

FDAPI 期货数据中心用户使用手册

文档版本：V2026.08

适配程序：FDAPI 期货数据服务动态库

运行平台：Windows x64

阅读对象：软件使用者、集成开发者、量化用户、策略开发者

文档用途：指导部署、配置、使用、维护整套期货行情数据服务

文档版权：永轩软件 www.yx10000.com

一、产品概述

1.1 主要功能简介

FDAPI 是一套基于上期所 CTP 官方接口二次封装的全功能期货本地数据服务系统，集行情接入、K 线自动合成、二进制文件持久化、通达信历史数据导入、多方式对外查询服务于一体。

核心能力：

- 实时连接期货行情前置，订阅全市场/指定合约 Tick 行情；
- 自动合成 5 秒、1 分钟、5 分钟、日 K；
- 交易所、品种、合约静态基础信息持久化 SQLite FutureBase.db 数据库；
- 支持通达信 .day/.1min/.5min 文件批量转换导入为本系统标准二进制 K 线
- 自动计算主力/次主力合约、夜盘品种标记
- 支持本地文件直读、DLL 函数调用、TCP 长连接、HTTP 网络访问四种使用模式
- 实盘/仿真目录物理隔离，数据互不干扰

1.2 业务需求与适用场景

- 本地长期存储完整 Tick、各级周期二进制 K 线
- 通达信存量历史行情批量迁移至 FDAPI 本地仓库
- 脱离行情软件，自建纯净本地数据源
- 策略实时读取、历史回测读取二进制 K/Tick 文件
- 本机/局域网多程序共享行情与本地二进制数据
- 易语言/Python/C++/C#第三方软件快速对接

1.3 开发 & 运行环境

- 操作系统：Windows 10 / Windows 11 / Windows Server x64
- 依赖组件：VC++运行库、sqlite3（仅存储静态基础信息）
- 数据存储分层：
 - SQLite FutureBase.db：仅存交易所、品种、合约静态基础信息，缓存动态行情；
 - 二进制文件：Tick、Sec5 逐日独立文件；Min1/Min5 单合约环形跨交易日文件；DayK 单合约全历史文件，磁盘推荐：SSD 固态硬盘，多合约+多落地类型场景大幅降低 IO 阻塞

1.4 适用群体

量化策略开发者、程序化交易者、C++/Python/易语言对接开发、自建本地数据中心运维

二、详细功能介绍

2.1 行情接入功能

- 手动/自动断线重连，断线自动恢复订阅
- 批量订阅、批量退订合约，支持按分类一键订阅
- 实盘/仿真数据目录自动隔离、基础信息库共享
- 实时接收 Tick、五档深度盘口快照,首档落地

2.2 数据落地与合成功能

所有行情时序数据持久化二进制文件，SQLite 保存静态基础信息及动态行情：

- Tick 二进制落地：支持「全量 tick」/「仅成交量变动 tick」双模式切换
- Sec5（5 秒 K 线）：Tick 实时聚合，定点窗口兜底强制闭合写入文件
- Min1/Min5：Tick 驱动内存聚合，
- DayK：每日收盘统一合成写入二进制文件
- 分时线：读取 Min1 二进制实时计算，不额外占用磁盘
- 自动过滤 CTP 脏 tick、极值、空成交数据，减少无效 IO 写入
- TDX 导入功能：解析通达信本地 K 线文件，转换为本系统标准 K 线到二进制文件

2.3 基础信息维护功能

- 一键拉取交易所、品种、合约完整参数入库
- 支持期货/期权合约过滤开关
- 后台定时更新主力/次主力、夜盘标记写入数据库

- 静态信息长期缓存，无需频繁重复拉取前置接口

2.4 多模式数据服务功能

- 本地直读：解析二进制文件 + SQLite 基础库
- DLL 函数调用：统一 Query 接口读取二进制与基础信息
- TCP 长连接：局域网远程查询 K/Tick、合约基础数据
- HTTP 接口：GET 请求返回 JSON 格式行情与 K 线数据

