

Compound V2 TVL计算

简介

相比于 Aave 的 TVL 计算，compound 就稍微复杂一些，需要进行一些特别且巧妙的处理。

在介绍复杂部分之前，我们还是先讲一下 compound v2 TVL 计算的基础构建块。

基础构建块

1. 使用 **compound Comptroller** 合约实例获取 Compound 所有的 market 信息（compound 为每种代币组合都维护了一个池子）
2. 通过 compound 价格预言机合约实例获取基础资产的价格（以 U 计价）
3. 通过 cToken 合约实例获取兑换比率（exchangeRate），从而求得所有 cToken 能够兑换的基础资产的数量（最终计算价值时，要用基础资产的价格来计算）

$$exchangeRate = \frac{underlyingToken}{cToken}$$

$$underlyingToken = cToken * exchangeRate$$

4. 通过 cToken 合约实例获取 cToken 对应的基础资产的代币合约地址（比如 cDAI 对应的基础资产为 DAI）
5. 通过 ERC20 合约实例获取 symbol、totalSupply 等

以下是上述基础构建块的脚本示例：

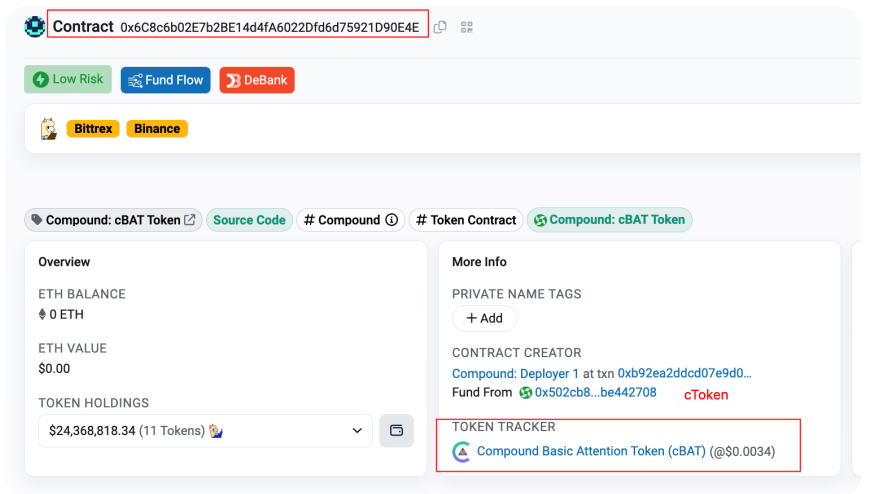
1. 获取所有的 cToken 合约地址

```
1 const Web3 = require('web3');
2
3 const compoundABI = require("../json/compound.json")
4 const web3 = new Web3('http://cloudflare-eth.com/');
5 const compoundContractInstance = new web3.eth.Contract(compoundABI, '0x3d9819210
6
7 const main = async function() {
8     // 获取所有 CToken 地址
9     const allMarkets = await compoundContractInstance.methods.getAllMarkets().ca
10     console.log(allMarkets);
11 }
12
13 main();
```

```

$ node compoundv2/getAllMarkets.js
[
  '0x6C8c6b02E7b2BE14d4fA6022Dfd6d75921D90E4E',
  '0x5d3a536E4D6DbD6114cc1Ead35777bAB948E3643',
  '0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5',
  '0x158079Ee67Fce2f58472A96584A73C7Ab9AC95c1',
  '0x39AA39c021dfbaE8faC545936693aC917d5E7563',
  '0xf650C3d88D12d8855b8bf7D11Be6C55A4e07dCC9',
  '0xC11b1268C1A384e55C48c2391d8d480264A3A7F4',
  '0x83319f5D18Bc0D84d1b4825Dcde5d5f7266d407',
  '0xF5DCe57282A584D2746FaF1593d3121Fcac444dC',
  '0x35A18000230DA775CAc24873d00Ff85BccdeD550',
  '0x70e36f6BF80a52b3B46b3aF8e106CC0ed743E8e4',
  '0xccF4429DB6322D5C611ee964527D42E5d685DD6a',
  '0x12392F67bdf24faE0AF363c24aC620a2f67DAd86',
  '0xFAce851a4921ce59e912d19329929CE6da6EB0c7',
  '0x95b4eF2869eBD948EB4eEE400a99824BF5DC325b',
  '0x4B0181102A0112A2ef11AbEE5563bb4a3176c9d7',
  '0xe65cdB6479BaC1e22340E4E755fAE7E509EcD06c',
  '0x80a2AE356fc9ef4305676f7a3E2Ed04e12C33946',
  '0x041171993284df560249B57358F931D9eB7b925D',
  '0x7713DD9Ca933848F6819F38B8352D9A15EA73F67'
]

