Solidity 面试题(一)

1. 私有、内部、公共和外部函数之间的区别?

答：私有private只能当前合约内部使用，子合约继承也不能调用；内部internal可以当前合约和子合约调用；公共public合约内部外部和子合约都可以调用；外部external只提供给外部调用，合约内部不能调用，合约接口的函数都要声明成external。

2. 智能合约大小大约可以有多大?

答：上海升级后，实现EIP-3860是48KB，原来24KB，扩大代码量解决以前复杂合约拆分后合约之间互相调用的高gas费问题。

3. create 和 create2 之间有什么区别?

答：CREATE和CREATE2是Solidity中的两个重要的操作码，它们都可以用于部署智能合约，但是它们之间有一些区别. CREATE操作码通过对发送者地址和nonce值进行哈希运算来计算新合约的地址，而CREATE2操作码则使用了更复杂的公式来计算新合约的地址，这个公式包括发送者地址、随机数、字节码等参数. CREATE2操作码的主要优点是，它可以在部署合约之前预测合约的地址，这对于需要提前知道合约地址的场景非常有用

4. Solidity 0.8.0 版本对算术运算有什么重大变化?

答：从Solidity 0.8.0版本开始，算术运算会在下溢和上溢时回滚。这意味着，如果一个算术运算的结果超出了其数据类型的范围，那么这个运算将会失败并回滚。在Solidity 0.8.0之前，整数下溢和上溢是允许的，不会导致错误，为了避免这种情况，Solidity 0.8.0默认开启了算术运算检查，如果您需要使用旧的算术运算方式，可以使用unchecked{...}语句块

5. 代理需要哪种特殊的 CALL 才能工作?

答：Solidity中有三种调用函数可以实现合约间的函数调用，它们分别是call、delegatecall和callcode。其中，delegatecall是一种特殊的调用方式，它允许我们在主合约的上下文中加载和调用另一个合约的代码。被调用合约的代码被执行，但被调用合约所做的任何状态改变实际上是在主合约的存储中进行的，而不是在被调用合约的存储中

6. 在 EIP-1559 之前，如何计算以太坊交易的成本?

答：在EIP-1559之前，以太坊交易的成本由矿工通过拍卖机制来决定。矿工会选择最高出价的交易，并将其包含在下一个区块中。交易的成本由两个因素决定：Gas Price和Gas Limit。Gas Price是以太坊网络中的一种计量单位，用于衡量交易的复杂性。Gas Limit是指交易可以使用的最大Gas数量。交易的成本等于Gas Price乘以Gas Limit。因此，交易的成本取决于Gas Price和Gas Limit的值，这些值由交易的发送者设置。

7. ETH转账的底层原理

答：底层原理就是修改链上数据，可以回答详细一点就是交易过程发生了什么。

以下是ETH转账的详细过程：

- 首先，您需要连接到以太坊网络并加载您的私钥。您可以使用go-ethereum客户端来完成此操作。
- 然后，您需要获取发送帐户的随机数（nonce）。每个新事务都需要一个nonce，这是一个仅使用一次的数字。如果是发送交易的新帐户，则该随机数将为“0”。
- 接下来，您需要将ETH转换为wei，因为这是以太坊区块链所使用的货币单位。以太网支持最多18个小数位，因此1个ETH为1加18个零。
- 然后，您需要设置您将要转移的ETH数量，以及燃气限额和燃气价格。燃气限额应设上限为“21000”单位，而燃气价格必须以wei为单位设定。
- 然后，您需要确定您要将ETH发送给哪个地址。这是接收地址。
- 接下来，您需要生成未签名的以太坊事务。这个函数需要接收nonce，地址，值，燃气上限值，燃气价格和可选发的数据。发送ETH的数据字段为“nil”（nil是go语言的null）。
- 然后，您需要使用发件人的私钥对事务进行签名。为此，您可以使用go-ethereum客户端提供的SignTx方法，该方法接受一个未签名的事务和您之前构造的私钥。
- 最后，您可以通过在客户端上调用“SendTransaction”将已签名的事务广播到整个网络。这将向网络中的所有节点发送交易，并等待矿工将其打包到区块中。

8. 在区块链上创建随机数的挑战是什么?

答：在区块链上创建随机数的挑战是确保生成的随机数是真正随机的。由于区块链是一个公开的分布式账本，因此任何人都可以查看和验证交易。这意味着，如果随机数生成算法不够随机，那么攻击者可能会通过分析交易来预测随机数的值，从而破坏系统的安全性。为了解决这个问题，一些技术已经被提出，例如使用区块链上的随机数生成器，或者使用多个随机数生成器来生成随机数，以确保生成的随机数是真正随机的 .

