Version 1.0
작성자:
적용 대상 거래소: Binance Futures (USDT-M)
대상 자산: DOGEUSDT, ALGOUSDT(기본), 확장 가능
전략 유형: 단방향 대세 + 손익 기반 헷지 + RSI2/MACD 중심선 신호 기반 단기 운용

---

1. 목적 (Purpose)

본 시스템의 목적은 다음 두 가지 원칙에 따라 **대세 한 방향 운용**을 수행하면서, 손실을 최소화하기 위한 **동일 수량 반대 포지션 헷지**를 자동으로 처리하는 것이다.

1. **일봉 대세 방향으로만 신규 포지션을 잡는다.**
2. **손실이 발생하면 동일 수량으로 반대 포지션을 헷지한다.**
3. **대세와 같은 신호가 재발하면 반대 포지션(헷지)은 손익 불문 즉시 제거한다.**

이를 통해 단방향 추세 수익 극대화 + 손실 구간 방어를 결합한 구조를 구현한다.

---

2. 전략 개요 (Strategy Concept)

2.1 대세(Regime)

- 일봉 기준 **RSI2 > MACD(1,1,1)** → 상승 대세(UP, 콜장)
- 일봉 기준 **RSI2 < MACD(1,1,1)** → 하락 대세(DOWN, 풋장)

대세는 하루에 한 번만 갱신하며,
대세가 바뀌면 **반대 포지션은 즉시 전부 청산**해 단방향 프레임을 유지한다.

2.2 5분봉 단기 신호 (Small Trend)

- RSI2 > MACD → 상승 신호(소세 상승)
- RSI2 < MACD → 하락 신호(소세 하락)

5분봉 신호는 **대세 방향에 맞는 신규 진입** 시에만 활용하고,
포지션 관리 및 헤지 해제에도 사용된다.

---

3. 주요 규칙 (Trading Rules)

3.1 상승 대세(UP: 콜장)

신규 콜 진입 조건

- 5분봉 상승 신호 발생
- pos_call == 0

콜 보유 관리

- 손익 ≥ 0 : exit_rule_hit() 충족 시 콜 청산
- 손익 < 0 : 풋 헤지 CALL_SIZE 추가 진입 (pos_put==0일 때)

새 상승 신호 발생 시

- pos_put > 0 → 모든 풋 헤지 즉시 청산
  (**손익과 무관하게 삭제**)

하락 신호 발생 시 (풋 신호)

- 콜 손익 ≥ 0 → 즉시 콜 청산 (이익 잠금)
- 콜 손익 < 0 → 풋 헤지 CALL_SIZE 진입

3.2 하락 대세(DOWN: 풋장)

신규 풋 진입 조건

- 5분봉 하락 신호 발생
- pos_put == 0

풋 보유 관리

- 손익 ≥ 0 : exit_rule_hit() 충족 시 풋 청산
- 손익 < 0 : 콜 헤지 CALL_SIZE 추가 진입 (pos_call==0일 때)

새 하락 신호 발생 시

- pos_call > 0 → 콜 헤지 즉시 청산
  (**손익 불문**)

상승 신호 발생 시 (콜 신호)

- 풋 손익 ≥ 0 → 풋 청산 (이익 잠금)
- 풋 손익 < 0 → 콜 헤지 CALL_SIZE 진입

---

4. 필수 API 기능 요구사항 (Requirements)

4.1 가격 및 지표

- get_last_price(symbol)
- get_indicator("RSI", symbol, tf, length=2)
- get_indicator("MACD_line", symbol, tf, fast=1, slow=1, signal=1)

4.2 주문

- buy_call(size) / sell_call(size)
- buy_put(size) / sell_put(size)

Binance에서는 "콜/풋" 개념이 없으므로:

- buy_call → Long position
- buy_put → Short position
  으로 매핑하여 개발자가 연결해야 한다.

