以太merkel proof问题分析

**背景：**

在以太solidity合约中做mapping存储时，如

mapping(uint256 => bytes32) **public** Transactions;

以太会将该storage的Keccak256（key）和value添加到其merkel树中，merkel树根hash在区块头的Root中。

例如：

Transactions[000000000000000000000000000000000000000000000000000000000000023e] = c9f550d3ba5a3c584de7e13d7b4ead29cc8bf09d530c04c054320789384d3394

通过RPC来查询存储状态

**Request**：

{

"jsonrpc":"2.0",

"method":"eth\_getStorageAt",

"params":[

"0x8D069f5a5B877D9ECc5EE28715982c12f3879414", **// 合约**

"0x1fcb2eebde5e81eef238fb4a50a1d5b718fbbb2174404a7f42c53a14e792f2b8", // **Keccak256（key）**

"latest"

],

"id":100

}

**Response**：

{

"jsonrpc": "2.0",

"id": 100,

"result": "0xc9f550d3ba5a3c584de7e13d7b4ead29cc8bf09d530c04c054320789384d3394"

}

通过RPC来查询proof

**Request**:

{

"jsonrpc":"2.0",

"method":"eth\_getProof",

"params":[

"0x8D069f5a5B877D9ECc5EE28715982c12f3879414",

[

"0x1fcb2eebde5e81eef238fb4a50a1d5b718fbbb2174404a7f42c53a14e792f2b8"

],

"latest"

],

"id":100

}

**Response**:

{

    "jsonrpc": "2.0",

    "id": 100,

    "result": {

        "accountProof": [

            "",

            "",

            "",

            "",

            "",

            "",

            "0xf85180a0db9b674ce8e64dcc246c4708218bc773662ae6b2de2d6e6953cbcb240b3709988080808080808080808080a0f60803ca53282fb28312d69ae9bc3dd29f0e131a19da8591b2382b7fee200219808080",

            "0xf8669d3492cbe0c4cc38b98898c3be4350da6c91ec0add9503273a55901cb4c8b846f8440180a05c485d8c06df97d5ae39ac0685a09af428162e12164f4eafef6085f62439ea69a0d5415eb1d2e74e08407476508707137c7e35dbf42995dd07e273c83b4c384c9d"

        ],

        "address": "0x8d069f5a5b877d9ecc5ee28715982c12f3879414",

        "balance": "0x0",

        "codeHash": "0xd5415eb1d2e74e08407476508707137c7e35dbf42995dd07e273c83b4c384c9d",

        "nonce": "0x1",

        "storageHash": "0x5c485d8c06df97d5ae39ac0685a09af428162e12164f4eafef6085f62439ea69",

        "storageProof": [

            {

                "key": "0x1fcb2eebde5e81eef238fb4a50a1d5b718fbbb2174404a7f42c53a14e792f2b8",

                "proof": [

                    "",

                    "",

                    "",

                    "0xf85180a0aff4c547385ac713d0f2b6dab38e28d444d3ece2d6de4914aeab4ee0b4e6c5e98080808080808080a0a1f1f7d3675acf565d621077d0619f0babe86b8631317c44a20927958e7074a6808080808080",

                    "0xf8429f20020928fbe56afa5fcb40252eb39ba9b27bfe8961de523c8b3f73268d9af2a1a0c9f550d3ba5a3c584de7e13d7b4ead29cc8bf09d530c04c054320789384d3394"

                ],

                "value": "0xc9f550d3ba5a3c584de7e13d7b4ead29cc8bf09d530c04c054320789384d3394"

            }

        ]

    }

}

可以看到RPC查询proof返回的value和查询storage返回的value一致，绿色部分标出。在其给出的mekel证明中，叶子节点上的value值和storage一致，黄色部分标出。