2.5 订阅合约&落地类型 联动性能调优

2.5.1 资源消耗对应关系

内存、CPU、磁盘 IO 压力由「订阅合约数量」+「开启落地类型」共同决定，负载从低到高分级：

- 低负载：≤30 个主力合约，关闭 Tick，仅开启 Min1+DayK
- 中等负载：20 ~ 60 合约，开启 Sec5+Min1，关闭 Tick
- 高负载：80+合约，同时开启 Tick+Sec5+Min1+Min5
- 极高负载：全市场合约订阅+全类型 Tick/K 线落地（机械硬盘禁止使用）

2.5.2 各落地类型资源开销明细

| 落地类型 | 内存占用 | 磁盘写入频率 | IO 压力 | 适用场景 |

|---|---|---|---|

| Tick (全量) | 极高 | 毫秒级高频写 | 最大 | 精细逐笔复盘、高频 tick 策略 |

| Tick (仅波动) | 中 | 仅成交变动写入 | 中低 | 绝大多数量化策略（推荐） |

| Sec5 5 秒 K | 中 | 每 5 秒批量写盘 | 中等 | 秒级短线、日内高频策略 |

| Min1 1 分 K | 低 | 每分钟批量写入 | 低 | 通用日内策略，默认推荐开启 |

| Min5 5 分 K | 极低 | 每 5 分钟批量写入 | 极低 | 波段、隔夜趋势策略 |

| DayK 日 K | 极低 | 每日收盘一次写入 | 可忽略 | 所有策略必开 |

2.5.3 分场景推荐落地组合

- 短线秒级（SSD，≤60 合约）：波动 Tick + Sec5 + Min1 + DayK
- 日内波段（机械硬盘/低配机器）：关闭 Tick、Sec5，仅 Min1+DayK
- 隔夜长线：关闭 Tick/Sec5，仅 Min5+DayK

2.5.4 性能约束与风险点

- CTP 行情回调 OnRtnDepthMarketData 为同步阻塞回调，所有落地逻辑异步投递 IO 线程；订阅过多叠加多类型落地会造成 IO 线程堆积、行情延迟；
- 每个订阅合约独立维护 tickBuffer、sec5Cache 等内存对象，合约越多内存线性增长；
- 二进制文件仅支持服务端单线程写入，外部程序仅允许只读，多写会文件损坏；

2.5.5 性能优化操作清单

- 及时退订不交易合约，从根源减少内存与文件数量；
- Tick 切换为「仅成交量波动」模式，降低 tick 缓存记录；
- 关闭自身策略不需要的周期落地开关；
- 机械硬盘场景订阅合约控制在 30 以内，可禁用全量 Tick；
- 开启合约白名单，仅白名单合约生成二进制文件；
- 7×24 长期运行建议闲时重启释放内存碎片。

三、基础使用方法与配置指南

3.1 首次初始化配置

3.1.1 数据目录配置规则

- 实盘模式：自定义根目录
- 仿真模式：目录自动追加 _Sim，二进制、SQLite 完全隔离
- SQLite FutureBase.db 统一存放于数据根目录
- 二进制分层目录：tick/sec5/min1/min5/dayk/log，程序自动递归创建

3.1.2 授权激活步骤

- 调用接口获取本机硬件授权码
- 官方网站 (www.yx10000.com) 自动开通对应版本授权
- 填入授权码，注入授权验证。

3.1.3 落地类型开关（性能核心配置）

可独立开启/关闭任意二进制存储，直接控制 IO 负载：

- Tick 存储开关（区分全量/波动两种模式）
- Sec5 5 秒 K 落地开关
- Min1 1 分钟 K 落地开关
- Min5 5 分钟 K 落地开关
- DayK 日 K 落地开关
- 合约白名单开关：仅白名单合约生成二进制文件
- 非 http 共享数据服务 Min1 和 Min5 可关闭定时刷新（设置为 0）

3.2 进阶配置

- 行情前置实盘/仿真 IP 自定义配置
- TCP、HTTP 服务启停开关
- 本地日志输出开关
- 合约白名单配置，过滤无需落地品种
- 定时任务开关：自动更新主力、夜盘标记写入 SQLite
- TDX 通达信文件扫描根目录配置

