计算 TVL 文档

简介

本文档主要介绍三种代币合约 TVL 的拉取方式。分别是

- AAVE V2
- AAVE V3
- Compound V2

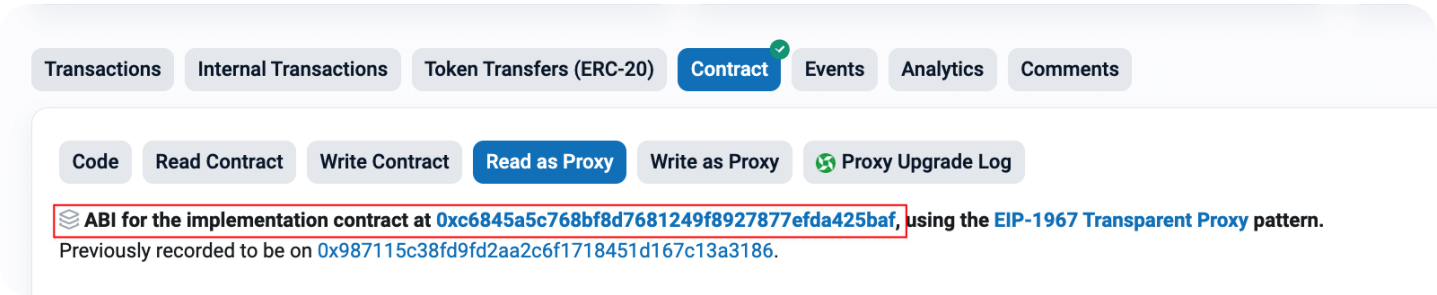
前置知识

ABI 获取

代理合约的 ABI 获取方式

1. 搜索 0x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9 代理合约信息
2. 搜索 **0xc6845a5c768bf8d7681249f8927877efda425baf** 逻辑合约信息
3. 找到 Contract —> Code -> 下拉找到 Contract ABI
4. 复制下来保存到 json 文件中引用即可。

以 AAVE V2 Lending Pool 合约为例，通过以太坊浏览器查询得到页面如下：

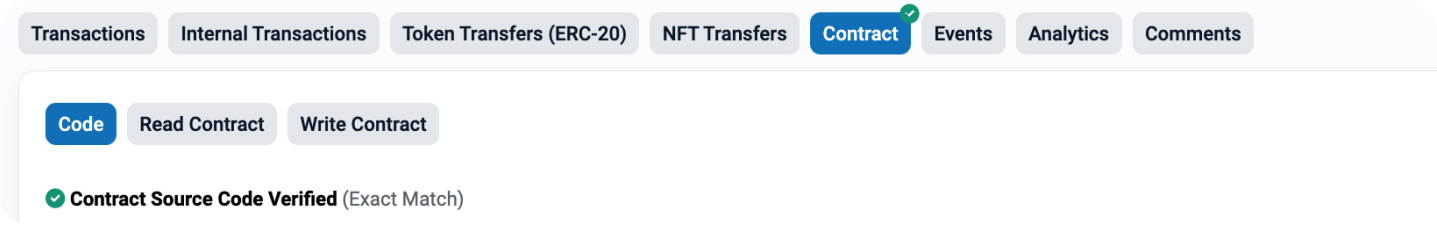


其真正的逻辑合约实现是位于合约地址 **0xc6845a5c768bf8d7681249f8927877efda425baf** ，我们获取该地址的 ABI 作为创建合约实例的参数。然后使用代理合约地址 0x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9 去代理调用真正的逻辑合约。

```
const lendingContractInstance = new web3.eth.Contract(lendingAbi, '0x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9');
```

