**TESTADOR INDEPENDENTE - DOCUMENTAÇÃO ATUALIZADA**

Acho que esta otimo, mas antes de continuar vc vai atualizar a documentacao. Observa cuidadosamente as mudancas e adicoes, como por exemplo o tema das mensagens. NAO TIRA NADA DA DOCUMENTACAO original a nao ser que tenha mudado. O que nao mudou deixa como esta. Uma forma de checar que vc nao apagou nada é contar as palavras da documentacao original e da documentacao final, excluindo o codigo e printouts. A final tem que ser maior pela quantidade de coisas que incluimos. Inclui tambem os exemplos das mensagens que te passei. Por ultimo, considera que essa documentacao sera a base da continuidade do desenvolvimento, mas, tambem, se este chat ficar muito longo, sera utilizada num novo chat. Entao, esta documentacao deve ser o suficientemente ampla, clara e precissa para que vc, no novo chat, a entenda e a interprete corretamente para continuar com o desenvolvimento. Tambem, essa doc deve ser em puro texto para poder fazer copy/paste num arquivo txt. Ta claro? Alguma duvida?

NAO TIRA NADA DA DOCUMENTACAO original a nao ser que tenha mudado. O que nao mudou deixa como esta. Por ultimo, considera que essa documentacao sera a base da continuidade do desenvolvimento num outro chat. Portanto, esta documentacao deve ser o suficientemente ampla, clara e precissa para que vc, no novo chat, a entenda e a interprete corretamente para poder continuar eficientemente. Tambem, essa doc deve ser em puro texto para poder fazer copy/paste num arquivo txt. Ta claro? Alguma duvida? Nao faz nada ainda, so me responde.

-----

Como temos feito muitas definicoes e correcoes do projeto original, antes de continuar, faca uma atualizacao da documentacao com as mudancas deste chat. Observa cuidadosamente as mudancas e adicoes, como por exemplo o tema dos tempos transcorridos. NAO TIRA NADA DA DOCUMENTACAO original a nao ser que tenha mudado. O que nao mudou deixa como esta. Uma forma de checar que nao apagou nada é contar as palavras da documentacao original e da documentacao final, excluindo o codigo. A final tem que ser bem maior pela quantidade de coisas que incluimos, principalmente o tema dos workers e do messagechannel, que deve ser muito bem documentado. A isso, agrega as partes relevantes dos codigos que te passei para utilizar como modelo e exemplo. Por ultimo, considera que essa documentacao sera a base da continuidade do desenvolvimente, mas, tambem, se este chat ficar muito longo, sera utilizada num novo chat. Entao, deve ser o suficientemente ampla, clara e precissa para que vc, no novo chat, a entenda e a interprete corretamente para continuar com o desenvolvimento. Tambem, essa doc deve ser em puro texto para poder fazer copy/paste num arquivo txt. Ta claro? Alguma duvida?Testador Independente. Estado inicial, data 28/03/2025.

Objetivo Geral

O objetivo é criar um script em Node.js (testadorIndependente.js) que sirva como base para o workerCoordenador.js, focado em executar ordens de arbitragem triangular em tempo real nas exchanges Bybit e OKX, com potencial expansão para até 6 ou 7 exchanges. O sistema é dividido em dois worker\_threads separados: um para gerenciar conexões WebSocket e outro para executar trades.

Arquitetura (Dois Workers)

Worker de Conexoes:

Mantem WebSockets ativos (Bybit: Trade e Order; OKX: Privado).

Envia pings (Bybit: a cada 20s; OKX: a cada 15s) e gerencia reconexoes apos 5s de falha.

Publica dados de status para o Worker de Trades via MessageChannel.

Atualmente configurado para logar ordens ouvidas apos autenticacao e subscricao.

Worker de Trades:

Planejado para:

Ficar ocioso ate ser ativado pelo main.

Executar ordens com base na matriz operacoes, onde T1 e postada e aguarda preenchimento, e T2 e T3 sao disparadas simultaneamente apos T1 ser preenchida.

Usar o Worker de Conexoes como proxy para postar ordens via MessageChannel.

Verificar o resultado da postagem de T2 e T3 (aceitacao pelo servidor) e logar mensagens completas recebidas.

