Aula 4 – A Ethereum Virtual Machine (EVM) por Dentro

Bem-vindo à Aula 4! Se você já se aventurou no universo do desenvolvimento web, provavelmente entende a importância de um servidor ou de um ambiente de execução para o seu código. No mundo blockchain, especificamente na Ethereum, essa função é desempenhada por uma peça central e fascinante: a Ethereum Virtual Machine, ou EVM. Ela é, literalmente, o coração que pulsa em cada transação e execução de smart contract.

Entender a EVM não é apenas um detalhe técnico; é mergulhar na essência de como a Ethereum funciona e, mais importante, como seus contratos inteligentes realmente ganham vida. É aqui que a mágica acontece, onde seu código Solidity se transforma em instruções de baixo nível que a rede global da Ethereum pode executar. Para um desenvolvedor blockchain, dominar a EVM é como um engenheiro de software dominar o sistema operacional: é a base para construir soluções robustas, eficientes e seguras.

Nesta aula, vamos desvendar os mistérios da EVM. Começaremos explorando sua arquitetura interna, compreendendo como ela gerencia dados através de sua pilha (Stack), memória (Memory) e armazenamento persistente (Storage). Em seguida, abordaremos o conceito crucial de "gas", que é o combustível que move a EVM, e como ele impacta os custos e a otimização de suas aplicações. Mergulharemos nos opcodes, as instruções atômicas que a EVM entende, e finalizaremos com uma análise detalhada das diferentes formas de um contrato interagir com outro: CALL, DELEGATECALL e STATICCALL, entendendo suas implicações de segurança e uso. Ao final, você terá uma visão aprofundada de como a Ethereum executa seu código, permitindo que você escreva contratos mais eficientes e seguros.

A EVM: O Computador Global da Ethereum

Computador Global

Um vasto sistema descentralizado e sempre ativo onde qualquer pessoa pode rodar programas

Execução Consistente

Garante que o mesmo código produza o mesmo resultado em qualquer nó da rede

Sistema Operacional Universal

Ambiente padronizado e determinístico para smart contracts

Imagine a Ethereum como um vasto computador global, descentralizado e sempre ativo, onde qualquer pessoa pode rodar programas. Mas como esse "computador" executa o código dos smart contracts de forma consistente e segura em milhares de máquinas ao redor do mundo? É aí que entra a Ethereum Virtual Machine (EVM). Ela é o motor que processa as transações e executa os contratos inteligentes, garantindo que o mesmo código produza o mesmo resultado em qualquer nó da rede.

Pense na EVM como um sistema operacional universal para a blockchain Ethereum. Assim como o Windows ou o macOS fornecem um ambiente padronizado para seus aplicativos, a EVM oferece um ambiente de execução isolado e determinístico para os smart contracts. Isso significa que, independentemente de onde o código do contrato esteja sendo executado – seja no seu computador local ou em um servidor na China –, ele se comportará exatamente da mesma maneira, garantindo a integridade e a previsibilidade que são pilares da tecnologia blockchain.

Por que isso importa? Essa padronização é fundamental para a segurança e a confiabilidade da rede. Sem a EVM, cada nó da Ethereum precisaria de uma maneira diferente de interpretar e executar o código, levando a inconsistências e potenciais falhas. Com ela, temos uma "máquina" abstrata que todos os nós concordam em simular, criando um estado global único e compartilhado que é a própria essência da blockchain Ethereum.

Arquitetura da EVM: Stack, Memória e Storage

Para entender como a EVM executa o código, precisamos olhar para seus componentes internos. A EVM não é um computador físico, mas uma máquina de estado baseada em pilha (stack-based machine). Isso significa que ela usa uma estrutura de dados chamada "pilha" para a maioria de suas operações. Além da pilha, ela possui dois outros locais importantes para armazenar dados: a memória e o armazenamento (storage). Cada um tem um propósito distinto e impacta diretamente a forma como você projeta e otimiza seus smart contracts.