普通合约的 ABI 获取方式

和代理合约 ABI 获取流程类似，只不过需要再去查询逻辑合约了。



1. 搜索合约
2. Contract -> Code -> Contract ABI

ERC20 ABI

```
[
    {
        "constant": true,
        "inputs": [],
        "name": "name",
        "outputs": [
            {
                "name": "",
                "type": "string"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": false,
        "inputs": [
            {
                "name": "_spender",
                "type": "address"
            },
            {
                "name": "_value",
                "type": "uint256"
            }
        ],
        "name": "approve",
        "outputs": [
            {
                "name": "",
```

```
                "type": "bool"
            }
        ],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [],
        "name": "totalSupply",
        "outputs": [
            {
                "name": "",
                "type": "uint256"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": false,
        "inputs": [
            {
                "name": "_from",
                "type": "address"
            },
            {
                "name": "_to",
                "type": "address"
            },
            {
                "name": "_value",
                "type": "uint256"
            }
        ],
        "name": "transferFrom",
        "outputs": [
            {
                "name": "",
                "type": "bool"
            }
        ],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
```

```
    },
    {
        "constant": true,
        "inputs": [],
        "name": "decimals",
        "outputs": [
            {
                "name": "",
                "type": "uint8"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [
            {
                "name": "_owner",
                "type": "address"
            }
        ],
        "name": "balanceOf",
        "outputs": [
            {
                "name": "balance",
                "type": "uint256"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [],
        "name": "symbol",
        "outputs": [
            {
                "name": "",
                "type": "string"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
```

```
    },
    {
        "constant": false,
        "inputs": [
            {
                "name": "_to",
                "type": "address"
            },
            {
                "name": "_value",
                "type": "uint256"
            }
        ],
        "name": "transfer",
        "outputs": [
            {
                "name": "",
                "type": "bool"
            }
        ],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [
            {
                "name": "_owner",
                "type": "address"
            },
            {
                "name": "_spender",
                "type": "address"
            }
        ],
        "name": "allowance",
        "outputs": [
            {
                "name": "",
                "type": "uint256"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
```

```
    {
        "payable": true,
        "stateMutability": "payable",
        "type": "fallback"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "name": "owner",
                "type": "address"
            },
            {
                "indexed": true,
                "name": "spender",
                "type": "address"
            },
            {
                "indexed": false,
                "name": "value",
                "type": "uint256"
            }
        ],
        "name": "Approval",
        "type": "event"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "name": "from",
                "type": "address"
            },
            {
                "indexed": true,
                "name": "to",
                "type": "address"
            },
            {
                "indexed": false,
                "name": "value",
                "type": "uint256"
            }
        ],
        "name": "Transfer",
```

```
        "type": "event"
    }
]
```

ERC20(MKR) ABI

后面处理 Compound 合约时，由于 MKR 的 symbol 方法返回值是 bytes32，和一般的 erc20 代币合约不同（返回 string），所以需要提供 ABI 单独处理。

```
[{"constant":true,"inputs":[],"name":"symbol","outputs":[{"name":"","type":"byte
```

阅读合约提供的 API

阅读合约的 API

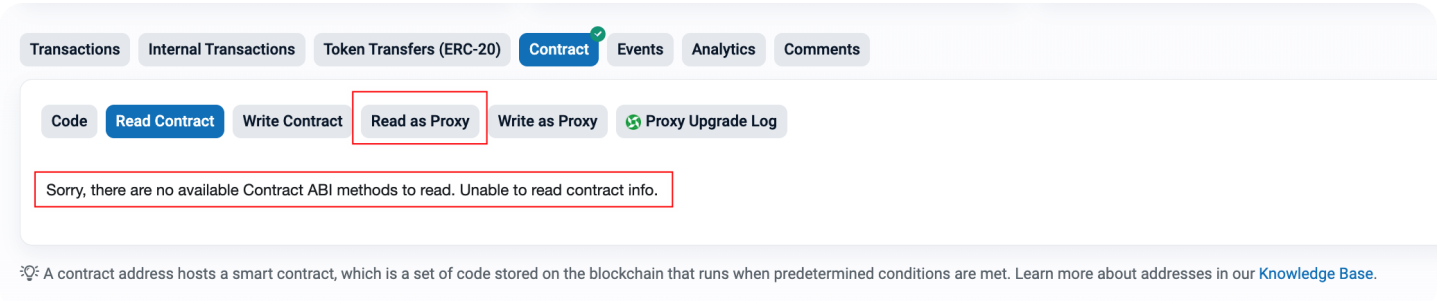
1. 在以太坊浏览器中搜索合约信息
2. 依次点击 Contract -> Read Contract