Implementar timeout de 3 segundos por ordem, sem retry.

Status: Ainda nao implementado na documentacao original, mas agora detalhado com base nas discussoes.

Razoes da Separacao:

Isolamento de carga para evitar bloqueios no event loop.

Resiliencia contra falhas (ex.: crash em uma exchange nao derruba tudo).

Escalabilidade para suportar mais instancias de trades no futuro.

Comunicacao:

Usa worker\_threads com MessageChannel (portToTrades e portToMain) para baixa latencia entre os workers.

O Worker de Conexoes ja cria o canal e passa uma porta ao main.

O Worker de Trades envia ordens ao Worker de Conexoes via portToTrades e recebe status de volta, com todas as mensagens logadas integralmente para analise.

Estrutura Geral

Matriz de Operacoes:

Define as ordens (ex.: T1: comprar BTC em OKX, T2: vender USDT em Bybit, T3: vender BTC em Bybit).

Particularidade: Na OKX, T1 usa ordType: 'ioc' (ignora type: 'limit'), enquanto Bybit usa type e timeInForce separadamente. T2 e T3, geralmente market, nao requerem timeInForce.

Exemplo:

    const operacoes = [

      { exchange: 'OKX', par: 'BTC-USDT', side: 'buy', type: 'limit', amount: '0.0001', price: '82300', timeInForce: 'IOC', orderTag: 'T1' },

      { exchange: 'Bybit', par: 'BTCBRL', side: 'Sell', type: 'limit', amount: '0.0001', price: '484000', timeInForce: null, orderTag: 'T2' },

      { exchange: 'Bybit', par: 'USDTBRL', side: 'Buy', type: 'limit', amount: '10', price: '5.800', timeInForce: null, orderTag: 'T3' }

Matriz de Exchanges:

Contem configuracoes e funcoes especificas (ex.: URLs WebSocket, chaves API).

Atualmente configurada para Bybit e OKX no workerConexoes.js.

Estrutura deve ser modular para escalar ate 7+ exchanges.

WebSockets:

Bybit:

Trade (wss://stream.bybit.com/v5/trade): Para postagem de ordens (ainda nao implementado no Worker de Trades).

Order (wss://stream.bybit.com/v5/private): Para status de ordens (topic: 'order').

Autenticacao com apiKey, expires (timestamp + 10s) e HMAC-SHA256 de GET/realtime${expires}.

OKX:

Privado (wss://ws.okx.com:8443/ws/v5/private): Para postagem e status (channel: 'orders').

Autenticacao com apiKey, passphrase, timestamp e HMAC-SHA256 de timestamp + GET/users/self/verify.

Componentes Principais

1. Inicializacao das Conexoes:

Funcao inicializarConexoes() autentica e subscreve os WebSockets.

Usa Promise.all para garantir que todas as conexoes estejam prontas.

Atualmente logs mostram apenas conexao, autenticacao, subscricao, pings e ordens ouvidas.

Em producao, o main abre todas as conexoes (trades, orders e publicas como orderbook).

2. Postagem de Ordens:

Planejado: Bybit usara op: 'order.create' via tradeWs; OKX usara op: 'order' via wsTrade.

T1 e postada e aguarda 'Filled'; T2 e T3 sao disparadas juntas apos T1, geralmente como market orders.

Resultado da postagem (OK ou erro) e logado.

3. Verificacao de Status:

Bybit: Escuta 'order' e verifica orderStatus: 'Filled'.

OKX: Escuta 'orders' e verifica state: 'filled'.

Timeout de 3s por ordem, sem retry. Se T1 nao for preenchida, para e avisa; se T2 ou T3 falharem, loga sem cancelar.

Todas as mensagens recebidas sao logadas integralmente com tempo transcorrido desde T1.

4. Fluxo de Execucao:

Worker de Conexoes sinaliza prontidao ao main via parentPort.postMessage({ type: 'conexoesProntas' }).

Worker de Trades executara as ordens:

Para testes, usa matriz operacoes no inicio e dispara apos conexoes prontas.

Em producao, espera { type: 'start', operacoes } do main.

Posta T1, verifica status, dispara T2 e T3 simultaneamente se T1 for 'Filled'.

