solidity面试题(二)

1. transfer 和 send 之间有什么区别？为什么不应该使用它们?

答：transfer和send函数都是用于将以太币从一个地址转移到另一个地址的函数。它们之间的区别在于它们的gas限制不同，这可能会导致一些安全问题。因此，您应该使用call函数来转移以太币,但是会有重入攻击风险。

```
详细：（transfer函数的gas限制为2300 gas，这意味着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。这可以防止重入攻击，但也可能导致一些问题，例如无法向某些合约发送以太币。
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。
由于这些限制，transfer和send函数已经被认为是不安全的，因此不应该使用它们。相反，您应该使用call函数来转移以太币。call函数没有gas限制，可以向任何地址发送以太币，并且可以指定要发送的gas数量。但是，您应该小心使用call函数，因为它可能会导致一些安全问题，例如重入攻击。）
```

2. 如何在 Solidity 中编写节省gas的高效循环?

答：

1.避免无限循环。

2.避免重复计算：多次计算相同的值，应该将该值存储在变量中，并在需要时重复使用该变量。

3.避免使用昂贵的操作：例如除法和取模。尝试使用更便宜的操作来代替它们。

4.避免使用大型数组：如果可能的话，您应该使用映射或其他数据结构来代替数组。

5.避免使用复杂的嵌套循环

6.使用constant和immutable关键字：如果您需要在循环中使用常量或不可变变量，请使用constant和immutable关键字来声明它们。

3. 代理合约中的存储冲突是什么?

答：存储冲突是指多个合约尝试访问同一存储位置时发生的问题。当多个合约同时尝试更新同一存储位置时，可能会发生存储冲突，导致数据不一致或合约无法正常工作。为了避免这种情况，使用代理合约来管理存储位置，并确保只有一个合约可以访问每个存储位置。代理合约可以充当存储位置的所有者，并使用访问控制机制来限制对存储位置的访问。这可以帮助确保数据的一致性，并提高智能合约的安全性和可靠性。

4. abi.encode 和 abi.encodePacked 之间有什么区别?

答:

abi.encode会在每个参数之间添加填充，以确保每个参数占用32个字节。这可以确保编码后的字节数组具有固定的长度，并且可以正确地解码回原始参数。但是，由于填充的存在，编码后的字节数组可能会比实际需要的更大。

abi.encodePacked不会添加填充，而是将所有参数拼接在一起。这可以确保编码后的字节数组尽可能小，但是可能会导致解码时出现问题，因为无法确定每个参数的确切位置。

5. uint8、uint32、uint64、uint128、uint256 都是有效的 uint 大小。还有其他的吗?

答：这些类型分别表示8位、32位、64位、128位和256位的无符号整数。除了这些类型之外，Solidity还支持其他整数类型，例如int8、int16、int32、int64、int128和int256。这些类型分别表示8位、16位、32位、64位、128位和256位的有符号整数。

6. 在权益证明之前后，block.timestamp 发生了什么变化?

答：在PoW协议中，block.timestamp表示矿工开始挖掘新块的时间戳。在PoS协议中，block.timestamp表示验证器开始验证新块的时间戳。因此，block.timestamp的含义在两种协议中都与块的创建时间有关，但在PoS协议中，它与验证器的行为有关，而不是矿工的行为有关。

7. 什么是抢跑（frontrunning）?

答：抢跑（frontrunning）是一种攻击行为，指在一笔正常交易等待打包的过程中，抢跑机器人通过设置更高 Gas 费用抢先完成攻击交易。在所有 Front-Running 中，最典型最具危害性的就是针对 AMM 交易的 Sandwich Attacks （三明治攻击）

注意：夹子交易有时也成三明治攻击，但是它是有矿工或验证节点端完成的，能把被攻击的那笔交易夹在中间打包区块。

8. 什么是提交-揭示方案，何时使用它?

答：提交-揭示方案（Commit-Reveal Scheme）是一种用于在区块链上进行投票或竞标的协议。该协议的目的是防止参与者在提交投票或竞标之前查看其他参与者的提交，从而保护投票或竞标的公正性。

具体：

- 提交-揭示方案的基本思想是将投票或竞标分为两个阶段：提交阶段和揭示阶段。在提交阶段，参与者将加密的投票或竞标提交到智能合约中。在揭示阶段，参与者揭示他们的加密投票或竞标，并将其与提交阶段中的哈希值进行比较。如果哈希值匹配，则投票或竞标被接受。否则，投票或竞标将被拒绝。
- 提交-揭示方案可以防止参与者在提交投票或竞标之前查看其他参与者的提交，从而保护投票或竞标的公正性。它还可以防止恶意参与者在提交阶段提交虚假的投票或竞标，因为他们无法预测其他参与者的投票或竞标。
- 提交-揭示方案通常用于加密货币中的投票或竞标，例如DAO（去中心化自治组织）的投票。它也可以用于其他需要保护公正性的场景，例如拍卖和投标。

9. 在什么情况下，abi.encodePacked 可能会产生漏洞?

