下半节

一、ABI.encode系列方法

（1）介绍abi.encode是什么

（2）演示不同encode的用法

（3）讲解什么时候需要使用

（4） 思考题 在testAbiSeletor时为什么要使用byte4进行转化

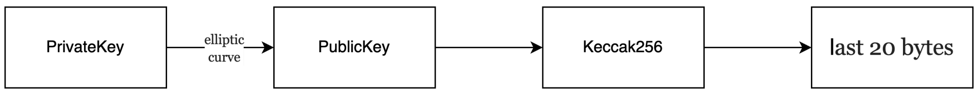
```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;
import "https://github.com/NomicFoundation/hardhat/blob/main/packages/hardhat-core/console.sol";

contract TestAbiEncode{

// 0x04d2
// 0x1234
// 0x31323334
     function testAbiEncodePacked() pure external {
        console.logBytes(abi.encodePacked(uint16(1234)));
        console.logBytes(abi.encodePacked(bytes2(0x1234)));
        console.logBytes(abi.encodePacked("1234"));
    }

// 0x00000000000000000000000000000000000000000000000000000000000004d2
// 0x0000000000000000000000000000000000000000000000000000000000001234
// 0x000000000000000000000000000000000000000000000000000000000000002000000000000000
000000000000000000000000000000000000000000000000043132333400000000000000000000000
000000000000000000000000000000000
   function testAbiEncode() pure external {
        console.logBytes(abi.encode(uint16(1234)));
        console.logBytes(abi.encode(0x1234));
        console.logBytes(abi.encode("1234"));
    }


// 0x00000000000000000000000000000000000000000000000000000000000004d200000000000000
000000000000000000000000000000000000000000000000040000000000000000000000000000000
000000000000000000000000000000000431323334000000000000000000000000000000000000000
0000000000000000
// 0x04d231323334
     function testAbiEncodeCombined() pure external {
        console.logBytes(abi.encode(1234,"1234"));
        console.logBytes(abi.encodePacked(uint16(1234),"1234"));

    }

    function testAbiDecode() pure external {
        bytes memory _encode=abi.encode(1234,"1234");
        (uint256 u,string memory s)=abi.decode(_encode,(uint256,string));
        console.logUint(u);
        console.logString(s);

        //  _encode=abi.encodePacked(uint16(1234),"1234");
        // (uint256 u2,string memory s2)=abi.decode(_encode,(uint256,string));
        // console.logUint(u2);
        // console.logString(s2);
    }

    function testAbiMethod(uint a,bytes2 b) external {

    }

    //函数签名=函数名+完整的参数定义
    function testAbiSignature() pure external {
        console.logBytes(abi.encodeWithSignature("testAbiMethod(uint256,bytes2)",1,bytes2(0x1234)));
    }
    //selector=bytes4(kaccake246(函数签名))
    function testAbiSeletor() pure external {

        console.logBytes(abi.encodeWithSelector(TestAbiEncode.testAbiMethod.selector,1,bytes2(0x1234)));
        //selector
        bytes4 _selector=bytes4(keccak256("testAbiMethod(uint256,bytes2)"));
        //argument
        console.logBytes(abi.encodePacked(_selector,abi.encode(1,bytes2(0x1234))));
    }

    //根据调用进行编码
    function testAbiCall() pure external{
         console.logBytes(abi.encodeCall(TestAbiEncode.testAbiMethod,(1,0x1234)));
    }

}
```

二、Create和Create2的区别

1、什么是以太坊的地址

features	EOA	CONTRACT
properties	1.balance2.nonce(tx counts)	1.balance 2.nonce(counts for creating contract) 3.code 4.storage
Initialization	Free	cost fee
SendTransaction	✅can send eth initiatively	❌due to contract has no pk for signing
ExecuteExternalFunction	✅ Transaction	✅ Message call
CreateContract	✅	✅
CanExecuteLogic	❌ no code here	✅

2、EOA地址如何生成



Ethereum Address

(1) Compute PBK From PRK

```
const hre = require("hardhat");
const ethers = require("ethers");

function convertPBK(privateKey) {
  const keyPair = new ethers.SigningKey(privateKey);
  console.log(keyPair.publicKey);
  return keyPair.publicKey;
}
```