Definicoes de T1, T2 e T3

T1: Sempre LIMIT IOC (executa ou cancela imediatamente).

T2/T3: Geralmente MARKET, mas podem ser LIMIT, dependendo da estrategia.

Proposito: T1 inicia a arbitragem com preco controlado; T2/T3 completam o ciclo, disparadas juntas apos T1.

Consideracoes Finais

Testes:

O Worker de Conexoes foi validado: conecta, autentica e escuta ordens enviadas pela plataforma web em Bybit e OKX.

Logs foram ajustados para mostrar apenas eventos principais e ordens ouvidas.

Evolucao:

Proximo passo: Implementar o Worker de Trades para usar o Worker de Conexoes como proxy via MessageChannel.

Definir formato exato das mensagens do MessageChannel com base nos logs completos.

Adicoes e Atualizacoes do Chat

Matriz de Operacoes - Particularidades

Exchanges como Bybit usam type (ex.: limit, market) e timeInForce (ex.: IOC, GTC) separadamente.

Na OKX, o campo ordType substitui essa combinacao:

LIMIT normal: ordType: 'limit'.

LIMIT IOC (como T1): ordType: 'ioc', ignorando type: 'limit' da matriz.

T2 e T3 sao geralmente market orders, entao timeInForce nao se aplica; o sistema apenas loga o resultado da postagem (OK ou erro).

Worker de Trades - Detalhamento

O Worker de Trades gerencia a execucao de T1, T2 e T3:

Para testes: Recebe a matriz operacoes no inicio e dispara ordens apos confirmar que os WebSockets do Worker de Conexoes estao prontos.

Em producao: Fica ocioso ate receber { type: 'start', operacoes } do main via parentPort.

Fluxo:

1. Posta T1 via portToTrades ao Worker de Conexoes.

2. Aguarda T1 ser 'Filled' (timeout de 3s); se nao preenchida, para e loga aviso com mensagem completa.

3. Apos T1 'Filled', posta T2 e T3 simultaneamente.

4. Loga resultado da postagem de T2 e T3 (OK ou erro); se falharem, avisa via log sem cancelar.

Escalabilidade: Estrutura modular para suportar ate 7+ exchanges, usando matriz operacoes e configuracoes genericas.

MessageChannel - Comunicacao entre Workers

O Worker de Conexoes cria um MessageChannel com portToTrades (para o Worker de Trades) e portToMain (para o main).

O Worker de Trades envia ordens ao Worker de Conexoes via portToTrades (ex.: { type: 'postOrder', exchange, order }).

O Worker de Conexoes retorna status via portToTrades (ex.: { type: 'orderStatus', exchange, orderId, status }).

Formato exato sera definido com base nos logs completos de todas as mensagens recebidas.

Todas as mensagens trocadas sao logadas integralmente com tempo transcorrido desde o envio de T1.

Tempos Transcorridos

O envio de T1 marca o inicio da arbitragem (t=0).

Cada mensagem recebida (WebSocket ou MessageChannel) e logada com o tempo transcorrido desde t=0 (ex.: [t=1500ms] Mensagem recebida).

Isso ajuda a monitorar a latencia e o progresso da arbitragem.

Validacao e Tratamento de Falhas

T1: Se nao for 'Filled' em 3s, a arbitragem para e um aviso e logado com a mensagem completa.

T2 e T3: Apos T1 'Filled', sao disparadas juntas. Validade e baseada na aceitacao pelo servidor (postagem OK), nao no preenchimento. Se falharem, o sistema loga a mensagem completa, mas nao cancela.

Norma: Toda mensagem recebida e impressa na integra com timestamp (ex.: [t=2500ms] { type: 'orderStatus', ... }).

Notas Finais

Esta documentacao reflete todas as decisoes do chat ate 25/03/2025.

Serve como base para o desenvolvimento do Worker de Trades e ajustes no Worker de Conexoes.

Proximo passo: Implementar o Worker de Trades com a logica definida.

Observacao sobre orderId

O orderId no Worker de Trades e gerado localmente como um identificador temporario (ex.: T1\_timestamp) para rastreamento interno durante testes. Em producao, o orderId oficial sera fornecido pela exchange e retornado pelo Worker de Conexoes na mensagem de status.