答：abi.encodePacked可能会产生漏洞，因为它不会在参数之间添加填充，而是将所有参数拼接在一起。这可能会导致哈希碰撞，从而使攻击者能够伪造交易或执行其他恶意操作。例如，如果攻击者知道您使用abi.encodePacked来编码交易数据，他们可以构造一个具有相同哈希值的交易，从而欺骗您的智能合约。

为了避免这种情况，用abi.encode来编码交易数据，因为它会在每个参数之间添加填充，以确保每个参数占用32个字节。这可以防止哈希碰撞，并提高智能合约的安全性。

10. 以太坊如何确定 EIP-1559 中的 BASEFEE?

答：在以太坊的EIP-1559协议中，BASEFEE是由以太坊网络根据交易需求和区块大小动态调整的。BASEFEE的计算方式是通过一个名为“基础费用追踪器”的算法来实现的。该算法会根据当前区块的交易需求和区块大小来动态调整BASEFEE，以确保交易能够在合理的时间内得到确认。

具体来说，基础费用追踪器会根据当前区块的交易需求和区块大小计算出一个目标基础费用。如果当前的BASEFEE高于目标基础费用，那么BASEFEE将会下降。如果当前的BASEFEE低于目标基础费用，那么BASEFEE将会上升。这种动态调整机制可以确保BASEFEE始终能够反映当前的交易需求和区块大小，从而提高以太坊的交易效率和可靠性。

11. 冷读（cold read）和热读（warm read）之间有什么区别?

答：冷读（cold read）和热读（warm read）是两种不同的访问存储变量的方式。冷读是指第一次读取存储变量时，需要从存储中读取变量的值，这需要较高的gas费用。而热读是指在第一次读取存储变量之后，再次读取存储变量时，可以从缓存中读取变量的值，这需要较低的gas费用。热读和冷读是由Ethereum虚拟机（EVM）自动处理的。

12. AMM 如何定价资产?

答：通过恒定乘积算法，a * b = k ,兑换 m个a 需要的b数量算法 = k/(a+m) - b ,这里没计算手续费。

13. 代理中的函数选择器冲突是什么，它是如何发生的?

答：函数选择器是一个用于标识函数的哈希值。在Solidity中，当您调用一个合约中的函数时，您需要提供该函数的选择器。函数选择器由函数名称和参数类型组成，并使用Keccak-256哈希算法进行哈希处理。当您在代理合约中调用另一个合约的函数时，您需要将该函数的选择器传递给代理合约。如果您在代理合约中定义了具有相同名称和参数类型的函数，则会发生函数选择器冲突。这意味着当您调用代理合约中的函数时，Solidity无法确定您要调用哪个函数，因为它们具有相同的函数选择器。为了避免函数选择器冲突，您可以使用不同的函数名称或参数类型来定义代理合约中的函数。或者使用管理员校验来调用，只有管理员才调用代理合约定义的函数。

14. payable 函数对 gas 的影响是什么?

答：payable函数是一种特殊类型的Solidity函数，它允许合约接受以太币作为支付。当您在Solidity中定义一个payable函数时，您可以在函数调用中包含以太币，并将其存储在合约的余额中。由于以太币是一种有价值的加密货币，因此在调用payable函数时需要支付一定的gas费用。这是因为在以太坊网络中，每个操作都需要消耗一定数量的gas，以保证网络的安全性和可靠性。

当您在Solidity中使用payable函数时，需要注意以下几点：

- 确保您的合约具有足够的余额来处理以太币支付。
- 确保您的合约具有足够的gas来处理以太币支付。
- 确保您的合约具有足够的安全性来处理以太币支付。

15. 什么是签名重放攻击?

答：签名重放攻击是一种网络攻击，攻击者通过重复使用已经被验证的数字签名来欺骗系统。数字签名是一种用于验证数据完整性和身份验证的技术，它使用公钥密码学来生成和验证签名。在签名重放攻击中，攻击者截获了一个数字签名并将其重复使用，以便在未经授权的情况下执行某些操作。例如，攻击者可以使用重放攻击来多次执行某个交易，从而导致资金损失。为了防止签名重放攻击，您可以使用以下方法：

使用时间戳或随机数来确保每个数字签名只能使用一次。

使用序列号来确保数字签名按顺序使用。

使用加密协议来保护数字签名，例如TLS或SSL。

16. 什么是 gas griefing ?

