⚙️ 项目运行逻辑手册 (Project Execution Logic Manual)

> **版本**: v3.1.1 (Async Core)
> **适用对象**: 开发者、高级用户

本文档详细拆解了 CryptoOracle 的启动流程、异步调度机制以及核心数据流转逻辑，帮助用户理解系统是如何"跑"起来的。

一、 系统启动流程 (Startup Flow)

当您运行 `OKXBot_Plus.py` (即原来的 main.py) 时，系统会按照以下顺序进行初始化：

1. **环境引导 (Bootstrap)**:
    - **日志初始化**: `setup_logger()` 启动，确保所有输出都有迹可循。
    - **配置加载**: `Config()` 读取 `.env` 和 `config.json`。如果配置缺失，系统会立即报错退出。
2. **核心组件实例化 (Component Instantiation)**:
    - **AI Agent**: 初始化 `DeepSeekAgent` (异步客户端)，配置 API Key 和 Proxy。
    - **Exchange**: 初始化 `ccxt.okx` (异步模式)，加载市场数据 (`load_markets`) 以获取最新的交易对精度和限制。
    - **Traders**: 根据 `config.json` 中的 `symbols` 列表，为每个交易对创建一个 `DeepSeekTrader` 实例。
    - **Risk Manager**: 初始化全局风控管理器，优先连接 OKX 统一账户接口获取 `totalEq` (总权益)。
3. **自检与预热 (Self-Check & Warm-up)**:
    - **连通性测试**: 测试 OKX API 和 DeepSeek API 是否通畅。
    - **资金盘点**: 扫描 USDT 余额和编外资产。
    - **数据预热**: 并发拉取所有交易对的 K 线数据，填满 `history_limit`，确保第一次分析时指标（RSI/MACD）是准确的。
    - *(可选)*: 如遇到连接问题，可先运行 `python test/test_connection.py` 进行独立诊断。
4. **进入主循环 (Main Loop)**:
    - 系统进入 `while True` 循环，开始周期性轮询。
    - **注意**: 这里的"轮询"并非传统的串行阻塞式轮询，而是**异步并发轮询。**
        - 系统会在每个周期内，通过 `asyncio.gather` 同时发起所有任务（市场数据拉取、AI 推理、风控检查）。
        - "周期性"仅指任务触发的频率（如每 60 秒触发一次），而"异步"指任务执行的方式（并行不等待）。

二、 异步调度机制 (Async Scheduling)

v3.0 的核心在于**非阻塞并发。**在每个轮询周期（Interval）内，系统执行以下操作：

1. 批次开始 (Batch Start)

- 记录当前时间戳，打印分割线。

2. 全局风控检查 (Global Risk Check)

- **await risk_manager.check()**:
    - 获取账户总权益（Total Equity）。
    - 计算相对于基准线（Smart Baseline）的盈亏。
    - 如果触发**全局止盈**或**全局止损**，立即触发熔断，平仓所有持仓并退出程序。

3. 并发交易执行 (Parallel Execution)

- 系统使用 `asyncio.gather(*tasks)` 同时启动所有 Trader 的 `run()` 方法。
- **这意味着**：如果有 10 个币种，系统会**同时**向 OKX 请求这 10 个币种的 K 线，并**同时**向 DeepSeek 发送分析请求（如果触发分析条件）。
- **优势**：总耗时取决于最慢的那一个请求，而不是所有请求之和。

4. 休眠与定频 (Sleep & Interval)

- 计算本轮执行耗时 `elapsed`。
- 休眠时间 `sleep_time = interval - elapsed`。
- **毫秒级支持**: 如果配置了 `1s` 或 `500ms`，系统会精准控制休眠时间，确保高频轮询的稳定性。

三、 核心数据流转 (Core Data Flow)

对于单个交易对（`DeepSeekTrader`），数据流转如下：

1. 感知 (Perception)

- **获取 K 线**: 调用 `exchange.fetch_ohlcv`。
- **计算指标**: 本地计算 RSI, MACD, Bollinger Bands, ADX。
- **获取持仓**: 调用 `exchange.fetch_positions` 了解当前盈亏。

2. 决策 (Decision)

- **构建 Prompt**: 将 K 线数据、指标数值、持仓状态、账户余额打包成一段 Prompt。
    - *Prompt* 包含: 最近 5 根 K 线数据、RSI/MACD/ADX/布林带具体数值、当前市场波动率状态 (High Trend/Choppy/Normal)。
    - 角色设定: 根据 ADX 自动注入不同的 System Prompt（如“激进趋势交易员”或“保守避险者”）。
- **AI 推理**: 发送给 DeepSeek-V3。
- **解析信号**: 接收 AI 返回的 JSON（BUY/SELL/HOLD, 信心, 止损位）。
    - 鲁棒性: 使用正则提取 JSON 块，防止 AI 回复多余的寒暄语导致解析失败。

3. 执行 (Execution)

- **信心过滤**: 检查 `confidence` 是否达到 `min_confidence` 门槛。
- **滑点保护**: 再次获取实时 Ticker，对比分析时的价格。如果偏差过大 (>1%)，放弃交易。
- **微利拦截**: (仅卖出) 检查浮盈是否覆盖手续费。
- **资金计算**: 计算 `min(AI建议, 配置限额, 余额)`。

- **下单**: 调用 `exchange.create_order`。

四、 异常处理 (Error Handling)

系统设计了多层防护网：

1. **网络异常**: `aiohttp` / `httpx` 超时或连接失败会触发重试或跳过本轮，不会导致程序崩溃。
2. **API 限流**: `ccxt` 开启了 `enableRateLimit=True`，自动处理 429 Too Many Requests。
3. **下单失败**: 余额不足或参数错误会触发 `Diagnostic Report`（诊断报告）推送。
4. **致命错误**: 只有配置错误或严重的运行时错误（如内存溢出）才会导致主程序退出。

五、 状态管理与持久化 (State Persistence)

系统使用本地文件来保证重启后的数据连续性：

1. **Bot State (`data/bot_state.json`)**:
    - 存储 `smart_baseline` (基准资金)。
    - 每次启动时读取，确保风控计算（总盈亏）是基于最初投入的本金，而不是基于上次重启时的余额。
2. **PnL History (`data/pnl_history.csv`)**:
    - 每轮风控检查时追加一条记录（时间戳、总权益、盈亏比例）。
    - `plotter.py` 读取此文件生成可视化图表。

六、 优雅退出机制 (Graceful Shutdown)

当用户按下 `Ctrl+C` (SIGINT) 时，系统不会立即杀死进程，而是执行以下清理：

1. **捕获信号**: `OKXBot_Plus.py` 捕获 `KeyboardInterrupt`。
2. **停止循环**: 跳出 `while True` 循环。
3. **资源释放**:
    - 关闭 `ccxt` 的异步连接池。
    - 关闭 `aiohttp` 的 Session。
    - 刷新日志缓冲区。
4. **退出日志**: 打印 "User Manual Stop" 确认安全退出。

七、 常见问题 (FAQ)

Q: 为什么日志里有时会看到 "Sleep 0.01s"?
A: 这说明您的轮询任务执行时间超过了设定的间隔（例如设了 1s，但网络请求花了 1.2s）。系统会立即开始下一轮，不等待。

Q: 增加币种会变慢吗?
A: 在 v3.0 异步架构下，增加币种对总耗时的影响很小（主要受限于带宽和 CPU 解析 JSON 的速度），但要注意交易所的 API 限流（Rate Limit）。