**问题：**

我们先看看如果value最前端为0时，查询数据的结果是什么。下面给出了case：

ransactions[000000000000000000000000000000000000000000000000000000000000023d] = 00b93cdc2c48e0574cd27a2fcb580843fc6c3441bcbe467532a9c52aa6055460

注意value最左端是0开始的（十六进制表示）

通过RPC查询存储状态

**Request**：

{

"jsonrpc":"2.0",

"method":"eth\_getStorageAt",

"params":[

"0x8D069f5a5B877D9ECc5EE28715982c12f3879414",

"0x9a215a401a334e0d4544e852c160ee647d373a69d4f8652344f59d6ab6019d14",

"latest"

],

"id":100

}

**Response**：

{

    "jsonrpc": "2.0",

    "id": 100,

    "result": "0x00b93cdc2c48e0574cd27a2fcb580843fc6c3441bcbe467532a9c52aa6055460"

}

查询存储返回的value正确

通过RPC来查询proof

**Request**:

{

"jsonrpc":"2.0",

"method":"eth\_getProof",

"params":[

"0x8D069f5a5B877D9ECc5EE28715982c12f3879414",

[

"0x9a215a401a334e0d4544e852c160ee647d373a69d4f8652344f59d6ab6019d14"

],

"latest"

],

"id":100

}

**Response**:

{

    "jsonrpc": "2.0",

    "id": 100,

    "result": {

        "accountProof": [

            "",

            "",

            "",

            "",

            "",

            "",

            "0xf85180a0db9b674ce8e64dcc246c4708218bc773662ae6b2de2d6e6953cbcb240b3709988080808080808080808080a0f60803ca53282fb28312d69ae9bc3dd29f0e131a19da8591b2382b7fee200219808080",

            "0xf8669d3492cbe0c4cc38b98898c3be4350da6c91ec0add9503273a55901cb4c8b846f8440180a05c485d8c06df97d5ae39ac0685a09af428162e12164f4eafef6085f62439ea69a0d5415eb1d2e74e08407476508707137c7e35dbf42995dd07e273c83b4c384c9d"

        ],

        "address": "0x8d069f5a5b877d9ecc5ee28715982c12f3879414",

        "balance": "0x0",

        "codeHash": "0xd5415eb1d2e74e08407476508707137c7e35dbf42995dd07e273c83b4c384c9d",

        "nonce": "0x1",

        "storageHash": "0x5c485d8c06df97d5ae39ac0685a09af428162e12164f4eafef6085f62439ea69",

        "storageProof": [

            {

                "key": "0x9a215a401a334e0d4544e852c160ee647d373a69d4f8652344f59d6ab6019d14",

                "proof": [

                    "",

                    "",

                    "",

                    "0xf851808080808080808080808080a0bf160cb5d412ac51aaab38102ed4a3e642e84bbb6ea7ef16a85fee6ca3ddaea780a0acbfbae5393615847e1b039a42d544adde87c52b9d98523640bacf4a0935571a8080",

                    "0xf8419f2094275d2782da6748b94922bbcd1c251ada014f58c11c9e6fb131ceafa97ca09fb93cdc2c48e0574cd27a2fcb580843fc6c3441bcbe467532a9c52aa6055460"

                ],

                "value": "0xb93cdc2c48e0574cd27a2fcb580843fc6c3441bcbe467532a9c52aa6055460"

            }

        ]