答：Gas griefing是一种智能合约攻击，攻击者通过发送恰好足够的gas来执行主要智能合约，但未为其子调用或外部通信提供足够的gas，从而导致未受控制的行为并在某些情况下对合约的业务逻辑造成严重破坏1. 这种攻击可能会导致合约的不可预测行为，例如无限循环或拒绝服务攻击。为了防止gas griefing攻击，您可以使用以下方法：

- 在智能合约中检查子调用所需的gas量，并确保为其提供足够的gas。
- 使用安全的编程实践来编写智能合约，例如避免使用循环和递归等高消耗操作。
- 使用Solidity的require和assert语句来确保智能合约的正确性和安全性。

17. 自由内存指针是什么，它存储在哪里?

答：自由内存指针是指指向已经释放的内存地址的指针。当您释放内存时，内存中的数据并不会被删除，而是被标记为可用。如果您在稍后的时间内使用已经释放的内存地址，您可能会访问到已经被其他程序或数据覆盖的数据。这可能会导致程序崩溃或产生不可预测的行为。自由内存指针通常是由于编程错误或不正确的内存管理而引起的。为了避免自由内存指针，您可以使用以下方法：

- 在释放内存后，将指针设置为NULL或0。
- 使用动态内存分配函数（例如malloc和calloc）来分配内存，并使用free函数来释放内存。
- 使用智能指针或垃圾回收器等工具来管理内存。

18. 接口中有效的函数修饰符有哪些?

答：在Solidity中，接口是一种抽象合约，它定义了合约应该实现的函数。接口中的函数没有实现，只有函数签名。函数签名包括函数名称、参数类型和返回类型。接口中的函数可以使用以下修饰符：

external：指定函数只能从合约外部调用。

view：指定函数不会修改合约状态。

pure：指定函数既不会修改合约状态，也不会读取合约状态。

payable：指定函数可以接受以太币作为支付。

19. 函数参数中的 memory 和 calldata 有什么区别?

答：memory：用于声明函数参数将被存储在内存中。内存中的数据只在函数执行期间存在，执行完毕后就被销毁。在函数内部，您可以使用memory关键字来创建临时变量，但是不能在函数之外使用它们。在函数调用期间，函数参数的值将从调用方复制到内存中，并在函数执行完毕后被销毁。

calldata：用于声明函数参数将被存储在调用数据区域中。调用数据区域是一个不可修改的区域，用于保存函数参数。在函数内部，您可以使用calldata关键字来访问函数参数，但是不能在函数之外使用它们。在函数调用期间，函数参数的值将从调用方复制到调用数据区域中，并在函数执行完毕后被销毁。

20. 描述三种存储 gas 成本类型。

答：

内存变量：内存变量是指在函数执行期间分配的临时变量。它们的gas成本取决于它们的大小，通常比存储变量更便宜。在函数执行结束后，内存变量将被销毁。

存储变量：存储变量是指在合约存储器中永久存储的变量。它们的gas成本取决于它们的大小，通常比内存变量更昂贵。存储变量的gas成本还取决于它们的位置，例如，如果它们位于映射中，则访问映射中的元素的成本更高。

状态变量：状态变量是指在合约存储器中永久存储的变量，但它们是在合约创建时定义的。与存储变量相比，它们的gas成本更便宜，因为它们只需要在合约创建时初始化一次。

21. 为什么可升级合约不应该使用构造函数?

答：使用构造函数来初始化合约状态变量是一种常见的做法，但是这会导致合约的状态变量无法升级。因为在升级合约时，新的合约代码将会被部署到一个新的地址，而旧的合约状态变量将无法被传递到新的合约中。因此，为了使合约状态变量能够升级，应该使用初始化函数来初始化合约状态变量，而不是构造函数。初始化函数可以在合约部署后随时调用，因此可以在升级合约时重新初始化合约状态变量。

22. UUPS 和 Transparent Upgradeable Proxy 模式之间有什么区别?

答：在Transparent Upgradeable Proxy模式中，代理合约只负责将所有调用转发到实现合约，并将实现合约的地址存储在代理合约的状态变量中。在升级合约时，新的实现合约将被部署到新的地址，然后将新的实现合约的地址存储在代理合约的状态变量中。这种方法的缺点是，每次升级合约时都需要部署一个新的代理合约。

