;              // 48.club 端点（默认
https://puissant-bsc.48.club）
  gasPriceGwei?: string;                // 覆盖 gasPrice（gwei）
  gasLimit?: bigint;                    // 原生注资/转发 gasLimit（默认
25000n）
  blockOffset?: number;                 // 截止区块偏移（默认 +100）
  spMode?: 'none' | 'timestampPersonalSign' | 'concatTxHash' |
'rawTimestamp';
  spPrivateKey?: string;
  spVMode?: '27_28' | '0_1';
};
```

示例（原生币多跳）

```
const { bundleUuid, hopAddresses } = await stealthTransfer({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  chainId: 56,
  rootPrivateKey: '0xROOT...',
  finalTo: '0xFinalTo...',
  hopCount: 4,
```

```
  amount: '0.001',                          // 原生金额（自动 parseEther）
  bundleEndpoint: 'https://puissant-bsc.48.club',
  gasPriceGwei: '0.1'
});
console.log('bundleUuid:', bundleUuid);
console.log('hops:', hopAddresses);
```

示例（ERC-20 多跳，任意标准代币）

```
const { bundleUuid, hopAddresses } = await stealthTransfer({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  chainId: 56,
  rootPrivateKey: '0xROOT...',          // 用于给每个 hop 注入原生 gas
  finalTo: '0xFinalTo...',
  hopCount: 3,
  amount: '1.0',                         // 代币人类可读数量；自动读取 decimals
→ parseUnits
  tokenAddress: '0xToken...',
  tokenHolderPrivateKey: '0xTOKEN_OWNER...', // 可选，不填默认 root
  bundleEndpoint: 'https://puissant-bsc.48.club',
  gasPriceGwei: '0.1'
});
```

说明与限制

- 原生模式：根地址先向第一跳注资（含总 gas+净额），各 hop 依次原生转发；最终到手 amount（自动转成 wei）。
- 代币模式：为每个 hop 注入原生 gas；代币先从持有人转至 hop0，随后各 hop 依次 `transfer`，最终转给 `finalTo`。decimals 将自动读取，`amount` 会按 decimals 转为 wei。
- 代币需符合标准 `transfer(address,uint256) returns (bool)`；带税/黑名单/可暂停等代币可能回退或需更高 gas。
- 单次仅支持一种代币；如需多种代币同捆，可扩展成批量版本。
- root 地址需有足够原生币用于 hop 注资；代币持有者需有足够代币余额。

一键分散（空投）与一键归集（原生 + ERC-20，Bundle）

通过 48.club Bundle 将多笔转账原子提交；默认使用 Legacy 交易（type:0）。原生模式按 parseEther，代币模式自动读取 decimals 后 parseUnits。

通用 API（推荐）

```
import { disperseWithBundle, sweepWithBundle } from 'four-flap-meme-sdk';

// 一键分散（支持相同金额或 amounts 数组逐个金额）
type DisperseParams = {
  rpcUrl: string;
  chainId: number;
```

```
  fromPrivateKey: string;
  recipients: string[];
  amount?: string;                      // 兼容：单一金额（原生 '0.001' / 代币
'123.45'）
  amounts?: string[];                   // 新：与 recipients 一一对应的人类可读
金额
  tokenAddress?: string;                // 不传=原生；传入=ERC20
  bundleEndpoint?: string;
  gasPriceGwei?: string;
  transferGasLimit?: bigint;            // ERC20（默认 65000n）
  nativeGasLimit?: bigint;              // 原生（默认 21000n）
  blockOffset?: number;
  spMode?: 'none' | 'timestampPersonalSign' | 'concatTxHash' |
'rawTimestamp';
  spPrivateKey?: string;
  spVMode?: '27_28' | '0_1';
};

// 说明：若提供 amounts，则优先逐个金额；否则使用单一 amount 自动扩展为与 recipients
同长度。
// 若 recipients 与 amounts 长度不一致将抛出错误。

// 一键归集（按余额比例或固定数量归集到目标地址）
type SweepParams = {
  rpcUrl: string;
  chainId: number;
  sourcePrivateKeys: string[];
  target: string;
  ratioPct?: number;                    // 可选：按余额比例（百分比），100=100%,
50=50%
  amount?: string;                      // 可选：固定数量（向后兼容），原生
'0.001' / 代币 '123.45'
  tokenAddress?: string;                // 不传=原生；传入=ERC20
  bundleEndpoint?: string;
  gasPriceGwei?: string;
  transferGasLimit?: bigint;            // ERC20（默认 65000n）
  nativeGasLimit?: bigint;              // 原生（默认 21000n）
  blockOffset?: number;
  skipIfInsufficient?: boolean;         // 默认 true（余额不足跳过）
  spMode?: 'none' | 'timestampPersonalSign' | 'concatTxHash' |
'rawTimestamp';
  spPrivateKey?: string;
  spVMode?: '27_28' | '0_1';
};
```

示例：原生币分散