01

Stack (Pilha)

Bloco de notas temporário para operações rápidas e cálculos imediatos

02

Memory (Memória)

Lousa volátil para dados durante a execução de uma função

03

Storage (Armazenamento)

Disco rígido persistente para dados permanentes na blockchain

Esses três componentes – Stack, Memory e Storage – são como as diferentes partes de uma bancada de trabalho de um artesão. Cada ferramenta e material tem seu lugar específico e sua função. Entender onde e como os dados são manipulados em cada um desses locais é crucial para escrever contratos eficientes e evitar surpresas com os custos de transação, que são diretamente influenciados pelo uso desses recursos.

Vamos explorar cada um desses componentes em detalhes, começando pela estrutura mais fundamental da EVM: a pilha.

A Pilha (Stack): O Bloco de Notas Temporário da EVM

Imagine que você está resolvendo um problema complexo e precisa fazer alguns cálculos intermediários ou guardar pequenas informações por um curto período. Você usaria um bloco de notas, escrevendo e apagando rapidamente. Na EVM, a pilha (Stack) funciona de maneira muito similar. Ela é uma área de dados volátil e temporária, usada para armazenar valores durante a execução de opcodes.

A pilha opera sob o princípio **LIFO (Last-In, First-Out)**, o que significa que o último item adicionado é o primeiro a ser removido. Pense em uma pilha de pratos: você sempre adiciona um prato no topo e sempre pega o prato que está no topo. Na EVM, opcodes como PUSH adicionam um item à pilha, e opcodes que realizam operações (como ADD ou MUL) retiram os operandos do topo da pilha, realizam a operação e colocam o resultado de volta.

Características da Stack

- **Tamanho:** Limitado a 1024 itens
- **Velocidade:** Acesso extremamente rápido
- **Uso:** Operações aritméticas e cálculos temporários
- **Persistência:** Volátil (não permanente)

Essa estrutura é extremamente eficiente para operações rápidas e cálculos aritméticos, pois o acesso aos dados é imediato. No entanto, a pilha tem um tamanho limitado (1024 itens) e não é ideal para armazenar grandes volumes de dados ou para acesso aleatório. Ela é o local de trabalho imediato da EVM, onde as operações atômicas são processadas.

Memória (Memory): A Lousa Volátil do Contrato

Enquanto a pilha é excelente para operações rápidas e temporárias, ela não é adequada para armazenar dados que precisam ser acessados de forma mais flexível ou em maior quantidade durante a execução de uma função. Para isso, a EVM utiliza a Memória. Pense na memória como uma lousa que o contrato pode usar para escrever e ler informações enquanto uma função está sendo executada.

Endereçamento Direto

Área de bytes endereçável - você pode acessar qualquer posição de memória diretamente, ao contrário da pilha

Volatilidade

Todos os dados são apagados ao final de cada chamada de função externa

Uso Ideal

Armazenar arrays, strings e estruturas de dados complexas dentro do escopo de uma transação

O acesso à memória tem um custo de gas, e esse custo aumenta à medida que a memória utilizada se expande, pois a EVM precisa alocar mais recursos. Portanto, gerenciar o uso da memória de forma eficiente é uma parte importante da otimização de contratos inteligentes. Diferente da pilha, que é mais para "rascunhos" rápidos, a memória é para "anotações" mais elaboradas que duram o tempo de uma tarefa específica.

Storage: O Disco Rígido Persistente da Blockchain

Se a pilha é o bloco de notas e a memória é a lousa, o Storage (Armazenamento) é o disco rígido da blockchain. Ele é o único local onde os dados de um smart contract são armazenados de forma persistente na blockchain, ou seja, eles permanecem lá mesmo após a transação ser concluída e por todas as transações futuras. Variáveis de estado declaradas em Solidity (como `uint public myVariable;`) são armazenadas no Storage.

