ANEXO: CÓDIGOS DE EXEMPLO E REFERÊNCIA ---

[Adicionado em 6/04/2025]

Mensagem Grok: Estamos fazendo um sistema de trading de criptoativos em node.js via API e websockets. Precisamos definir uma nomenclatura padrao para parametros de trade de diferentes exchanges. As exchanges sao: okx, bybit, binance, kucoin, bitpreco, novadax, foxit e mercadobitcoin Os parametros sao: par ou symbol: ex, na binance é BTCUSDT, na oxk BTC-USDT side: ex na okx = sell, na bybit = Sell type: na bybit = Market, na okx o campo nao existe timeInforce: ex: ioc x IOC na okx o campo nao existe, e quantidade. Me da a sintaxis correta desses campos para cada Exchange

Para criar um sistema de trading de criptoativos em Node.js utilizando APIs e WebSockets, é essencial padronizar a nomenclatura dos parâmetros de trade entre as diferentes exchanges, já que cada uma possui variações em seus formatos e requisitos. Vou detalhar a sintaxe correta dos parâmetros solicitados (symbol/par, side, type, e timeInForce) para cada uma das exchanges mencionadas: OKX, Bybit, Binance, KuCoin, BitPreço, NovaDAX, Foxbit e Mercado Bitcoin. As informações são baseadas nas documentações oficiais das APIs de cada exchange, conforme disponíveis até a data atual (6 de abril de 2025), e refletem os padrões mais comuns para ordens de mercado e limite.

**Resumo Comparativo**

| **Exchange** | **Symbol** | **Side** | **Type** | **Quantity** | **TimeInForce** |
| --- | --- | --- | --- | --- | --- |
| OKX | BTC-USDT | buy/sell | market/limit | sz | ioc/fok (via ordType) |
| Bybit | BTCUSDT | Buy/Sell | Market/Limit | qty | GTC/IOC/FOK |
| Binance | BTCUSDT | BUY/SELL | MARKET/LIMIT | quantity | GTC/IOC/FOK |
| KuCoin | BTC-USDT | buy/sell | market/limit | size | GTC/IOC/FOK |
| BitPreço | BTC-BRL | buy/sell | market/limit | amount | Não especificado (GTC) |
| NovaDAX | BTC\_BRL | BUY/SELL | MARKET/LIMIT | amount | Não especificado (GTC) |
| Foxbit | BTC\_BRL | buy/sell | market/limit | quantity | Não especificado (GTC) |
| Mercado Bitcoin | BTCBRL | buy/sell | market/limit | quantity | Não especificado (GTC) |

Detalhes:

**1. OKX**

* **Parâmetros**:
  + **symbol/par**: BTC-USDT (com hífen, maiúsculas).
  + **side**: buy ou sell (minúsculas).
  + **type**: Implícito no ordType: market ou limit.
  + **quantity**: Campo chamado sz (size), ex.: "0.01" (string, em unidades da moeda base, como BTC).
  + **timeInForce**: Via ordType: ioc, fok (ou omitido para GTC em ordens limite).
* **Exemplo**:
  + Mercado: { "instId": "BTC-USDT", "side": "sell", "ordType": "market", "sz": "0.01" }
  + Limite: { "instId": "BTC-USDT", "side": "buy", "ordType": "limit", "px": "30000", "sz": "0.01" }

**2. Bybit**

* **Parâmetros**:
  + **symbol/par**: BTCUSDT (sem hífen, maiúsculas).
  + **side**: Buy ou Sell (capitalizado).
  + **type**: Market ou Limit (capitalizado).
  + **quantity**: Campo chamado qty, ex.: "0.01" (string, em unidades da moeda base).
  + **timeInForce**: GTC, IOC, FOK.
* **Exemplo**:
  + Mercado: { "symbol": "BTCUSDT", "side": "Sell", "order\_type": "Market", "qty": "0.01" }
  + Limite: { "symbol": "BTCUSDT", "side": "Buy", "order\_type": "Limit", "price": "30000", "qty": "0.01", "time\_in\_force": "GTC" }