Dinamica de Postagem e Status das Ordens

As exchanges OKX e Bybit utilizam dois canais WebSocket distintos para gerenciar ordens: o canal de postagem (Trade) e o canal de ordens (Orders).

Canal de Postagem (Trade): Responde se a ordem foi aceita ou rejeitada tecnicamente pelo servidor. Aceitacao significa que a ordem foi recebida sem erros tecnicos (ex.: sintaxe invalida, campos obrigatorios ausentes). Se aceita, retorna o orderId oficial da exchange. Se rejeitada, indica o motivo do erro.

Canal de Ordens (Orders): Reporta atualizacoes de estado da ordem apos a aceitacao (ex.: new, live, filled, cancelled) e detalhes adicionais, como motivo de cancelamento. Este canal envia mensagens sempre que o estado da ordem muda.

Fluxo para T1:

1. O Worker de Trades posta T1 via Worker de Conexoes.

2. O canal de postagem confirma aceitacao ou rejeicao. Se rejeitada, a arbitragem para e loga o erro.

3. Se aceita, o canal de ordens e monitorado para verificar o estado. T1 deve atingir state: 'filled' (OKX) ou orderStatus: 'Filled' (Bybit) para prosseguir. Se atingir 'cancelled'/'Cancelled', a arbitragem para.

4. Timeout de 3s e aplicado desde a postagem de T1. Se nao houver 'filled'/'Filled' em 3s, a arbitragem para e loga a mensagem completa.

Fluxo para T2 e T3:

1. Apos T1 atingir 'filled'/'Filled', T2 e T3 sao postadas simultaneamente.

2. O canal de postagem confirma aceitacao ou rejeicao. Aceitacao significa que a ordem foi enviada com sucesso ao servidor.

3. Como T2 e T3 sao geralmente ordens de mercado, a aceitacao e suficiente para considerar a postagem valida, mas o estado final (ex.: filled) deve ser registrado para logs futuros.

Campos a testar:

OKX: state (filled ou cancelled) no canal orders; code e msg no canal de postagem para aceitacao/rejeicao.

Bybit: orderStatus (Filled ou Cancelled) no canal order; retCode e retMsg no canal trade para aceitacao/rejeicao.

Motivos de cancelamento: OKX usa cancelSource (ex.: '14' para preco inexistente em IOC); Bybit usa rejectReason (ex.: 'EC\_NoImmediateQtyToFill').

Exemplos de Mensagens

OKX - Canal de Postagem:

{

id: 'T1Buy14h12m24s981',

op: 'order',

code: '0',

msg: '',

data: [

{

tag: '',

ts: '1743095545544',

ordId: '2368712901851389952',

clOrdId: '',

sCode: '0',

sMsg: 'Order successfully placed.'

}

],

inTime: '1743095545543725',

outTime: '1743095545545624'

}

OKX - Canal Orders (live):

{

arg: { channel: 'orders', instType: 'SPOT', uid: '670299194443692254' },

data: [

{

instType: 'SPOT',

instId: 'BTC-USDT',

tgtCcy: '',

ccy: '',

ordId: '2368712901851389952',

clOrdId: '',

algoClOrdId: '',

algoId: '',

tag: '',

px: '85300',

sz: '0.0001',

notionalUsd: '8.528891100000001',

ordType: 'ioc',

side: 'buy',

posSide: '',

tdMode: 'cash',

accFillSz: '0',

fillNotionalUsd: '',

avgPx: '0',

state: 'live',

lever: '0',

pnl: '0',

feeCcy: 'BTC',

fee: '0',

rebateCcy: 'USDT',

rebate: '0',

category: 'normal',

uTime: '1743095545544',

cTime: '1743095545544',

source: '',

reduceOnly: 'false',

cancelSource: '',

quickMgnType: '',

stpId: '',

stpMode: 'cancel\_maker',

attachAlgoClOrdId: '',

lastPx: '86981',

isTpLimit: 'false',

slTriggerPx: '',

slTriggerPxType: '',

tpOrdPx: '',

tpTriggerPx: '',

tpTriggerPxType: '',

slOrdPx: '',

fillPx: '',

tradeId: '',

fillSz: '0',

fillTime: '',

fillPnl: '0',

fillFee: '0',

fillFeeCcy: '',

execType: '',

fillPxVol: '',

fillPxUsd: '',

fillMarkVol: '',

fillFwdPx: '',

fillMarkPx: '',

fillIdxPx: '',

amendSource: '',

reqId: '',

amendResult: '',

code: '0',

msg: '',

pxType: '',

pxUsd: '',

pxVol: '',

linkedAlgoOrd: [],

attachAlgoOrds: []

