AAVE 解析 aToken 的精度（适用于 V2 V3）

简介

在 AAVE 代币价格计算过程中，需要考虑到精度问题，由于不同代币的精度不同，我们需要获取一个 AAVE 币对（比如 DAI-aDAI）的精度信息，这样才能得到准确的价格信息。

而在 AAVE V2 合约中并没有直接获取 aToken 精度的 API，所以我们需要自己解析。通过查看合约，可以发现 aToken 的精度信息是位于一个名为 tuple 元组内的 data 字段中，形式如下：

```
tuple:  [
  [ '368942858936744584216O4', data: '368942858936744584216O4' ],
  '10105850162771793574932852OO',
  '10577618626068197481235263OO',
  '15882029846737835339315 69',
  '16336535921852014514914587',
  '53337908459788592164163695',
  '1686110963',
  '0x05Ec93c0365baAeAbF7AefFb0972ea7ECdD39CF1',
  '0x277f8676FAcf4dAA5a6EA38ba511B7F65AA02f9F',
  '0xfc218A6Dfe6901CB34B1a5281FC6f1b8e7E56877',
  '0xBdfC85b140edF1FeaFd6eD664027AA4C23b4A29F',
  '7',
  configuration: [ '368942858936744584216O4', data: '368942858936744584216O4' ],
  liquidityIndex: '10105850162771793574932852OO',
  variableBorrowIndex: '10577618626068197481235263OO',
  currentLiquidityRate: '15882029846737835339315 69',
  currentVariableBorrowRate: '16336535921852014514914587',
  currentStableBorrowRate: '53337908459788592164163695',
  lastUpdateTimestamp: '1686110963',
  aTokenAddress: '0x05Ec93c0365baAeAbF7AefFb0972ea7ECdD39CF1',
  stableDebtTokenAddress: '0x277f8676FAcf4dAA5a6EA38ba511B7F65AA02f9F',
  variableDebtTokenAddress: '0xfc218A6Dfe6901CB34B1a5281FC6f1b8e7E56877',
  interestRateStrategyAddress: '0xBdfC85b140edF1FeaFd6eD664027AA4C23b4A29F',
  id: '7'
]
```