9. solidity实现随机数生成器

答：在这个示例中，我们使用了Solidity中的keccak256哈希函数来生成随机数。我们将当前时间戳、发送方地址和nonce值作为输入，然后对它们进行哈希运算，得到一个随机数。nonce值是一个递增的计数器，用于确保每次调用random函数时都会生成一个新的随机数。请注意，这种方法并不是完全随机的，因为它仍然依赖于输入参数的值。但是，它可以提供足够的随机性，以满足大多数应用程序的需求。

```
pragma solidity ^0.8.0;

contract RandomNumberGenerator {uint256 private nonce = 0;
function random() public returns (uint256) {uint256 randomNumber =
uint256(keccak256(abi.encodePacked(block.timestamp, msg.sender, nonce)));
        nonce++;return randomNumber;
    }
}
```

10. 什么是EIP-1559

答：Ethereum Improvement Proposal (EIP) 1559是以太坊的一个升级，旨在改变以太坊计算和处理网络交易费用（称为“gas费用”）的方式。该升级通过使用基于区块的基础费用和发送方指定的最大费用，而不是对gas价格进行竞价，来更加平衡地激励矿工在高或低网络拥塞期间进行挖矿，从而使以太坊交易更加高效。EIP-1559是一个提案，它改变了gas费用的结构和矿工的奖励方式。该提案于2021年8月5日作为以太坊伦敦硬分叉的一部分实施。

gas费 = 基础费（Basefee）+ 矿工费（Tip），基础费会根据区块的Gas利用率动态调整，如每个区块的Gas费利用率低于50%，就降低手续费，高于50%，就提高手续费。

11. 荷兰式拍卖和英式拍卖之间有什么区别?

答：荷兰式拍卖和英式拍卖是两种不同的拍卖方式。在荷兰式拍卖中，拍卖师会先宣布一个高价，然后逐渐降低价格，直到有人愿意出价购买物品为止。在这种拍卖中，第一个出价的人通常会赢得拍卖，因为他们愿意支付的价格最高。荷兰式拍卖通常用于出售艺术品、古董和其他昂贵的物品.

英式拍卖是一种更传统的拍卖方式。在这种拍卖中，拍卖师会宣布一个起始价，然后买家可以逐渐提高价格，直到没有人再愿意出价为止。在英式拍卖中，最后一个出价的人通常会赢得拍卖，因为他们愿意支付的价格最高。英式拍卖通常用于出售房地产、汽车和其他大型物品.

总的来说，荷兰式拍卖和英式拍卖都是有效的拍卖方式，但它们的实现方式略有不同。荷兰式拍卖通常更适合出售昂贵的物品，而英式拍卖则更适合出售大型物品。

12. ERC20 中的 transfer 和 transferFrom 之间有什么区别?

答：transfer是从当前合约转账给目标账户，transferFrom是可设置发送账户和目标账户

transfer(address recipient, uint256 amount)

transferFrom(address sender, address recipient, uint256 amount)

13. 对于地址 allowlist，使用映射还是数组更好？为什么?

答：地址 allowlist 是一个存储地址的列表，用于限制只有在列表中的地址才能执行某些操作。在 Solidity 中，可以使用映射或数组来实现地址 allowlist。使用映射的优点是可以快速查找地址是否在 allowlist 中，而使用数组的优点是可以更容易地遍历整个 allowlist。

如果您需要快速查找地址是否在 allowlist 中，那么使用映射可能更好。映射使用哈希表实现，可以在 O(1) 的时间内查找地址是否在 allowlist 中。这对于大型 allowlist 尤其有用，因为它可以避免遍历整个列表来查找地址。

如果您需要遍历整个 allowlist，那么使用数组可能更好。数组可以更容易地遍历整个列表，因为它们是连续的内存块。这对于小型 allowlist 尤其有用，因为它可以避免使用哈希表的额外开销。

14. 为什么不应该使用 tx.origin 进行身份验证?

答：在 Solidity 中，tx.origin 是一个全局变量，它返回发送交易的账户地址。在合约代码中，最常用的是使用 msg.sender 来检查授权，但有时由于有些程序员不熟悉 tx.origin 和 msg.sender 的区别，如果使用了 tx.origin 可能导致合约的安全问题。黑客最典型的攻击场景是利用 tx.origin 的代码问题常

与钓鱼攻击相结合的组合拳的方式进行攻击。因为 tx.origin 返回交易的原始发送者，因为攻击的调用链可能是原始发送者 -> 攻击合约 -> 受攻击合约。在受攻击合约中，tx.origin 是原始发送者。因此，通过调用 tx.origin 来检查授权可能会导致合约受到攻击。为了避免这种情况，建议使用 msg.sender 来检查授权

15. 以太坊主要使用什么哈希函数?