阅读代理合约的 API

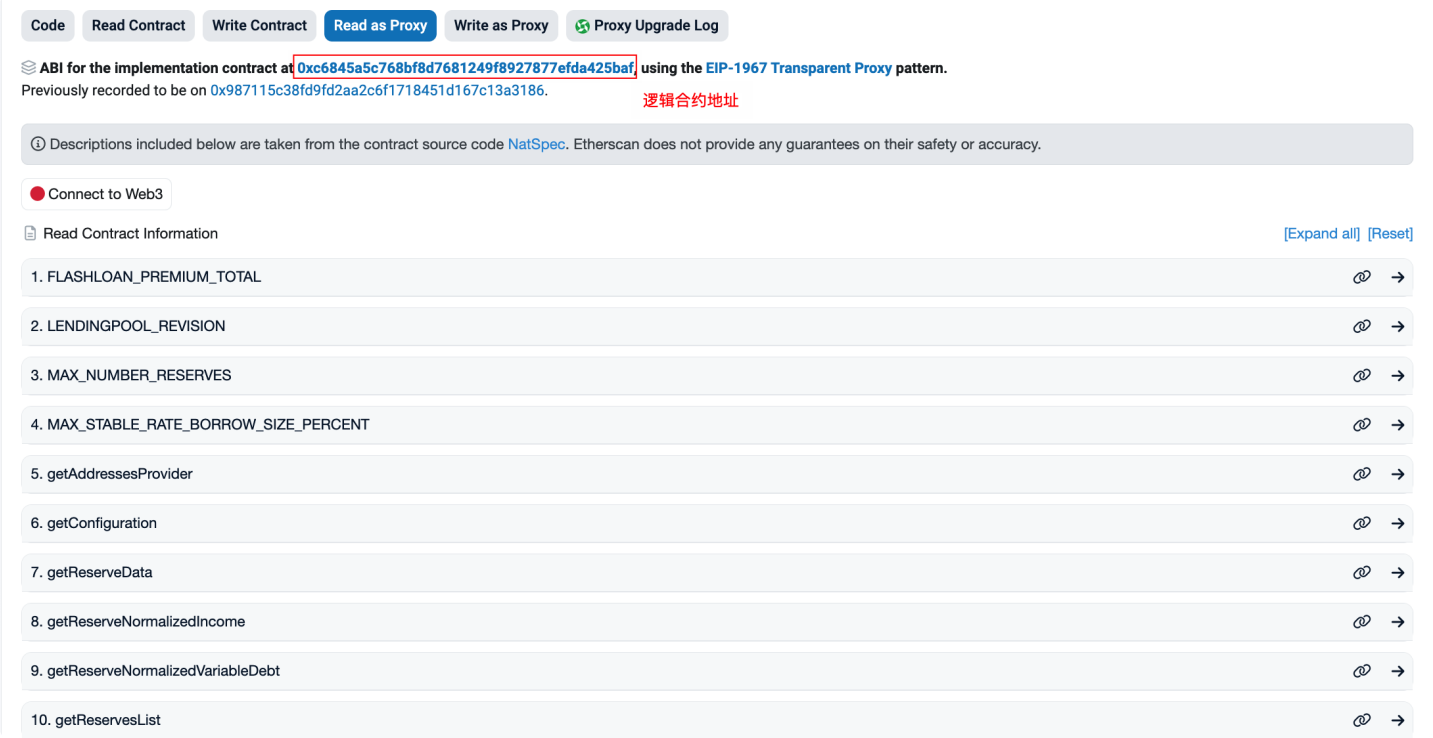
1. 在以太坊浏览器中搜索合约信息
2. 点击 Contract -> Read as Proxy 阅读 API

