

“谁是大富豪” 编程练习

说明：

本文档用“谁是大富豪”游戏为例，学习 solidity 语言。为了完成和理解游戏代码，本文档中针对代码需要的功能设计了一些练习，希望读者逐个完成这些练习，并测试这些代码。建议不要逐句抄写练习代码，需要看懂后代码后，自行编写和测试。

需求：

两个玩家比谁投入到合约中的以太币多，赢家通吃：

设计：

设计阶段是将需求转换为实现的重要阶段，主要设计软件的结构、模块接口、数据结构等内容。我们先分析程序的数据结构，然后再分析程序的流程。

1. 首先我们对参与游戏的角色分析：

- 游戏参与方有两个人：playerA 和 playerB。因此可以设计两个变量：

```
address public playerA;  
address public playerB;
```

但两行相似的声明有些麻烦，我们也可以用**数组**表示玩家的这个变量：

```
address[2] player;
```

其中第 1 个玩家为 player[0]，第 2 个玩家为 player[1]。

由于玩家需要能够参与交易，完成转账功能。因此需要特殊声明：

```
address payable[2] player;
```

为了便于调制，可以用 public 随时确定 player 的值。

```
address payable[2] public player;
```

- 由于 player 数组只能记录玩家地址，无法记录每个玩家投入多少金额，因此需要一个账本进行记录：

```
mapping(address => uint) public table;
```

- 程序中，总数是一个很重要的参数，常需要纳入考虑。因此，可以定义一个变量，参与游戏的人数：numberOfPlayer

```
uint numberOfPlayer;
```

2.分析游戏过程，并确定程序的模块。

这个游戏比较简单，程序的模块可以与函数一一对应。 分析如下：

参与游戏的玩家需要下注，然后比较大小，最后转账。因此可以设计以下函数完成上述功能：

- 下注：**bet()**
- 比较和转账：**whoisRicher()**

程序的分析、学习和实现：

下面分别实现概要设计的两个函数：bet()和 whoisRicher()。与此同时，可以进一步学习 solidity 语言。另外，和课件 solidity 用例的版本有所区别，我们这里用的是 0.5.0 版本。

1. 实现用于下注的 bet()：

bet()函数支持下注，即需要接收以太币，因此该函数需要用 payable。该函数附加的以太币可以用 msg.value 表示。调用者可以用 msg.sender 表示。

bet()下注后，为了 whoisRicher()的转账功能，还需要记录和判断下注者地址。

综上所述，下面将 bet()的几个功能单独剥离开，做几个练习，热热身。

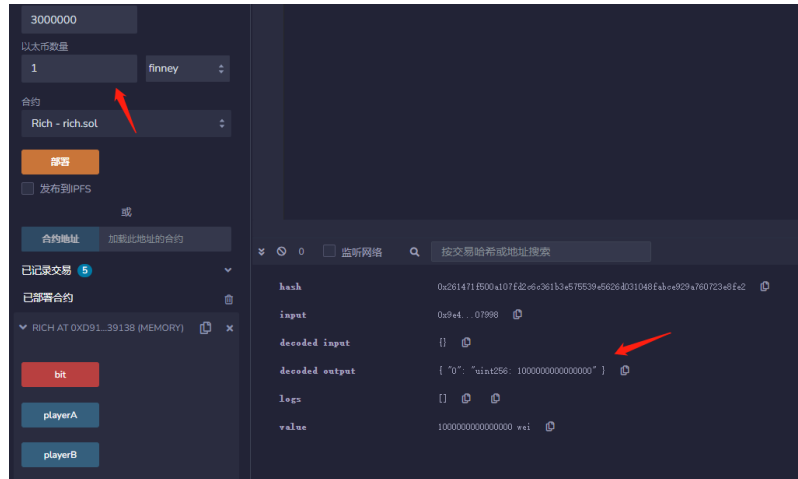
- **实现下注的基本代码：**通过 payable 支持下注，用 msg.value 做为返回值，该值为调用函数的消息（msg）给合约地址支付的以太币。

练习 1

```
pragma solidity ^0.5.0;
contract msgValue {

    function bet() public payable returns(uint){
        return msg.value;
    }
}
```