4.3 기타

- exit_rule_hit(): 목표가/트레일링/RSI 롤오버 등 자유 구성
- state 영속화(DB/파일)
- 오류/부분체결 처리
- 재시작 데이터 복구

---

5. 변수 설명 및 구조

Parameters

변수	설명
CALL_SIZE	기본 진입 및 헤지 수량

변수	설명
RSI_LEN	RSI 기간, 기본=2
SYMBOL	거래 심볼

State

변수	설명
regime	"UP" or "DOWN"
pos_call	콜 보유 수량
pos_put	풋 보유 수량
entry_price_call	콜 진입가
entry_price_put	풋 진입가

---

6. 예외 처리 및 시스템 요구사항

6.1 재시작/프로그램 종료 대비

- state를 JSON 파일 또는 DB에 자동 저장
- 재부팅 후 마지막 포지션 상태를 정확히 복구해야 한다

6.2 비정상 주문 실패

- 주문 실패 시 재시도 3회
- 오더 체결 확인 후 상태 업데이트

6.3 심볼 확장성

- DOGEUSDT, ALGOUSDT → 배열 구조로 확장 가능

---

7. 참조 코드 (Copy-Ready Skeleton)

아래 코드는 전략 로직을 그대로 구현한 기본 코드 프레임워크다.
개발자는 이걸 기반으로 API 레이어를 추가하면 바로 실전 적용 가능하다.

```
# --- Parameters & state ---
CALL_SIZE = 10
RSI_LEN = 2

state = {
    "regime": "UP",
    "pos_call": 0,
    "pos_put": 0,
    "entry_price_call": None,
    "entry_price_put": None,
}

# --- Exchange/API placeholders ---
def get_last_price(symbol: str) -> float: ...
def get_indicator(name: str, symbol: str, tf: str, **kwargs) -> float: ...
def buy_call(symbol: str, size: int): ...
def sell_call(symbol: str, size: int): ...
def buy_put(symbol: str, size: int): ...
def sell_put(symbol: str, size: int): ...
def exit_rule_hit(symbol: str) -> bool: ...

SYMBOL = "YOUR_SYMBOL"

# --- Signals ---
```

```
def macd_centerline(symbol: str, tf: str) -> float:
    return get_indicator("MACD_line", symbol=symbol, tf=tf,
                                    fast=1, slow=1, signal=1)

def rsi2(symbol: str, tf: str) -> float:
    return get_indicator("RSI", symbol=symbol, tf=tf,
                                    length=RSI_LEN)

def signal_long(symbol: str, tf: str) -> bool:
    return rsi2(symbol, tf) > macd_centerline(symbol, tf)

def signal_short(symbol: str, tf: str) -> bool:
    return rsi2(symbol, tf) < macd_centerline(symbol, tf)


# --- PnL helpers ---
def pnl_call(current_price: float) -> float:
    if state["entry_price_call"] is None:
        return 0.0
    return current_price - state["entry_price_call"]

def pnl_put(current_price: float) -> float:
    if state["entry_price_put"] is None:
        return 0.0
    return state["entry_price_put"] - current_price
```

```
# --- Daily regime update ---
def on_new_daily():
    up = signal_long(SYMBOL, tf="1D")
    down = signal_short(SYMBOL, tf="1D")

    if up and not down:
        state["regime"] = "UP"
    elif down and not up:
        state["regime"] = "DOWN"

    if state["regime"] == "UP" and state["pos_put"] > 0:
        sell_put(SYMBOL, state["pos_put"])
        state["pos_put"] = 0
        state["entry_price_put"] = None

    if state["regime"] == "DOWN" and state["pos_call"] > 0:
        sell_call(SYMBOL, state["pos_call"])
        state["pos_call"] = 0
        state["entry_price_call"] = None


# --- 5m execution loop ---
def on_new_5m():
```

```
price = get_last_price(SYMBOL)

# ===== 상승 대세 =====
if state["regime"] == "UP":

    if signal_long(SYMBOL, "5m") and state["pos_call"] == 0:
        buy_call(SYMBOL, CALL_SIZE)
        state["pos_call"] = CALL_SIZE
        state["entry_price_call"] = price

        if state["pos_put"] > 0:
            sell_put(SYMBOL, state["pos_put"])
            state["pos_put"] = 0
            state["entry_price_put"] = None

    if state["pos_call"] > 0:
        if pnl_call(price) >= 0:
            if exit_rule_hit(SYMBOL):
                sell_call(SYMBOL, state["pos_call"])
                state["pos_call"] = 0
                state["entry_price_call"] = None

                if state["pos_put"] > 0:
                    sell_put(SYMBOL, state["pos_put"])
                    state["pos_put"] = 0
```

```
                    state["entry_price_put"] = None
            else:
                if state["pos_put"] == 0:
                    buy_put(SYMBOL, CALL_SIZE)
                    state["pos_put"] = CALL_SIZE
                    state["entry_price_put"] = price

        if state["pos_put"] > 0 and signal_long(SYMBOL, "5m"):
            sell_put(SYMBOL, state["pos_put"])
            state["pos_put"] = 0
            state["entry_price_put"] = None

        if signal_short(SYMBOL, "5m") and state["pos_call"] > 0:
            if pnl_call(price) >= 0:
                sell_call(SYMBOL, state["pos_call"])
                state["pos_call"] = 0
                state["entry_price_call"] = None
            else:
                if state["pos_put"] == 0:
                    buy_put(SYMBOL, CALL_SIZE)
                    state["pos_put"] = CALL_SIZE
                    state["entry_price_put"] = price


# ===== 하락 대세 =====
```

```
elif state["regime"] == "DOWN":

    if signal_short(SYMBOL, "5m") and state["pos_put"] == 0:
        buy_put(SYMBOL, CALL_SIZE)
        state["pos_put"] = CALL_SIZE
        state["entry_price_put"] = price

        if state["pos_call"] > 0:
            sell_call(SYMBOL, state["pos_call"])
            state["pos_call"] = 0
            state["entry_price_call"] = None

    if state["pos_put"] > 0:
        if pnl_put(price) >= 0:
            if exit_rule_hit(SYMBOL):
                sell_put(SYMBOL, state["pos_put"])
                state["pos_put"] = 0
                state["entry_price_put"] = None

                if state["pos_call"] > 0:
                    sell_call(SYMBOL, state["pos_call"])
                    state["pos_call"] = 0
                    state["entry_price_call"] = None
        else:
            if state["pos_call"] == 0:
```

```
                buy_call(SYMBOL, CALL_SIZE)
                state["pos_call"] = CALL_SIZE
                state["entry_price_call"] = price

    if state["pos_call"] > 0 and signal_short(SYMBOL, "5m"):
        sell_call(SYMBOL, state["pos_call"])
        state["pos_call"] = 0
        state["entry_price_call"] = None

    if signal_long(SYMBOL, "5m") and state["pos_put"] > 0:
        if pnl_put(price) >= 0:
            sell_put(SYMBOL, state["pos_put"])
            state["pos_put"] = 0
            state["entry_price_put"] = None
        else:
            if state["pos_call"] == 0:
                buy_call(SYMBOL, CALL_SIZE)
                state["pos_call"] = CALL_SIZE
                state["entry_price_call"] = price
```