代理合约的 Read Contract 选项不会提供任何的 API 方法详情，需要去 Read as Proxy 选项中获取（实际上获取的是逻辑合约提供的 API）。

- Read Contract



- Read as Proxy



AAVE V2

AAVE V2 Lending Pool： 0x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9（proxy contract）

AAVE V2 Lending Pool Logic Contract：**0xc6845a5c768bf8d7681249f8927877efda425baf**（提供 ABI）

AAVE V2 Price Oracle： 0xA50ba011c48153De246E5192C8f9258A2ba79Ca9（价格预言机）

Uniswap V2：Route2：0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D（获取相对于 USDT 或者 ETH 的价格）

AAVE V3

AAVE Pool V3 : 0x87870Bca3F3fD6335C3F4ce8392D69350B4fA4E2

AAVE Pool V3 Logic Contract : 0xF1Cd4193bbc1aD4a23E833170f49d60f3D35a621

AAVE V3 Price Oracle : 0x54586bE62E3c3580375aE3723C145253060Ca0C2

Compound V2

Compound Comptroller : 0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B

Compound Price Oracle : 0x50ce56A3239671Ab62f185704Caedf626352741e

获取 TVL 脚本

AAVE V2

脚本依赖

1. npm init
2. npm install web3
3. 创建 json 目录，并将所有需要的 ABI 放入到 json 目录中（所有需要的 ABI 都可以根据上述合约地址获取）

脚本方法调用

- getTokensTVL(address);

获取指定 Token 的 TVL，如果当前内存中没有存储该地址 TVL 的数据，就调用 main 函数获取所有 Token 的 TVL，然后返回 address 对应的 TVL；

- main();