测试该代码：如果调用 bet()时，附加以太币，交易返回值中能看到下注的值：

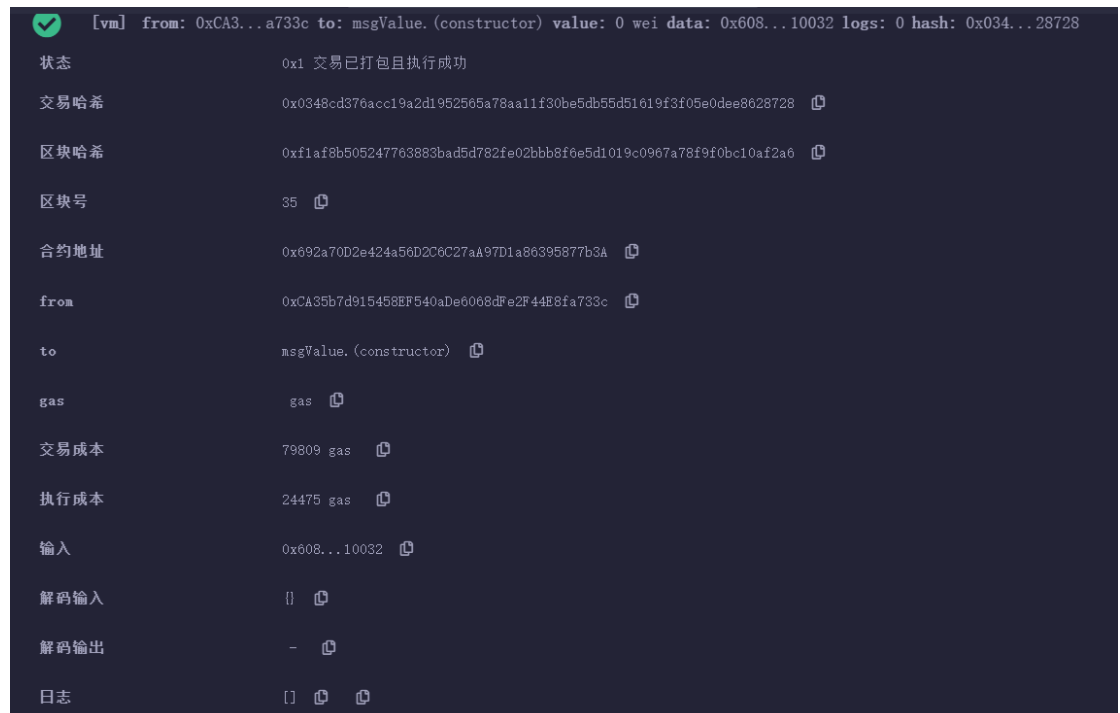


程序代码

```
pragma solidity ^0.5.0;
contract msgValue {

    function bet() public payable returns(uint){ 166 gas
        return msg.value;
    }
}
```

部署



准备调用 bet

以太坊数量

5

▲▼

Finney

合约

msgValue - contracts/HelloWorld1.sol

evm 版本: istanbul

部署

☐ 发布到 IPFS

At Address

加载此地址的合约

已记录的交易 35 ⓘ >

已部署的合约 ⓘ

▼ MSGVALUE AT 0X049...A1FD3 (MEMORY) ⓘ

余额: 0.1 ETH

bet

低级交互 ⓘ

CALLDATA

Transact

成功返回结果

✓	[vm] from: 0xCA3...a733c to: msgValue.bet() 0x049...A1Fd3 value: 5000000000000000 wei data: 0x116...10c25 logs: 0 hash: 0x8b3...c7a69	
状态	0x1 交易已打包且执行成功	
交易哈希	0x8b30c935a218930b962865734474dcd150050a40cc9af105fe39aacb4dec7a69 ⓘ	
区块哈希	0x433eea8265c1b4a587ed33399419142f5b6b46fc371c1538b017ee64d76c724b ⓘ	
区块号	36 ⓘ	
from	0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c ⓘ	
to	msgValue.bet() 0x0498E7c793D7432Cd9dB27fb02fc9cfdEaFA1Fd3 ⓘ	
gas	gas ⓘ	
交易成本	21230 gas ⓘ	
执行成本	166 gas ⓘ	
输入	0x116...10c25 ⓘ	
解码输入	{ ⓘ	
解码输出	{ "0": "uint256: 5000000000000000" } ⓘ	

- 如果限制玩家为 2，需要在 bet() 中，添加参与人数的限制：
因此需要练习一下 if 语句和 revert() 的使用：

练习 2

```
pragma solidity ^0.5.0;
contract ifAndRevert {
    uint numberOfPlayer = 0;

    function bet() public payable returns(uint){
        if(numberOfPlayer>=2){
            revert();
        }
        numberOfPlayer++;
        return msg.value;
    }
}
```

程序代码

```
pragma solidity ^0.5.0;
contract ifAndRevert {
    uint numberOfPlayer = 0;

    function bet() public payable returns(uint){
        if(numberOfPlayer>=2){
            revert();
        }
        numberOfPlayer++;
        return msg.value;
    }
}
```

部署

[vm] from: 0xCA3...a733c to: ifAndRevert. (constructor) value: 0 wei data: 0x608...10032 logs: 0 hash: 0xf3a...64d59	
状态	0x1 交易已打包且执行成功
交易哈希	0xf3a782a014dff50458dda9fa54024271582e3f63f8d56bcb24c288a56c64d59
区块哈希	0x843f494a38e014c69631ec7b1b2420e7df8fa3dc9da29babbbc4a0543299d161
区块号	40
合约地址	0xef55BfAc4228981E850936AAf042951F7b146e41
from	0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c
to	ifAndRevert. (constructor)
gas	gas
交易成本	88735 gas
执行成本	32887 gas
输入	0x608...10032
解码输入	[]

