

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：



Transactions Internal Transactions Token Transfers (ERC-20) **Contract** Events Analytics Comments

Code Read Contract Write Contract **Read as Proxy** Write as Proxy Proxy Upgrade Log

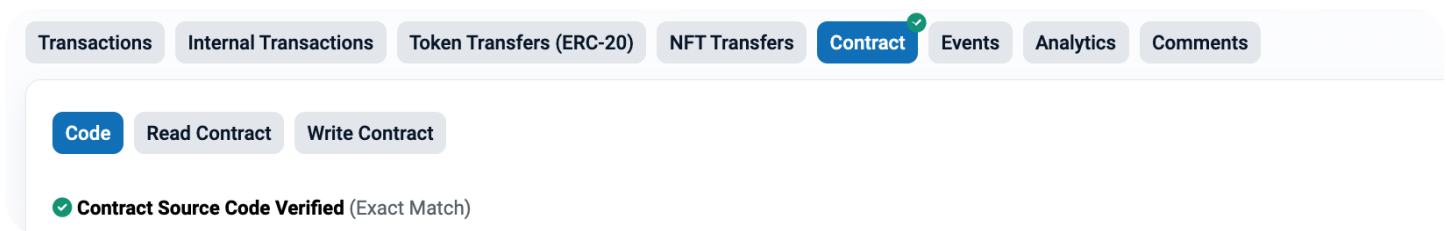
 ABI for the implementation contract at [0xc6845a5c768bf8d7681249f8927877efda425baf](#), using the EIP-1967 Transparent Proxy pattern.
Previously recorded to be on [0x987115c38fd9fd2aa2c6f1718451d167c13a3186](#).

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。

```
1 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

```
1  [
2      {
3          "constant": true,
4          "inputs": [],
5          "name": "name",
6          "outputs": [
7              {
8                  "name": "",
9                  "type": "string"
10             }
11         ],
12         "payable": false,
13         "stateMutability": "view",
14         "type": "function"
15     },
16     {
17         "constant": false,
18         "inputs": [
19             {
20                 "name": "_spender",
21                 "type": "address"
22             },
23             {
24                 "name": "_value",
25                 "type": "uint256"
26             }
27         ],
28         "name": "approve",
29         "outputs": [
30             {
31                 "name": "",
```

```
32         "type": "bool"
33     }
34 ],
35     "payable": false,
36     "stateMutability": "nonpayable",
37     "type": "function"
38 },
39 {
40     "constant": true,
41     "inputs": [],
42     "name": "totalSupply",
43     "outputs": [
44         {
45             "name": "",
46             "type": "uint256"
47         }
48     ],
49     "payable": false,
50     "stateMutability": "view",
51     "type": "function"
52 },
53 {
54     "constant": false,
55     "inputs": [
56         {
57             "name": "_from",
58             "type": "address"
59         },
60         {
61             "name": "_to",
62             "type": "address"
63         },
64         {
65             "name": "_value",
66             "type": "uint256"
67         }
68     ],
69     "name": "transferFrom",
70     "outputs": [
71         {
72             "name": "",
73             "type": "bool"
74         }
75     ],
76     "payable": false,
77     "stateMutability": "nonpayable",
78     "type": "function"
```

```
79     },
80     {
81         "constant": true,
82         "inputs": [],
83         "name": "decimals",
84         "outputs": [
85             {
86                 "name": "",
87                 "type": "uint8"
88             }
89         ],
90         "payable": false,
91         "stateMutability": "view",
92         "type": "function"
93     },
94     {
95         "constant": true,
96         "inputs": [
97             {
98                 "name": "_owner",
99                 "type": "address"
100             }
101         ],
102         "name": "balanceOf",
103         "outputs": [
104             {
105                 "name": "balance",
106                 "type": "uint256"
107             }
108         ],
109         "payable": false,
110         "stateMutability": "view",
111         "type": "function"
112     },
113     {
114         "constant": true,
115         "inputs": [],
116         "name": "symbol",
117         "outputs": [
118             {
119                 "name": "",
120                 "type": "string"
121             }
122         ],
123         "payable": false,
124         "stateMutability": "view",
125         "type": "function"
```

```
126     },
127     {
128         "constant": false,
129         "inputs": [
130             {
131                 "name": "_to",
132                 "type": "address"
133             },
134             {
135                 "name": "_value",
136                 "type": "uint256"
137             }
138         ],
139         "name": "transfer",
140         "outputs": [
141             {
142                 "name": "",
143                 "type": "bool"
144             }
145         ],
146         "payable": false,
147         "stateMutability": "nonpayable",
148         "type": "function"
149     },
150     {
151         "constant": true,
152         "inputs": [
153             {
154                 "name": "_owner",
155                 "type": "address"
156             },
157             {
158                 "name": "_spender",
159                 "type": "address"
160             }
161         ],
162         "name": "allowance",
163         "outputs": [
164             {
165                 "name": "",
166                 "type": "uint256"
167             }
168         ],
169         "payable": false,
170         "stateMutability": "view",
171         "type": "function"
172     },
```