(2) Compute Address From PBK

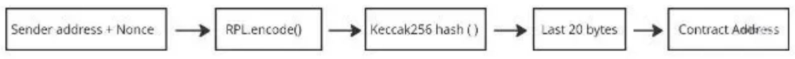
```
function computeAddress(privateKey) {
  const pbk = convertPBK(privateKey);
  const address =
    "0x" + ethers.keccak256("0x" + pbk.substring(4)).substring(26);
  console.log(address);
  console.log(ethers.computeAddress(pbk));
}
```

3、合约地址如何生成

两种方法：Create和Create2

Create1

Create方法创建的合约是跟创建人地址+nonce确定的，所以多次构建，只要sender address和nonce一致，创建出来的地址也就一致。



💡演示 从代码上演示如何生成合约地址

演示1:通过EOA创建合约

```
  const _contract = await ethers.getContractFactory("Factory");
  const _deploy = await _contract.deploy();
  await _deploy.waitForDeployment();
  console.log(await _deploy.getAddress());
```

演示2:通过合约创建合约

- 使用new关键字
- 通过yul语句

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;
import "hardhat/console.sol";
contract Factory{
    event ContractCreated(address);
    function deployWithNew() external {
        console.logAddress(address(new Demo(300)));
    }

    function deployWithYul() external {
         bytes memory _bytes=abi.encodePacked(type(Demo).creationCode,abi.encode(300));
        address deployAddress;
        assembly{
            deployAddress:=create(callvalue(),add(_bytes,0x20),mload(_bytes))
        }
        console.log(deployAddress);
        emit ContractCreated(deployAddress);
    }
}

contract Demo{
    uint256 public i;
    constructor(uint256 _i){
        i=_i;
    }
}
```

- 测试过程

```
const { ethers } = require("hardhat");
const hre = require("hardhat");
async function show() {
  //0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 →0x00 : nonce0
  const _contract = await ethers.getContractFactory("Factory");
  const _deploy = await _contract.deploy();
  await _deploy.waitForDeployment();
  console.log(await _deploy.getAddress());
  //0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266→0x5FbDB2315678afecb367f032d93F642f64180
aa3 :nonce 1
  // console.log(await _deploy.deployV1());
  const _reponse = await _deploy.deployWithYul();
  const _receipt = await _reponse.wait();
  //console.log(ethers.keccak256(ethers.toUtf8Bytes("ContractCreated(address)"))); index0 是什么
  console.log(_receipt.logs[0].topics[1]);
  const _demo = await ethers.getContractAt(
    "Demo",
    "0x" + _receipt.logs[0].topics[1].substring(26)
  );
  console.log(await _demo.i());
}
```

```
show();
```

🧑‍🏫讲解

通过两种方式部署的合约地址相同，

说明只要nonce和address确认 通过create1创建的地址就是相同的。

EOA账户创建的nonce从0开始

但是我们通常很难控制nonce的生成，有没有一种方式可以让我们预测地址的生成呢？🤔

Create2

因为create2需要传入salt 所以无法通过eoa账户创建

```
keccak256( 0xff ++ address ++ salt ++ keccak256(init_code))[12:]
```

演示1:通过合约创建合约

- 使用new关键字
- 通过yul语句

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;
import "hardhat/console.sol";
contract Factory2{

    event ContractCreated(address indexed);

    function deployV2(uint256 salt) external {
        bytes memory _bytes = abi.encodePacked(type(Demo).creationCode, abi.encode(uint256(300)));

        address deployAddress;
        assembly {
            deployAddress := create2(callvalue(), add(_bytes, 0x20), mload(_bytes), salt)
            if iszero(extcodesize(deployAddress)){
                revert(0,0)
            }
        }

        console.logAddress(deployAddress);
        // 使用 emit 来触发事件
        emit ContractCreated(deployAddress);
    }
    function deployV1(uint256 salt) external {
      address _deployAddress=address(new Demo{salt:bytes32(salt)}(300));
      console.logAddress(_deployAddress);
      emit ContractCreated(_deployAddress);
    }

   function predict(uint256 salt) external view returns(address){
       bytes32 hash = keccak256(
            abi.encodePacked(
            bytes1(0xff), address(this), salt, keccak256(abi.encodePacked(type(Demo).creationCode,abi.enco
de(300)))
          )
      );
      return  address(uint160(uint256(hash)));

    }




}


contract Demo{
    uint256 public i;
    constructor(uint256 _i){
        i=_i;
    }
}
```

🧪实验1 如果先执行deployV2再执行deployV1，deployV1会失败，但是先执行deployV1再执行deploy V1，deployV2不会失败但是返回空地址，为什么