    }

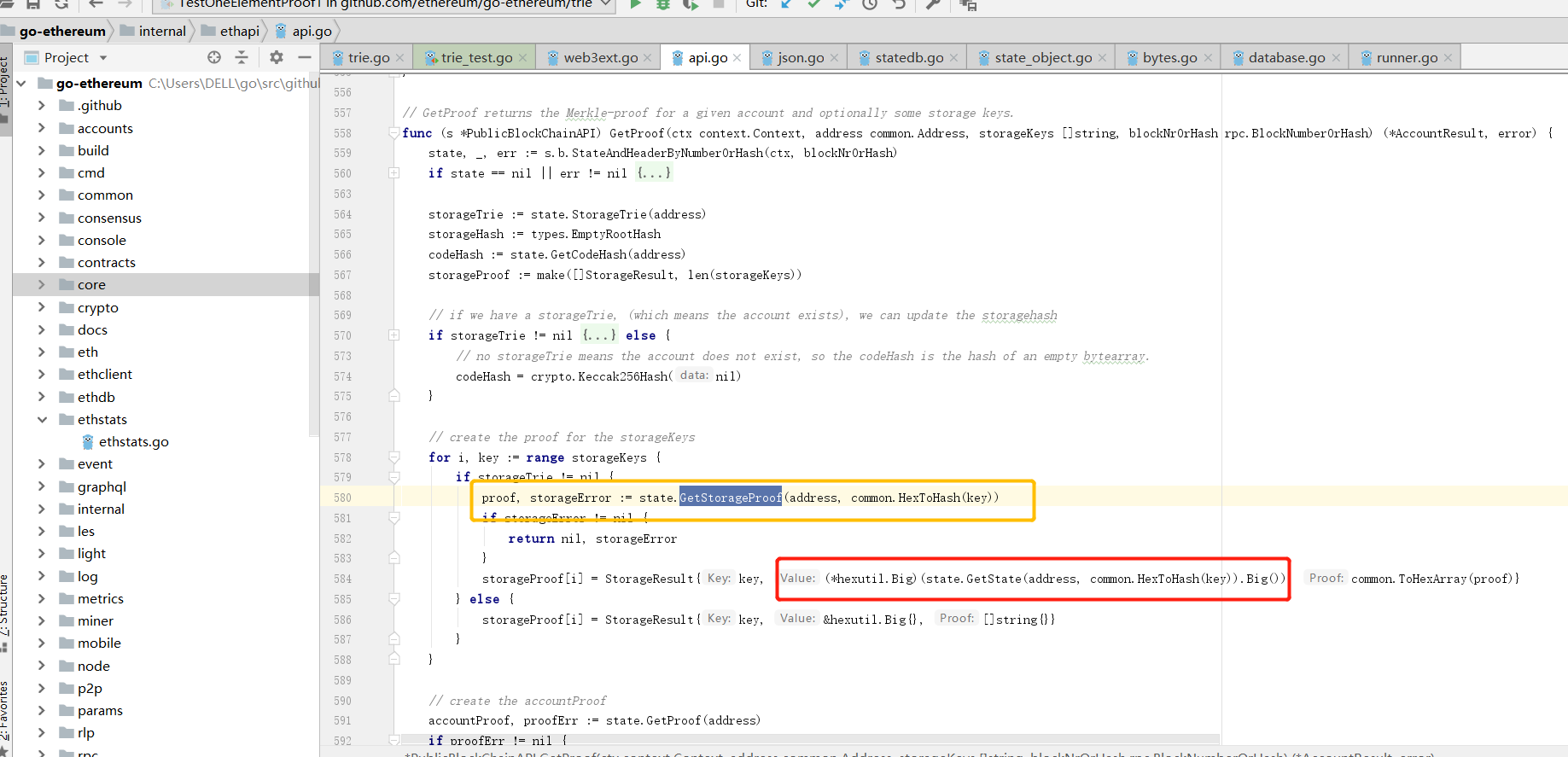
}

可以看到查询proof返回的value和查询存储返回的value有点区别，在上面绿色标出。在其给出的merkel证明中，叶子节点上的数据在上面黄色标出。