---

8. 개발자가 반드시 구현해야 하는 부분

1. Binance 주문 API 연결
2. 지표 계산(RSI, MACD) → 직접 구현 또는 라이브러리 활용
3. exit_rule_hit() 구체화
4. 포지션 상태 저장/복구

5. UI(수량 변경, 시작/종료 버튼)
6. 오류처리 + partial fill 처리

1. EXE 실행 구조 설계서

1-1. 전체 구조 개요 (모듈 단위)

윈도우 기준, 폴더 구조 예시:

```
autohedge_bot/
  config/
    settings.yaml          # 심볼, 레버리지, TP%, SL%, 호출 주기 등
  core/
    strategy.py            # on_new_daily / on_new_5m, state, 신호로직
    rules.py               # exit_rule_hit, 리스크 룰
    backtest.py            # 백테스트 엔진
  adapters/
    binance_futures.py     # Binance API 연동 (buy_call/buy_put 등 구현)
    indicators.py          # RSI, MACD 계산 (백테스트/실시간 공통)
  ui/
    main_ui.py             # PyQt/PySide GUI
  runner/
    service.py             # 실시간 루프(스케줄러), 로그 관리
  logs/
    ...
  main.py                  # 진입점 (GUI/CLI 선택)
  requirements.txt
  build_exe.bat
```