**3. Binance**

* **Parâmetros**:
  + **symbol/par**: BTCUSDT (sem hífen, maiúsculas).
  + **side**: BUY ou SELL (maiúsculas).
  + **type**: MARKET ou LIMIT (maiúsculas).
  + **quantity**: Campo chamado quantity, ex.: "0.01" (string, em unidades da moeda base).
  + **timeInForce**: GTC, IOC, FOK.
* **Exemplo**:
  + Mercado: { "symbol": "BTCUSDT", "side": "SELL", "type": "MARKET", "quantity": "0.01" }
  + Limite: { "symbol": "BTCUSDT", "side": "BUY", "type": "LIMIT", "price": "30000", "quantity": "0.01", "timeInForce": "GTC" }

**4. KuCoin**

* **Parâmetros**:
  + **symbol/par**: BTC-USDT (com hífen, maiúsculas).
  + **side**: buy ou sell (minúsculas).
  + **type**: market ou limit (minúsculas).
  + **quantity**: Campo chamado size, ex.: "0.01" (string, em unidades da moeda base).
  + **timeInForce**: GTC, IOC, FOK.
* **Exemplo**:
  + Mercado: { "symbol": "BTC-USDT", "side": "sell", "type": "market", "size": "0.01" }
  + Limite: { "symbol": "BTC-USDT", "side": "buy", "type": "limit", "price": "30000", "size": "0.01", "timeInForce": "GTC" }

**5. BitPreço**

* **Parâmetros**:
  + **symbol/par**: BTC-BRL (com hífen, maiúsculas).
  + **side**: buy ou sell (minúsculas).
  + **type**: market ou limit (minúsculas).
  + **quantity**: Campo chamado amount, ex.: "0.01" (string, em unidades da moeda base).
  + **timeInForce**: Não especificado (assume-se GTC para ordens limite).
* **Exemplo**:
  + Mercado: { "pair": "BTC-BRL", "side": "sell", "type": "market", "amount": "0.01" }
  + Limite: { "pair": "BTC-BRL", "side": "buy", "type": "limit", "price": "150000", "amount": "0.01" }

**6. NovaDAX**

* **Parâmetros**:
  + **symbol/par**: BTC\_BRL (com underline, maiúsculas).
  + **side**: BUY ou SELL (maiúsculas).
  + **type**: MARKET ou LIMIT (maiúsculas).
  + **quantity**: Campo chamado amount, ex.: "0.01" (string, em unidades da moeda base).
  + **timeInForce**: Não especificado (assume-se GTC para ordens limite).
* **Exemplo**:
  + Mercado: { "symbol": "BTC\_BRL", "side": "SELL", "type": "MARKET", "amount": "0.01" }
  + Limite: { "symbol": "BTC\_BRL", "side": "BUY", "type": "LIMIT", "price": "150000", "amount": "0.01" }

**7. Foxbit**

* **Parâmetros**:
  + **symbol/par**: BTC\_BRL (com underline, maiúsculas).
  + **side**: buy ou sell (minúsculas).
  + **type**: market ou limit (minúsculas).
  + **quantity**: Campo chamado quantity, ex.: "0.01" (string, em unidades da moeda base).
  + **timeInForce**: Não especificado (assume-se GTC para ordens limite).
* **Exemplo**:
  + Mercado: { "symbol": "BTC\_BRL", "side": "sell", "type": "market", "quantity": "0.01" }
  + Limite: { "symbol": "BTC\_BRL", "side": "buy", "type": "limitA", "price": "150000", "quantity": "0.01" }

**8. Mercado Bitcoin**

* **Parâmetros**:
  + **symbol/par**: BTCBRL (sem separador, maiúsculas).
  + **side**: buy ou sell (minúsculas).
  + **type**: market ou limit (minúsculas).
  + **quantity**: Campo chamado quantity, ex.: "0.01" (string, em unidades da moeda base).
  + **timeInForce**: Não especificado (assume-se GTC para ordens limite).
* **Exemplo**:
  + Mercado: { "symbol": "BTCBRL", "side": "sell", "type": "market", "quantity": "0.01" }
  + Limite: { "symbol": "BTCBRL", "side": "buy", "type": "limit", "price": "150000", "quantity": "0.01" }

[Adicionado em 30/03/2025]

Esta seção contém códigos de exemplo e modelos fornecidos como apoio ao desenvolvimento do sistema de arbitragem triangular. Diferentemente dos códigos vigentes (mainTester.js, workerConexoes.js, workerTrades.js), que representam o estado atual do sistema, os códigos aqui servem como referências práticas para entender as APIs das exchanges e validar implementações no workerConexoes.js e workerTrades.js. Incluem mensagens explicativas e printouts de execução.