看样子以太生成proof时对value处理机制不简单。下面为proof对value的处理机制描述。

**原因**：

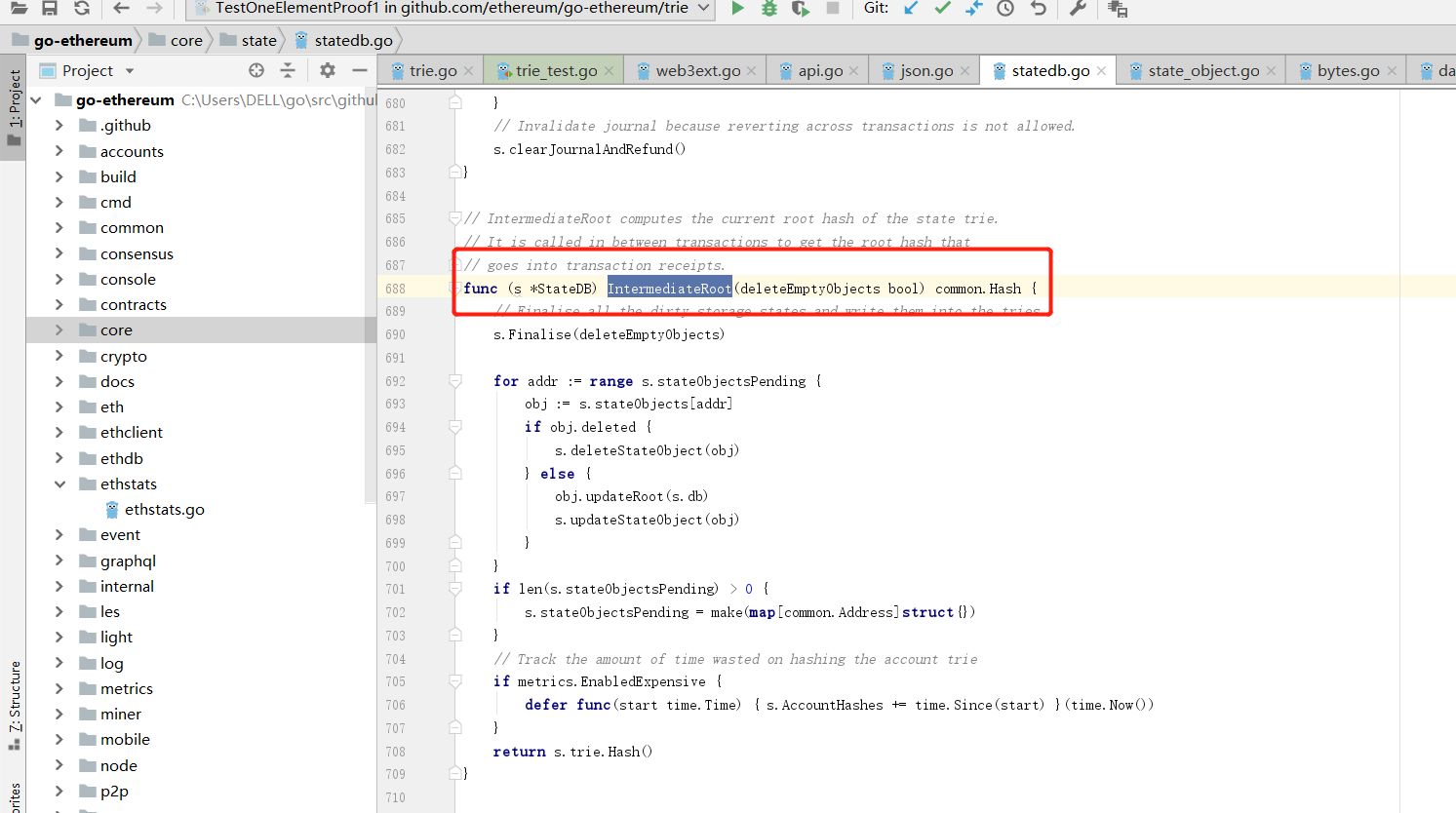