仍然能够成功返回

以太币数量

2

Finney

合约

ifAndRevert - contracts/HelloWorld1.sol

evm 版本: istanbul

部署

☐ 发布到 IPFS

合约地址

At Address

加载此地址的合约

已记录的交易 40 ⓘ >

已部署的合约

IFANDREVERT AT 0XEF5...46E41 (MEMORY)

余额 0. ETH

bet

低级交互 ⓘ

CALLDATA

Transact

	[vm] from: 0xCA3...a733c to: ifAndRevert.bet() 0xef5...46e41 value: 2000000000000000 wei data: 0x116...10c25 logs: 0 hash: 0x6de...27353		
状态	0x1 交易已打包且执行成功		
交易哈希	0x6de3d01a2dae8e1d353e5aef648b3a5f2796cacd88924341c6feb93013727353		
区块哈希	0xb2dcb2239494c2468bc804538528bdc78e1d28f268f63d01b5846de1e5e5d9b0		
区块号	41		
from	0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c		
to	ifAndRevert.bet() 0xef55BfAc4228981E850936AAf042951F7b146e41		
gas	gas		
交易成本	43490 gas		
执行成本	22426 gas		
输入	0x116...10c25		
解码输入	{		
解码输出	{ "0": "uint256: 2000000000000000" }		
日志	[]		
val	2000000000000000 wei		

- 如何记录
bet()需要将下注的玩家和投注需要记录下来。可以考虑用 mapping 记录，可以声明一个字典类型的状态变量 table。

```
mapping(address => uint) public table;
```

字典 table 中的 address 列是玩家的地址，uint 列是玩家的投注。玩家的地址保存在前面定义的数组 player[2]中。

可以尝试编写 bet()记录玩家和投注的代码：

练习 3

```
pragma solidity ^0.5.0;
contract putETH {
    address[2] player;
    mapping(address => uint) public table;
    uint numberOfPlayer = 0;

    function bet() public payable returns(uint){
        player[numberOfPlayer] = msg.sender;
        numberOfPlayer++;
        table[msg.sender] = msg.value;
        return numberOfPlayer;
    }
}
```
















注：为了便于测试，table 状态变量我们标注上 public，这样可以随时查询玩家投入了多少钱。你可以尝试检查一下 table 的状态。

程序代码

```
pragma solidity ^0.5.0;
contract putETH {
    address[2] player;
    mapping(address => uint) public table;
    uint numberOfPlayer = 0;

    function bet() public payable returns(uint){ 63623 gas
        player[numberOfPlayer] = msg.sender;
        numberOfPlayer++;
        table[msg.sender] = msg.value;
        return numberOfPlayer;
    }
}
```

部署情况

 [vm] from: 0xCA3...a733c to: putETH. (constructor) value: 0 wei data: 0x608...10032 logs: 0 hash: 0xf14...e782d	
状态	0x1 交易已打包且执行成功
交易哈希	0xf14c5b03d493fd3fc0be59859a7e87345ad6a9b5dc9085d9e98da02ed4ee782d 
区块哈希	0x66b3103cf8ed07adb9b8e1026b0f484ea7772f1670062feb9e3b240425d6d6bb 
区块号	46 
合约地址	0x35eF07393b57464e93dEE59175fF72E6499450cF 
from	0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c 
to	putETH. (constructor) 
gas	gas 
交易成本	148243 gas 
执行成本	88141 gas 
输入	0x608...10032 
解码输入	[] 
解码输出	- 
日志	[]  

投入 5

状态	0x1 交易已打包且执行成功
交易哈希	0xdef293ac48a54529bac71d256f2b3890cd742f4f8b38e27254cba69787868de0
区块哈希	0x0ef3cd989885992d580d5ab194f810e38a460781336517bbac54e264d9606c90
区块号	47
from	0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c
to	putETH.bet() 0x35eF07393b57464e93dEB59175fF72E6499450cF
gas	gas
交易成本	87987 gas
执行成本	66923 gas
输入	0x116...10c25
解码输入	
解码输出	{ "0": "uint256: 1" }
日志	

查询投入

▼ PUTETH AT 0X35E...450CF (MEMORY)

余额: 0.005 ETH

bet

table

0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c

▼

0: uint256: 5000000000000000

CALL [call] from: 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c to: putETH.table(address) data: 0x010...a733c

from

0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c

to

putETH.table(address) 0x35eF07393b57464e93dEB59175fF72E6499450cF

执行成本

2447 gas (当被一个合约调用是需要费用)

输入

0x010...a733c

解码输入

{
 "address ": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c"
}

解码输出

{
 "0": "uint256: 5000000000000000"
}

日志

- 完整的 bet()
前期工作准备完毕，可以写一个完整的 bet()函数了。本游戏只支持两个人充值，因此编写代码时考虑该场景下不同情况的处理方案，包括玩家初始化，某些玩家多