**1. Bybit - testerbybt1t2.js**

* Descrição: Código funcional que conecta à Bybit, autentica dois WebSockets (tradeWs para envio, orderWs para status), posta T1 (LIMIT IOC) e dispara T2 (MARKET) se T1 for preenchida.
* Mensagem Associada:

Dois WebSockets:

tradeWs (wss://stream.bybit.com/v5/trade): Usado para postar ordens (T1, T2, etc.).

orderWs (wss://stream.bybit.com/v5/private): Usado para receber atualizações de status das ordens.

Autenticação:

Ambos os WebSockets se autenticam com apiKey, expires (timestamp + 10s) e uma assinatura HMAC-SHA256 baseada em GET/realtime${expires} e apiSecret.

Após a autenticação no tradeWs, ele posta T1. No orderWs, inscreve no tópico order para atualizações.

Postagem de Ordens:

postOrderT1(): Ordem LIMIT, IOC, exemplo: BUY BTCUSDT, qty 0.0001, preço 83000.

Enviada via tradeWs com op: 'order.create'.

postOrderT2(): Ordem MARKET, exemplo: SELL BTCUSDT, qty 0.0001, sem preço (market).

Só é chamada se T1 for preenchida (Filled).

Fluxo de Controle:

orderWs monitora o status da ordem via topic: 'order'.

Se orderStatus === 'Filled' para o orderId de T1, dispara T2.

Se não preencher (ex.: cancelada), loga o status e não prossegue.

Reconexão: Ambos os WebSockets tentam reconectar após 5s se fecharem.

Logs: Mostra tempos relativos (new Date() - horaInicial) e detalhes das mensagens/respostas.

17

* Código:

const WebSocket = require('ws');

const crypto = require('crypto');

*// Configurações*

const bybitTradeWsUrl = 'wss://stream.bybit.com/v5/trade';

const bybitOrderWsUrl = 'wss://stream.bybit.com/v5/private';

const apiKey = 'b6uS8UAyMoRPImNnAU';

const apiSecret = '47gHJE2gGonqxJlXz8SxOqqLhTDABAbJdaYB';

let tradeWs;

let orderWs;

let orderIdT1; *// ID da ordem T1*

let orderIdT2; *// ID da ordem T2*

let hasPostedT2 = false; *// Controle para evitar múltiplas postagens de T2*

let orderTimeout; *// Timeout para aguardar atualização da ordem*

const horaInicial = new Date();

const TIMEOUT\_MS = 5000; *// 5 segundos de espera máxima para atualização da ordem*

*// Função para gerar a assinatura*

function generateSignature(expires) {

const signatureString = `GET/realtime${expires}`;

return crypto.createHmac('sha256', apiSecret).update(signatureString).digest('hex');

}

*// Função para conectar ao WebSocket de trades*

function connectTradeWebSocket() {

tradeWs = new WebSocket(bybitTradeWsUrl);

tradeWs.on('open', () => {

console.log('[Bybit Trade] Conectado ao WebSocket de trades.');

const expires = (Date.now() + 10000).toString();

const signature = generateSignature(expires);

tradeWs.send(JSON.stringify({

op: 'auth',

args: [apiKey, expires, signature],

}));

});

tradeWs.on('message', (data) => {

const message = JSON.parse(data);

console.log('[Bybit Trade] Mensagem recebida:', message.retMsg, 'Tempo:', new Date() - horaInicial);

if (message.op === 'auth' && message.retCode === 0) {

console.log('[Bybit Trades] Autenticado no WebSocket de trades.Tempo:', new Date() - horaInicial, Date.now().toString());

console.log('[Debug] Estado do WebSocket de ordens antes de postar T1:', orderWs.readyState === WebSocket.OPEN ? 'Aberto' : 'Fechado ou conectando');

postOrderT1();

}

if (message.op === 'order.create') {

if (message.retCode === 0) {

console.log('-------------->>> mensage completo',message);

console.log('[Bybit Trade] Ordem postada com sucesso. OrderId:', message.data.orderId,

'Tempo:', new Date() - horaInicial, Date.now().toString());

if (!orderIdT1) {

orderIdT1 = message.data.orderId; *// Salva o ID de T1*

orderTimeout = setTimeout(() => {

console.log('[Bybit Order] Timeout: Nenhuma atualização recebida para a ordem T1:', orderIdT1, 'Tempo:', new Date() - horaInicial);

process.exit(1);