```
173     {
174         "payable": true,
175         "stateMutability": "payable",
176         "type": "fallback"
177     },
178     {
179         "anonymous": false,
180         "inputs": [
181             {
182                 "indexed": true,
183                 "name": "owner",
184                 "type": "address"
185             },
186             {
187                 "indexed": true,
188                 "name": "spender",
189                 "type": "address"
190             },
191             {
192                 "indexed": false,
193                 "name": "value",
194                 "type": "uint256"
195             }
196         ],
197         "name": "Approval",
198         "type": "event"
199     },
200     {
201         "anonymous": false,
202         "inputs": [
203             {
204                 "indexed": true,
205                 "name": "from",
206                 "type": "address"
207             },
208             {
209                 "indexed": true,
210                 "name": "to",
211                 "type": "address"
212             },
213             {
214                 "indexed": false,
215                 "name": "value",
216                 "type": "uint256"
217             }
218         ],
219         "name": "Transfer",
```

```
220         "type": "event"
221     }
222 ]
```

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），所以需要单独提供 ABI 单独处理。

```
1 [{"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 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

The screenshot shows the Etherscan interface for a contract. The 'Contract' tab is selected in the top navigation bar. Below the navigation bar, there are several buttons: 'Code', 'Read Contract', 'Write Contract', 'Read as Proxy', 'Write as Proxy', and 'Proxy Upgrade Log'. The 'Read as Proxy' button is highlighted with a red box. Below these buttons, a message box states: 'Sorry, there are no available Contract ABI methods to read. Unable to read contract info.'

- Read as Proxy

CodeRead ContractWrite ContractRead as ProxyWrite as ProxyProxy Upgrade Log

ABI for the implementation contract at **0xc6845a5c768bf8d7681249f8927877efda425baf** using the EIP-1967 Transparent Proxy pattern.
Previously recorded to be on **0x987115c38fd9fd2aa2c6f1718451d167c13a3186**.

逻辑合约地址

ⓘ Descriptions included below are taken from the contract source code [NatSpec](#). Etherscan does not provide any guarantees on their safety or accuracy.

Connect to Web3

Read Contract Information

[Expand all] [Reset]

1. FLASHLOAN_PREMIUM_TOTAL	🔗 →
2. LENDINGPOOL_REVISION	🔗 →
3. MAX_NUMBER_RESERVES	🔗 →
4. MAX_STABLE_RATE_BORROW_SIZE_PERCENT	🔗 →
5. getAddressesProvider	🔗 →
6. getConfiguration	🔗 →
7. getReserveData	🔗 →
8. getReserveNormalizedIncome	🔗 →
9. getReserveNormalizedVariableDebt	🔗 →
10. getReservesList	🔗 →

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

	USDT Tether	Reserve Size \$527.94M	527940000 Available liquidity \$128.16M	Utilization Rate 75.72 %	Oracle price \$1.00
	WBTC Wrapped BTC	Reserve Size \$720.34M	720340000 Available liquidity \$625.37M	Utilization Rate 13.18 %	Oracle price \$26,376.44
	WETH Wrapped ETH	Reserve Size \$1.03B	10 0000 0000 Available liquidity \$433.00M	Utilization Rate 58.10 %	Oracle price \$1,835.66

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

	WETH Wrapped ETH	490510000	Reserve Size \$490.51M	Available liquidity \$178.46M	Utilization Rate 63.62 %	Oracle price \$1,835.66
	wstETH Wrapped liquid staked Ether 2.0	532460000	Reserve Size \$532.46M	Available liquidity \$20.04M	Utilization Rate 0.90 %	Oracle price \$2,067.43
	WBTC Wrapped BTC	140780000	Reserve Size \$140.78M	Available liquidity \$99.89M	Utilization Rate 29.05 %	Oracle price \$26,352.25

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



Basic Attention Token





2835000000

Total Earning	Earn APR	Earn Distribution	Reserves
\$28.35M	 0.00%	 0.00%	\$695.70K
Total Borrowing	Borrow APR	Borrow Distribution	Borrow Cap
\$42,507.92	 2.51%	 0.00%	\$176.69K
Collateral Factor	Reserve Factor	Oracle Price	
60.00%	25.00%	\$0.20	



Dai

275540000

Total Earning	Earn APR	Earn Distribution	Reserves
\$274.54M	 1.70%	 0.68%	\$21.93M
Total Borrowing	Borrow APR	Borrow Distribution	Borrow Cap
\$156.81M	 3.51%	 1.18%	No Limit
Collateral Factor	Reserve Factor	Oracle Price	
83.50%	15.00%	\$1.00	