1-2. 실행 흐름

1. main.exe 실행
2. config/settings.yaml 로드
3. UI 모드라면 ui/main_ui.py 실행 후:
    - API 키/심볼/전략 파라미터를 UI에서 입력 or 수정
    - "시작" 버튼 클릭 시 → runner/service.py에 설정 전달
4. service.py에서:
    - Binance 시간 동기화
    - 1D 캔들 클로즈 감지 → strategy.on_new_daily()
    - 5m 캔들 클로즈 감지 → strategy.on_new_5m()
    - 주문함수는 adapters/binance_futures.py 의 buy_call, buy_put 등을 호출
5. 로그 및 UI 상태 패널에 상태 업데이트

1-3. EXE 빌드 방식

- Python + PyQt(Pyside6) 기준:
    - pip install pyinstaller
    - pyinstaller --noconsole --onefile main.py
- build_exe.bat 예시:

```
@echo off
cd /d %~dp0
pyinstaller --noconfirm --noconsole --onefile main.py
echo 빌드 완료: dist₩main.exe
pause
```

환경설정은 .env 또는 settings.yaml 사용하고,
실 계정용 API 키는 절대 코드에 하드코딩하지 않는 구조로 고정.

---

2. exit_rule_hit() 상세 설계

2-1. 설계 목표

- "이익이 어느 정도 나면 욕심부리지 말고 정리"
- "너무 오래 들고 있지 않기"
- "단기 모멘텀이 꺾인 느낌(RSI2 롤오버)일 때 정리"

세 가지를 OR 조건으로 묶는다.

2-2. 파라미터 제안

```
TAKE_PROFIT_PCT = 0.01        # +1% 이상 수익 시 청산 후보
MAX_HOLD_BARS    = 24          # 5분봉 24개 ≒ 2시간 이상 보유 시 정리 후보
TRAILING_PCT     = 0.005       # +0.5% 이상 이익 구간에서 트레일링 스탑
```

추가로 state에 다음 필드 몇 개를 붙이는 것을 추천:

```
state.update({
    "entry_time_call": None,
    "entry_time_put": None,
    "max_favorable_price_call": None,   # 진입 이후 최고가
    "max_favorable_price_put": None,    # 진입 이후 최저가(풋 기준)
})
```

2-3. 결정 로직

콜 보유 시:

1. 현재 수익률 = (price - entry) / entry
2. 수익률 ≥ TAKE_PROFIT_PCT → 이익 실현 OK
3. 현재 시각 – 진입 시각 ≥ MAX_HOLD_BARS * 5분 → 시간 초과 청산
4. 수익 구간(수익률 ≥ TRAILING_PCT)에서,
    - price가 max_favorable_price_call * (1 - TRAILING_PCT) 아래로 떨어지면 → 모멘텀 꺾임으로 보고 청산
5. RSI2 롤오버:

- 콜(상승장)에서 RSI2가 MACD 센터라인 아래로 내려가면(signal_short) → 추세 꺾임 신호로 간주, 청산 후보

풋도 반대로 적용.