次充值和第 3 个玩家的加入等场景。

当处理多种情况时，可以用“if else if”语句完成条件判断。代码如下：

练习 4

```
pragma solidity ^0.5.0;
contract putETH {
    address[2] player;
    mapping(address => uint) public table;
    uint numberOfPlayer = 0;

    function bet() public payable returns(uint){
        if (msg.sender == player[0]){
            table[msg.sender] += msg.value;
        } else if (msg.sender == player[1]){
            table[msg.sender] += msg.value;
        } else if (numberOfPlayer == 0) {
            player[0] = msg.sender;
            numberOfPlayer++;
            table[msg.sender] += msg.value;
        } else if (numberOfPlayer == 1) {
            player[1] = msg.sender;
            numberOfPlayer++;
            table[msg.sender] += msg.value;
        } else revert();

        return address(this).balance;
    }
}
```

如果看不懂上述代码，那可以看下面对代码的详细解释：

a. 当有一个玩家调用这个函数时，我们首先会检查该玩家是否是 player[0]，

```
if (msg.sender == player[0]){                //针对 0 号玩家， player[0]
    table[msg.sender] += msg.value; //在 table 中更新玩家的账户余额
}
```

如果是 player[0]，我们就会在 table 中， player[0] 账户上记录转过来的费用。

若不是，再检查是否是 player[1]：

```
else if (msg.sender == player[1]){            //针对 1 号玩家， player[0]
    table[msg.sender] += msg.value; //在 table 中更新玩家的账户余额
}
```

如果是 player[1]，我们就会在 table 中， player[1] 账户上记录转过来的费用。

- b. 若上述两者情况都不是，则说明 player 还没有初始化或第 3 个玩家试图加入这个游戏。因此先考虑在没有初始化的情况下，尝试初始化 table 表中 player 的信息：

```
else if (numberOfPlayer == 0) {           //初始化 player[0]
    player[0] = msg.sender; //将调用函数的玩家地址放入 player[0]
    numberOfPlayer++;        //更新 numberOfPlayer
    table[msg.sender] += msg.value; //在 table 中增加一行
} else if (numberOfPlayer == 1) { //下面与 player[0]初始化类似
    player[1] = msg.sender;
    numberOfPlayer++;
    table[msg.sender] += msg.value;
}
```

- c. 若既不是玩家重复充值，也不是初始玩家的情况，就只可能是第 3 个玩家试图加入游戏的情况了，因此用 revert()方法终止合约的执行。

```
else revert();
```













程序代码

```
pragma solidity ^0.5.0;
contract putETH {
    address[2] player;
    mapping(address => uint) public table;
    uint numberOfPlayer = 0;















    function bet() public payable returns(uint){ 66314 gas
        if (msg.sender == player[0]){
            table[msg.sender] += msg.value;
        } else if (msg.sender == player[1]){
            table[msg.sender] += msg.value;
        } else if (numberOfPlayer == 0) {
            player[0] = msg.sender;
            numberOfPlayer++;
            table[msg.sender] += msg.value;
        } else if (numberOfPlayer == 1) {
            player[1] = msg.sender;
            numberOfPlayer++;
            table[msg.sender] += msg.value;
        } else revert();

        return address(this).balance;
    }
}
```

部署


	[vm] from: 0xCA3...a733c to: putETH. (constructor) value: 0 wei data: 0x608...10032 logs: 0 hash: 0x949...fdfb2	
状态	0x1 交易已打包且执行成功	
交易哈希	0x949dd909906ad1fbcfc7f5ff3b7c570e34fb8cdd7021e053f58f6ec0a71fdfb2	
区块哈希	0xa50d17b873c5176e34d8f8c7e2f415b6cc25bb7b33adf666701e5e771d979d00	
区块号	48	
合约地址	0x0c2E77121DaF0270d26Bf0a7E9ab0fAA8Bf739Ef	
from	0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c	
to	putETH. (constructor)	
gas	gas	
交易成本	273189 gas	
执行成本	204051 gas	
输入	0x608...10032	
解码输入	{}	
解码输出	-	
日志	[]	 

第一个账户 bet 2Finney 及查询

	[vm] from: 0x5B3...eddC4 to: putETH.bet() 0x0c2...739Ef value: 2000000000000000 wei data: 0x116...10c25 logs: 0 hash: 0x1aa...ed44d	
状态	0x1 交易已打包且执行成功	
交易哈希	0x1aa3bc59b577c304026ff0ca525bc291a1115ae46c9d3c056fed3a5045aed44d	
区块哈希	0xdefc0ef68f50242d0444c465d46c6816cad34bbddae548f5240401a278d6394a	
区块号	49	
from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4	
to	putETH.bet() 0x0c2E77121DaF0270d26Bf0a7E9ab0fAA8Bf739Ef	
gas	gas	
交易成本	90351 gas	
执行成本	69287 gas	
输入	0x116...10c25	
解码输入	{}	
解码输出	{ "0": "uint256: 2000000000000000" }	
日志	[]	 
val	2000000000000000 wei	
		