2^256	32	20K
Slots Disponíveis	**Bytes por Slot**	**Gas para SSTORE**
Mapeamento chave-valor gigantesco	Capacidade de cada slot de armazenamento	Custo aproximado de escrita (modificação)

O Storage é um mapeamento chave-valor de 2^256 slots, onde cada slot pode armazenar 32 bytes de dados. É um espaço gigantesco, mas cada operação de leitura (SLOAD) e, especialmente, de escrita (SSTORE) no Storage é significativamente mais cara em termos de gas do que as operações na pilha ou na memória. Isso ocorre porque as alterações no Storage precisam ser gravadas permanentemente na blockchain e replicadas por todos os nós da rede.

> **Otimização Crítica:** Devido ao alto custo, a otimização do uso do Storage é uma das áreas mais críticas para a eficiência de um smart contract. Desenvolvedores experientes buscam minimizar as escritas no Storage e empacotar dados de forma inteligente para economizar gas. Entender essa persistência e o custo associado é fundamental para projetar contratos que sejam economicamente viáveis e sustentáveis a longo prazo.

O Conceito de Gas na Ethereum: O Combustível da Rede

Agora que entendemos onde a EVM guarda e manipula os dados, precisamos falar sobre como ela se move: o gas. Na Ethereum, cada operação, desde uma simples adição na pilha até uma complexa escrita no Storage, consome uma quantidade específica de "gas". O gas não é uma criptomoeda em si, mas uma unidade de medida do esforço computacional necessário para executar uma operação na EVM.

Unidade de Medida

Pense no gas como a gasolina que seu carro precisa para andar. Cada quilômetro rodado (ou cada operação na EVM) consome uma certa quantidade de gasolina (gas).

Proteção da Rede

O objetivo principal do gas é evitar loops infinitos e spam na rede, além de remunerar os mineradores ou validadores que processam e validam as transações.

Cálculo de Custo

O custo total de uma transação é determinado pela quantidade de gas consumida multiplicada pelo "preço do gas" (Gas Price), expresso em Gwei.

Se você não tiver gasolina suficiente, seu carro não anda; se uma transação não tiver gas suficiente, ela falha. O custo total de uma transação é determinado pela quantidade de gas consumida multiplicada pelo "preço do gas" (Gas Price), que é o valor que você está disposto a pagar por unidade de gas, geralmente expresso em Gwei (uma fração de Ether). Esse preço varia de acordo com a demanda da rede: quanto mais congestionada a rede, maior o preço do gas.

Cálculo de Custos de Transação e Otimização

Fórmula do Custo

Custo Total (em Ether) = Gas Usado × Gas Price

- **Gas Usado:** Soma do gas de todos os opcodes executados e do gas para armazenamento de dados
- **Gas Price:** Valor que você define (ou que sua carteira sugere) para cada unidade de gas

Estratégias de Otimização

1. Minimizar escritas no Storage
2. Usar variáveis `view` ou `pure` quando possível
3. Evitar loops desnecessários ou ineficientes
4. Empacotar dados de forma inteligente

Compreender o gas é o primeiro passo; saber como calcular e otimizar seus custos é o segundo, e talvez o mais importante, para um desenvolvedor. O Gas Usado é a soma do gas de todos os opcodes executados e do gas para armazenamento de dados. O Gas Price é o valor que você define (ou que sua carteira sugere) para cada unidade de gas.

A otimização de gas é uma arte e uma ciência. Pequenas mudanças no código podem ter um impacto significativo nos custos. Por exemplo, armazenar dados no Storage é muito mais caro do que na Memory ou na Stack. Loops desnecessários ou ineficientes podem consumir grandes quantidades de gas. Uma estratégia comum é minimizar as escritas no Storage e, sempre que possível, usar variáveis view ou pure em Solidity, que não modificam o estado da blockchain e, portanto, não custam gas (exceto pelo custo de execução do nó que as processa).