2-4. 예시 코드

```
from datetime import datetime, timedelta


TAKE_PROFIT_PCT = 0.01     # 1%
MAX_HOLD_BARS     = 24        # 24 * 5분
TRAILING_PCT       = 0.005    # 0.5%


def now_utc():
    # 실제로는 exchange 서버 시간 또는 로컬 UTC 기준
    return datetime.utcnow()


def exit_rule_hit(symbol: str) -> bool:
    price = get_last_price(symbol)

    # 콜 포지션이 메인일 때
    if state["pos_call"] > 0 and state["entry_price_call"] is not None:
        entry = state["entry_price_call"]
        pnl_pct = (price - entry) / entry

        # 최초 진입 시각 없으면 지금 세팅
        if state.get("entry_time_call") is None:
            state["entry_time_call"] = now_utc()
```

```
# 최대 유리 가격 갱신
if state.get("max_favorable_price_call") is None:
    state["max_favorable_price_call"] = price
else:
    state["max_favorable_price_call"] = max(
        state["max_favorable_price_call"], price
    )

hold_minutes = (now_utc() - state["entry_time_call"]).total_seconds() / 60.0
hold_bars = hold_minutes / 5.0

# 1) 고정 TP
if pnl_pct >= TAKE_PROFIT_PCT:
    return True

# 2) 시간 초과
if hold_bars >= MAX_HOLD_BARS:
    return True

# 3) 트레일링 스탑
max_price = state["max_favorable_price_call"]
if pnl_pct > TRAILING_PCT:
    if price <= max_price * (1 - TRAILING_PCT):
        return True
```

```
    # 4) RSI2 롤오버 (상승장 기준)
    if signal_short(symbol, tf="5m"):
        # 단기 모멘텀이 아래로 꺾였다고 보고 정리
        return True

    return False

# 풋 포지션이 메인일 때
if state["pos_put"] > 0 and state["entry_price_put"] is not None:
    entry = state["entry_price_put"]
    pnl_pct = (entry - price) / entry  # 가격 하락 시 이익

    if state.get("entry_time_put") is None:
        state["entry_time_put"] = now_utc()

    if state.get("max_favorable_price_put") is None:
        state["max_favorable_price_put"] = price
    else:
        # 풋 기준 유리한 방향은 price가 내려가는 것 → 최저가 기록
        state["max_favorable_price_put"] = min(
            state["max_favorable_price_put"], price
        )

    hold_minutes = (now_utc() - state["entry_time_put"]).total_seconds() / 60.0
```

```
        hold_bars = hold_minutes / 5.0

        if pnl_pct >= TAKE_PROFIT_PCT:
            return True

        if hold_bars >= MAX_HOLD_BARS:
            return True

        min_price = state["max_favorable_price_put"]
        if pnl_pct > TRAILING_PCT:
            # 풋에서 트레일링: 최저가에서 일정 비율 위로 반등하면 종료
            if price >= min_price * (1 + TRAILING_PCT):
                return True

        # 풋 메인일 땐 상승 신호가 롤오버로 작동
        if signal_long(symbol, tf="5m"):
            return True

        return False

    # 포지션 없으면 항상 False
    return False
```

---

3. 멀티코인 백테스트 코드 스켈레톤

여기서는 **클래스 기반 전략 인스턴스**를 정의해서

각 심볼별로 독립적인 state를 갖게 만든다.

3-1. Strategy 클래스

```
import pandas as pd


class HedgeStrategy:
    def __init__(self, symbol: str, call_size: int = 10, rsi_len: int = 2):
        self.symbol = symbol
        self.CALL_SIZE = call_size
        self.RSI_LEN = rsi_len

        self.state = {
            "regime": "UP",
            "pos_call": 0,
            "pos_put": 0,
            "entry_price_call": None,
            "entry_price_put": None,
            "entry_time_call": None,
            "entry_time_put": None,
            "max_favorable_price_call": None,
            "max_favorable_price_put": None,
        }

        self.trades = []  # 백테스트용 체결 기록

    # 여기에 rsi2, macd_centerline, signal_long/short, pnl_call/put,
```

```
    # on_new_daily, on_new_5m, exit_rule_hit 등을 인스턴스 메서드로 이식
```