CALL	[call] from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: putETH.table(address) data: 0x010...eddc4			
from	0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2	🔗		
to	putETH.table(address)	0x0c2E77121DaF0270d26Ef0a7E9ab0fAA8Bf739Ef	🔗	
执行成本	2447 gas	(当被一个合约调用是才需要费用) 🔗		
输入	0x010...eddc4	🔗		
解码输入	{ "address ": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4" }			🔗
解码输出	{ "0": "uint256: 2000000000000000" }			🔗
日志	[]			🔗 🔗

第二个账户 bet 5Finney 及查询

	[vm] from: 0xAb8...35cb2 to: putETH.bet() 0x0c2...739Ef value: 5000000000000000 wei data: 0x116...10c25 logs: 0 hash: 0xe34...38db0		
状态	0x1 交易已打包且执行成功		
交易哈希	0xe3434beedc098e26b6a5c0455bf571b6e31ed730b12a09a8517cd78455738db0	🔗	
区块哈希	0xb3a7e8e7231a648eef4bd232de8c0a9e246b15f5be353901dcb5a71486e7b5a4	🔗	
区块号	50	🔗	
from	0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2	🔗	
to	putETH.bet() 0x0c2E77121DaF0270d26Bf0a7E9ab0fAA8Bf739Ef	🔗	
gas	gas	🔗	
交易成本	73378 gas	🔗	
执行成本	52314 gas	🔗	
输入	0x116...10c25	🔗	
解码输入	{	🔗	
解码输出	{ "0": "uint256: 7000000000000000" }	🔗	
日志	[]	🔗 🔗	
val	5000000000000000 wei	🔗	

<i>CALL</i>	[call] from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: putETH.table(address) data: 0x010...35cb2
from	0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 🔗
to	putETH.table(address) 0x0c2E77121DaF0270d26Bf0a7E9ab0fAA8Bf739Ef 🔗
执行成本	2447 gas (当被一个合约调用是才需要费用) 🔗
输入	0x010...35cb2 🔗
解码输入	{ "address": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2" }
解码输出	{ "0": "uint256: 5000000000000000" }
日志	[] 🔗 🔗

尝试使用第三个账户转入，但是失败

	[vm] from: 0x4B2...C02db to: putETH.bet() 0x0c2...739Ef value: 1000000000000000 wei data: 0x116...10c25 logs: 0 hash: 0xf8d...8e39b
状态	0x0 交易已打包但执行失败
交易哈希	0xf8d1759bfe07b713068e177aad92ddc310030e95c45bf78750410f0e9e28e39b 🔗
区块哈希	0x87aa840ea909d503f11f3034d86af06b297889afe5f0486669e21088712f84e 🔗
区块号	51 🔗
from	0x4E20993Bc481177ec7E8f571ceCaE8A9e22C02db 🔗
to	putETH.bet() 0x0c2E77121DaF0270d26Bf0a7E9ab0fAA8Bf739Ef 🔗
gas	gas 🔗
交易成本	27840 gas 🔗
执行成本	6776 gas 🔗
输入	0x116...10c25 🔗
解码输入	[] 🔗
解码输出	{ "error": "Failed to decode output: Error: hex data is odd-length (argument=\"value\", value=\"0x0\", code=INVALID_ARGUMENT, version=bytes/5.7.0)" }
日志	[] 🔗 🔗
val	10000000000000000 wei 🔗

第一个账户追加 7Finney

✓ [vm]	from: 0x5B3...eddc4 to: putETH.bet() 0x0c2...739Ef value: 7000000000000000 wei data: 0x116...10c25 logs: 0 hash: 0x3e1...a3e4c
状态	0x1 交易已打包且执行成功
交易哈希	0x3e13192c12d2ad8942387ab963f9c5d285c9b04dae0224dd742f4fee2e1a3e4c
区块哈希	0xeb53b1af788ea60e0aa4042ff9b627cb370ecde0a4c59e2e36734eeafe0b36cd
区块号	52
from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to	putETH.bet() 0x0c2E77121DaF0270d26Bf0a7E9ab0fAA8Bf739Ef
gas	gas
交易成本	28589 gas
执行成本	7525 gas
输入	0x116...10c25
解码输入	{}
解码输出	{ "0": "uint256: 14000000000000000" }
日志	[]

CALL	[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: putETH.table(address) data: 0x010...eddc4
from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to	putETH.table(address) 0x0c2E77121DaF0270d26Bf0a7E9ab0fAA8Bf739Ef
执行成本	2447 gas (当被一个合约调用时才需要费用)
输入	0x010...eddc4
解码输入	{ "address ": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4" }
解码输出	{ "0": "uint256: 9000000000000000" }
日志	[]

2. 实现用于比较和转账: whoisRicher()

该函数需要比较 player[0]和 player[1]投注的多寡,然后将合约的全部资产交给投注最多的玩家。因此代码需要有两种功能,一个是比较投注,另一个是将合约资产转交给的玩家。

- 比较投注:

前面的代码中,用 table 字典保存了 player[0]和 player[1]的投注,因此可以对这两个地址中记录的本金进行比较。比较结果有三种,即 player[0]得钱最多、player[1]得钱最多和两者相等。这里可以用“if else if”语句完成比较。

```
if (table[player[0]] < table[player[1]]) {  
  
  
} else if (table[player[0]] > table[player[1]]) {
```

```
} else {  
  
}
```