}

]

}

OKX - Canal Orders (cancelled):

{

arg: { channel: 'orders', instType: 'SPOT', uid: '670299194443692254' },

data: [

{

instType: 'SPOT',

instId: 'BTC-USDT',

tgtCcy: '',

ccy: '',

ordId: '2368712901851389952',

clOrdId: '',

algoClOrdId: '',

algoId: '',

tag: '',

px: '85300',

sz: '0.0001',

notionalUsd: '8.528891100000001',

ordType: 'ioc',

side: 'buy',

posSide: '',

tdMode: 'cash',

accFillSz: '0',

fillNotionalUsd: '',

avgPx: '0',

state: 'canceled',

lever: '0',

pnl: '0',

feeCcy: 'BTC',

fee: '0',

rebateCcy: 'USDT',

rebate: '0',

category: 'normal',

uTime: '1743095545546',

cTime: '1743095545544',

source: '',

reduceOnly: 'false',

cancelSource: '14',

quickMgnType: '',

stpId: '',

stpMode: 'cancel\_maker',

attachAlgoClOrdId: '',

lastPx: '86981',

isTpLimit: 'false',

slTriggerPx: '',

slTriggerPxType: '',

tpOrdPx: '',

tpTriggerPx: '',

tpTriggerPxType: '',

slOrdPx: '',

fillPx: '',

tradeId: '',

fillSz: '0',

fillTime: '',

fillPnl: '0',

fillFee: '0',

fillFeeCcy: '',

execType: '',

fillPxVol: '',

fillPxUsd: '',

fillMarkVol: '',

fillFwdPx: '',

fillMarkPx: '',

fillIdxPx: '',

amendSource: '',

reqId: '',

amendResult: '',

code: '0',

msg: '',

pxType: '',

pxUsd: '',

pxVol: '',

linkedAlgoOrd: [],

attachAlgoOrds: []

}

]

}

Bybit - Canal Trade/Postagem:

{

retCode: 0,

retMsg: 'OK',

op: 'order.create',

data: {

orderId: '1915688408441948416',

orderLinkId: '1915688408441948417'

},

retExtInfo: {},

header: {

'X-Bapi-Limit': '20',

'X-Bapi-Limit-Status': '19',

'X-Bapi-Limit-Reset-Timestamp': '1743103854170',

Traceid: '4f6c365c56631832493a2d7f95d4056f',

Timenow: '1743103854171'

},

connId: 'curd3r9e8n8lp963d030-5ysxb'

}

Bybit - Canal Order:

{

topic: 'order',

id: '76027776\_22009\_68256961471',

creationTime: 1743104139484,

data: [

{

category: 'spot',

symbol: 'BTCUSDT',

orderId: '1915690801795696896',

orderLinkId: '1915690801795696897',

blockTradeId: '',

side: 'Buy',

positionIdx: 0,

orderStatus: 'Cancelled',

cancelType: 'UNKNOWN',

rejectReason: 'EC\_NoImmediateQtyToFill',

timeInForce: 'IOC',

isLeverage: '1',

price: '86000.0',

qty: '0.000100',

avgPrice: '',

leavesQty: '0.0001',

leavesValue: '8.6000000',

cumExecQty: '0',

cumExecValue: '0.0000000',

cumExecFee: '0',

orderType: 'Limit',

stopOrderType: '',

orderIv: '',

triggerPrice: '0.0',

takeProfit: '0.0',

stopLoss: '0.0',

triggerBy: '',

tpTriggerBy: '',

slTriggerBy: '',

triggerDirection: 0,

placeType: '',

lastPriceOnCreated: '87379.6',

closeOnTrigger: false,

reduceOnly: false,

smpGroup: 0,

smpType: 'None',

smpOrderId: '',

slLimitPrice: '0.0',

tpLimitPrice: '0.0',

marketUnit: '',

createdTime: '1743104139481',

updatedTime: '1743104139483',

feeCurrency: '',

slippageTolerance: '',

slippageToleranceType: 'UNKNOWN'

}

]

}

MessageChannel - Formato das Mensagens

O Worker de Trades e o Worker de Conexoes se comunicam via MessageChannel com dois tipos de mensagens: enviadas (ordens) e recebidas (status).

