

获取 AAVE V2 支持所有代币的价格

简介

获取代币价格是计算所有代币 TVL 的基础构建块，所以本节我们将设计脚本获取 AAVE V2 版本所有支持代币的价格，因为 AAVE V2 以以太价格结算，所以这里的价格都是相对于以太而言的。


AAVE V2 提供了价格预言机，我们将使用官方提供的价格预言机合约获取所有代币的价格。

这里先给出 AAVE V2 相关的合约地址：

- Aave: Lending Pool Provider V2: 0xB53C1a33016B2DC2f3653530bfF1848a515c8c5
- AAVE V2 Lending Pool : 0x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9
- AAVE V2 Lending Pool Logic Contract: [0xc6845a5c768bf8d7681249f8927877efda425baf](#)
- AAVE Price Oracle V2: 0xA50ba011c48153De246E5192C8f9258A2ba79Ca9

流程设计

1. 使用借贷池合约拿到代币支持列表
2. 循环处理代币列表中的每一个代币，使用价格预言机合约获取其价格

 价格预言机返回的已经是相对于以太的价格了（单位 wei），所以不需要考虑精度问题（仍然给出了求精度的逻辑）。在一些需要考虑到精度的情况下，我们还需要 erc20 的合约实例，通过调用 decimals 方法获取代币的精度，从而得到准确的价格。

脚本设计

ABI

- AAVE V2 Lending Pool 逻辑合约 ABI
- AAVE V2 Oracle Price 合约 ABI

Js 脚本

```
1 const Web3 = require('web3');  
2
```

```

3  const priceOracleAbi = require('../json/priceOracle.json');
4  const erc20Abi = require('../json/erc20.json');
5  const lendingAbi = require('../json/lending_pool.json');
6
7  // web3 instance
8  const web3 = new Web3('https://cloudflare-eth.com');
9
10 // aave lending pool v2 contract instance
11 const lendingContractInstance = new web3.eth.Contract(lendingAbi, '0x7d2768dE32b
12
13 const priceInstance = new web3.eth.Contract(priceOracleAbi, '0xA50ba011c48153De2
14
15 const main = async function() {
16     let allTokens = await lendingContractInstance.methods.getReservesList().call
17     for (let i = 0; i < allTokens.length; i++) {
18         // the logic to access decimal of token
19         const erc20Instance = new web3.eth.Contract(erc20Abi, allTokens[i]);
20         const decimal = await erc20Instance.methods.decimals().call();
21
22         let price = await priceInstance.methods.getAssetPrice(allTokens[i]).call
23         console.log(allTokens[i],':',web3.utils.fromWei(price),'eth');
24         console.log(allTokens[i],':', price,'wei','\n');
25     }
26 }
27
28 const testTime = async function() {
29     console.time('run time');
30     await main();
31     console.timeEnd('run time')
32 }
33
34 testTime()
35

```

运行结果

```
0xdAC17F958D2ee523a2206206994597C13D831ec7 : 0.000573801330580408 eth
0xdAC17F958D2ee523a2206206994597C13D831ec7 : 573801330580408 wei

0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 : 14.719345507596231298 eth
0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 : 14719345507596231298 wei

0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 : 1 eth
0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 : 1000000000000000000 wei

0x0bc529c00C6401aEF6D220BE8C6Ea1667F6Ad93e : 2.921172524255123 eth
0x0bc529c00C6401aEF6D220BE8C6Ea1667F6Ad93e : 2921172524255123000 wei

0xE41d2489571d322189246DaFA5ebDe1F4699F498 : 0.000095389056611039 eth
0xE41d2489571d322189246DaFA5ebDe1F4699F498 : 95389056611039 wei

0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984 : 0.002191653756820504 eth
0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984 : 2191653756820504 wei
```