```
const { bundleUuid } = await disperseWithBundle({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  chainId: 56,
  fromPrivateKey: '0xFROM...',
```

```
  recipients: ['0xA...', '0xB...', '0xC...'],
  amount: '0.001',                    // 每人 0.001 BNB
  gasPriceGwei: '1'
});
```

示例：ERC-20 分散

```
const { bundleUuid } = await disperseWithBundle({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  chainId: 56,
  fromPrivateKey: '0xFROM...',
  recipients: ['0xA...', '0xB...', '0xC...'],
  amount: '10',                       // 每人 10 个代币
  tokenAddress: '0xToken...',
  gasPriceGwei: '1'
});

// 示例：原生币分散（逐个金额）
const { bundleUuid: bundleUuid2 } = await disperseWithBundle({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  chainId: 56,
  fromPrivateKey: '0xFROM...',
  recipients: ['0xA...', '0xB...', '0xC...'],
  amounts: ['0.001', '0.002', '0.0005'], // 分别对应 A/B/C 的 BNB 数量
  gasPriceGwei: '1'
});

// 示例：ERC-20 分散（逐个金额）
const { bundleUuid: bundleUuid3 } = await disperseWithBundle({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  chainId: 56,
  fromPrivateKey: '0xFROM...',
  recipients: ['0xA...', '0xB...'],
  amounts: ['5', '12.34'],            // 分别对应 A/B 的代币数量
  tokenAddress: '0xToken...',
  gasPriceGwei: '1'
});
```

示例：原生币归集

```
const { bundleUuid } = await sweepWithBundle({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  chainId: 56,
  sourcePrivateKeys: ['0xK1...', '0xK2...'],
  target: '0xTARGET...',
  amount: '0.005',                    // 每个来源地址归集 0.005 BNB
  skipIfInsufficient: true
});
```

示例：ERC-20 归集

```
const { bundleUuid } = await sweepWithBundle({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  chainId: 56,
  sourcePrivateKeys: ['0xK1...', '0xK2...', '0xK3...'],
  target: '0xTARGET...',
  amount: '5',                          // 每个来源地址归集 5 个代币
  tokenAddress: '0xToken...',
  skipIfInsufficient: true,
  gasPriceGwei: '1'
});
```

备注：仍保留 ERC-20 专用方法 `disperseErc20WithBundle` 与 `sweepErc20WithBundle` 以兼容旧代码。

注意：

- 建议提供 48SP（spPrivateKey）以获得更高优先级与失败详情；
- 代币若带税/黑名单/可暂停，可能需要提高 transferGasLimit（如 90000~120000）。

获取最低 Gas 价格

```
const gasPrice = await client.getMinGasPrice();
console.log('最低 gas 价格:', gasPrice);
```

使用此方法确保你的交易满足最低 gas 价格要求。

签署 48 SoulPoint

```
import { Club48Client } from 'four-flap-meme-sdk';

// 用 48SP 签名交易
const signature = Club48Client.sign48SoulPoint(
  spPrivateKey,              // 48 SoulPoint 私钥
  [signedTx1, signedTx2]     // 已签名交易数组
);

console.log('48SP 签名:', signature);
```

Backrun 助手

在目标交易后立即执行交易。

```
import { Club48Client, sendBackrunBundle } from 'four-flap-meme-sdk';

const client = new Club48Client();

const bundleUuid = await sendBackrunBundle(client, {
  backrunTarget: '0xTARGET_TX_HASH',          // 要 backrun 的交易
  txs: [signedTx1, signedTx2],                // 你的交易
  soulPointSignature: Club48Client.sign48SoulPoint(spPrivateKey,
[signedTx1, signedTx2]),
  maxTimestamp: Math.floor(Date.now() / 1000) + 300  // 截止时间（5 分钟）
});

console.log('Backrun bundle UUID:', bundleUuid);

// 等待 bundle 结果
const status = await client.waitForBundle(bundleUuid);
console.log('Bundle 状态:', status);
```

使用场景：

- 在代币发行后立即执行
- 对特定链上事件作出反应
- 保证在目标交易后的排序

Permit 自动签名：支持读取代币 name

`buildPermitPiggybackAuto` 现支持链上读取代币名称或手动覆盖：

```
import { buildPermitPiggybackAuto } from 'four-flap-meme-sdk';

const permitData = await buildPermitPiggybackAuto(
  'BSC',
  privateKey,
  tokenAddress,
  value,
  deadline,
  nonce,
  { rpcUrl, tokenNameOverride: 'MyToken' } // 可选
);
```

附录

常量与地址

SDK 提供所有支持网络的预配置地址：

```
import { CHAIN } from 'four-flap-meme-sdk';

// 使用链枚举调用 SDK 方法（地址由 SDK 内部处理）
// 示例：
// const tm2 = TM2.connectByChain('BSC', rpcUrl);
// const portal = new FlapPortal({ chain: 'BSC', rpcUrl });
// await tryBuy('BSC', rpcUrl, token, amount, funds);