Mensagens Enviadas (Worker de Trades para Worker de Conexoes):

Contem os parametros da ordem baseados na matriz operacoes. Os campos minimos sao: exchange, par, side, type, amount, price, timeInForce.

Pendente de definicao: o cabecalho das mensagens (ex.: op para OKX, header com timestamp para Bybit) pode ser montado pelo Worker de Trades ou pelo Worker de Conexoes. Sugestao: o Worker de Trades envia apenas os parametros brutos, e o Worker de Conexoes formata conforme a API de cada exchange.

Exemplo para OKX (se montado no Worker de Trades):

{

id: 'T1Buy',

op: 'order',

args: [{

instId: 'BTC-USDT',

side: 'buy',

ordType: 'ioc',

sz: '0.0001',

px: '85300',

tdMode: 'cash'

}]

}

Exemplo para Bybit (se montado no Worker de Trades):

{

header: { "X-BAPI-TIMESTAMP": "1743103854170", "X-BAPI-RECV-WINDOW": "2000" },

op: 'order.create',

args: [{

symbol: 'BTCUSDT',

isLeverage: '1',

side: 'Buy',

orderType: 'Limit',

qty: '0.0001',

price: '86000',

timeInForce: 'IOC',

category: 'spot',

timestamp: '1743103854170'

}]

}

Mensagens Recebidas (Worker de Conexoes para Worker de Trades):

Formato simplificado que o Worker de Trades espera: { type: 'orderStatus', exchange, orderId, status, orderTag }. O Worker de Conexoes traduz as respostas brutas das exchanges para esse formato.

Mapeamento:

OKX - code: '0' (aceita) ou outro (rejeitada) no canal de postagem; state: 'filled' ou 'cancelled' no canal orders.

Bybit - retCode: 0 (aceita) ou outro (rejeitada) no canal trade; orderStatus: 'Filled' ou 'Cancelled' no canal order.

Registro de Resultados

Os resultados de T2 e T3 (preco executado, estado final) devem ser armazenados para logs futuros de arbitragens, pois, como ordens de mercado, os precos finais podem diferir dos detectados inicialmente. Campos a guardar: avgPx (OKX), avgPrice (Bybit), fillSz/cumExecQty, e timestamp de execucao. Por enquanto, esses dados devem ser logados pelo Worker de Trades.

Notas Finais Atualizadas

Esta documentacao reflete todas as decisoes do chat ate 28/03/2025.

Serve como base para o desenvolvimento do Worker de Trades, ajustes no Worker de Conexoes e continuidade em chats futuros.

Proximo passo: Implementar o Worker de Trades com a logica definida, validando o fluxo com simulacao antes de integrar com exchanges reais.

---

Administracao dos Formatos de Mensagens de Ordens das Exchanges

Objetivo

Administrar os diferentes formatos de mensagens de ordens das exchanges (inicialmente OKX e Bybit, expansivel para 6-8) de forma modular, organizada e escalavel, priorizando simplicidade inicial e garantindo clareza absoluta na distincao entre status internos e nativos.

Principios

Simplicidade Inicial: Focar em OKX e Bybit com simulacao para validar o conceito antes de integrar exchanges reais.

Modularidade: Separar logica generica (workerTrades.js) das especificidades das exchanges (workerConexoes.js) via adaptadores.

Clareza nos Status: Definir status internos fixos e mapear explicitamente os status nativos das exchanges para eles, com documentacao e codigo refletindo essa separacao.

1. Estrutura Modular no workerConexoes.js

Matriz de Adaptadores (exchanges):

Cada exchange (OKX, Bybit, etc.) tem um objeto com:

- Configuracoes: URLs WebSocket, chaves API (mantidas da documentacao original).