data 字段是按照位进行数据存储的，规则如下：

```
1 uint256 constant LTV_MASK =                       0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000;
2 uint256 constant LIQUIDATION_THRESHOLD_MASK = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000FFFF;
3 uint256 constant LIQUIDATION_BONUS_MASK =     0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000FFFF;
4 uint256 constant DECIMALS_MASK =              0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000FFFF;
5 uint256 constant ACTIVE_MASK =                0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000FFFF;
6 uint256 constant FROZEN_MASK =                0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000FFFF;
7 uint256 constant BORROWING_MASK =             0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000FFFF;
8 uint256 constant STABLE_BORROWING_MASK =      0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000FFFF;
```

```
uint256 constant RESERVE_FACTOR_MASK =         0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

我们只需要关注 DECIMALS_MASK 所处位置，不难看出其值位于低位第 7 个字节。有了这个信息，我们就可以解析 data 从而拿到精度信息了。具体解析思路如下：

1. 调用借贷池合约提供的 getReserveData(address) 方法获取该代币的 tuple 信息。
2. 取出 tuple 中的 data 数据（tuple.configuration.data）**uint256**
3. 将 data 转换为 16 进制表示，并且在左侧填充 0 使数据长度为 32 字节，然后将 16 进制数据转换字节形式
4. 精度位于低位第 7 个字节，也就是高位的第 26 字节（索引为 25）

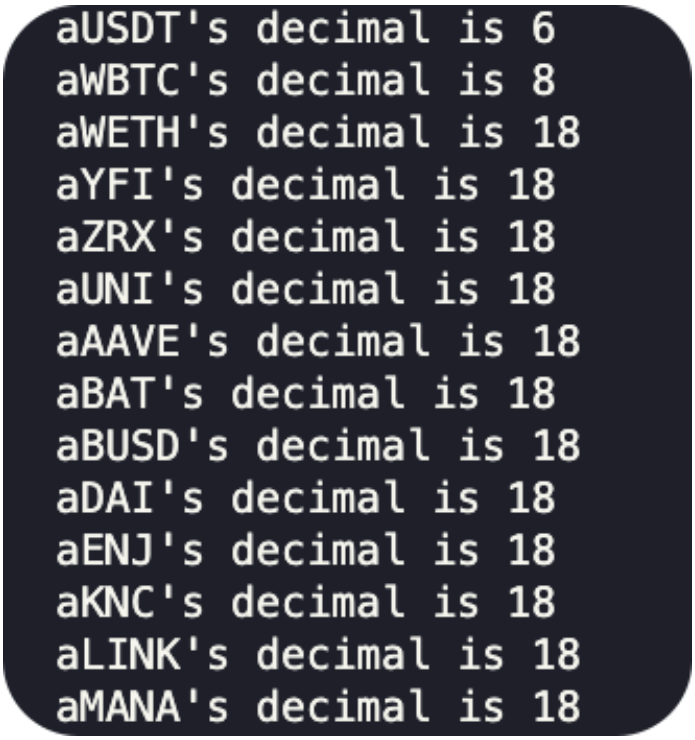
Js 脚本

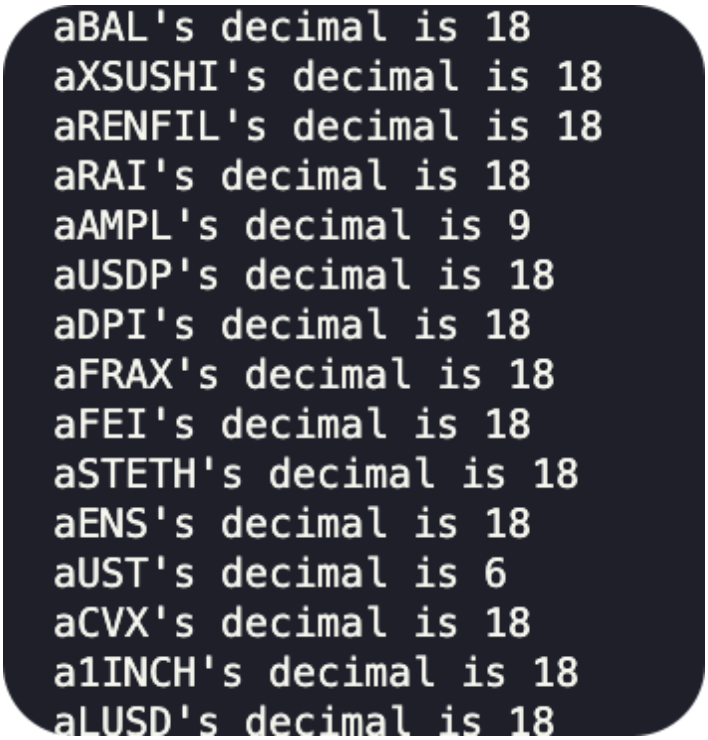
代码

```
const Web3 = require('web3');
const lendingAbi = require('../json/lending_pool.json');
const erc20ABI = require('../json/erc20.json');

const web3 = new Web3('http://cloudflare-eth.com/');

// aave v2 借贷池合约实例
const lendingContractInstance = new web3.eth.Contract(lendingAbi, '0x7d2768dE32b

const main = async function () {
    // 调用 getReservesList API 获取 AAVE V2 支持的代币列表（地址形式显示）
    let allTokens = await lendingContractInstance.methods.getReservesList().call
    console.log(allTokens);

    for (let i = 0; i < allTokens.length; i++) {
        // 获取 token 的 tuple 信息
        let tuple = await lendingContractInstance.methods.getReserveData(allToke
        let aTokenAddress = tuple.aTokenAddress;

         // erc20 contract instance
        const erc20Instance = new web3.eth.Contract(erc20ABI, aTokenAddress);
        // 获取类似 aUSDT 的名称
        let symbol = await erc20Instance.methods.symbol().call();

        // access decimals tuple[configuration[data]]
        let data = tuple.configuration.data;
```

```
        // 填充满 32 字节， 64 代表整个字符串中包含的字符数量
        let dataHex = web3.utils.padLeft(web3.utils.numberToHex(data), 64);
        let bytes = web3.utils.hexToBytes(dataHex);
        // decimals 位于低位第 7 个字节，即高位第 26 个字节 (索引 25)
        let decimal = bytes[25];
        console.log(`${symbol}'s decimal is ${decimal}`);
    }
}

main();
```

运行结果





从上面两个图就能更深入的理解：**代币的精度范围在 [6, 18] 内**。