```



这将返回 compound 支持的所有 cToken 合约地址，右边是其中一个的查询结果。

2. 获取 cToken 对应基础资产的价格

我们将借助构建块 1 获取所有的 cToken 合约地址，然后分别求出它们对应底层资产的价格。

为了使输出更加清晰明确，我们将使用 erc20 合约实例获取代币合约地址对应的 symbol。

```

1 const Web3 = require('web3');
2 const web3 = new Web3('http://cloudflare-eth.com/');
3
4 const compoundABI = require("../json/compound.json")
5 const compoundContractInstance = new web3.eth.Contract(compoundABI, '0x3d9819210
6
7 const compoundOracleABI = require("../json/compoundOracleABI.json");
8 const compoundOracleInstance = new web3.eth.Contract(compoundOracleABI, '0x50ce
9
10 const main = async function() {
11     const allMarkets = await compoundContractInstance.methods.getAllMarkets().ca
12     console.log(allMarkets);
13
14     for (let i = 0; i < allMarkets.length; i++) {
15         // 调用预言机合约的api求得该 ctoken 对应的基础资产的价格
16         const price = await compoundOracleInstance.methods.getUnderlyingPrice(a
17         console.log(allMarkets[i], ' underlying price', price);
18     }
19 }
20
21 main();

```

• 输出

```

0x6C8c6b02E7b2BE14d4fA6022Dfd6d75921D90E4E underlying price 164200000000000000
0x5d3a536E4D6DbD6114cc1Ead35777bAB948E3643 underlying price 999940000000000000
0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5 underlying price 1738275550000000000000
0x158079Ee67Fce2f58472A96584A73C7Ab9AC95c1 underlying price 64336800000000000000
0x39AA39c021dfbaE8faC545936693aC917d5E7563 underlying price 1000000000000000000000000000000000
0xf650C3d88D12dB855b8bf7D11Be6C55A4e07dCC9 underlying price 1000000000000000000000000000000000
0xC11b1268C1A384e55C48c2391d8d480264A3A7F4 underlying price 2568300000000000000000000000000000
0xB3319f5D18Bc0D84dD1b4825Dcde5d5f7266d407 underlying price 16775900000000000000
0xF5DCe57282A584D2746FaF1593d3121Fcac444dC underlying price 91867860000000000000
0x35A18000230DA775CAc24873d00Ff85BccdeD550 underlying price 39561000000000000000
0x70e36f6BF80a52b3B46b3aF8e106CC0ed743E8e4 underlying price 2657000000000000000000
0xcccF4429DB6322D5C611ee964527D42E5d685DD6a underlying price 2568300000000000000000000000000000
0x12392F67bdf24faE0AF363c24aC620a2f67DAd86 underlying price 1000000000000000000000000000000000
0xFAce851a4921ce59e912d19329929CE6da6EB0c7 underlying price 499625500000000000000000000000000
0x95b4eF2869eBD94BEb4eEE400a99824BF5DC325b underlying price 588105777000000000000000000000000
0x4B0181102A0112A2ef11AbEE5563bb4a3176c9d7 underlying price 651112000000000000000000000000000
0xe65cdB6479BaC1e22340E4E755fAE7E509EcD06c underlying price 526220420000000000000000000000000
0x80a2AE356fc9ef4305676f7a3E2Ed04e12C33946 underlying price 529128000000000000000000000000000
0x041171993284df560249B57358F931D9eB7b925D underlying price 1000000000000000000000000000000000
0x7713DD9Ca933848F6819F38B8352D9A15EA73F67 underlying price 1001094000000000000000000000000000

```

我们以第二行 cDAI 的输出进行分析，

0x5d3a536E4D6DbD6114cc1Ead35777bAB948E3643 underlying price 999940000000000000

这表示 cDAI 的基础类型 DAI 的价格为 999940000000000000，DAI 的精度为 18，除以精度后可以得到 1 DAI = 0.99994 USDT。