- 合约资产转交

table 字典中记录的是投注的情况，但实际的总资产是保存在合约中的。因此需要解决合约资产转交给数组的对象。下面用一个简单的示例说明：

练习 5

1	pragma solidity ^0.5.0;
2	contract msgValue {
3	address payable[2] public player;
4	
5	function input() public payable returns(uint){
6	player[0] = msg.sender;
7	return address(this).balance;
8	}
9	
10	function out() public returns(address, uint){
11	player[1] = msg.sender;
12	player[1].transfer(address(this).balance);
13	return (player[1], player[1].balance);
14	}
15	}

这个程序有两个函数，input()函数确定 player[0]的地址，并接收以太币。out()函数确定 player[1]，并转予合约资产。

注意第 3 行中，声明 player 数组，数组内部的变量的数据类型为 address，且数组变量能够接收交易。简单的说，可以让合约将资产转给 player[0]和 player[1]。

第 5 行，payable 表示该函数可以接收以太币。

第 6 行，数组变量 player[0] 赋予了调用该函数的玩家地址。

第 11 行，数组变量 player[1] 赋予了调用该函数的玩家地址。

第 12 行，player[1]接收该合约的所有资产。其中，合约资产用“address(this).balance”表示。

程序


```


pragma solidity ^0.5.0;
contract msgValue {
    address payable[2] public player;

    function input() public payable returns(uint){ 21108 gas
        player[0] = msg.sender;
        return address(this).balance;
    }

    function out() public returns(address, uint){ infinite gas
        player[1] = msg.sender;
        player[1].transfer(address(this).balance);
        return (player[1], player[1].balance);
    }
}

```

部署

<div>  [vm] from: 0x5B3...eddC4 to: msgValue. (constructor) value: 0 wei data: 0x608...10032 logs: 0 hash: 0x76a...cd57b </div>	
状态	0x1 交易已打包且执行成功
交易哈希	0x76aaa93e8beb036f661b3fd447e429847cb3800c2230c00ea22f49bcf9ccd57b 🔗
区块哈希	0xedc8bd3fb463aacbbcla5ceca9ca2ea5179c8e16d929609d32dbe5a21f7e2e5f 🔗
区块号	57 🔗
合约地址	0x9396B453Fad71816cA9f152Ae785276a1D578492 🔗
from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 🔗
to	msgValue. (constructor) 🔗
gas	gas 🔗
交易成本	230810 gas 🔗
执行成本	164608 gas 🔗
输入	0x608...10032 🔗
解码输入	🔗 🔗

调用 input 和 out

状态	0x1 交易已打包且执行成功
交易哈希	0x903da3a62d85d097f95f3776f985aa432da6c0e41c2e5d1796c0655121be1867
区块哈希	0xb0b8f53f8f2e7b4ef5a3e04faa3279537864a2fc955171cd363033512f6af8f1
区块号	58
from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to	msgValue.input() 0x9396B453Fad71816cA9f152Ae785276a1D578492
gas	gas
交易成本	43472 gas
执行成本	22408 gas
输入	0xae...d3f4f
解码输入	{}
解码输出	{ "0": "uint256: 0" }
日志	[]

状态	0x1 交易已打包且执行成功
交易哈希	0xaa6c27d13a650fe0a5e9fe90e2a779152ceca00fbf3454908acc9b3f8e5243d1
区块哈希	0xd7f7d1191a1bf362d570db41d1f5703f7e221c0c9a65482ff586e6804e111d99
区块号	59
from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to	msgValue.out() 0x9396B453Fad71816cA9f152Ae785276a1D578492
gas	gas
交易成本	44333 gas
执行成本	23269 gas
输入	0xb2a...1449b
解码输入	{}
解码输出	{ "0": "address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4", "1": "uint256: 99990999999995296751" }
日志	[]

```
player[0].transfer(address(this).balance);
```

另外，在 0.5.0 之前的版本，合约地址的资产可以用 `this.balance` 表示，在 0.5.0 之后的版本，需要用下面表示方式。即高版本的编译器使用 `address(this)`来访问

address 成员。

```
return address(this).balance;
```

在上述基础上，可以完成 whoisRicher()代码的编写：

```
1 function whoisRicher() public returns(uint){
2     if(table[player[0]]>table[player[1]]){
3         player[0].transfer(address(this).balance);
4     } else if(table[player[0]]<table[player[1]]){
5         player[1].transfer(address(this).balance);
6     } else {
7         player[0].transfer(address(this).balance/2);
8         player[1].transfer(address(this).balance/2);
9     }
10    table[player[0]] = 0;
11    table[player[1]] = 0;
    return(address(this).balance);
}
```