四、数据维护引导

4.1 日常数据维护

- ✓ 交易日实时增量写入对应二进制文件
- ✓ 每日收盘自动更新日 K、同步更新 SQLite 主力/夜盘标记
- ✓ 长期运行定期清理日志文件夹释放磁盘空间

4.2 数据修复与重置（区分 SQLite/二进制）

- ✓ 基础信息异常：重新拉取交易所/品种/合约，覆盖 SQLite 表数据
- ✓ 二进制文件损坏：直接删除对应交易日/合约文件，重启订阅自动重生成当日数据
- ✓ SQLite 数据库损坏：删除 FutureBase.db，手动重建基础信息数据，交易所、品种和合约
- ✓ TDX 导入异常：检查文件路径、合约编码匹配，清空目标文件后重新导入

4.3 磁盘与性能维护建议

- ✓ 高频多合约场景必须使用 SSD；机械硬盘仅适合≤30 合约、关闭全量 Tick；
- ✓ 不交易合约及时退订，减少内存缓存对象与二进制文件数量；
- ✓ 按需关闭无用周期落地开关，降低磁盘 IO 负载；
- ✓ 二进制文件仅服务端单线程写入，外部程序只允许只读查询，禁止并发写。

五、SQLite 数据库完整表字典

数据库文件名：FutureBase.db

5.1 实时行情缓存表 MarketCache

仅程序运行内存缓存，程序退出不落地磁盘；存储最新盘口快照，供实时查询
主键：ExchangeID + InstrumentData

字段名	数据类型	字段详细注释
ExchangeID	TEXT	交易所唯一编码 (CZCE/SHFE/DCE/CFFEX/INE/GFEX)
InstrumentID	TEXT	CTP 标准合约代码，如rb2610、au2608
ExchangeInstID	TEXT	交易所内部合约原始编码
TradingDay	TEXT	当前交易日 YYYYMMDD
ActionDay	TEXT	行情业务发生自然日

UpdateTime	TEXT	行情更新时分 HH:MM:SS
UpdateMillisec	INTEGER	行情更新毫秒 0~999
LastPrice	REAL	最新成交价格
AveragePrice	REAL	当日成交均价
PreClosePrice	REAL	昨收盘价格
OpenPrice	REAL	当日开盘价
HighestPrice	REAL	当日最高价
LowestPrice	REAL	当日最低价
ClosePrice	REAL	盘中临时收盘价，收盘后为正式收盘价
Volume	INTEGER	当日累计成交量（手）
Turnover	REAL	当日累计成交额
OpenInterest	INTEGER	当前总持仓量
PreOpenInterest	INTEGER	昨日收盘持仓量
PreSettlementPrice	REAL	昨日结算价
SettlementPrice	REAL	当日结算价（收盘后更新）
UpperLimitPrice	REAL	涨停价格
LowerLimitPrice	REAL	跌停价格
PreDelta	REAL	期权昨日虚实度
CurrDelta	REAL	期权当日虚实度

BandingUpperPrice	REAL	能源中心上带价
BandingLowerPrice	REAL	能源中心下带价
BidPrice1~5	REAL	买一至买五委托价格
BidVolume1~5	INTEGER	买一至买五委托手数
AskPrice1~5	REAL	卖一至卖五委托价格
AskVolume1~5	INTEGER	卖一至卖五委托手数

5.2 交易所信息表 Exchanges

主键：ExchangeID

字段名	数据类型	字段详细注释
ExchangeID	TEXT	交易所编码 CZCE/SHFE/DCE/CFFE X/INE/GFEX
ExchangeName	TEXT	交易所中文全称
ExchangeProperty	TEXT	交易所业务属性描述

5.3 品种信息表 Products

主键：ProductID + ExchangeID

字段名	数据类型	字段详细注释
ProductID	TEXT	品种代码 rb/au/cu
ProductName	TEXT	品种中文名称
ExchangeID	TEXT	所属交易所编码
ProductClass	TEXT	品种类型：期货/期权/组