> **Soluções de Escalabilidade 2025:** As soluções de escalabilidade de Layer 2, como Optimistic Rollups (Arbitrum, Optimism) e ZK-Rollups (zkSync, StarkNet), são cruciais para a otimização de custos em 2025. Elas processam transações fora da cadeia principal da Ethereum e as agrupam em um único pacote, reduzindo drasticamente o custo por transação. Entender a EVM ajuda a otimizar o código base, e as Layer 2s oferecem uma camada adicional de economia e escalabilidade.

Opcodes da EVM: Uma Visão de Baixo Nível

Por trás de cada linha de código Solidity que você escreve, há uma série de instruções de baixo nível que a EVM realmente executa. Essas instruções são chamadas de "opcodes" (operation codes). Cada opcode é uma operação atômica, como "adicionar dois números", "empurrar um valor para a pilha" ou "armazenar um valor no storage". A EVM entende e executa esses opcodes um por um.

Instruções Básicas

Como as instruções que um processador de computador entende - operações atômicas fundamentais

Compilação

O compilador Solidity traduz seu código de alto nível em uma sequência de opcodes

Custo de Gas

Cada opcode tem um custo de gas fixo associado

Pense nos opcodes como as instruções básicas que um processador de computador entende. Quando você escreve um programa em Python ou JavaScript, ele é compilado ou interpretado em instruções que a CPU pode executar. Da mesma forma, o compilador Solidity traduz seu código de alto nível em uma sequência de opcodes que a EVM pode processar. Cada opcode tem um custo de gas fixo associado, o que reforça a importância de escrever código eficiente.

Conhecer os opcodes não significa que você precisará escrever smart contracts em assembly da EVM (embora seja possível!). No entanto, ter uma compreensão básica de como eles funcionam e quais são os mais caros (como SSTORE para escrita no storage) pode ajudá-lo a depurar contratos, otimizar o uso de gas e entender melhor as vulnerabilidades de segurança. É a linguagem nativa da máquina que executa seus contratos.

Opcode	Função Principal	Custo de Gas (exemplo)
PUSH1	Empurra um byte para a pilha	3
ADD	Soma os dois itens do topo da pilha	3
MUL	Multiplica os dois itens do topo da pilha	5
SSTORE	Armazena um valor no Storage (persistente)	20000 (modificação)
SLOAD	Carrega um valor do Storage para a pilha	100
CALL	Chama outro contrato	700

Entendendo os Opcodes na Prática

Para ilustrar como os opcodes funcionam, vamos considerar um exemplo muito simplificado. Imagine que seu contrato Solidity tem uma linha como `uint result = a + b;`. Quando o compilador Solidity processa isso, ele pode gerar uma sequência de opcodes como:

PUSH1 0x0A

Empurra o valor 10 para a pilha (representando 'a')

PUSH1 0x05

Empurra o valor 5 para a pilha (representando 'b')

ADD

Retira 10 e 5 da pilha, soma, e empurra 15 de volta

PUSH1 0x00

Empurra o slot de storage 0 para a pilha

SSTORE

Retira 0 e 15 da pilha, armazena 15 no slot 0 do storage

Nesse pequeno trecho, você pode ver a interação entre a pilha (PUSH, ADD) e o storage (SSTORE). Cada uma dessas operações tem um custo de gas. Ferramentas como o Etherscan permitem que você visualize o bytecode de um contrato e, em alguns casos, até mesmo o fluxo de opcodes durante a execução de uma transação, o que é incrivelmente útil para depuração e auditoria de segurança.

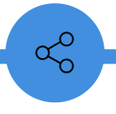
> **Insight Prático:** Essa visão de baixo nível nos ajuda a apreciar a complexidade por trás de cada transação e a entender por que certas operações são mais caras do que outras. É a base para escrever código Solidity que não apenas funcione, mas que seja eficiente e seguro, evitando surpresas com os custos de gas e potenciais vulnerabilidades.