不同的基础类型的精度不同，所以这就要求我们在之后计算 TVL 时，需要解析出基础类型的精度信息。

3. 通过 cToken 合约实例获取当前兑换率

我们将通过 cToken 合约实例提供的 API 方法获取到 exchangeRate，然后计算出 cToken 可以兑换的基础类型代币的数量，然后再乘以第二部分中计算出的价格，就可以得出锁仓量了。

```

1 const Web3 = require('web3');
2 const web3 = new Web3('http://cloudflare-eth.com/');
3
4 const erc20ABI = require('../json/erc20.json');
5 const cTokenABI = require('../json/cTokenABI.json');
6 const compoundABI = require("../json/compound.json")
7
8 const compoundContractInstance = new web3.eth.Contract(compoundABI, '0x3d9819210
9
10 const main = async function() {
11     const allMarkets = await compoundContractInstance.methods.getAllMarkets().ca
12     console.log(allMarkets);
13
14     for (let i = 0; i < allMarkets.length; i++) {
15         const cTokenInstance = new web3.eth.Contract(cTokenABI, allMarkets[i]);
16         const csymbol = await cTokenInstance.methods.symbol().call();
17         const exchangeRate = await cTokenInstance.methods.exchangeRateCurrent().
18         console.log(csymbol, " exchangeRate:", exchangeRate, exchangeRate.length)

```

```

19     }
20 }
21
22 main()

```

• 输出

```

cBAT  exchangeRate: 206767449033274182316142398 27
cDAI  exchangeRate: 222665421418149861568788309 27
cETH  exchangeRate: 200854447459369641247376983 27
cREP  exchangeRate: 200406592366299713774311439 27
cUSDC exchangeRate: 228824963745884 15
cUSDT exchangeRate: 223672625936549 15
cWBTC exchangeRate: 20204487898663672 17
cZRX  exchangeRate: 206284107596938252174657530 27
cDAI  exchangeRate: 214497546666950175599123870 27
cUNI  exchangeRate: 203543314521231371620801812 27
cCOMP exchangeRate: 204373596596548339895355424 27
cWBTC exchangeRate: 20075057343395933 17
cTUSD exchangeRate: 211670985486332322799414145 27
cLINK exchangeRate: 202089964111814812272084130 27
cMKR  exchangeRate: 201549923231356423366852034 27
cSUSHI exchangeRate: 205514984165763533270282469 27
cAAVE exchangeRate: 206212769112475227426451517 27
cYFI  exchangeRate: 203885288245935638218353823 27
cUSDP exchangeRate: 203806564740438548750841878 27
cFEI  exchangeRate: 201359211746649374022864719 27

```

```

cBAT  exchangeRate: 0.020676744907623744 27
cDAI  exchangeRate: 0.022266545003731444 27
cETH  exchangeRate: 0.020085444907351896 27
cREP  exchangeRate: 0.02004065923662997 27
cUSDC exchangeRate: 0.022882499238747 15
cUSDT exchangeRate: 0.0223672667661687 15
cWBTC exchangeRate: 0.02020448789866367 17
cZRX  exchangeRate: 0.02062841164795549 27
cDAI  exchangeRate: 0.021449754666695017 27
cUNI  exchangeRate: 0.020354331695064364 27
cCOMP exchangeRate: 0.020437360001115887 27
cWBTC exchangeRate: 0.020075057398260793 17
cTUSD exchangeRate: 0.02116710031194845 27
cLINK exchangeRate: 0.020208996466954697 27
cMKR  exchangeRate: 0.020154992331580062 27
cSUSHI exchangeRate: 0.02055149852384398 27
cAAVE exchangeRate: 0.020621277151261636 27
cYFI  exchangeRate: 0.02038852883171473 27
cUSDP exchangeRate: 0.020380688789757868 27
cFEI  exchangeRate: 0.02013592117466494 27

```

我们还是以 cDAI 为例进行分析，它的 exchangeRate 大约在 0.22 左右，因此我们可以得出 exchangeRate 的精度为 28 位 (exchangeRate.length+1)。