相比之下，UUPS代理模式使用了更加智能的方法。在UUPS代理模式中，代理合约只负责将所有调用转发到实现合约，并将实现合约的地址存储在代理合约的状态变量中。在升级合约时，只需要将新的实现合约的代码上传到现有的代理合约地址，而无需部署新的代理合约。这种方法的优点是，可以在不更改代理合约地址的情况下升级合约，从而避免了每次升级合约时都需要部署一个新的代理合约的问题。

23. 如果合约通过 delegatecall 调用一个空地址或之前已自毁的实现，会发生什么？如果是常规调用而不是 delegatecall 呢？

答：如果合约通过delegatecall调用一个空地址或之前已自毁的实现，会导致delegatecall返回false，并且不会发生任何状态更改。如果是常规调用而不是delegatecall，则会导致交易失败并回滚，因为您不能调用一个不存在的合约或已自毁的合约。

24. ERC777 代币存在什么危险？

答：ERC777 代币是一种功能型代币，它在 ERC20 标准的基础上进行了改进，解决了一些 ERC20 标准存在的问题。ERC777 代币的主要优势是它支持发送代币时携带额外的信息，同时也支持代币的操作员功能。此外，ERC777 代币还可以通过 ERC1820 接口注册表合约来实现代币转账的监听，增强了代币的安全性.

虽然 ERC777 代币有很多优点，但是它也存在一些潜在的危险。由于 ERC777 代币是一种相对较新的代币标准，因此它的生态系统相对较小，可能存在一些安全漏洞。此外，ERC777 代币的操作员功能也可能会被滥用，导致代币被盗或者其他安全问题。因此，在使用 ERC777 代币时，需要谨慎选择代币合约，同时也需要注意代币的安全性.

25. 根据 Solidity 风格指南，函数应该如何排序？

答：函数应根据其可见性和顺序进行分组：

构造函数

receive 函数（如果存在）

fallback 函数（如果存在）

外部函数(external)

公共函数(public)

内部函数(internal)

私有函数(private)

在一个分组中，把 view 和 pure 函数放在最后。

26. 根据 Solidity 风格指南，函数修饰符应该如何排序？

答：函数修改器的顺序应该是:

可见性（Visibility）

可变性（Mutability）

虚拟（Virtual）

重载（Override）

自定义修改器（Custom modifiers）

function balance(uint from) public view override onlyowner returns (uint) { // onlyowner是自定义修改器

return balanceOf[from];

}

27. 什么是债券曲线(bonding curve)?

答：债券曲线是指一种数学模型，用于描述债券价格与到期时间之间的关系。它通常是一个连续的曲线，横坐标表示债券的到期时间，纵坐标表示债券的价格。债券曲线可以用来计算债券的收益率、估算债券价格以及预测市场利率的变化等。在加密货币领域，债券曲线也被用于构建一些新型的金融工具，如预测市场价格的算法交易、代币发行等。

28. OpenZeppelin ERC721 实现中的 safeMint 与 mint 有何不同?

答：在OpenZeppelin ERC721实现中，safeMint和mint都是用于创建新的ERC721代币的函数，但它们之间有一些区别。mint函数只是简单地创建一个新的ERC721代币，并将其分配给指定的地址。而safeMint函数则会在创建新的ERC721代币之前检查目标地址是否支持ERC721转移。如果目标地址不支持ERC721转移，则safeMint函数会抛出异常并阻止创建新的ERC721代币。因此，safeMint函数比mint函数更安全，因为它可以防止ERC721代币被锁定在不支持ERC721转移的合约中。

29. Solidity 提供哪些关键字来测量时间?

答：

- now：返回当前区块的时间戳（以秒为单位）。
- 时间单位：Solidity提供了几个时间单位，包括seconds、minutes、hours、days、weeks和years。这些单位可以与数字一起使用，例如5 minutes或1 hours。
- block.timestamp：与now关键字类似，返回当前区块的时间戳。
- block.number：返回当前区块的块号。

30. 什么是三明治(sandwich)攻击?

答：三明治攻击是去中心化的一种攻击手段，主要出现在swap交易中。黑客节点把用户的交易夹在中间，前后是攻击者的交易，利用用户的交易改变价格，获取价差套利。例如用户买入eth, 黑客会在创建一个eth买单，和卖单，然后打包交易，区块内的交易顺序是：黑客买单 - 用户买单 - 黑客卖单。导致用户真正的买入价格偏高。解决：可设置低滑点，如果价格超出预算范围交易失败。

31. 如果向一个会回滚的函数进行 delegatecall，delegatecall 会怎么做?