答：以太坊主要使用两种哈希函数，分别是 Keccak-256 和 SHA-3。Keccak-256 是以太坊最初采用的哈希函数，它是 SHA-3 标准的一个变种。在以太坊中，Keccak-256 用于计算地址、交易哈希和区块哈希等重要数据的哈希值。SHA-3 是一种新的哈希函数标准，它比 Keccak-256 更加安全和高效。在以太坊中，SHA-3 用于计算合约代码的哈希值。

16. 1 个Ether 相当于 多少个 gwei ?

答：1 ether = 109109 gwei

17. 1 个Ether 相当于 多少个 wei ?

答：1 ether = 10181018 wei

18. assert 和 require 之间有什么区别?

答：在 Solidity 中，assert 和 require 都是用于检查条件的函数。当条件不满足时，它们会抛出错误或异常。它们之间的区别在于，require 用于在执行前验证输入和条件，而 assert 用于检查不应该为假的代码，失败的断言可能意味着代码层面存在错误。当我们想在执行之前验证输入和条件时，我们使用 require。当我们想要检查可以而且永远不应该为 false 的代码时，我们使用 assert。如果失败，则意味着存在错误，在这种情况下，我们应该使用 assert。

当在选择 assert 和 require 之间做出决定时，需要考虑两个方面：

Gas 优化的效率：如果 assert 返回一个 false 语句，它会编译为 0xfe，这是一个无效的操作码，它会用完所有剩余的 gas，从而完全恢复更改。如果 require 返回错误语句，它会编译为 0xfd，这是 REVERT 的操作码，这意味着它将返回剩余的气体。

Bytecode 分析：如果你使用 assert，你的代码将会更小，因为它不会返回任何东西。如果你使用 require，你的代码将会更大，因为它需要返回错误信息。

19. 什么是闪电贷?

答：闪电贷是一种无抵押借贷的defi产品，主要给开发者提供的，它允许在一笔交易内进行借款还款，只要支付一点闪电贷设定的手续费。很多攻击者利用闪电贷进行攻击和套利。

20. 什么是检查效果（check-effects）模式?

答：检查效果（check-effects）模式是一种编程模式，用于确保函数在执行时不会对状态造成意外的影响。在这种模式下，函数应该只读取输入参数，并且不应该修改任何状态。如果函数需要修改状态，它应该返回一个包含所有修改的对象，而不是直接修改状态。这种模式可以帮助开发人员编写更安全、更可靠的代码，因为它可以减少由于状态修改而导致的错误。

21. 运行独立验证节点所需的最小以太数量是多少?

答：在以太坊网络中，要运行一个独立的验证节点，需要至少 32 个以太币作为抵押品。这是因为在以太坊的 Proof of Stake（PoS）共识机制中，验证节点需要抵押一定数量的以太币来证明他们对网络的贡献，同时也会获得一定的奖励

22. fallback 和 receive 之间有什么区别?

答：fallback和receive都是特殊的回调函数，用于处理合约中不存在的函数调用和接收以太币。它们之间的区别在于，fallback函数会在调用合约不存在的函数时被触发，而receive函数只用于处理接收以太币。

23. 什么是重入?

答：重入攻击是一种安全漏洞，攻击者利用合约的 fallback 函数和多余的 gas 将本不属于自己的以太币转走的攻击手段。攻击者通过在攻击合约中调用受攻击合约的函数，然后在受攻击合约中再次调用攻击合约的函数，从而实现重复调用，造成重入攻击。重入攻击的本质是递归调用，攻击者利用这种递归调用来绕过原代码中的限制条件，从而造成攻击。为了避免重入攻击，可以使用 Solidity 内置的 transfer() 函数，或者使用检查-生效-交互模式 (checks-effects-interactions) 来确保状态变量的修改要早于转账操作。

被攻击合约取款方法：

```
function withdraw() public {uint bal = balances[msg.sender];require(bal > 0);// 用call给攻击合约发送eth,// 促发了攻击合约中定义了fallback函数,因为攻击合约中没有""方法，fallback中并实现调用withdraw方法// call一直执行，一直触发fallback
        (bool sent, ) = msg.sender.call{value: bal}(""); require(sent, "Failed to send Ether");// 这里的修改余额应该放到发送eth步骤之前，这样反复攻击时前面的balance校验就会不通过
        balances[msg.sender] = 0;
    }
```

24. 上海升级后，每个区块的 gas 限制是多少?

答：在以太坊上，每个区块的 gas limit 是由矿工决定的。在上海升级后，每个区块的 gas limit 已经从 15,000,000 增加到了 30,000,000

25. 什么阻止无限循环永远运行?

答：这个问题原文是：What prevents infinite loops from running forever?