3-2. 백테스트 루프 (다중 심볼)

```
def backtest_multi(symbol_dfs_1d: dict, symbol_dfs_5m: dict):
    """
    symbol_dfs_1d: { "DOGEUSDT": df_1d, "ALGOUSDT": df_1d, ... }
    symbol_dfs_5m: { "DOGEUSDT": df_5m, "ALGOUSDT": df_5m, ... }
    df_1d / df_5m는 반드시 'open_time', 'close' 컬럼 포함
    """
    strategies = {
        sym: HedgeStrategy(sym) for sym in symbol_dfs_5m.keys()
    }

    # 1D와 5m의 시간축을 맞춰야 하는데, 단순하게 5m 루프 돌면서
    # 해당 시점까지의 일봉이 새로 생겼는지만 체크하는 방식으로 구현 가능
    results = {}

    for symbol, df_5m in symbol_dfs_5m.items():
        strat = strategies[symbol]
        df_daily = symbol_dfs_1d[symbol]

        # 일봉 인덱스 포인터
        daily_idx = 0
        current_daily_time = df_daily["open_time"].iloc[daily_idx]

        for _, row in df_5m.iterrows():
```

```
            ts = row["open_time"]
            price = row["close"]

            # 1) 일봉 갱신 체크
            # (ts가 다음 일봉 구간으로 넘어가면 on_new_daily)
            while (daily_idx + 1 < len(df_daily) and
                   ts >= df_daily["open_time"].iloc[daily_idx + 1]):
                daily_idx += 1
                current_daily_time = df_daily["open_time"].iloc[daily_idx]
                strat.on_new_daily(df_daily.iloc[:daily_idx+1])

            # 2) 5m 실행
            strat.on_new_5m(price, ts, df_5m, idx=None)  # 구현 방식에 따라 인자 조정

        results[symbol] = strat.trades

    return results
```

여기서는 전체 구조만 잡은 거고,
실제로는 HedgeStrategy 안에 지표 계산/신호/매수·매도 기록을 채워 넣으면 된다.

---

4. Binance 실전 주문용 실행 코드 스켈레톤

여기서는 python-binance 라이브러리를 사용해서
buy_call, buy_put, sell_call, sell_put, get_last_price를 **실제 선물 주문**에 매핑하는 예를 보여줄게. python-binance.readthedocs.io+1

전제:

- USDT-M 선물
- 듀얼 포지션 모드(hedge mode)에서
    - “콜” = positionSide="LONG"
    - “풋” = positionSide="SHORT"

4-1. 설치

pip install python-binance

4-2. Binance 어댑터 예시 (adapters/binance_futures.py)

```
import os
from binance.client import Client

API_KEY    = os.getenv("BINANCE_API_KEY")
API_SECRET = os.getenv("BINANCE_API_SECRET")

# testnet 사용을 권장 (실계좌 전에)
client = Client(API_KEY, API_SECRET, testnet=True)

def get_last_price(symbol: str) -> float:
    ticker = client.futures_symbol_ticker(symbol=symbol)
    return float(ticker["price"])

def buy_call(symbol: str, size: float):
    """
    콜 = LONG 포지션
    """
    order = client.futures_create_order(
```

```
        symbol=symbol,
        side="BUY",
        type="MARKET",
        quantity=size,
        positionSide="LONG"
    )
    return order

def sell_call(symbol: str, size: float):
    """
    콜 청산 = LONG reduceOnly SELL
    """
    order = client.futures_create_order(
        symbol=symbol,
        side="SELL",
        type="MARKET",
        quantity=size,
        positionSide="LONG",
        reduceOnly=True
    )
    return order

def buy_put(symbol: str, size: float):
    """
    풋 = SHORT 포지션
```

```
    """
    order = client.futures_create_order(
        symbol=symbol,
        side="SELL",
        type="MARKET",
        quantity=size,
        positionSide="SHORT"
    )
    return order


def sell_put(symbol: str, size: float):
    """
    풋 청산 = SHORT reduceOnly BUY
    """
    order = client.futures_create_order(
        symbol=symbol,
        side="BUY",
        type="MARKET",
        quantity=size,
        positionSide="SHORT",
        reduceOnly=True
    )
    return order
```

futures_create_order 는 바이낸스 USDT-M 선물 REST의 New Order 엔드포인트를 래핑한 것으로, side, type, quantity, positionSide, reduceOnly 등을 인자로 받는다. [python-binance.readthedocs.io+1]

위 함수를 그대로 기존 전략 코드의 placeholder 자리에 연결하면 된다.