4. 根据 cToken 合约实例求基础类型合约地址

```

1  const Web3 = require('web3');
2  const web3 = new Web3('http://cloudflare-eth.com/');
3
4  const erc20ABI = require('../json/erc20.json');
5  const cTokenABI = require('../json/cTokenABI.json');
6  const compoundABI = require("../json/compound.json")
7
8  const compoundContractInstance = new web3.eth.Contract(compoundABI, '0x3d9819210
9
10 const main = async function() {
11     const allMarkets = await compoundContractInstance.methods.getAllMarkets().ca
12     console.log(allMarkets);
13
14     for (let i = 0; i < allMarkets.length; i++) {
15         const cTokenInstance = new web3.eth.Contract(cTokenABI, allMarkets[i]);
16         const symbol = await cTokenInstance.methods.symbol().call();
17         const underlying = await cTokenInstance.methods.underlying().call();
18
19         console.log(symbol, "'s underlying type: ", underlying);

```

```

20     }
21 }
22
23 main();

```

- 输出结果

```


cBAT 's underlying type: 0x0D8775F648430679A709E98d2b0Cb6250d2887EF
cDAI 's underlying type: 0x6B175474E89094C44Da98b954EedeAC495271d0F
/Users/apple/Study/BlockChain/ETH/web3.js/node_modules/web3-eth-abi/lib/index.js:304
    throw new Error('Returned values aren\'t valid, did it run Out of Gas? ' +
    ^


```


Error: Returned values aren't valid, did it run Out of Gas? You might also see this error if you are not using the correct ABI for the contract


到三个地址调用时报错，我们看下第三个地址是什么合约代币。


0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5



Contract 0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5


 Low Risk

 Fund Flow

 DeBank


Coinbase



Binance


Compound: cETH Token

[Source Code](#)

[# Compound](#)

[# Token Contract](#)


Compound: cETH Token

Overview

ETH BALANCE

251,509.022947126678310366 ETH

ETH VALUE

\$438,073,385.99 (@ \$1,741.78/ETH)

TOKEN HOLDINGS

\$38,787.48 (88 Tokens)

More Info

PRIVATE NAME TAGS

+ Add

CONTRACT CREATOR

Compound: Deployer 1 at txn 0xe60e30c7131f043b...

Fund From 0x502cb8...be442708

TOKEN TRACKER

Compound Ether (cETH) (@\$34.9832)

cETH 不同于其他类型的 ERC20 代币，它的基础类型就是它本身，所以它的合约中没有提供 underlying 方法调用，因此这里我们需要对该地址进行单独处理。

```

1 const Web3 = require('web3');
2 const web3 = new Web3('http://cloudflare-eth.com/');
3
4 const erc20ABI = require('../json/erc20.json');
5 const cTokenABI = require('../json/cTokenABI.json');
6 const compoundABI = require("../json/compound.json")
7
8 const compoundContractInstance = new web3.eth.Contract(compoundABI, '0x3d9819210

```



```

9
10 const main = async function() {
11     const allMarkets = await compoundContractInstance.methods.getAllMarkets().ca
12     console.log(allMarkets);
13
14     var underlying = '';
15     var decimal = 0;
16     var underSymbol = '';
17
18     for (let i = 0; i < allMarkets.length; i++) {
19         const cTokenInstance = new web3.eth.Contract(cTokenABI, allMarkets[i]);
20         const symbol = await cTokenInstance.methods.symbol().call();
21         if (allMarkets[i] == '0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5') { //
22             underlying = '0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5';
23             decimal = 18;
24         } else {
25             underlying = await cTokenInstance.methods.underlying().call();
26             const underERC = new web3.eth.Contract(erc20ABI, underlying);
27             decimal = await underERC.methods.decimals().call();
28             underSymbol = await underERC.methods.symbol().call();
29         }
30
31         console.log(symbol, "underlying type:", underSymbol, "underlying address:
32     }
33 }
34
35 main();