打印所有 Token 的 TVL并以 map 的形式将 Token 的 TVL 存储到 map 中。

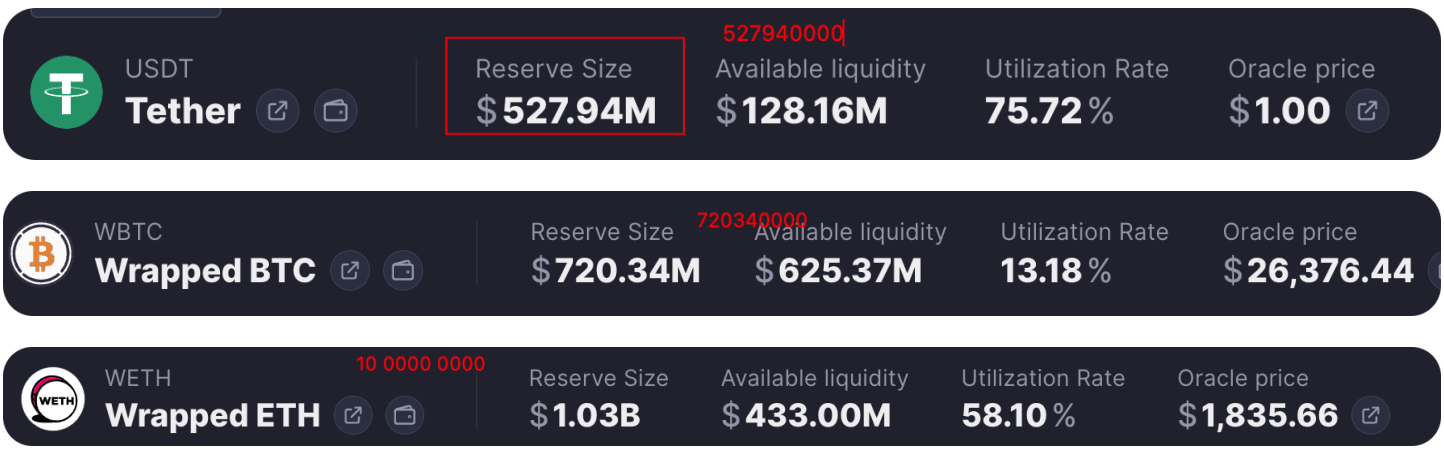
结果比对

官方 TVL 查询地址：https://app.aave.com/markets/?marketName=proto_mainnet

- 脚本输出

```
symbol: aUSDT totalSupply: 528939877120315 tvl: 526542977.04023826
symbol: aWBTC totalSupply: 2730982939358 tvl: 718448482.2721442
symbol: aWETH totalSupply: 563199533726709431615018 tvl: 1031130868.412428
```

- 官方提供的 TVL 数据



AAVE V3

脚本依赖

同 AAVE V2

脚本方法调用

- main();

获取所有 AAVE V3 支持代币的 TVL。

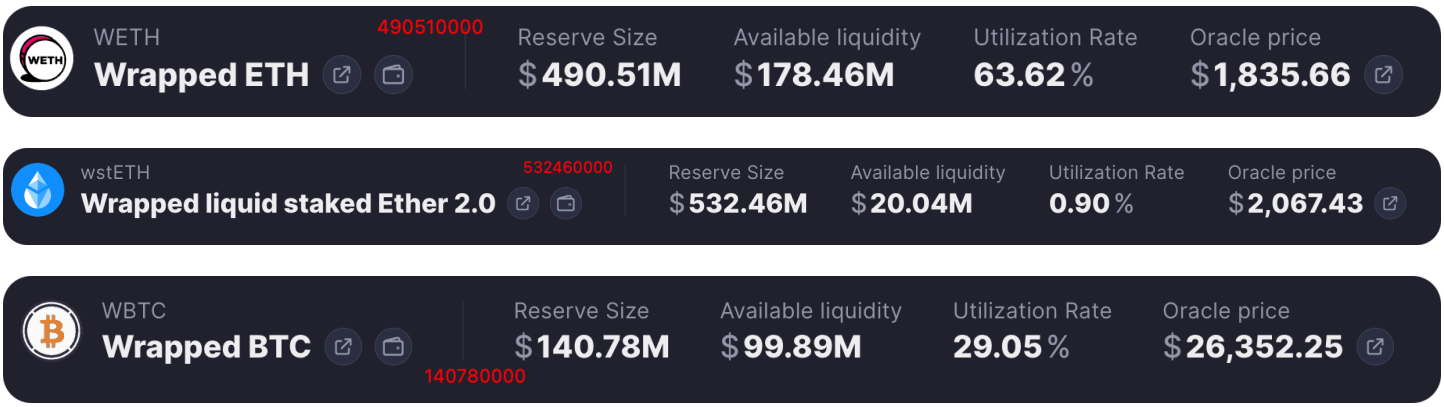
结果比对

官方 TVL 查询网址：https://app.aave.com/markets/?marketName=proto_mainnet_v3

- 脚本输出

```
tokens = aEthWETH, price = 183565520000, tvl = 490317173.2979192
tokens = aEthwstETH, price = 206743493457, tvl = 532461838.6472047
tokens = aEthWBTC, price = 2635224569500, tvl = 140783428.0750146
```

- 官方提供的 TVL



Compound V2

脚本依赖

同 AAVE V2

脚本方法调用

- main();

获取所有 AAVE V3 支持代币的 TVL。

- getSpecifyMarketsTVL(address);

获取指定 Token 的 TVL。

结果比对

Compound V2 Markets TVL 查询地址：https://app.compound.finance/markets/?market=v2

- 脚本输出

```
cBAT collateralFactorMantissa: 0.6
ctoken =  cBAT oraclePrice =  194338 tvl =  28060630.402532615

cDAI collateralFactorMantissa: 0.835
ctoken =  cDAI oraclePrice =  999969 tvl =  274531145.85762686
```

- 官方提供的 BAT 和 DAI 的 TVL