4-3. 실행 러너 예시 (runner/service.py)

아주 단순하게:

```
import time
from datetime import datetime
from core.strategy import on_new_daily, on_new_5m
from adapters.binance_futures import get_last_price

SYMBOLS = ["DOGEUSDT", "ALGOUSDT"]

def is_new_5m_candle(prev_ts):
    now = datetime.utcnow()
    minute = now.minute
    if prev_ts is None:
        return True, now.replace(second=0, microsecond=0)
    if now.minute % 5 == 0 and now.second < 5:
        if now.replace(second=0, microsecond=0) != prev_ts:
            return True, now.replace(second=0, microsecond=0)
    return False, prev_ts

def main_loop():
    prev_5m_ts = None
    prev_daily_date = None

    while True:
```

```
        now = datetime.utcnow()

        # 일봉 마감(예: 00:00 UTC 기준) 체크
        if prev_daily_date is None or now.date() != prev_daily_date:
            # 실제로는 바이낸스 일봉 클로즈 시각에 맞추는 게 더 정확
            on_new_daily()
            prev_daily_date = now.date()

        # 5분봉 마감 체크
        is_new, prev_5m_ts = is_new_5m_candle(prev_5m_ts)
        if is_new:
            for sym in SYMBOLS:
                on_new_5m()  # 내부에서 SYMBOL 전역 또는 인자 이용

        time.sleep(1)

if __name__ == "__main__":
    main_loop()
```

실전용으로는 WebSocket 캔들 스트림을 붙여서 정확한 캔들 클로즈에 반응하도록 확장하면 더 좋다. algotrading101.com

---

5. UI 목업 (텍스트 와이어프레임 + PyQt 구조)

5-1. 화면 구조 와이어프레임

텍스트 기반으로 그리면 대략 이런 느낌:

```
+-----------------------------------------------------------+
```

```
| [ Binance Auto Hedge Trader ]                                  |
+----------------------------------------------------------------+
| API 설정                                                       |
|   API Key    [_____________________________]                   |
|   Secret     [_____________________________]  [연결 테스트]    |
+----------------------------------------------------------------+
| 전략 설정                                                      |
|   심볼 선택    [DOGEUSDT] [ALGOUSDT] [추가...]                 |
|   기본 수량    CALL_SIZE: [ 10 ]                               |
|   RSI 길이     RSI_LEN :  [  2 ]                               |
|   레버리지     [ 10x ▼ ]                                       |
|                                                                |
| 리스크 / 청산 설정                                             |
|   TP%          [ 1.0 ]     SL(트레일링)% [ 0.5 ]               |
|   최대 보유시간 (5분봉 개수) [ 24 ]                            |
+----------------------------------------------------------------+
| 실시간 상태                                                    |
|   심볼      대세   포지션   수량   진입가   현재가   PnL   헷지 |
|  ------------------------------------------------------------  |
|   DOGEUSDT  UP     LONG     10     0.1234  0.1250  +1.3%  SHORT |
|   ALGOUSDT  DOWN   SHORT    10     0.1450  0.1400  +3.4%  LONG  |
+----------------------------------------------------------------+
| 로그                                                           |
|  [2025-11-19 10:05] DOGEUSDT: 콜 10계약 진입                   |
|  [2025-11-19 10:10] DOGEUSDT: 풋 10계약 헷지 진입              |
```

```
|  ...                                                        |
+-------------------------------------------------------------+
| [전략 시작]  [일시정지]  [즉시 종료]                          |
+-------------------------------------------------------------+
```

5-2. PyQt/PySide 구조 제안

클래스 구조:

- MainWindow(QMainWindow)
    - ApiConfigWidget (API 키 입력 + 테스트 버튼)
    - StrategyConfigWidget (CALL_SIZE, RSI_LEN, TP%, MaxHold 등)
    - SymbolTableWidget (QTableWidget, 심볼별 상태 표현)
    - LogWidget (QPlainTextEdit 또는 QListWidget)
    - 하단에 Start / Pause / Stop 버튼

핵심 시그널/슬롯:

- API 설정 변경 → 내부 config 객체 업데이트
- 전략 시작 → service.py 쪽에 쓰레드 or QTimer로 루프 시작
- 5분마다 상태 업데이트 → UI 스레드로 emit → 테이블 갱신
- 로그 → append_log(str) 메서드로 UI에 쌓기