```

• 输出结果

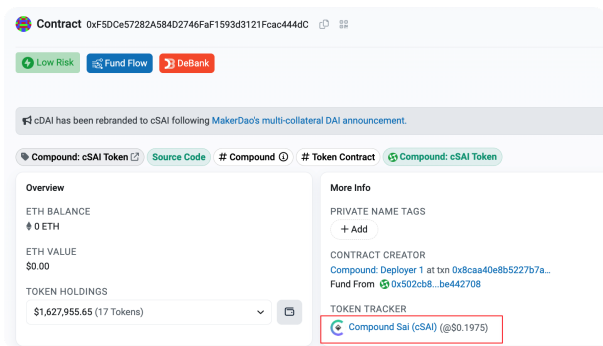
```

cBAT underlying type: BAT underlying address: 0x0D8775F648430679A709E98d2b0Cb6250d2887EF decimal: 18
cDAI underlying type: DAI underlying address: 0x6B175474E89094C44Da98b954EedeAC495271d0F decimal: 18
cETH underlying type: DAI underlying address: 0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5 decimal: 18
cREP underlying type: REP underlying address: 0x1985365e9f78359a9B6AD760e32412f4a445E862 decimal: 18
cUSDC underlying type: USDC underlying address: 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48 decimal: 6
cUSDT underlying type: USDT underlying address: 0xdAC17F958D2ee523a2206206994597C13D831ec7 decimal: 6
cWBTC underlying type: WBTC underlying address: 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 decimal: 8
cZRX underlying type: ZRX underlying address: 0xE41d2489571d322189246DaFA5ebDe1F4699F498 decimal: 18
/Users/apple/Study/BlockChain/ETH/web3.js/node_modules/@ethersproject/logger/lib/index.js:238
    var error = new Error(message);
                  ^
Error: overflow [ See: https://links.ethers.org/v5-errors-NUMERIC_FAULT-overflow ] (fault="overflow", operation="toNumber"

```

处理到第 9 个地址报错，我们依旧是查询一下该 cToken 代币合约。

0xF5DCe57282A584D2746FaF1593d3121Fcac444dC



Sai (Deprecated)			
Total Earning	Earn APR	Earn Distribution	Reserves
\$1.56M	0.00%	0.00%	\$71.55
Total Borrowing	Borrow APR	Borrow Distribution	Borrow Cap
\$2,295.95	6.17%	0.00%	No Limit
Collateral Factor	Reserve Factor	Oracle Price	
0.00%	100.00%	\$9.22	

本质上是 cDAI，但是在 compound 中已经弃用，所以我们不再处理该代币合约地址。

• 代码修改

我们只需要在 for 循环中添加一个判断逻辑即可

```
1 if (allMarkets[i] == '0xF5DCE57282A584D2746FaF1593d3121Fcac444dC') {  
2     continue  
3 }
```

新的输出结果

```
cBAT underlying type: BAT underlying address: 0x0D8775F648430679A709E98d2b0Cb6250d2887EF decimal: 18  
cDAI underlying type: DAI underlying address: 0x6B175474E89094C44Da98b954EedeAC495271d0F decimal: 18  
cETH underlying type: DAI underlying address: 0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5 decimal: 18  
cREP underlying type: REP underlying address: 0x1985365e9f78359a9B6AD760e32412f4a445E862 decimal: 18  
cUSDC underlying type: USDC underlying address: 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48 decimal: 6  
cUSDT underlying type: USDT underlying address: 0xdAC17F958D2ee523a2206206994597C13D831ec7 decimal: 6  
cWBTC underlying type: WBTC underlying address: 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 decimal: 8  
cZRX underlying type: ZRX underlying address: 0xE41d2489571d322189246DaFA5ebDe1F4699F498 decimal: 18  
cUNI underlying type: UNI underlying address: 0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984 decimal: 18  
cCOMP underlying type: COMP underlying address: 0xc00e94Cb662C3520282E6f5717214004A7f26888 decimal: 18  
cWBTC underlying type: WBTC underlying address: 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 decimal: 8  
cTUSD underlying type: TUSD underlying address: 0x00000000000085d4780B73119b644AE5ecd22b376 decimal: 18  
cLINK underlying type: LINK underlying address: 0x514910771AF9Ca656af840dff83E8264EcF986CA decimal: 18  
/Users/apple/Study/BlockChain/ETH/web3.js/node_modules/@ethersproject/logger/lib/index.js:238  
    var error = new Error(message);  
                ^
```

```
Error: overflow [ See: https://links.ethers.org/v5-errors-NUMERIC_FAULT-overflow ] (fault="overflow", operation="toNumber")
```

这次在处理到第 14 个地址时报错，第 14 个地址为

cMKR: 0x95b4eF2869eBD94BEb4eEE400a99824BF5DC325b

MKR 的 symbol 返回值竟然不是 string，那么我们的 ERC20 ABI 就无法匹配，因此调用时就会报错。

解决办法，创建一份新的关于 MKR 的 ABI，然后使用新的 ABI 创建合约实例去调用这个方法。


```

1  [
2    {
3      "constant":true,
4      "inputs":[
5
6      ],
7      "name":"symbol",
8      "outputs":[
9        {
10         "name": "",
11         "type": "bytes32"
12       }
13     ],
14     "payable":false,
15     "stateMutability":"view",
16     "type":"function"
17   }
18 ]

