**Documentação da API**

**DistributedMessagingSystem**

Classe principal que gerencia todo o sistema de mensagens distribuídas, incluindo clientes, canais e recursos compartilhados.

# Inicializa o sistema de mensageria distribuída

system = DistributedMessagingSystem()

# Registra um novo cliente

system.register\_client(client\_id: str) -> bool

# Cria um novo canal de comunicação

system.create\_channel(channel\_name: str) -> bool

# Cria um novo recurso compartilhado

system.create\_resource(resource\_id: str) -> bool

# Inscreve um cliente em um canal

system.subscribe\_to\_channel(client\_id: str, channel\_name: str) -> bool

# Envia uma mensagem unicast (diretamente para um cliente específico)

system.unicast(sender\_id: str, receiver\_id: str, message: Message) -> bool

# Envia uma mensagem multicast (para todos os clientes de um canal)

system.multicast(sender\_id: str, channel\_name: str, content: str) -> bool

# Envia uma mensagem broadcast (para todos os clientes registrados)

system.broadcast(sender\_id: str, content: str) -> bool

\*\*Atributos:\*\*

- `clients`: Dicionário que mapeia IDs de clientes para objetos Client

- `channels`: Dicionário que mapeia nomes de canais para objetos Channel

- `resources`: Dicionário que mapeia IDs de recursos para objetos ResourceManager

- `global\_clock`: Relógio lógico global do sistema (LogicalClock)

- `lock`: Lock para sincronização de threads

**Client**

Representa um cliente no sistema, capaz de enviar e receber mensagens e gerenciar recursos.

# Inicializa um cliente

client = Client(client\_id: str, messaging\_system: DistributedMessagingSystem)

# Envia uma mensagem unicast

client.send\_unicast(receiver\_id: str, content: str, msg\_type: str = "normal", request\_id: str = None) -> bool

# Envia uma mensagem multicast

client.send\_multicast(channel\_name: str, content: str) -> bool

# Envia uma mensagem broadcast

client.send\_broadcast(content: str) -> bool

# Requisita acesso a um recurso compartilhado

client.request\_resource(resource\_id: str) -> bool

# Libera um recurso compartilhado

client.release\_resource(resource\_id: str) -> bool

# Consome a próxima mensagem disponível na fila

client.consume\_message() -> Optional[Message]

# Obtém o histórico de mensagens enviadas e recebidas

client.get\_message\_log() -> List[str]

\*\*Atributos:\*\*

- `id`: Identificador único do cliente

- `messaging\_system`: Referência ao sistema de mensagens distribuídas

- `inbox`: Buffer de mensagens recebidas (MessageBuffer)

- `local\_clock`: Relógio lógico local do cliente (LogicalClock)

**ResourceManager**

Gerenciador de recursos compartilhados que implementa o algoritmo de exclusão mútua distribuída.

# Inicializa um gerenciador de recursos

resource\_manager = ResourceManager(resource\_id: str, messaging\_system: DistributedMessagingSystem)

# Solicita o uso de um recurso para um cliente

resource\_manager.request\_resource(client\_id: str) -> bool

# Libera o uso de um recurso para um cliente

resource\_manager.release\_resource(client\_id: str) -> bool

# Processa uma mensagem de requisição de recurso

resource\_manager.handle\_request(message: Message) -> None

# Processa uma mensagem de resposta de recurso

resource\_manager.handle\_reply(message: Message) -> None

# Processa uma mensagem de liberação de recurso

resource\_manager.handle\_release(message: Message) -> None

\*\*Atributos:\*\*

- `resource\_id`: Identificador do recurso gerenciado

- `messaging\_system`: Referência ao sistema de mensagens

- `lock`: Lock para sincronização

- `request\_queue`: Fila de solicitações pendentes

- `replied`: Conjunto de clientes que responderam

- `my\_request`: Tupla contendo (timestamp, client\_id, request\_id) da requisição atual

- `using\_resource`: Flag indicando se o recurso está em uso

- `deferred\_replies`: Respostas diferidas organizadas por cliente

**Message**

Classe que representa uma mensagem no sistema.

# Cria uma nova mensagem

message = Message(sender: str, content: str, timestamp: int, msg\_type: str = "normal", request\_id: str = None)

\*\*Atributos:\*\*

- `sender`: ID do cliente remetente

- `content`: Conteúdo da mensagem

- `timestamp`: Timestamp lógico da mensagem

- `consumption\_time`: Timestamp de consumo (preenchido quando a mensagem é consumida)

- `msg\_type`: Tipo da mensagem (normal, request, reply, release)

- `request\_id`: ID único para mensagens de recurso

**Channel**

Canais de comunicação para mensagens multicast.

# Cria um novo canal

channel = Channel(name: str)

\*\*Atributos:\*\*

- `name`: Nome do canal

- `buffer`: Buffer de mensagens do canal (MessageBuffer)

- `subscribers`: Conjunto de IDs de clientes inscritos

**MessageBuffer**

Buffer para armazenamento e ordenação de mensagens por timestamp lógico.

# Inicializa um buffer de mensagens

buffer = MessageBuffer()

\*\*Métodos:\*\*

- `add\_message(message: Message)`: Adiciona uma mensagem ao buffer

- `get\_message(consumer\_id: str) -> Message`: Obtém a próxima mensagem ordenada

- `get\_log() -> List[str]`: Retorna o histórico de mensagens

**LogicalClock**

Implementação de relógio lógico para ordenação de eventos.

# Inicializa um relógio lógico

clock = LogicalClock()

\*\*Métodos:\*\*

- `increment() -> int`: Incrementa e retorna o contador

- `update(received\_time: int) -> int`: Atualiza o contador com base em um timestamp recebido