  回答：因为对于已经存在的地址，如果使用create2创建，会失败，但是V2使用yul方式执行的并不会返回错误。

🧪实验2 在上面例子中，如何让v2也报错,增加返回值判断

```
        assembly {
            deployAddress := create2(callvalue(), add(_bytes, 0x20), mload(_bytes), salt)
            if iszero(extcodesize(deployAddress)){
                revert(0,0)
            }
        }
```

🧪实验3 思考为什么每次重复部署，得到的demo的地址都不一样

  回答：因为factory合约变了

- 测试过程

```
const { ethers } = require("hardhat");
const hre = require("hardhat");
async function show() {
  //0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 ->0x00：nonce0
  const _contract = await ethers.getContractFactory("Factory2");
  const _deploy = await _contract.deploy();
  await _deploy.waitForDeployment();
  console.log(await _deploy.getAddress());
  //0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266->0x68bbb25d542e358cf022bf49252c19dea462cfe5
:nonce 1
  const _response = await _deploy.deployV1(2024);
  const _receipt = await _response.wait();
  //console.log(await _deploy.deployV2(2024));
  console.log(await _deploy.predict(2024));
  const _demo = await ethers.getContractAt(
    "Demo",
    "0x" + _receipt.logs[0].topics[1].substring(26)
  );
  console.log(await _demo.a());
}

show();
```

🧑‍🏫讲解

通过两种方式部署的合约地址相同，

通过create2创建合约，只需要确定sender+salt就能唯一确定一个合约的地址

有什么好处呢？

✨合约可预测

```
   function predict(uint256 salt) external view returns(address){
      bytes32 hash = keccak256(
          abi.encodePacked(
             bytes1(0xff), address(this), salt, keccak256(abi.encodePacked(abi.encodePacked(type(Demo).crea
tionCode,abi.encode(1))))
          )
      );
      return  address(uint160(uint256(hash)));
   }
```

💡有什么作用呢

(1) 合约工厂

对于平台合约，在常见合约时，当需要根据某些参数再次找到构造的合约地址的时候，可以用create2

https://github.com/Uniswap/v2-core/blob/master/contracts/UniswapV2Factory.sol

https://github.com/Uniswap/v2-periphery/blob/master/contracts/libraries/UniswapV2Library.sol

（2）链下预测地址

```
const { bytecode } = require("../artifacts/contracts/Demo.sol/Demo.json");
async function predict() {
  const from = "0x5FbDB2315678afecb367f032d93F642f64180aa3";
  const salt = 2024;

  const initCodeHash = ethers.keccak256(
    bytecode +
      ethers.AbiCoder.defaultAbiCoder().encode(["uint256"], [1]).substring(2)
  );
  //  console.log(initCodeHash);
  console.log(
    "predictoffline" +
      ethers.getCreate2Address(
      from,
      ethers.AbiCoder.defaultAbiCoder().encode(["uint256"], [2024]),
      initCodeHash
    )
  );
}

predict();
```

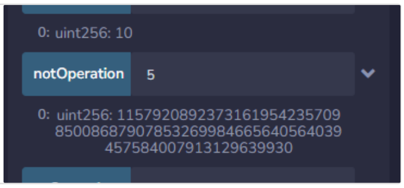
附录

bytesN with bytes

Bytes in Solidity - GeeksforGeeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.

https://www.geeksforgeeks.org/bytes-in-solidity/



```
0: uint256: 10
notOperation  5
0: uint256: 115792089237316195423570985008687907853269984665640564039457584007913129639930
```

Learn Solidity lesson 37. Creating and destroying contracts.

There are three ways to create a contract account on Ethereum. Externally owned accounts can create contract accounts by sending a...

https://medium.com/coinmonks/learn-solidity-lesson-37-creating-and-destroying-contracts-6921ae32413a

Predicting Contract Addresses on Multiple Networks with Solidity's CREATE2 | QuickNode Guides