这段代码在“if else if”构成的逻辑架构下，完成比较和赋值的操作。在第 10，11 行完成 table 的初始化。请根据这个代码学习数组的转账功能。

游戏代码

至此，我们就可以完成整个游戏的代码：

```
pragma solidity ^0.5.0;
contract putETH {
    address payable[2] player;
    mapping(address => uint) public table;
    uint numberOfPlayer = 0;

    function bet() public payable returns(uint){
        if (msg.sender == player[0]){
            table[msg.sender] += msg.value;
        } else if (msg.sender == player[1]){
            table[msg.sender] += msg.value;
        } else if (numberOfPlayer == 0) {
            player[0] = msg.sender;
            numberOfPlayer++;
            table[msg.sender] += msg.value;
        } else if (numberOfPlayer == 1) {
```

```
        player[1] = msg.sender;
        numberOfPlayer++;
        table[msg.sender] += msg.value;
    } else revert();

    return address(this).balance;
}

function whoisRicher() public returns(uint){

    if(table[player[0]]>table[player[1]]){
        player[0].transfer(address(this).balance);
    } else if(table[player[0]]<table[player[1]]){
        player[1].transfer(address(this).balance);
    } else {
        player[0].transfer(address(this).balance/2);
        player[1].transfer(address(this).balance/2);
    }
    table[player[0]] = 0;
    table[player[1]] = 0;
    return (address(this).balance);
}
}
```

程序

```
pragma solidity ^0.5.0;
contract putETH {
    address payable[2] player;
    mapping(address => uint) public table;
    uint numberOfPlayer = 0;
















    function bet() public payable returns(uint){ 66314 gas
        if (msg.sender == player[0]){
            table[msg.sender] += msg.value;
        } else if (msg.sender == player[1]){
            table[msg.sender] += msg.value;
        } else if (numberOfPlayer == 0) {
            player[0] = msg.sender;
            numberOfPlayer++;
            table[msg.sender] += msg.value;
        } else if (numberOfPlayer == 1) {
            player[1] = msg.sender;
            numberOfPlayer++;
            table[msg.sender] += msg.value;
        } else revert();

        return address(this).balance;
    }


    function whoisRicher() public returns(uint){ infinite gas



        if(table[player[0]]>table[player[1]]){
            player[0].transfer(address(this).balance);
        } else if(table[player[0]]<table[player[1]]){
            player[1].transfer(address(this).balance);
        } else {
            player[0].transfer(address(this).balance/2);
            player[1].transfer(address(this).balance/2);
        }
        table[player[0]] = 0;
        table[player[1]] = 0;
        return (address(this).balance);
    }
}
```

部署


 [vm] from: 0x5B3...eddC4 to: putETH. (constructor) value: 0 wei data: 0x608...10032 logs: 0 hash: 0xf98...54a6b	
状态	0x1 交易已打包且执行成功
交易哈希	0xf9842c289555b546c2463941b98cb887b849552d7444c0346b1fa7b02954a6b 
区块哈希	0x2afaeabe14ade6fb2454566175b45d33b4c4d9c490ca7997dac3eeffb9cd1832 
区块号	60 
合约地址	0x9a2E12340354d2532b4247da3704D2A5d73Bd189 
from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 
to	putETH. (constructor) 
gas	gas 
交易成本	541960 gas 
执行成本	453492 gas 
输入	0x608...10032 
解码输入	[] 
解码输出	- 
日志	[]  




账户 1 投入 5Finney

环境 

Remix VM (Shanghai)  

VM


账户 


0x5B3...eddC4 (100.005999999998745274 ether)   

GAS 上限


3000000

以太币数量

5 

Finney 

合约

putETH - contracts/HelloWorld1.sol 

evm 版本: istanbul


部署


☐ 发布到 IPFS


At Address



加载此地址的合约

已记录的交易

9 



已部署的合约 


▼ PUTETH AT 0X7EF...8CB47 (MEMORY)  















余额: 0 ETH

bet

whoisRicher

table

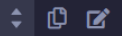
address 

	[vm] from: 0x5B3...eddC4 to: putETH.bet() 0xd91...39138 value: 5000000000000000 wei data: 0x116...10c25 logs: 0 hash: 0xacd...32128	
状态	0x1 交易已打包且执行成功	
交易哈希	0xacdf898e7b8473e9b1d88f7de54c0c9e922839d48b8dfcbe684326ae69032128 	
区块哈希	0x886f5d7db43b09d940c86ef9ec62e68f0574bdf151f39a54a2e159dffee25ca8 	
区块号	2 	
from	0x5E38Da6a701c568545dCfcB03FcB875f56beddC4 	
to	putETH.bet() 0xd9145CCE52D386f254917e481eB44e9943F39138 	
gas	gas 	
交易成本	90351 gas 	
执行成本	69287 gas 	
输入	0x116...10c25 	
解码输入	{ 	
解码输出	{ "0": "uint256: 5000000000000000" } 	
日志	[]  	
val	5000000000000000 wei 	

账户 2 投入 3Finney

账户 

0xAb8...35cb2 (99.990999999999733781 ether)