}, TIMEOUT\_MS);

} else if (hasPostedT2 && !orderIdT2) {

orderIdT2 = message.data.orderId; *// Salva o ID de T2*

}

} else {

console.error('[Bybit Trade] Erro ao postar a ordem:', message);

}

}

});

tradeWs.on('error', (err) => {

console.error('[Bybit Trade] Erro no WebSocket:', err);

});

tradeWs.on('close', () => {

console.log('[Bybit Trade] WebSocket desconectado. Tentando reconectar...');

setTimeout(connectTradeWebSocket, 5000);

});

}

*// Função para conectar ao WebSocket de ordens*

function connectOrderWebSocket() {

orderWs = new WebSocket(bybitOrderWsUrl);

orderWs.on('open', () => {

console.log('[Bybit Order] Conectado ao WebSocket de ordens.', 'Tempo:', new Date() - horaInicial);

const expires = (Date.now() + 10000).toString();

const signature = generateSignature(expires);

orderWs.send(JSON.stringify({

op: 'auth',

args: [apiKey, expires, signature],

}));

});

orderWs.on('message', (data) => {

const message = JSON.parse(data);

console.log('[Bybit Order] Mensagem recebida:', message.success || message.retMsg || message.topic || 'Desconhecido', message.op || '', 'Tempo:', new Date() - horaInicial, Date.now().toString());

if (message.op === 'auth' && message.success) {

console.log('[Bybit Order] Autenticado no WebSocket de ordens. Tempo:', new Date() - horaInicial, Date.now().toString());

orderWs.send(JSON.stringify({

op: 'subscribe',

args: ['order'],

}));

}

if (message.op === 'subscribe' && message.success) {

console.log('[Bybit Order] Inscrito no tópico de ordens.');

}

if (message.topic === 'order') {

const orderData = message.data[0];

console.log('===================================> message completo', message);

console.log('[Debug] Atualização recebida no canal de ordens:', orderData.orderId, 'Status:', orderData.orderStatus, 'Tempo:', new Date() - horaInicial);

if (orderData.orderId === orderIdT1) {

console.log('[Bybit Order] Atualização da ordem T1:', orderData, 'Tempo:', new Date() - horaInicial, Date.now().toString());

clearTimeout(orderTimeout);

if (orderData.orderStatus === 'Filled') {

console.log('[Bybit Order] Ordem T1 preenchida com sucesso!', 'Tempo:', new Date() - horaInicial, Date.now().toString());

if (!hasPostedT2) {

console.log('---------------------------------- COMEÇA POSTAGEM T2 ----------------------------');

postOrderT2();

hasPostedT2 = true;

}

} else if (orderData.orderStatus === 'Cancelled') {

console.log('[Bybit Order] Ordem T1 cancelada! Motivo:', orderData.rejectReason, 'Tempo:', new Date() - horaInicial, Date.now().toString());

process.exit(0);

} else {

console.log('[Bybit Order] Status da ordem T1:', orderData.orderStatus, 'Tempo:', new Date() - horaInicial, Date.now().toString());

}

} else if (orderData.orderId === orderIdT2) {

console.log('[Bybit Order] Atualização da ordem T2:', orderData, 'Tempo:', new Date() - horaInicial, Date.now().toString());

console.log('[Bybit Order] Status da ordem T2:', orderData.orderStatus, 'Tempo:', new Date() - horaInicial, Date.now().toString());

} else {

console.log('[Debug] Atualização de ordem não rastreada. OrderId recebido:', orderData.orderId, 'T1 esperado:', orderIdT1, 'T2 esperado:', orderIdT2);

}

}

});

orderWs.on('error', (err) => {

console.error('[Bybit Order] Erro no WebSocket:', err, 'Tempo:', new Date() - horaInicial);

});

orderWs.on('close', () => {

console.log('[Bybit Order] WebSocket desconectado. Tentando reconectar... Tempo:', new Date() - horaInicial);

setTimeout(connectOrderWebSocket, 5000);

});