		合
VolumeMultiple	INTEGER	合约乘数，每手对应标的数量
PriceTick	REAL	最小变动价位
MaxMarketOrderVolume	INTEGER	市价单最大下单手数
MinMarketOrderVolume	INTEGER	市价单最小下单手数
MaxLimitOrderVolume	INTEGER	限价单最大下单手数
MinLimitOrderVolume	INTEGER	限价单最小下单手数
PositionType	TEXT	持仓类型（平今/平昨区分）
PositionDateType	TEXT	持仓日期区分规则
CloseDealType	TEXT	平仓成交匹配规则
TradeCurrencyID	TEXT	交易货币代码
MortgageFundUseRange	TEXT	保证金可用范围说明
UnderlyingMultiple	REAL	期权标的乘数
ExchangeProductID	TEXT	交易所内部品种编码
OpenLimitControlLevel	INTEGER	开仓限额管控等级
OrderFreqControlLevel	INTEGER	报单频率管控等级
IsNightTradingFlag	INTEGER	夜盘标记：1=有夜盘，0=无夜盘
ProductTradStatus	INTEGER	品种交易状态，1 正常交易

5.4 合约信息表 Instruments

主键：InstrumentID + ExchangeID

字段名	数据类型	字段详细注释
InstrumentID	TEXT	完整合约代码 au2608/rb2610
ExchangeID	TEXT	所属交易所编码
InstrumentName	TEXT	合约中文名称
ProductClass	TEXT	品种分类 期货/期权
DeliveryYear	INTEGER	交割年份
DeliveryMonth	INTEGER	交割月份
MaxMarketOrderVolume	INTEGER	市价单最大下单手数
MinMarketOrderVolume	INTEGER	市价单最小下单手数
MaxLimitOrderVolume	INTEGER	限价单最大下单手数
MinLimitOrderVolume	INTEGER	限价单最小下单手数
VolumeMultiple	INTEGER	合约乘数
PriceTick	REAL	最小变动价位
CreateDate	TEXT	合约创建日期 YYYYMMDD
OpenDate	TEXT	合约上市开盘日期
ExpireDate	TEXT	合约到期日
StartDelivDate	TEXT	交割起始日

EndDelivDate	TEXT	交割结束日
InstLifePhase	TEXT	合约生命周期（上市/交易/交割/过期）
IsTrading	INTEGER	是否可交易 1 正常
PositionType	TEXT	持仓类型
PositionDateType	TEXT	今仓昨仓区分
LongMarginRatio	REAL	多头保证金比例
ShortMarginRatio	REAL	空头保证金比例
MaxMarginSideAlgorithm	INTEGER	大额单边保证金计算开关
StrikePrice	REAL	期权行权价（期货无效）
OptionsType	TEXT	认购/认沽期权类型
UnderlyingMultiple	REAL	期权标的乘数
CombinationType	TEXT	组合合约类型
ExchangeInstID	TEXT	交易所内部合约编码
ProductID	TEXT	所属品种代码
UnderlyingInstrID	TEXT	期权对应标的合约代码
IsMainSecond	INTEGER	主力标记：1 主力，2 次主力，0 普通合约

六、通达信 TDX 本地 K 线导入转换

6.1 功能概述

读取通达信软件本地 `.day/.1min/.5min` 二进制 K 线文件，自动转换为本系统标准 `KlineData` 结构体，写入 `Min1/Min5/DayK` 环形/单交易日二进制文件，用于历史回测补全缺失数据。

6.2 数据转换规则

- ✧ TD 复合 timestamp 编码 → 转换为 UTC+8 标准 Unix 秒赋值给 timestamp;
- ✧ TD 浮点价格、成交量、成交额映射到标准 K 线对应字段;
- ✧ TD 无持仓变化、成交笔数据，固定填充=0;
- ✧ 按合约 Key 匹配写入对应 min1/min5/dayk 二进制文件;

6.3 使用约束

- ✧ 仅支持期货 TD 文件，股票、期权文件不兼容;
- ✧ 导入前建议关闭对应合约实时落地，避免实时数据与导入历史数据冲突覆盖;
- ✧ 接口为增量追加模式，不存在重复追加;
- ✧ TD 合约编码必须和 CTP 标准完全一致，不匹配合约直接跳过不导入;
- ✧ 导入完成可调用数据中心接口读取对应周期数据校验结果。

七、四大使用模式详细教程

7.1 模式一：本地文件直读

静态信息读取 SQLite，行情时序直接解析二进制文件，无行情数据库依赖，纯本地离线读取，无网络、无服务进程即可使用。适用于历史数据复盘、离线数据校验、无网络环境下的数据查看场景。