- Formatador de Ordem: Funcao que transforma os parametros genericos da matriz operacoes no formato especifico da exchange, incluindo cabecalho (ex.: op para OKX, header com timestamp para Bybit).

- Parser de Status: Funcao que traduz status nativos dos canais Trade e Orders em status internos padronizados.

Separação de Canais:

- Canal Trade: Lida com aceitacao/rejeicao tecnica da postagem.

- Canal Orders: Lida com o estado da ordem apos aceitacao (ex.: preenchimento, cancelamento).

O parser diferencia os status nativos por canal e mapeia para os internos.

Exemplo de Mapeamento:

- OKX:

- Canal Trade: code: '0' mapeado para accepted; code: '51000' (erro) mapeado para rejected.

- Canal Orders: state: 'live' mapeado para accepted (ordem ativa); state: 'filled' mapeado para filled; state: 'canceled' mapeado para cancelled.

- Bybit:

- Canal Trade: retCode: 0 mapeado para accepted; retCode: 10001 (erro) mapeado para rejected.

- Canal Orders: orderStatus: 'New' mapeado para accepted; orderStatus: 'Filled' mapeado para filled; orderStatus: 'Cancelled' mapeado para cancelled.

Cabecalho: Montado pelo adaptador no workerConexoes.js, mantendo o workerTrades.js generico.

2. Formato Generico no workerTrades.js

Mensagens Enviadas:

- Formato: { type: 'postOrder', exchange: 'OKX', order: { par, side, type, amount, price, timeInForce }, orderTag: 'T1' }.

- Parametros sao os basicos da matriz operacoes. Parametros extras (ex.: leverage) serao tratados no adaptador do workerConexoes.js.

Mensagens Recebidas:

- Formato: { type: 'orderStatus', exchange, orderId, status, orderTag }.

- Status Internos Fixos:

- accepted: Ordem aceita tecnicamente (Trade) ou ativa (Orders).

- rejected: Ordem rejeitada por erro tecnico (Trade).

- filled: Ordem preenchida (Orders).

- cancelled: Ordem cancelada apos aceitacao (Orders).

- Regra: O workerTrades.js usa apenas esses status internos para decisoes, nunca os nativos diretamente.

3. Fluxo de Comunicacao

Postagem:

1. workerTrades.js envia ordem generica via portToTrades.

2. workerConexoes.js identifica a exchange, usa o adaptador para formatar a mensagem (com cabecalho) e simula o envio ao WebSocket.

Resposta:

1. workerConexoes.js (simulado) gera respostas ficticias para os canais Trade e Orders.

2. O parser mapeia os status nativos (ex.: state: 'filled', orderStatus: 'Cancelled') para os internos (filled, cancelled).

3. Envia ao workerTrades.js o status interno.

Decisoes no workerTrades.js:

- T1:

- Se status: 'rejected' (Trade), para e loga.

- Se status: 'accepted' (Trade), continua monitorando Orders.

- Se status: 'filled' (Orders), dispara T2 e T3.

- Se status: 'cancelled' (Orders) ou timeout de 3s, para e loga.

- T2 e T3:

- Verifica status: 'accepted' (Trade) como sucesso inicial.

- Loga o resultado (incluindo filled ou cancelled de Orders para analise futura).

4. Simulacao Inicial

Usar um workerConexoes.js simulado para OKX e Bybit:

- Simula respostas do canal Trade (ex.: accepted em 0,5s) e Orders (ex.: filled ou cancelled em 1s).

- Garante que o mapeamento de status nativos para internos seja testado antes de integrar WebSockets reais.

5. Escalabilidade Futura

Adicionar uma nova exchange exige apenas um novo adaptador em exchanges com mapeamento de status nativos para internos, preservando o fluxo do workerTrades.js.

6. Organizacao e Clareza

Documentacao: Detalhar os status internos e o mapeamento de cada exchange (ex.: OKX: state: 'live' mapeado para accepted).

Codigo: O parser no workerConexoes.js tera comentarios explicitos sobre o mapeamento (ex.: // OKX Trade: code '0' mapeado para 'accepted').

Logs: Incluir tanto o status interno quanto o nativo (ex.: [t=1000ms] T1 status: filled (nativo: state: 'filled')).