Diferença entre CALL, DELEGATECALL e STATICCALL – Introdução

No desenvolvimento de smart contracts, é comum que um contrato precise interagir com outro. Seja para chamar uma função, transferir Ether ou reutilizar lógica, a forma como essa interação acontece é crucial. A EVM oferece três opcodes principais para chamadas entre contratos: CALL, DELEGATECALL e STATICCALL. Embora todos permitam que um contrato execute o código de outro, eles fazem isso de maneiras fundamentalmente diferentes, com implicações significativas para a segurança e o comportamento do contrato.

CALL

A chamada padrão - executa código no contexto do contrato chamado



DELEGATECALL

Reutilização de lógica - executa código no contexto do contrato chamador

STATICCALL

Chamada somente leitura - não pode modificar o estado da blockchain

A escolha do tipo de chamada é uma das decisões mais importantes que um desenvolvedor de smart contracts pode tomar, pois um erro aqui pode levar a vulnerabilidades graves, como roubo de fundos ou manipulação de estado. Pense nessas chamadas como três maneiras distintas de "pedir um favor" a outro contrato: cada uma com suas próprias regras sobre quem é o "remetente", quem paga a conta e quem tem permissão para mudar as coisas.

Compreender as nuances entre CALL, DELEGATECALL e STATICCALL é essencial para construir sistemas robustos e seguros na Ethereum. Vamos desvendar cada um deles, entendendo seus mecanismos, casos de uso e, principalmente, os riscos associados.

Detalhando CALL: A Chamada Padrão

O CALL é o tipo de chamada mais comum e fundamental na EVM. Quando um contrato A faz um CALL para um contrato B, ele está pedindo ao contrato B para executar uma de suas funções. A característica mais importante do CALL é que o código do contrato B é executado no *contexto* do próprio contrato B. Isso significa que:

1 msg.sender e msg.value

Dentro da função chamada no contrato B, o `msg.sender` será o endereço do contrato A, e o `msg.value` será qualquer Ether enviado pelo contrato A na chamada.

2 Estado

Qualquer modificação de estado (escrita no Storage) feita pelo contrato B afetará o Storage do *próprio contrato B*. O Storage do contrato A permanece inalterado.

3 Gas

O contrato A pode especificar uma quantidade de gas para a chamada. Se o contrato B consumir mais gas do que o permitido, a chamada falha.

O CALL é usado para interações padrão, como transferir Ether para outro contrato, chamar uma função para obter um valor ou interagir com um token ERC-20. No entanto, ele também é a raiz de algumas vulnerabilidades, como o ataque de reentrancy, onde um contrato malicioso pode chamar de volta o contrato original repetidamente antes que a transação seja concluída.

> **Caso de Uso Típico:** Transferir Ether, chamar funções de tokens ERC-20, interações padrão entre contratos onde cada um mantém seu próprio estado.

Detalhando DELEGATECALL: Reutilizando Lógica com Cuidado

O DELEGATECALL é uma chamada mais poderosa e, consequentemente, mais perigosa. Quando um contrato A faz um DELEGATECALL para um contrato B, ele está pedindo ao contrato B para executar seu código, mas com uma diferença crucial: o código do contrato B é executado no *contexto* do contrato *chamador* (contrato A). Isso significa que:

Características

- **msg.sender e msg.value:** Dentro da função chamada no contrato B, o `msg.sender` e o `msg.value` serão os do *chamador original* do contrato A, não o contrato A em si.
- **Estado:** Qualquer modificação de estado (escrita no Storage) feita pelo contrato B afetará o Storage do *contrato A*. O Storage do contrato B permanece inalterado.

Uso Principal

O DELEGATECALL é amplamente utilizado em padrões de **upgradeability** de contratos (contratos proxy), onde um contrato proxy (A) delega a lógica para um contrato de implementação (B). Isso permite que a lógica do contrato seja atualizada sem mudar o endereço do contrato proxy.