先看看通过RPC获取proof时，以太怎么给出proof的。



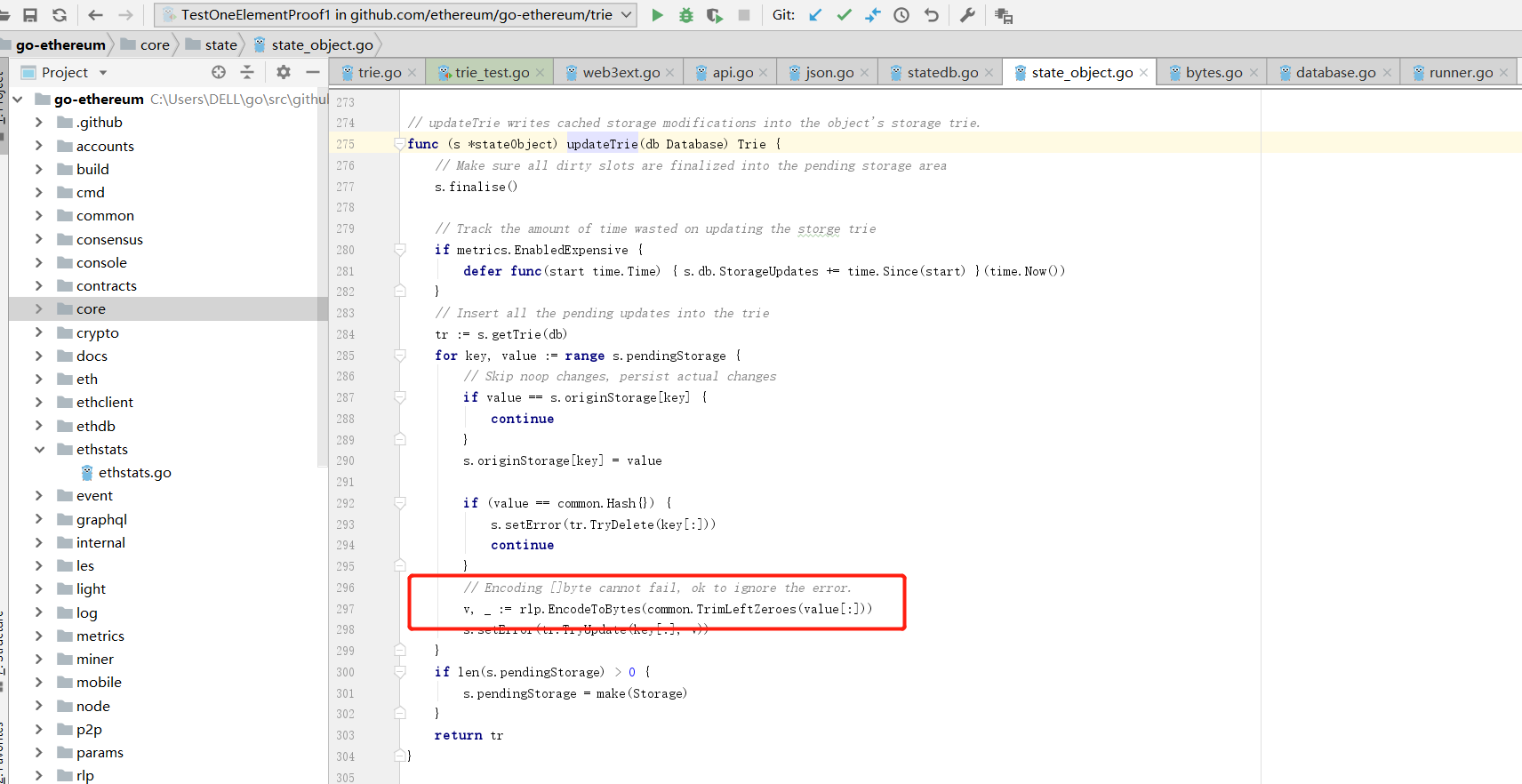
在给出proof的value时，将存储数据转为big.int，那后续json后，最左端0会被剔除掉。