}

*// Função para postar T1*

function postOrderT1() {

const timestamp = Date.now().toString();

const order = {

category: 'spot',

symbol: 'BTCUSDT',

isLeverage: '1',

side: 'Buy',

orderType: 'Limit',

qty: '0.0001',

price: '83000',

timeInForce: 'IOC',

timestamp: timestamp,

};

console.log('[Bybit Order] Postando ordem T1:', order, 'Tempo:', new Date() - horaInicial, Date.now().toString());

tradeWs.send(

JSON.stringify({

header: {

"X-BAPI-TIMESTAMP": timestamp,

"X-BAPI-RECV-WINDOW": "2000",

},

op: 'order.create',

args: [order],

})

);

}

*// Função para postar T2*

function postOrderT2() {

const timestamp = Date.now().toString();

const order = {

symbol: 'BTCUSDT',

side: 'Sell',

orderType: 'Market',

marketUnit: 'baseCoin',

qty: '0.0001',

price: '88000',

category: 'spot',

isLeverage: 1,

timestamp: timestamp,

};

console.log('[Bybit Order] Postando ordem T2:', order, 'Tempo:', new Date() - horaInicial, Date.now().toString());

tradeWs.send(

JSON.stringify({

header: {

"X-BAPI-TIMESTAMP": timestamp,

"X-BAPI-RECV-WINDOW": "2000",

},

op: 'order.create',

args: [order],

})

);

}

*// Iniciar as conexões com os WebSockets*

connectOrderWebSocket();

setTimeout(() => {

connectTradeWebSocket();

}, 3000);

210

* Printout de Execução:

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\coordenador>node testerbybt1t2

[Bybit Order] Conectado ao WebSocket de ordens. Tempo: 1101

[Bybit Order] Mensagem recebida: true auth Tempo: 1443 1743189813262

[Bybit Order] Autenticado no WebSocket de ordens. Tempo: 1443 1743189813262

[Bybit Order] Mensagem recebida: true subscribe Tempo: 1774 1743189813593

[Bybit Order] Inscrito no tópico de ordens.

[Bybit Trade] Conectado ao WebSocket de trades.

[Bybit Trade] Mensagem recebida: OK Tempo: 4409

[Bybit Trades] Autenticado no WebSocket de trades.Tempo: 4410 1743189816229

[Debug] Estado do WebSocket de ordens antes de postar T1: Aberto

[Bybit Order] Postando ordem T1: {

category: 'spot',

symbol: 'BTCUSDT',

isLeverage: '1',

side: 'Buy',

orderType: 'Limit',

qty: '0.0001',

price: '83000',

timeInForce: 'IOC',

timestamp: '1743189816230'

} Tempo: 4411 1743189816230

[Bybit Trade] Mensagem recebida: OK Tempo: 4762

-------------->>> mensage completo {

retCode: 0,

retMsg: 'OK',

op: 'order.create',

data: {

orderId: '1916409516187191552',

orderLinkId: '1916409516187191553'

},

retExtInfo: {},

header: {

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

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

'X-Bapi-Limit-Reset-Timestamp': '1743189816912',

Traceid: '3387eccf39f1fafb3d342fc81f99fe4b',

Timenow: '1743189816913'

},

connId: 'cuulsa46g6hfrgrgnvj0-58nu4'

}

[Bybit Trade] Ordem postada com sucesso. OrderId: 1916409516187191552 Tempo: 4765 1743189816584

[Bybit Order] Mensagem recebida: order Tempo: 4767 1743189816586

===================================> message completo {

topic: 'order',

id: '76027776\_22009\_68405198872',

creationTime: 1743189816915,

data: [

{

category: 'spot',

symbol: 'BTCUSDT',

orderId: '1916409516187191552',

orderLinkId: '1916409516187191553',

blockTradeId: '',

side: 'Buy',

positionIdx: 0,

orderStatus: 'Cancelled',

cancelType: 'UNKNOWN',

rejectReason: 'EC\_NoImmediateQtyToFill',

timeInForce: 'IOC',

isLeverage: '1',

price: '83000.0',

qty: '0.000100',

avgPrice: '',

leavesQty: '0.0001',

leavesValue: '8.3000000',

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: '83692.9',

closeOnTrigger: false,

reduceOnly: false,

smpGroup: 0,

smpType: 'None',

smpOrderId: '',

slLimitPrice: '0.0',

tpLimitPrice: '0.0',

marketUnit: '',

createdTime: '1743189816912',

updatedTime: '1743189816914',

feeCurrency: '',

slippageTolerance: '',

slippageToleranceType: 'UNKNOWN'