예시 스켈레톤:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout,
    QHBoxLayout, QLabel, QLineEdit, QPushButton, QTableWidget,
    QTableWidgetItem, QSpinBox, QDoubleSpinBox, QPlainTextEdit
)


class MainWindow(QMainWindow):
```

```
def __init__(self):
    super().__init__()
    self.setWindowTitle("Binance Auto Hedge Trader")

    root = QWidget()
    root_layout = QVBoxLayout(root)

    # API 설정
    api_layout = QHBoxLayout()
    api_layout.addWidget(QLabel("API Key"))
    self.api_key_edit = QLineEdit()
    api_layout.addWidget(self.api_key_edit)

    api_layout.addWidget(QLabel("Secret"))
    self.api_secret_edit = QLineEdit()
    self.api_secret_edit.setEchoMode(QLineEdit.Password)
    api_layout.addWidget(self.api_secret_edit)

    self.btn_test = QPushButton("연결 테스트")
    api_layout.addWidget(self.btn_test)

    root_layout.addLayout(api_layout)

    # 전략 설정
    strat_layout = QHBoxLayout()
```

```
strat_layout.addWidget(QLabel("CALL_SIZE"))
self.spin_call_size = QSpinBox()
self.spin_call_size.setRange(1, 100000)
self.spin_call_size.setValue(10)
strat_layout.addWidget(self.spin_call_size)

strat_layout.addWidget(QLabel("RSI_LEN"))
self.spin_rsi_len = QSpinBox()
self.spin_rsi_len.setRange(1, 50)
self.spin_rsi_len.setValue(2)
strat_layout.addWidget(self.spin_rsi_len)

strat_layout.addWidget(QLabel("TP%"))
self.spin_tp = QDoubleSpinBox()
self.spin_tp.setRange(0.1, 100.0)
self.spin_tp.setValue(1.0)
strat_layout.addWidget(self.spin_tp)

strat_layout.addWidget(QLabel("MaxHold(5m bars)"))
self.spin_max_hold = QSpinBox()
self.spin_max_hold.setRange(1, 500)
self.spin_max_hold.setValue(24)
strat_layout.addWidget(self.spin_max_hold)

root_layout.addLayout(strat_layout)
```

```
# 심볼 상태 테이블
self.table = QTableWidget(0, 8)
self.table.setHorizontalHeaderLabels(
      ["심볼", "대세", "포지션", "수량", "진입가",
       "현재가", "PnL", "헷지"]
)
root_layout.addWidget(self.table)

# 로그
self.log = QPlainTextEdit()
self.log.setReadOnly(True)
root_layout.addWidget(self.log)

# 버튼
btn_layout = QHBoxLayout()
self.btn_start = QPushButton("전략 시작")
self.btn_pause = QPushButton("일시정지")
self.btn_stop  = QPushButton("즉시 종료")
btn_layout.addWidget(self.btn_start)
btn_layout.addWidget(self.btn_pause)
btn_layout.addWidget(self.btn_stop)
root_layout.addLayout(btn_layout)

self.setCentralWidget(root)
```

```
    def append_log(self, text: str):
        self.log.appendPlainText(text)
```

이 정도 구조면 너가 "디자인 관점에서 뭐 더 넣을지, 빼줄지" 판단하기 좋을 거야.

정리

요청한 네 가지 + UI 목업까지 한 번에 정리하면:

1. **EXE 실행 구조 설계서**
    - 모듈 구조, 실행 흐름, PyInstaller 빌드 관점까지 포함
2. **exit_rule_hit() 상세 설계**
    - TP%, 최대 보유 시간, 트레일링, RSI2 롤오버를 조합한 실제 구현 예시 제공
3. **멀티코인 백테스트 코드**
    - HedgeStrategy 클래스로 심볼별 state 분리, 루프 구조 예시 제공
4. **Binance 실전 주문용 코드**
    - python-binance 기반으로 futures_create_order를 사용한 buy_call/buy_put/sell_call/sell_put 구현
5. **UI 목업**
    - 텍스트 와이어프레임 + PyQt 스켈레톤