那merkel树的叶子节点上的value为什么左端0也被剔除掉？再看生成block的root hash时以太怎么计算的。

下面是计算以太merkel树的入口



在最后处理value时



在计算merkel树时，对叶子节点上的value，把最左端的0剔除掉。

**结论**：

在验证value的merkel proof时，需要先剔除value最左端的0，或者将proof中给出的value的最左端补齐0到value的长度。这样才能通过merkel proof的验证。

**以太Proof中Key的处理分析：**

**现象：**

先看一个查询

proof req is:

{  
    **"jsonrpc"**:**"2.0"**,  
    **"method"**:**"eth\_getProof"**,  
    **"params"**:[  
        **"0x10BCC4B6C2555fF48540571ebe5aBa6D32915250"**, //合约地址  
        [  
            **"0x02d9f8353bca53bc9b195aa186ab6d98b49a9120c00257ee2c7d860c26f864ea" // store的key**  
        ],  
        **"0x79ad94" // 查询的高度**  
    ],  
    **"id"**:**1**  
}

proof rsp is:

{  
    **"jsonrpc"**:**"2.0"**,  
    **"result"**:{  
        **"accountProof"**:[  
            **""**,  
            **""**,  
            **""**,  
            **""**,  
            **""**,  
            **""**,  
            **"0xf87180a08fbdd95c5250c6d309bf062d1718aa67e07e2b8959c38915865930facd5e515a8080808080808080a00f867817b68ba58735bcf2635efb8a25b41d79a5ccd6bafb8c76bd20f4b9d3c88080a05d16ec4ebca0c90149fb623fa8d5d0b8340e02cba62f0800b75c445e0d1c0775808080"**,  
            **"0xf8669d311010770374ee72a5655c343d4c3fc9914ddbc79b4f4647dd8257d668b846f8440180a070efb9dc368254b0645df7d904d0e3318cadd9fa9c154677a0c968fce9c63d9fa004c3063bdff2748cfc1e9690ebe993875894b4843918014099f30abe04ffe65d"**  
        ],  
        **"address"**:**"0x10bcc4b6c2555ff48540571ebe5aba6d32915250"**,  
        **"balance"**:**"0x0"**,  
        **"codeHash"**:**"0x04c3063bdff2748cfc1e9690ebe993875894b4843918014099f30abe04ffe65d"**,  
        **"nonce"**:**"0x1"**,  
        **"storageHash"**:**"0x70efb9dc368254b0645df7d904d0e3318cadd9fa9c154677a0c968fce9c63d9f"**,  
        **"storageProof"**:[  
            {  
                **"key"**:**"0x2d9f8353bca53bc9b195aa186ab6d98b49a9120c00257ee2c7d860c26f864ea"**,  
                **"proof"**:[  
                    **""**,  
                    **""**,  
                    **"0xf8d1a04b04eb1bb042990ec098db0d02f51062bc76d71b26ec1789253a7c70329e2f9980a0816499520f593d56ae99bdba3eb3ae1651a9c75298083a0aa2b99aa68a20ec4380808080a0a77d00d3a762bf0ed1e7ae78fe8b1e2a8b09d956e3e25bd3209d1af4d0a6252fa00bbd63bfa772ca4b9f7883ca83abdb7595dd40f868482b3b02446f1f9fb377f6808080a01f83049ba2377c55daa3445a11998a88802357079ef7c25f82e95dacee50456a80a0f2109e9a0e37462aaa074eacdb16c37bc04114b41aa67541cce065ed8d74acee8080"**,  
                    **"0xf85180808080a0f1db927de76f2a5d7a019999a5bfa62557502a9c4aa0a2d9a55d0186e483ae6e808080a073fbe663f4fe64f58f96db116df67e3115ab3813c1c61026e58fef797bd183c38080808080808080"**,  
                    **"0xf8429f2040a2ede5d9b19fd502c4f2105d43540bbc6f91660d7fcbc47963f8db6939a1a06f754faf1fc54b6b6204402972a1066335a37bb89f817f86ec34d4fe8991b2c3"**  
                ],  
                **"value"**:**"0x6f754faf1fc54b6b6204402972a1066335a37bb89f817f86ec34d4fe8991b2c3"**  
            }  
        ]  
    },  
    **"id"**:**1**  
}

