简介

为了提高智能合约安全性，EIP214增加了一个新的操作码STATICCALL，该操作码允许用来调用其它合约或者合约自身，当时该调用或者它的子调用不允许改变任何状态，任何使用STATICCALL进行的状态修改操作都会抛出异常。(EIP 214 2017-02-13)

以太坊并没有对合约调用进行限制，因此只要gas充足你可以进行任何调用操作。在这种情况下，如果我们不知道被调用的合约源码，则当我们进行一次常规调用后我们无法确认合约的状态是否发生改变，这可能引发一些安全问题。

因此该提案引入了一种调用其它合约的方法，并且限制了该调用对数据状态地修改操作，该调用发生前后数据状态不会发生改变。

基本原理

向虚拟机中引入新的标志**“STATIC”**，这个标志默认设置为false，该标志的值总是会被传给子调用(除了新操作吗码**0xfa**)。

STATICALL的功能和CALL相同，但它只需要6个参数（“value”参数不包括在内，被设置为0），并且发起子调用时也会将子调用的STATIC标签设置为true，当调用结束后标签值恢复成调用前的状态。

任何尝试在STATIC标签设置为true的运行实例内改变状态的操作都会导致虚拟机抛出异常，这些操作包括**CREATE**，**CREATE2**，**LOGO**，**LOG1**，**LOG2**，**LOG3**，**LOG4**，**SSTORE**，**SELFDESTRUCT**和value为非零的**CALL**。**CALLCODE**不被认为是一种状态改变操作（无论value是否为0）。

STATICCALL允许合约进行不改变状态的操作，因此该调用不可能出现重入攻击或其他问题。STATICCALL除了返回一个输出外不会执行任何操作。

用例(合约层面)

```
pragma solidity >=0.8.0;
//靶子合约，用于被调用
contract target{
    int public state;
    //不改变状态
    function getState() view public returns(int){
        return state;
    }

    //改变状态
    function changeState(int data) public returns(int){
        state = data;
        return state;
    }

    //回调函数，用于判断调用是否真的指向目标函数
```

```
17      fallback() external {
18          state = 6;
19      }
20  }
21
22  //发起调用的合约
23  contract arrow{
24      address public target;
25      constructor(address _target){
26          target=_target;
27      }
28
29      //不改变状态的staticcall，调用成功，获取traget合约中的状态数据
30      function getState() public returns(bool,bytes memory){
31          return target.staticcall(abi.encodeWithSignature("getState()"));
32      }
33
34      //试图改变target状态的staticcall，调用失败
35      function changeState(int data) public returns(bool,bytes memory){
36          return
    target.staticcall(abi.encodeWithSignature("changeState(int256)",data));
37      }
38
39      //对照组，试图改变target状态的call，调用成功并能改变target的状态变量
40      function changeStatePlus(int data) public returns(bool,bytes memory){
41          return
    target.call(abi.encodeWithSignature("changeState(int256)",data));
42      }
43  }
```

资料来源

EIP-214: New opcode STATICCALL (ethereum.org)