```

修改后的代码

```

1  const Web3 = require('web3');
2  const web3 = new Web3('http://cloudflare-eth.com/');
3
4  const erc20ABI = require('../json/erc20.json');
5  const cTokenABI = require('../json/cTokenABI.json');
6  const compoundABI = require("../json/compound.json")
7  const MKRABI = require("../json/MKR.json")
8
9  const compoundContractInstance = new web3.eth.Contract(compoundABI, '0x3d9819210
10
11  const main = async function() {
12    const allMarkets = await compoundContractInstance.methods.getAllMarkets().ca
13    console.log(allMarkets);
14
15    var underlying = '';
16    var decimal = 0;
17    var underSymbol = '';
18
19    for (let i = 0; i < allMarkets.length; i++) {
20      if (allMarkets[i] == '0xF5DCe57282A584D2746FaF1593d3121Fcac444dC') {
21        continue
22      }
23

```

```

24     const cTokenInstance = new web3.eth.Contract(cTokenABI, allMarkets[i]);
25     const symbol = await cTokenInstance.methods.symbol().call();
26
27     if (allMarkets[i] == '0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5') { //
28         underlying = '0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5';
29         decimal = 18;
30     } else {
31         underlying = await cTokenInstance.methods.underlying().call();
32         const underERC = new web3.eth.Contract(erc20ABI, underlying);
33         decimal = await underERC.methods.decimals().call();
34     }
35
36     if (allMarkets[i] == '0x95b4eF2869eBD94BEb4eEE400a99824BF5DC325b') {
37         // 字节数据转换为字符串 todo
38         //const erc20 = new web3.eth.Contract(MKRABI, underlying)
39         // let bytess = await erc20.methods.symbol().call();
40         //underSymbol = web3.utils.bytesToHex(bytess);
41         //underSymbol = underERC20.slice(0, 20); // just need 20 bytes
42         underSymbol = "MKR"
43     } else {
44         const erc20 = new web3.eth.Contract(erc20ABI, underlying);
45         underSymbol = await erc20.methods.symbol().call()
46     }
47
48     console.log(symbol, "underlying type:", underSymbol, "underlying address:
49 }
50 }
51
52 main();

```

- 最终输出结果

```

cBAT underlying type: BAT underlying address: 0x0D8775F648430679A709E98d2b0Cb6250d2887EF decimal: 18
cDAI underlying type: DAI underlying address: 0x6B175474E89094C44Da98b954EedeAC495271d0F decimal: 18
cETH underlying type: cETH underlying address: 0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5 decimal: 18
cREP underlying type: REP underlying address: 0x1985365e9f78359a9B6AD760e32412f4a445E862 decimal: 18
cUSDC underlying type: USDC underlying address: 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48 decimal: 6
cUSDT underlying type: USDT underlying address: 0xdAC17F958D2ee523a2206206994597C13D831ec7 decimal: 6
cWBTC underlying type: WBTC underlying address: 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 decimal: 8
cZRX underlying type: ZRX underlying address: 0xE41d2489571d322189246DaFA5ebDe1F4699F498 decimal: 18
cUNI underlying type: UNI underlying address: 0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984 decimal: 18
cCOMP underlying type: COMP underlying address: 0xc00e94Cb662C3520282E6f5717214004A7f26888 decimal: 18
cWBTC underlying type: WBTC underlying address: 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 decimal: 8
cTUSD underlying type: TUSD underlying address: 0x0000000000085d4780B73119b644AE5ecd22b376 decimal: 18
cLINK underlying type: LINK underlying address: 0x514910771AF9Ca656af840dff83E8264EcF986CA decimal: 18
cMKR underlying type: MKR underlying address: 0x9f8F72aA9304c8B593d555F12eF6589cC3A579A2 decimal: 18
cSUSHI underlying type: SUSHI underlying address: 0x6B3595068778DD592e39A122f4f5a5cF09C90fE2 decimal: 18
cAAVE underlying type: AAVE underlying address: 0x7Fc66500c84A76Ad7e9c93437bFc5Ac33E2DDaE9 decimal: 18
cYFI underlying type: YFI underlying address: 0x0bc529c00C6401aEF6D220BE8C6Ea1667F6Ad93e decimal: 18
cUSDP underlying type: USDP underlying address: 0x8E870D67F660D95d5be530380D0eC0bd388289E1 decimal: 18
cFEI underlying type: FEI underlying address: 0x956F47F50A910163D8BF957Cf5846D573E7f87CA decimal: 18