答：如果向一个会回滚的函数进行 delegatecall，delegatecall 会返回 false，并且不会发生任何状态更改。如果是常规调用而不是 delegatecall，则会导致交易失败并回滚，因为您不能调用一个不存在的合约或已自毁的合约。

32. 乘以和除以二的倍数的 gas 高效替代方法是什么?

答：在Solidity中，将一个数乘以或除以2的倍数可以通过移位运算来实现，这比使用乘法或除法运算更高效。具体来说，将一个数左移n位相当于将它乘以2的n次方，而将一个数右移n位相当于将它除以2的n次方。例如，将一个数除以8可以通过将它右移3位来实现，而将一个数乘以16可以通过将它左移4位来实现。在Solidity 0.8.3及更高版本中，编译器会自动将乘法和除法运算转换为移位运算，从而提高代码的效率。

33. 多大 uint 可以与一个地址在一个槽中?

答：一个地址是20个字节（160位），而一个uint256是32个字节（256位）, 256-160 = 96。uint96及以下的都可以

34. 哪些操作会部分退还 gas?

答：

SSTORE 操作：如果将一个存储槽从非零值更改为零值，则会退还一部分gas。

SLOAD 操作：如果从存储中读取一个非零值，则会退还一部分gas。

CALL 操作：如果调用的合约执行成功，则会退还一部分gas。

RETURN 操作：如果从函数中返回，则会退还一部分gas。

SELFDESTRUCT 操作：如果自毁合约，则会退还其余的gas。

需要注意的是，这些操作的退还gas的数量是固定的，具体取决于操作的类型和执行的环境。

35. ERC165 作用于什么?

答：ERC165是一个标准，用于检测和发布智能合约实现的接口。它标准化了如何识别接口，如何检测它们是否实现了ERC165或其他接口，以及合约将如何发布它们实现的接口。它可以帮助您查询合约实现的特定接口，以及更重要的是，该智能合约实现的版本。

36. 如果代理对 A 进行 delegatecall，而 A 执行 address(this).balance，返回的是代理的余额还是 A 的余额?

答：返回的是代理的余额。因为在 delegatecall 中，调用的A会共享代理合约的存储空间，因此 address(this) 将返回代理合约的地址，而不是被调用合约的地址。

37. 滑点参数有什么用?

答：滑点参数是指在交易时允许的价格波动范围。在去中心化交易所（DEX）中，滑点参数通常用于保护交易免受价格波动的影响。如果价格波动超过了滑点参数的范围，交易将被取消。滑点参数可以帮助交易者在不受过度波动的影响下进行交易，并减少交易失败的风险。

38. ERC721A 如何减少铸造成本？有什么权衡?

答：ERC721A是一个改进的ERC721标准，旨在通过减少铸造成本来提高NFT的可扩展性。ERC721A通过引入批量铸造API来实现这一点，从而将铸造成本降低到O(1)的时间复杂度。与OZ的单独铸造方式不同，ERC721A的批量铸造API可以同时铸造多个NFT，而不需要循环调用单独的铸造方法。这种方法可以显著减少铸造成本，但需要权衡的是，它可能会降低NFT的安全性。

39. 为什么 Solidity 不支持浮点数运算?

答：Solidity不支持浮点数运算是因为浮点数运算需要大量的计算资源，而以太坊虚拟机的计算资源是有限的。此外，浮点数运算可能会导致精度丢失和舍入错误，这可能会影响智能合约的正确性和安全性。为了避免这些问题，Solidity使用整数运算来代替浮点数运算。如果您需要进行浮点数运算，可以使用一些库，如FixedPoint.sol，来模拟浮点数运算。这些库使用整数运算来实现浮点数运算，从而避免了精度丢失和舍入错误的问题。

40. 什么是 TWAP?

答：TWAP（Time Weighted Average Price）时间加权平均价格是一种最简单的传统算法交易策略之一。该算法将交易时间均匀分割，并在每个分割节点上将均匀拆分的订单进行提交。其目的是使交易对市场影响减小的同时提供一个较低的平均成交价格，从而达到减小交易成本的目的。在分时成交量无法准确估计的情况下，该模型可以较好地实现算法交易的基本目的。

41. Compound Finance 如何计算利用率?

答：Compound Finance 是一个算法化的、自治的利率协议，旨在为开发者解锁一系列开放式金融应用。该协议的利用率是指借款人从 Compound 借入资产的数量与抵押品价值的比率。具体地，它是通过将所有借款人的借款总额除以所有抵押品的总价值来计算的。这个比率越高，代表 Compound 的借款人越多，市场上的资金也越紧张。（从compound中借款，可通过增加抵押品价值降低借款利率）。