> ⚠️ **Alerta de Segurança:** No entanto, a execução de código externo no seu próprio contexto de Storage é um risco enorme. Se o contrato B for malicioso ou tiver um bug, ele pode corromper ou roubar os dados do contrato A.

Conceito	Âmbito/Aplicação	Base/Origem
CALL	Interação padrão, transferência de Ether	Executa no contexto do contrato chamado
DELEGATECALL	Reutilização de lógica, upgradeability (proxies)	Executa no contexto do contrato chamador

Detalhando STATICCALL: A Chamada Somente Leitura

O STATICCALL é uma versão mais segura do CALL, introduzida para mitigar certos tipos de ataques e garantir a integridade do estado. Quando um contrato A faz um STATICCALL para um contrato B, ele está pedindo ao contrato B para executar uma de suas funções, mas com uma restrição fundamental: a chamada é "somente leitura". Isso significa que:

msg.sender e msg.value

Assim como no CALL, o `msg.sender` será o endereço do contrato A, e o `msg.value` será qualquer Ether enviado (embora enviar Ether em um STATICCALL geralmente resultará em falha).

Restrição de Estado

Qualquer tentativa de modificação de estado (escrita no Storage, emissão de eventos, criação de novos contratos, etc.) feita pelo contrato B resultará em uma reversão da transação.

O STATICCALL é ideal para situações onde um contrato precisa consultar dados de outro contrato sem o risco de que o contrato chamado altere o estado da blockchain. Isso é particularmente útil para oráculos, bibliotecas de dados ou qualquer interação que deva ser puramente informativa. Ele oferece uma garantia de segurança importante, pois impede efeitos colaterais indesejados.

Conceito	Âmbito/Aplicação	Base/Origem	Restrição Principal
CALL	Interação padrão, transferência de Ether	Executa no contexto do contrato chamado	Pode modificar o estado do contrato chamado
DELEGATECALL	Reutilização de lógica, upgradeability (proxies)	Executa no contexto do contrato *chamador*	Pode modificar o estado do contrato *chamador* (risco alto)
STATICCALL	Consulta de dados, leitura segura	Executa no contexto do contrato chamado	Não pode modificar o estado da blockchain

Tendências e Futuro da EVM: Evoluindo o Ecossistema

A EVM é a espinha dorsal da Ethereum, mas o ecossistema blockchain está em constante evolução. As limitações da EVM original em termos de escalabilidade e experiência do usuário levaram ao desenvolvimento de novas soluções que complementam e expandem suas capacidades. Para um desenvolvedor em 2025, é crucial estar ciente dessas tendências e como elas se integram com o conhecimento da EVM.

Abstração de Contas (ERC-4337)

Tradicionalmente, as contas na Ethereum são de dois tipos: EOA (Externally Owned Accounts), controladas por chaves privadas, e contas de contrato. A Abstração de Contas permite que as carteiras sejam smart contracts por si só, eliminando a necessidade de seed phrases e possibilitando recursos como recuperação social, pagamentos de gas por terceiros e autenticação multifator nativa. Isso melhora drasticamente a experiência do usuário (UX) em dApps, tornando a interação com a blockchain mais familiar e menos intimidadora.

Soluções de Escalabilidade (Layer 2)

Como discutimos, a EVM na Layer 1 tem limitações de throughput e custos de gas. Optimistic Rollups (como Arbitrum e Optimism) e ZK-Rollups (como zkSync e StarkNet) processam transações fora da cadeia principal, mas herdam sua segurança, permitindo milhares de transações por segundo a custos muito mais baixos. Eles são, em essência, "EVMs em miniatura" ou ambientes compatíveis com EVM que se comunicam com a Layer 1.