// 链信息
console.log(CHAIN.BSC.chainId);          // 56
console.log(CHAIN.BASE.chainId);         // 8453
console.log(CHAIN.XLAYER.chainId);       // 196
console.log(CHAIN.MORPH.chainId);        // 2818
console.log(CHAIN.ARBITRUM_ONE.chainId); // 42161
```

BigInt 与单位

理解 BigInt

所有金额都以 wei 为单位（1 ETH = 10^18 wei）。使用带 `n` 后缀的 `bigint` 类型。

```
// 1 BNB/ETH (wei)
const oneEth = 1n * 10n ** 18n;

// 0.5 BNB/ETH (wei)
const halfEth = 5n * 10n ** 17n;

// 使用 ethers.js 转换
import { formatEther, parseEther } from 'ethers';
const readable = formatEther(1000000000000000000n);  // '1.0'
const wei = parseEther('1.0');                        //
1000000000000000000n
```

常用模式

模式 1: 估算 → 执行

始终在执行昂贵操作前估算：

```
// 1. 估算
const estimate = await tryBuy('BSC', rpcUrl, token, 0n, 1n * 10n ** 18n);
console.log('将获得:', estimate.estimatedAmount, '代币');

// 2. 检查是否可接受
```

```
if (estimate.estimatedAmount < 1000n * 10n ** 18n) {
  throw new Error('代币不够! ');
}

// 3. 执行
await tradeBuy('BSC', rpcUrl, token, {
  type: 'funds',
  funds: estimate.amountFunds,
  minAmount: estimate.estimatedAmount * 95n / 100n  // 5% 滑点
});
```

模式 2: 授权 → 卖出

卖出需要两步：

```
const tokenAddress = '0x...';
const amount = 1_000n * 10n ** 18n;

// 步骤 1: 授权（仅需一次，或当授权额度不足时）
await ensureSellApproval('BSC', rpcUrl, privateKey, tokenAddress,
yourAddress, amount);

// 步骤 2: 卖出
await tradeSell('BSC', rpcUrl, tokenAddress, {
  type: 'direct',
  amount: amount,
  minFunds: 0n
});
```

模式 3: 错误处理

始终将交易操作包装在 try-catch 中：

```
import { parseFourError } from 'four-flap-meme-sdk';

try {
  await tradeBuy(...);
} catch (error) {
  const { code, message } = parseFourError(error);

  switch (code) {
    case 'Slippage':
      console.error('价格变动太大，用更高滑点重试');
      break;
    case 'More BNB':
      console.error('BNB 不足，添加更多资金');
      break;
    case 'SO':
      console.error('订单太小，增加金额');
```

```
        break;
      default:
        console.error('未知错误:', message);
    }
  }
```

常见问题

问：four.meme 和 Flap Protocol 有什么区别?

答: 两者都是代币发行平台，但:

- **four.meme**: 更简单，专注 BSC，REST API + 合约
- **Flap Protocol**: 高级 CDPV2 曲线，多链，更多自定义选项

问：发币时可以自己先购买吗（preSale）？

答: 可以！使用 `preSale` 参数:

```
const result = await createTokenFlow({
  // ... 其他参数
  payload: {
    name: '我的代币',
    shortName: 'MTK',
    // ... 其他字段
    preSale: '0.5',  // ✅ 发币时自己先购买 0.5 BNB 的代币
  }
});
```

工作原理:

1. 你设置 `preSale: '0.5'`
2. SDK 自动在创建交易时发送 0.5 BNB
3. 合约部署代币后，立即用这 0.5 BNB 从联合曲线买入代币
4. 购买的代币自动发送到你的地址

常见用途:

- 防止被狙击（自己先占据初始供应）
- 测试代币交易功能
- 作为项目方持有初始代币

注意：确保钱包有足够的 BNB（需要 gas fee + preSale 金额）

问：为什么卖出需要两笔交易?

答: ERC20 安全特性。第一笔 `approve()`，第二笔 `sell()`。使用 `ensureSellApproval()` 自动处理。

问：什么是好的滑点容忍度?

答: 正常市场 1-5%。波动代币 10-20%。

问：DEX 迁移后还能交易吗?

答: 不能。迁移后，在 DEX（PancakeSwap、Uniswap）上交易。联合曲线关闭。

问：买入时出现"More BNB"错误

答: 调用 `tradeBuy()` 时使用 `estimate.amountFunds`（不是 `estimatedCost`）。

问：卖出时交易失败无错误

答: 你忘记授权了。始终在 `tradeSell()` 前调用 `ensureSellApproval()`。

许可与支持

- **许可**: MIT
- **GitHub**: 报告问题或贡献
- **文档**: 本 README

🎉 **你现在已准备好在 four.meme 和 Flap Protocol 上构建了！**