}

]

}

[Debug] Atualização recebida no canal de ordens: 1916409516187191552 Status: Cancelled Tempo: 4772

[Bybit Order] Atualização da ordem T1: {

category: 'spot',

symbol: 'BTCUSDT',

orderId: '1916409516187191552',

orderLinkId: '1916409516187191553',

blockTradeId: '',

side: 'Buy',

positionIdx: 0,

orderStatus: 'Cancelled',

cancelType: 'UNKNOWN',

rejectReason: 'EC\_NoImmediateQtyToFill',

timeInForce: 'IOC',

isLeverage: '1',

price: '83000.0',

qty: '0.000100',

avgPrice: '',

leavesQty: '0.0001',

leavesValue: '8.3000000',

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: '83692.9',

closeOnTrigger: false,

reduceOnly: false,

smpGroup: 0,

smpType: 'None',

smpOrderId: '',

slLimitPrice: '0.0',

tpLimitPrice: '0.0',

marketUnit: '',

createdTime: '1743189816912',

updatedTime: '1743189816914',

feeCurrency: '',

slippageTolerance: '',

slippageToleranceType: 'UNKNOWN'

} Tempo: 4773 1743189816592

[Bybit Order] Ordem T1 cancelada! Motivo: EC\_NoImmediateQtyToFill Tempo: 4776 1743189816595

145

**2. OKX - conectaOkxTradesePostaT.js**

Descrição: Código funcional que conecta à OKX via WebSocket único, autentica, subscreve o canal orders, e posta uma ordem T1 (LIMIT IOC), recebendo atualizações no mesmo fluxo.

* Mensagem Associada:

WebSocket Único: wss://ws.okx.com:8443/ws/v5/private para envio de ordens e atualizações.

Canais:

Envio de ordens via op: "order".

Atualizações via canal orders (subscrito com op: "subscribe").

Autenticação: Usa apiKey, passphrase, timestamp, e assinatura HMAC-SHA256 com GET/users/self/verify.

Fluxo:

Conecta, autentica, subscreve o canal orders, e envia uma ordem (IOC, LIMIT, BUY BTC-USDT, 0.00001, $82300).

Recebe atualizações no mesmo WebSocket (state: live, filled, canceled).

Ping/Pong: Envia ping a cada 25s para manter a conexão viva.

Respostas:

Confirmação de envio com ordId.

Atualizações detalhadas no canal orders.

Reconexão: Tenta reconectar após 5s em caso de desconexão.

13

* Código:

const WebSocket = require('ws');

const crypto = require('crypto');

*// Configurações da OKX*

const okxTradeWsUrl = 'wss://ws.okx.com:8443/ws/v5/private';

const okxApiKey = 'ac0bc774-1bad-4da2-83f9-55b8eebb697d';

const okxApiSecret = '4AD9EBBD4A8EEB6526F31B9527545ADC';

const okxPassphrase = 'Aa@066466646';

*// Função para gerar a assinatura da OKX*

function generateOKXSignature(timestamp, method, path, body) {

const prehashString = timestamp + method + path + (body || '');

return crypto.createHmac('sha256', okxApiSecret).update(prehashString).digest('base64');

}

*// Função para gerar o ID único no formato T1Sellhhhmmsmilissegundos*

function gerarIdUnico() {

const agora = new Date();

const horas = String(agora.getHours()).padStart(2, '0');

const minutos = String(agora.getMinutes()).padStart(2, '0');

const segundos = String(agora.getSeconds()).padStart(2, '0');

const milissegundos = String(agora.getMilliseconds()).padStart(3, '0');

return `T1${horas}h${minutos}m${segundos}s${milissegundos}`;

}

*// Função para conectar ao WebSocket de trades da OKX*

function conectarWebSocketTradesOKX() {

const ws = new WebSocket(okxTradeWsUrl);

let pingInterval;

ws.on('open', () => {

console.log('[OKX] Conectado ao WebSocket de trades.');

*// Autenticar no WebSocket de trades*

const timestamp = (Date.now() / 1000).toString();

const sign = generateOKXSignature(timestamp, 'GET', '/users/self/verify', '');

ws.send(JSON.stringify({

op: 'login',

args: [{

apiKey: okxApiKey,

passphrase: okxPassphrase,

timestamp: timestamp,

sign: sign,

}],

}));