Interoperabilidade e Cross-Chain

Protocolos como Chainlink CCIP e LayerZero permitem que smart contracts em diferentes blockchains (incluindo EVMs e outras VMs) se comuniquem e troquem dados e ativos de forma segura. Isso transforma a Ethereum de uma ilha em um continente conectado, abrindo novas possibilidades para aplicações descentralizadas que abrangem múltiplas redes. A EVM continua sendo o motor, mas agora ela faz parte de uma rede de motores interconectados.

Consolidação e Próximos Passos

Chegamos ao fim da nossa jornada pela Ethereum Virtual Machine. Vimos que a EVM é o ambiente de execução determinístico que dá vida aos smart contracts, gerenciando dados através de sua pilha (Stack) para operações rápidas, memória (Memory) para dados temporários de função e armazenamento (Storage) para dados persistentes na blockchain. Compreendemos o papel vital do "gas" como combustível e mecanismo de segurança, e exploramos os opcodes, as instruções de baixo nível que a EVM executa. Finalmente, desvendamos as diferenças cruciais entre CALL, DELEGATECALL e STATICCALL, entendendo suas implicações para a segurança e o design de contratos.

Em prática

Com este conhecimento, você está mais apto a escrever smart contracts mais eficientes em gas, depurar problemas de execução e, crucialmente, projetar interações entre contratos de forma segura, evitando vulnerabilidades comuns. A compreensão da EVM é a base para se tornar um desenvolvedor blockchain de alto nível, capaz de construir soluções robustas e inovadoras.

Autoavaliação

1. Qual dos componentes da arquitetura da EVM é utilizado para armazenar dados de forma persistente na blockchain, mesmo após a conclusão de uma transação?
   a) Stack
   b) Memory
   c) Storage
   d) Cache
2. O que o conceito de "gas" representa na Ethereum?
   a) Uma criptomoeda utilizada para pagar taxas de transação.
   b) Uma unidade de medida do esforço computacional para executar operações na EVM.
   c) O limite máximo de Ether que pode ser enviado em uma transação.
   d) Um mecanismo para aumentar a velocidade das transações na rede.
3. Um desenvolvedor deseja que um contrato A chame uma função em um contrato B, mas que essa chamada não possa, de forma alguma, modificar o estado da blockchain. Qual tipo de chamada ele deve utilizar?
   a) CALL
   b) DELEGATECALL
   c) STATICCALL
   d) TRANSFER
4. Qual das seguintes afirmações sobre o DELEGATECALL é verdadeira?
   a) Ele executa o código do contrato chamado no contexto do contrato chamado.
   b) Ele permite que o contrato chamado modifique o storage do contrato chamador.
   c) Ele é usado principalmente para transferir Ether entre contratos.
   d) Ele proíbe qualquer modificação de estado durante a execução.
5. Explique como as soluções de escalabilidade de Layer 2 (como Rollups) se relacionam com a EVM e qual o seu principal benefício para o ecossistema Ethereum.

Gabarito

1. c) Storage
2. b) Uma unidade de medida do esforço computacional para executar operações na EVM
3. c) STATICCALL
4. b) Ele permite que o contrato chamado modifique o storage do contrato chamador

Próxima Aula e Recursos Adicionais

Próxima Aula

Aula 5 – Solidity Avançado: Estruturas e Padrões de Dados

Aprofundaremos no Solidity, explorando estruturas de dados complexas e padrões de design que otimizam a eficiência e a segurança dos seus contratos, construindo sobre o conhecimento da EVM adquirido hoje.

Recursos Adicionais

- **Documentação Oficial da Ethereum**

  Para detalhes técnicos aprofundados sobre a EVM e seus opcodes.

- **Artigos sobre ERC-4337**

  Para entender as últimas tendências em abstração de contas e UX.

- **Whitepapers de Rollups (Arbitrum, zkSync)**

  Para explorar as soluções de escalabilidade de Layer 2.

NOTA IMPORTANTE: As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.