GAS 上限

3000000

以太币数量

3



Finney



合约

putETH - contracts/HelloWorld1.sol




evm 版本: istanbul

部署

☐ 发布到 IPFS

At Address

加载此地址的合约

已记录的交易 11 



已部署的合约



▼ PUTETH AT 0X7EF...8CB47 (MEMORY)



余额: 0.008 ETH

bet

whoisRicher

table

address



[vm] from: 0xAb8...35cb2 to: putETH.bet() 0xd91...39138 value: 3000000000000000 wei data: 0x116...10c25 logs: 0 hash: 0x1bf...99cd5

状态

0x1 交易已打包且执行成功

交易哈希

0x1bfa15c85c9ee2ff06ee03bdc81754d9f4e758e87e5f7cad8da3e672bb599cd5 [🔗](#)

区块哈希

0x0cf7baddfb35b00bde0e4a9c35ff3b34b0ecccc245ad88efb1c54cd38bfbbae90 [🔗](#)

区块号

3 [🔗](#)

from

0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 [🔗](#)

to

putETH.bet() 0xd91450CE52D386f254917e481eB44e9943F39138 [🔗](#)

gas

gas [🔗](#)

交易成本

73378 gas [🔗](#)

执行成本

52314 gas [🔗](#)

输入

0x116...10c25 [🔗](#)

解码输入

{ [🔗](#)

解码输出

{
 "0": "uint256: 8000000000000000"
}
[🔗](#)

日志

[] [🔗](#) [🔗](#)

val

3000000000000000000 wei [🔗](#)

进行比较和转账

余额: 0.008 ETH

bet

whoisRicher

table

address [▼](#)

[vm] from: 0xAb8...35cb2 to: putETH.whoisRicher() 0xd91...39138 value: 0 wei data: 0xf85...c5dda logs: 0 hash: 0x131...27eee

状态

0x1 交易已打包且执行成功

交易哈希

0x1312af13067e7a9a5a7f4c9d5cad61fd025c36ffa6e38dbcfb1d5e1c2fd27eee [🔗](#)

区块哈希

0x3d58cf4f0b9539d2b30459233001b9460087f52fe5bb0e7f7472eed5932d0405 [🔗](#)

区块号

4 [🔗](#)

from

0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 [🔗](#)

to

putETH.whoisRicher() 0xd91450CE52D386f254917e481eB44e9943F39138 [🔗](#)

gas

gas [🔗](#)

交易成本

36766 gas [🔗](#)

执行成本

24893 gas [🔗](#)

输入

0xf85...c5dda [🔗](#)

解码输入

{ [🔗](#)

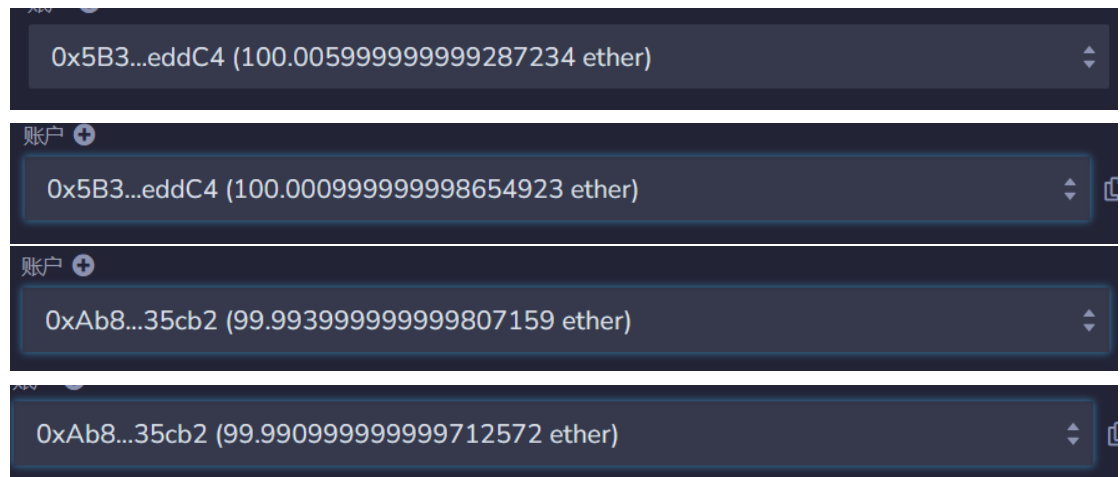
解码输出

{
 "0": "uint256: 0"
}
[🔗](#)

日志

[] [🔗](#) [🔗](#)

查看各个账户前后情况



思考：

1. 请思考，如何测试这个代码呢？

测试如上述过程所示

2. 不知道大家注意到没有，这个游戏漏洞比较大，因为投注的以太坊额度是可以通过交易查看到，第二个玩家可以作弊。因此，投注保密的问题尚需要解决，那这个程序如何修改呢？

第二个玩家投注后，可以看到连同自己投注的总和。将返回值与 table 函数结合，仅仅返回自己账户的结果。

当完成这个文档中的各个练习，理解游戏代码后，能够尝试自己编写一个骰子游戏，就可以动手编写剪刀石头布游戏代码了。