```

5. 第五步已经穿插在前面几部分了。

ok, 基础构建块大致就这么多, 我们将基础构建块合并在一起, 组成完整的 compound TVL 计算脚本。

完整代码

```
1  const Web3 = require('web3');
2
3  const erc20ABI = require('../json/erc20.json');
4  const compoundOracleABI = require('../json/compoundOracleABI.json');
5  const compoundABI = require('../json/compound.json');
6  const cTokenABI = require('../json/cTokenABI.json');
7  const MKRABI = require('../json/MKR.json');
8
9  const web3 = new Web3('http://cloudflare-eth.com/');
10
11 const compoundContractInstance = new web3.eth.Contract(compoundABI, '0x3d9819210
12 const compouondOracleInstance = new web3.eth.Contract(compoundOracleABI, '0x50ce
13
14 /** cTokenToTVL
15  * @param key address
16  * @param value Total Value Locked (TVL: 总锁仓价值)
17  */
18
19 const Struct = {
20   chainID: Number,
21   cTokenToTVL: new Map(),
22   cTokenAddressToSymbol: new Map(),
23   cTokenToUnderlying: new Map(),
24 }
25
26 const Data = Object.create(Struct);
27
28 const main = async function() {
29   // 获取所有 CToken 地址
30   const allMarkets = await compoundContractInstance.methods.getAllMarkets().ca
31   console.log(allMarkets);
32   // 遍历处理所有 CToken 并获取 TVL
33   for (let i = 0; i < allMarkets.length; i++) {
34     if (allMarkets[i] == '0xF5DCe57282A584D2746FaF1593d3121Fcac444dC') { //
35       continue
36     }
37
38     // 打印 ctoken、供应量信息, 因为下面 erc20Instance 还会被其他合约实现, 声明为
39     const erc20Instance = new web3.eth.Contract(erc20ABI, allMarkets[i]); //
40     const symbol = await erc20Instance.methods.symbol().call(); //
```

```

41     const totalSupply = await erc20Instance.methods.totalSupply().call(); //
42
43     // map ctoken address to symbol
44     Data.cTokenAddressToSymbol.set(allMarkets[i], symbol);
45
46     // 质押因子信息
47     const results = await compoundContractInstance.methods.markets(allMarket
48     console.log(symbol, 'collateralFactorMantissa:', results[1] / 10 ** 18);
49
50     // compound v2 合约实例
51     const cTokenInstance = new web3.eth.Contract(cTokenABI, allMarkets[i]);
52
53     // cToken 精度和底层 erc20 代币类型
54     var decimal = 0;
55     var underlying = '';
56
57     // cETH 底层类型就是自己
58     if (allMarkets[i] == '0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5') { //
59         underlying = '0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5';
60         decimal = 18;
61     } else { // 获取 ctoken 的底层代币类型, 比如 cDAI 的底层代币为 DAI
62         underlying = await cTokenInstance.methods.underlying().call(); // 获
63         decimal = await erc20Instance.methods.decimals().call(); // 获取该代
64     }
65
66     var underERC20 = '';
67     // 由于 MKR 的 symbol() 返回值不是标准 erc20 标准中的 string 类型而是 bytes3.
68     if (allMarkets[i] == '0x95b4eF2869eBD94BEb4eEE400a99824BF5DC325b') {
69         // const underERC20Instance = new web3.eth.Contract(MKRABI, underlyi
70         // const bytes = await underERC20Instance.methods.symbol().call(); /
71         // underERC20 = web3.utils.bytesToHex(bytes);
72         // underERC20 = underERC20.slice(0, 20); // just need 20 bytes
73         underERC20 = 'MKR';
74     } else {
75         const underERC20Instance = new web3.eth.Contract(erc20ABI, underlyin
76         underERC20 = await underERC20Instance.methods.symbol().call();
77     }
78     // map ctoken symbol to underlying symbol
79     Data.cTokenToUnderlying.set(symbol, underERC20);
80
81     // access underlying price by Oracle (decimal 在打印时考虑)
82     const price = await compouondOracleInstance.methods.getUnderlyingPrice(a
83     // access the exchangeRate between ctoken and underlying token
84     // let exchangeRate = await cTokenContract.methods.exchangeRateCurrent()
85     const exchangeRate = await cTokenInstance.methods.exchangeRateStored().c
86
87     // totalsupply * exchangeRate = underlyingAmount