如果理解成：是什么不让无限循环永远执行的话？答案就很明显，方法会受到gas费限制，栈深度的限制，所以循环不会一直永远运行，会报out of gas错误。

如果题目理解成：做什么可以阻止无限循环。那就是代码实现，在循坏中使用计数器：require判断是否达到满足中断循坏次数。（感谢评论区指出错误）

26. tx.origin 和 msg.sender 之间有什么区别?

答：tx.orgin是这笔交易的发起者，msg.sender是前一个调用发起者，如交易流程：张三发起交易-合约A-合约B，在合约B中msg.sender = 合约A，tx.orgin = 张三

27. 如何向没有payable 函数、receive 或回退的合约发送以太?

答：这里是评论区给出的答案

①通过其他合约自毁将自毁的eth发送目标合约

②将挖矿获取的区块奖励费用的接收者设置为目标合约地址

③将Beacon信标链上质押后的提现地址设置为目标合约地址

这里补充一个：在solidity0.4之前的合约，也可以直接接收ETH。

28. view 和 pure 之间有什么区别?

答：view和pure都是函数的修饰符，view可以访问合约中的状态变量，不能修改；pure不能访问也不能修改状态变量。

29. ERC721 中的 transferFrom 和 safeTransferFrom 之间有什么区别?

答：transferFrom和safeTransferFrom都是ERC721合约中的函数，用于将代币从一个地址转移到另一个地址。它们之间的区别在于，safeTransferFrom函数会检查接收合约是否实现了onERC721Received函数，并且该函数返回的值是否为特定的魔数。如果接收合约没有实现onERC721Received函数或者返回的值不是特定的魔数，safeTransferFrom函数将抛出异常并回滚所有更改，以确保代币不会永久丢失。而transferFrom函数则不会进行这种检查，因此可能会导致代币永久丢失。

30. 如何将 ERC1155 代币转换为非同质化代币?

答：ERC1155代币是一种可替代和不可替代的代币，可以在同一合约中管理多个资产。ERC721代币是一种非可替代代币，每个代币都是唯一的。因此，将ERC1155代币转换为ERC721代币需要将每个ERC1155代币转换为一个唯一的ERC721代币。

要将ERC1155代币转换为ERC721代币，您需要执行以下步骤：

- 创建一个新的ERC721合约。
- 为每个ERC1155代币创建一个新的ERC721代币。
- 将ERC1155代币的所有权转移到新的ERC721代币。
- 销毁原始ERC1155代币

31. 访问控制是什么，为什么重要?

答：访问控制是一种重要的机制，用于限制对智能合约的访问。通过使用访问控制，您可以确保只有经过授权的用户才能执行特定操作或访问敏感信息。这可以帮助保护您的智能合约免受未经授权的访问和攻击。

Solidity提供了几种访问控制修饰符，例如public、private、internal和external。这些修饰符用于控制函数和状态变量的可见性和访问权限。

32. 修饰符（modifier）的作用是什么?

答：modifier是一种函数修饰符，用于声明一个函数修改器。函数修改器的作用与Spring中的切面功能很相似，当它作用于一个函数上，可以在函数执行前或后（依赖于具体实现）预先执行modifier中的逻辑，以增强其功能。使用modifier可以将一些通用的操作提取出来，提高编码效率，降低代码耦合性。

modifier可以用于控制函数的访问权限和行为。例如，如果您希望只有合约的所有者才能访问某个函数，您可以将该函数标记为private，并使用modifier来检查调用者是否为合约的所有者。如果调用者不是合约的所有者，则函数将停止执行并回滚所有更改。这可以帮助保护合约免受未经授权的访问和攻击。

```
contract MyContract {address public owner;
modifier onlyOwner() {require(msg.sender == owner, "Only the contract owner
can call this function.");
        _; // 这里才执行myFunction代码
    }
function myFunction() public onlyOwner {// do something
    }
}
```

33. uint256 可以存储的最大值是多少?

答：最大2^256-1 。为什么减1？0占用了一个空间，256位的表示的数是从000...00到111...11 。例如：如果有uint2 , 那么2^2可存储的数可以是 00 = 0 ， 01 = 1,10 = 2,11 = 3，这里最大是3，也就是2^-1,而不是4。

34. 什么是浮动利率和固定利率?

答：固定利率是指在贷款期间内，贷款利率保持不变。而浮动利率是指贷款利率会随着市场利率的变化而变化。浮动利率通常基于某个基准利率，例如央行基准利率或LIBOR（伦敦银行间同业拆借利率）。如果基准利率上升，贷款利率也会上升，反之亦然。

send函数的gas限制也为2300 gas，但它返回一个布尔值，指示转账是否成功。如果转账失败，它将返回false。但是，如果接收方合约没有实现fallback函数，或者fallback函数消耗的gas超过了2300，那么转账将失败并回滚所有更改。