*// Iniciar o intervalo de ping (a cada 25 segundos)*

pingInterval = setInterval(() => {

if (ws.readyState === WebSocket.OPEN) {

ws.send(JSON.stringify({ op: 'ping' }));

console.log('[OKX] Ping enviado.');

}

}, 25000);

});

ws.on('message', (mensagem) => {

const data = JSON.parse(mensagem);

*// Processar autenticação*

if (data.event === 'login' && data.code === '0') {

console.log('[OKX] Autenticação bem-sucedida no WebSocket de trades.');

*// Assinar o canal de ordens privadas*

const subscribeOrderChannel = {

op: 'subscribe',

args: [{

channel: 'orders',

instType: 'SPOT',

instId: 'BTC-USDT'

}]

};

ws.send(JSON.stringify(subscribeOrderChannel));

console.log('[OKX] Assinatura do canal de ordens enviada:', subscribeOrderChannel);

*// Postar a operação após a autenticação*

const orderMessage = {

id: gerarIdUnico(),

op: 'order',

args: [{

instId: 'BTC-USDT',

side: 'buy',

ordType: 'ioc',

sz: '0.00001',

px: '82300',

tdMode: 'cash',

ccy: 'USDT',

}],

};

console.log('[OKX] Enviando ordem:', orderMessage);

ws.send(JSON.stringify(orderMessage));

}

*// Processar resposta de subscrição*

if (data.event === 'subscribe' && data.arg?.channel === 'orders') {

console.log('[OKX] Subscrição ao canal de ordens confirmada:', data);

}

*// Processar atualizações de ordens*

if (data.arg?.channel === 'orders' && data.data) {

console.log('[OKX] Atualização de ordem recebida:', data.data);

data.data.forEach(order => {

console.log(`[OKX] Ordem ${order.ordId}: Estado: ${order.state}, Preço: ${order.px}, Quantidade: ${order.sz}`);

});

}

*// Processar pong (resposta ao ping)*

if (data.event === 'pong') {

console.log('[OKX] Pong recebido:', data);

}

*// Processar outras mensagens (debug)*

console.log('[OKX] Mensagem recebida:', data);

});

ws.on('close', () => {

console.log('[OKX] WebSocket de trades desconectado. Tentando reconectar em 5 segundos...');

clearInterval(pingInterval);

setTimeout(conectarWebSocketTradesOKX, 5000);

});

ws.on('error', (erro) => {

console.error('[OKX] Erro no WebSocket de trades:', erro);

clearInterval(pingInterval);

});

}

*// Iniciar a conexão*

conectarWebSocketTradesOKX();

128

* Printout de Execução:

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\coordenador>node conectaokxtradesepostat

[OKX] Conectado ao WebSocket de trades.

[OKX] Autenticação bem-sucedida no WebSocket de trades.

[OKX] Assinatura do canal de ordens enviada: {

op: 'subscribe',

args: [ { channel: 'orders', instType: 'SPOT', instId: 'BTC-USDT' } ]

}

[OKX] Enviando ordem: {

id: 'T116h21m06s386',

op: 'order',

args: [

{

instId: 'BTC-USDT',

side: 'buy',

ordType: 'ioc',

sz: '0.00001',

px: '82300',

tdMode: 'cash',

ccy: 'USDT'

}

]

}

[OKX] Mensagem recebida: { event: 'login', msg: '', code: '0', connId: '2241d76c' }

[OKX] Subscrição ao canal de ordens confirmada: {

event: 'subscribe',

arg: { channel: 'orders', instType: 'SPOT', instId: 'BTC-USDT' },

connId: '2241d76c'

}

[OKX] Mensagem recebida: {

event: 'subscribe',

arg: { channel: 'orders', instType: 'SPOT', instId: 'BTC-USDT' },

connId: '2241d76c'

}

[OKX] Mensagem recebida: {

event: 'channel-conn-count',

channel: 'orders',

connCount: '1',

connId: '2241d76c'

}

[OKX] Mensagem recebida: {

id: 'T116h21m06s386',

op: 'order',

code: '0',

msg: '',

data: [

{

tag: '',

ts: '1743189667090',

ordId: '2371871096866381824',

clOrdId: '',

sCode: '0',

sMsg: 'Order successfully placed.'