```

```

88     // tvl = underlyingAmount * underlyingPrice
89     const tvl = web3.utils.toBN(totalSupply) * web3.utils.toBN(exchangeRate)
90     // store cToken -> tvl into memory
91     Data.cTokenToTVL.set(allMarkets[i], tvl/1e6);
92
93     // todo: oraclePrice calculate : price * 10 ** decimal / 1e30
94
95     // print detail information
96     console.log('ctoken = ', symbol, 'oraclePrice = ', web3.utils.toBN(price
97   }
98 }
99
100 /**
101  *
102  * @param {*} cToken : 要查询的 cToken 地址
103  */
104 // 获取指定 cToken 地址对应的 Token name、底层代币类型以及对应的 TVL 信息
105 const getSpecifyMarketsTVL = async function(cToken) {
106     console.time('flag');
107     if (Data.cTokenToTVL.size == 0) {
108         // 执行 main 函数获取所有 ctoken 的 tvl 数据并存入内存中
109         await main();
110     }
111
112     // 取值返回, 给出查询地址对应 ctoken 的 symbol、underlying token symbol、 以及
113     const symbol = Data.cTokenAddressToSymbol.get(cToken);
114     const underlying = Data.cTokenToUnderlying.get(symbol);
115     const tvl = Data.cTokenToTVL.get(cToken);
116
117     console.log(`
118 查询的 cToken 地址为 ${cToken}
119 对应的 cToken Symbol 为 ${symbol}
120 cToken 对应的底层代币 Symbol 为 ${underlying}
121 cToken 对应的 TVL 为: ${tvl}$
122 `)
123     console.timeEnd('flag');
124 }
125
126 getSpecifyMarketsTVL('0x95b4eF2869eBD94BEb4eEE400a99824BF5DC325b').catch((err) =
127     console.error(err);
128 });
129
130 // main() 获取所有 Market 的 TVL
131 // main().catch((err) => {
132 //     console.error(err);
133 // });

```



• 输出结果

```
ctoken = cBAT tvl = 24298198.51902858
ctoken = cDAI tvl = 270469610.8797952
ctoken = cETH tvl = 459694173.73437357
ctoken = cREP tvl = 7053.382002665154
ctoken = cUSDC tvl = 396593030.90885276
ctoken = cUSDT tvl = 198419172.7785795
ctoken = cWBTC tvl = 3602428.5500522847
ctoken = cZRX tvl = 1295544.913683222
ctoken = cUNI tvl = 18298137.912276827
ctoken = cCOMP tvl = 2201373.3767672707
ctoken = cWBTC tvl = 366448200.3992414
ctoken = cTUSD tvl = 784134.4081792649
ctoken = cLINK tvl = 2127503.60032742
ctoken = cMKR tvl = 299206.5533351194
ctoken = cSUSHI tvl = 1867856.7403037474
ctoken = cAAVE tvl = 1018132.8064288375
ctoken = cYFI tvl = 284778.13613724895
ctoken = cUSDP tvl = 22348.70717591629
ctoken = cFEI tvl = 1737.4886866997022




查询的 cToken 地址为 0x95b4eF2869eBD94BEb4eEE400a99824BF5DC325b
对应的 cToken Symbol 为 cMKR
cToken 对应的底层代币 Symbol 为 MKR
cToken 对应的 TVL 为：299206.5533351194$

flag: 49.574s
```

• 官方提供的 TVL

 **Maker**

302250

Total Earning	Earn APR	Earn Distribution	Reserves	Utilization
\$302.25K	 0.03%	 0.00%	\$1,837.83	1.44% ●
Total Borrowing	Borrow APR	Borrow Distribution	Borrow Cap	Borrow APR
\$4,359.81	 2.86%	 0.00%	\$188.22K	2.86%
Collateral Factor	Reserve Factor	Oracle Price	Earn APR	
73.00%	25.00%	\$627.38	0.03% ○	

• 误差率

略