可以看到查询proof的指定的key为

0x02d9f8353bca53bc9b195aa186ab6d98b49a9120c00257ee2c7d860c26f864ea

而返回的proof的结果中key为

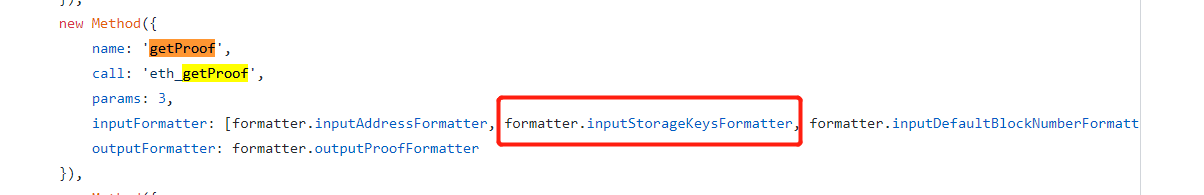
0x2d9f8353bca53bc9b195aa186ab6d98b49a9120c00257ee2c7d860c26f864ea

**结果：**

去掉了最前端的0

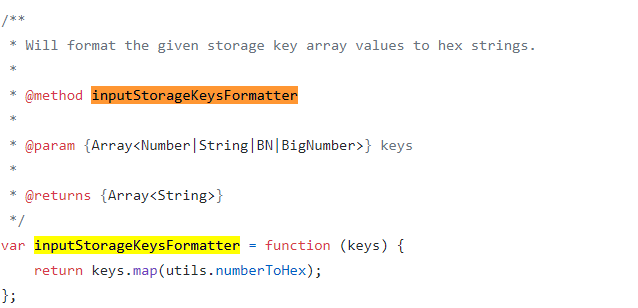
**原因：**

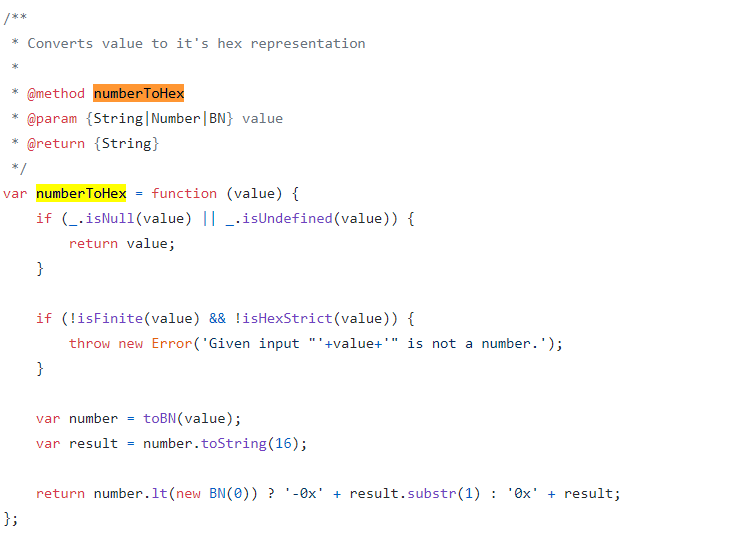
1. Rpc参数格式化

****

Getproof的RPC参数首先使用formatter.inputStorageKeysFormatter进行格式化

1. 格式化处理





主要处理过程：

var number = toBN(value);

var result = number.toString(16);

return number.lt(new BN(0)) ? '-0x' + result.substr(1) : '0x' + result;

将字符串的key转为big number，然后转为16进制的hex string，如果是正数，则最前端加0x。

这样就去掉了最前端的0.