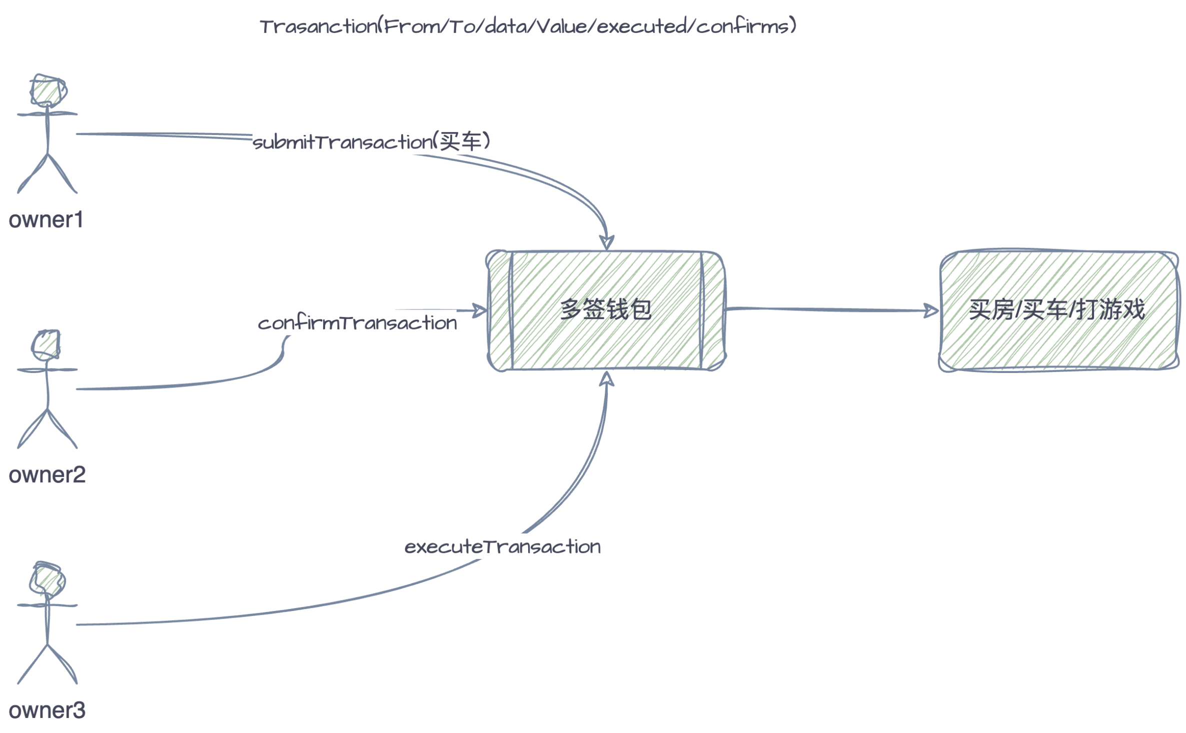
In this guide, you will learn how to predict smart contract addresses before deployment using Hardhat, OpenZeppelin, and Solidity.

https://www.quicknode.com/guides/ethereum-development/smart-contracts/how-to-use-create2-to-predetermine-contract-addresses

> Let's keep digging down the rabbit hole.
>
> Additionally, some use cases CREATE2 enables are:
>
> - Allow off-chain transactions between parties. They only deploy contracts on-chain if disputes arise, saving costs and improving efficiency.
> - Smart contracts are set up but only deployed (and paid for) when needed, enabling efficient resource use.
> - Services like pre-configuring smart contracts for users without deploying them. Smart contracts are only deployed when the user becomes active, reducing initial costs.
> - Enable predictable complex operations in DeFi by ensuring certain contracts will be deployed in future transactions.

3、多签钱包

(1）业务场景



（2）走读多签钱包的代码及演示

- 走读代码

Solidity by Example
https://solidity-by-example.org/app/multi-sig-wallet/

- 部署合约

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;
contract TestContract {
    uint256 public i;

    function callMe(uint256 j) payable public {
        i += j;
    }

    function getData() public pure returns (bytes memory) {
        return abi.encodeWithSignature("callMe(uint256)", 123);
    }

}
```

```
 （1）部署多签钱包合约，
设置owner为["0x5B38Da6a701c568545dCfcB03FcB875f56beddC4","0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2","0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db"]
设置require为2
（2）向多签钱包存400wei（注意，构造函数没有payable，使用合约下面的Calldata TX执行）
（3）部署TestContract合约
（4）提交交易
to为TestContract合约
value为执行这笔合约需要的金额100wei
data为调用TestContract.getData的值
（5）确认交易(使用C4,B2)
（6）执行交易（使用DB执行）
```

（3）商用多签钱包

提前部署的合约

0x5b80dee27015e1dd137813f19aa77577a58a2f9c

https://sepolia.etherscan.io/tx/0x5a74dd6ad0f7b581d8e062405270070e440ebdb42849680dfbccb4957c79a104

master:0x6307AdC71cB53dbb9E874F553E43A6db2C8c2a05

second:0x06E668133D7F3EE7DBC2B5371f71E443EA173693

https://app.safe.global/