}

],

inTime: '1743189667086783',

outTime: '1743189667091719'

}

[OKX] Atualização de ordem recebida: [

{

instType: 'SPOT',

instId: 'BTC-USDT',

tgtCcy: '',

ccy: '',

ordId: '2371871096866381824',

clOrdId: '',

algoClOrdId: '',

algoId: '',

tag: '',

px: '82300',

sz: '0.00001',

notionalUsd: '0.8223498300000001',

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: '1743189667090',

cTime: '1743189667090',

source: '',

reduceOnly: 'false',

cancelSource: '',

quickMgnType: '',

stpId: '',

stpMode: 'cancel\_maker',

attachAlgoClOrdId: '',

lastPx: '83651',

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: { algoId: '' },

attachAlgoOrds: []

}

]

[OKX] Ordem 2371871096866381824: Estado: live, Preço: 82300, Quantidade: 0.00001

[OKX] Mensagem recebida: {

arg: {

channel: 'orders',

instType: 'SPOT',

instId: 'BTC-USDT',

uid: '670299194443692254'

},

data: [

{

instType: 'SPOT',

instId: 'BTC-USDT',

tgtCcy: '',

ccy: '',

ordId: '2371871096866381824',

clOrdId: '',

algoClOrdId: '',

algoId: '',

tag: '',

px: '82300',

sz: '0.00001',

notionalUsd: '0.8223498300000001',

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: '1743189667090',

cTime: '1743189667090',

source: '',

reduceOnly: 'false',

cancelSource: '',

quickMgnType: '',

stpId: '',

stpMode: 'cancel\_maker',

attachAlgoClOrdId: '',

lastPx: '83651',

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: [Object],

attachAlgoOrds: []

}

]

}

[OKX] Atualização de ordem recebida: [

{

instType: 'SPOT',

instId: 'BTC-USDT',

tgtCcy: '',

ccy: '',

ordId: '2371871096866381824',

clOrdId: '',

algoClOrdId: '',

algoId: '',

tag: '',

px: '82300',

sz: '0.00001',

notionalUsd: '0.8223498300000001"',

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: '1743189667092',

cTime: '1743189667090',

source: '',

reduceOnly: 'false',

cancelSource: '14',

quickMgnType: '',

stpId: '',

stpMode: 'cancel\_maker',

attachAlgoClOrdId: '',

lastPx: '83651',

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: { algoId: '' },

attachAlgoOrds: []

}

]

[OKX] Ordem 2371871096866381824: Estado: canceled,

CODIGOS DE CANCELAMENTO OKX - cancelSource

https://www.okx.com/docs-v5/en/#order-book-trading-trade-ws-order-channel

Source of the order cancellation. Valid values and the corresponding meanings are:

0: Order canceled by system  
1: Order canceled by user  
2: Order canceled: Pre reduce-only order canceled, due to insufficient margin in user position  
3: Order canceled: Risk cancellation was triggered. Pending order was canceled due to insufficient margin ratio and forced-liquidation risk.  
4: Order canceled: Borrowings of crypto reached hard cap, order was canceled by system.  
6: Order canceled: ADL order cancellation was triggered. Pending order was canceled due to a low margin ratio and forced-liquidation risk.  
7: Order canceled: Futures contract delivery.  
9: Order canceled: Insufficient balance after funding fees deducted.  
10: Order canceled: Option contract expiration.  
13: Order canceled: FOK order was canceled due to incompletely filled.  
14: Order canceled: IOC order was partially canceled due to incompletely filled.  
15: Order canceled: The order price is beyond the limit  
17: Order canceled: Close order was canceled, due to the position was already closed at market price.  
20: Cancel all after triggered  
21: Order canceled: The TP/SL order was canceled because the position had been closed  
22, 23: Order canceled: Reduce-only orders only allow reducing your current position. System has already canceled this order.  
27: Order canceled: Price limit verification failed because the price difference between counterparties exceeds 5%  
31: The post-only order will take liquidity in taker orders  
32: Self trade prevention  
33: The order exceeds the maximum number of order matches per taker order  
36: Your TP limit order was canceled because the corresponding SL order was triggered.  
37: Your TP limit order was canceled because the corresponding SL order was canceled.  
38: You have canceled market maker protection (MMP) orders.  
39: Your order was canceled because market maker protection (MMP) was triggered.  
42: Your order was canceled because the difference between the initial and current best bid or ask prices reached the maximum chase difference.