如果想要打印的内容更加直观，可以通过 erc20 合约的 symbol 方法获取每个代币的标识，然后进行格式化输出。

并发查询

为了获取更快的查询速度，我们尝试使用并发来完成价格的查询，并比较并发与非并发性能差距。

并发测试代码

```
1 const Web3 = require('web3');
2
3 const priceOracleAbi = require('../json/priceOracle.json');
4 var lendingAbi = require('../json/lending_pool.json');
5 var aaveAbi = require('../json/aave2.json');
6
7 // web3 instance
8 const web3 = new Web3('https://cloudflare-eth.com');
9
10 const lendingPoolInstance = new web3.eth.Contract(lendingAbi, '0x7d2768dE32b0b80
11 const priceOracleInstance = new web3.eth.Contract(priceOracleAbi, '0xA50ba011c48
12
13 // get all tokens price
14 const testTime = async function() {
15     console.time('concurrency run time')
16     // getAllTokens
17     let allTokens = await lendingPoolInstance.methods.getReservesList().call();
18
19     // 2. get all tokens price by concurrency
20     await main(allTokens, 8);
21     console.timeEnd('concurrency run time');
```

```

22 }
23
24 // concurrency process
25 const process = (address) => {
26     let price = priceOracleInstance.methods.getAssetPrice(address).call();
27
28     return new Promise((resolve) => {
29         setTimeout(() => {
30             resolve(price);
31         }, 500);
32     })
33 }
34
35 // tokens = allTokensList
36 const main = async function(tokens, max) {
37     // 任务组池
38     let pool = [];
39     for (let i = 0; i < tokens.length; i++) {
40         // concurrency process task
41         const task = process(tokens[i]);
42         // task push into pool
43         pool.push(task);
44
45         // data is price which is return from process resolver
46         task.then((data) => {
47             console.log(`${tokens[i]}: ${data} 并发数 ${pool.length}`);
48             console.log(`${tokens[i]}: ${data/1e18} eth 并发数 ${pool.length}`);
49             console.log();
50             // this task is resolved , delete from pool
51             pool.splice(pool.indexOf(task), 1);
52         })
53
54         // control concurrency task number: if the length of pool is equal to max
55         // once there is a free slot in the pool, continue process other tasks
56         if (pool.length == max) {
57             await Promise.race(pool);
58         }
59     }
60
61     // process other task when pool's length minus 8
62     while (pool.length > 0) {
63         await Promise.race(pool);
64     }
65 }
66
67 testTime();

```

测试结果

- 运行时间（普通查询和并发查询）

run time: 30.338s

concurrency run time: 5.027s

- 执行结果

```
0xdAC17F958D2ee523a2206206994597C13D831ec7 : 0.000573801330580408 eth
0xdAC17F958D2ee523a2206206994597C13D831ec7 : 573801330580408 wei

0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 : 14.719345507596231298 eth
0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 : 14719345507596231298 wei

0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 : 1 eth
0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 : 1000000000000000000 wei

0x0bc529c00C6401aEF6D220BE8C6Ea1667F6Ad93e : 2.95145558439167 eth
0x0bc529c00C6401aEF6D220BE8C6Ea1667F6Ad93e : 2951455584391670000 wei

0xE41d2489571d322189246DaFA5ebDe1F4699F498 : 0.000095389056611039 eth
0xE41d2489571d322189246DaFA5ebDe1F4699F498 : 95389056611039 wei
```

```
$ node aave2/getToken/getToken_concurrency.js
0xdAC17F958D2ee523a2206206994597C13D831ec7: 573801330580408 并发数 8
0xdAC17F958D2ee523a2206206994597C13D831ec7: 0.000573801330580408 eth 并发数 8

0x008775F648430679A709E98d2b0Cb6250d2887EF: 93560512132960 并发数 8
0x008775F648430679A709E98d2b0Cb6250d2887EF: 0.00009356051213296 eth 并发数 8

0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2: 1000000000000000000 并发数 8
0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2: 1 eth 并发数 8

0x0bc529c00C6401aEF6D220BE8C6Ea1667F6Ad93e: 2951455584391670000 并发数 8
0x0bc529c00C6401aEF6D220BE8C6Ea1667F6Ad93e: 2.95145558439167 eth 并发数 8

0x1f9840a85d5aF5bf1D1762F92580AddC4201F984: 2191653756820504 并发数 8
0x1f9840a85d5aF5bf1D1762F92580AddC4201F984: 0.002191653756820504 eth 并发数 8

0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599: 14719345507596231298 并发数 8
0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599: 14.71934550759623 eth 并发数 8
```

左侧是按照支持代币列表顺序查询返回的结果，而右侧是并发查询返回的结果（无序）。

- 结果比对

普通查询: 0xdAC17F958D2ee523a2206206994597C13D831ec7: 0.000573801330580408 eth

并发查询: 0xdAC17F958D2ee523a2206206994597C13D831ec7: 0.000573801330580408 eth

普通查询: 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599 : 14.719345507596231298 eth

并发查询: 0xdAC17F958D2ee523a2206206994597C13D831ec7: 14.71934550759623 eth