7.2 模式二：接口函数类调用（DLL 函数调用）

本模式适配 C++、C#、Python、易语言等所有支持 DLL 调用的开发场景，是所

有对接方式中**最常用、最稳定、性能最高**的对接方案，本地无网络开销，直接读取本地 SQLite 静态库与二进制时序数据文件。

7.2.1 核心调用流程

- 初始化服务：完成数据目录、授权密钥、缓存策略等基础配置；
- 行情登录：连接 CTP 行情前置，完成实盘/仿真环境登录校验；
- 订阅合约：按需订阅目标期货、期权合约，开启行情落地；
- 基础信息入库：一键拉取交易所、品种、合约静态信息写入 SQLite 数据库；
- 调用查询函数：通过统一 DLL 接口查询各类静态信息、实时行情、历史时序数据。

7.2.2 核心查询函数声明

```
C++
/**
 * @brief 统一本地数据库查询入口
 * @param[out] outDataPtr 输出堆内存结构体数组；调用完成必须
delete[]释放
 * @param dbDataTypeStr 查询数据类型标识：
Exchanges/Products/Instruments/MarketCache/Ticks/Sec5Kline/Min1Kline/Min5Kline/DayKline
 * @param param1 主筛选条件，多个编码逗号分隔
 * @param param2 次要筛选（交易日等）
 * @param param3 扩展过滤参数
 * @param intParam 数字参数：返回条数、偏移量
 * @param boolParam true 仅期货 false 包含期权
 * @param IsOutJson true 输出 JSON false 输出二进制结构体
 * @return >=0 有效记录条数；负数代表查询失败
 */
virtual TAPIINT32 WINAPI QueryFutureData(void*& outDataPtr,
char* dbDataTypeStr, char* param1, char* param2, char* param3, int
intParam, bool boolParam, bool IsOutJson);

/**
 * @brief 获取主力/次主力合约
 * @param contractType 1=主力合约 2=次主力合约
 * @return 逗号分隔合约字符串（动态内存，调用方释放）
 */
virtual char* WINAPI GetMainSubContract(int contractType);
```

```

/**
 * @brief 获取所有支持夜盘的品种列表
 * @return 静态字符串，无需释放
 */
virtual char* WINAPI GetNightProductList(void);

/**
 * @brief 获取所有夜盘合约完整列表
 * @return 静态字符串，无需释放
 */
virtual char* WINAPI GetNightInstrumentList(void);

/**
 * @brief 获取夜盘品种主力/次主力合约
 * @param contractType 1 主力，2 次主力
 * @return 静态字符串
 */
virtual char* WINAPI GetNightMainSubInstrument(int
contractType);

```

7.2.3 通用查询数据结构

所有 DLL 查询接口统一返回二进制结构体数组或 UTF8 标准 JSON 字符串，所有数据结构采用 1 字节紧凑对齐，完美适配易语言、C#、Python 等跨语言解析。

7.2.3.1 行情数据 (FixedQuoteData)

```

C++
struct FixedQuoteData {
    char InstrumentID[81];           ///< 合约代码
    char ExchangeInstID[81];         ///< 合约在交易所内代码
    char ExchangeID[9];              ///< CTP 标准交易所代码

    char TradingDay[9];              ///< 交易日 YYYYMMDD
    char ActionDay[9];               ///< 业务日期
    char UpdateTime[9];              ///< 更新时间 HH:MM:SS
    int UpdateMillisec;              ///< 更新毫秒

    double LastPrice;                ///< 最新价
    double AveragePrice;             ///< 当日均价

```

```

double PreClosePrice;          ///< 昨收盘
double OpenPrice;              ///< 今开盘
double HighestPrice;           ///< 最高价
double LowestPrice;            ///< 最低价
double ClosePrice;             ///< 今收盘
int Volume;                    ///< 成交量
double Turnover;               ///< 成交金额
long long OpenInterest;        ///< 当前持仓量
long long PreOpenInterest;     ///< 昨持仓量
double PreSettlementPrice;     ///< 上次结算价
double SettlementPrice;        ///< 本次结算价
double UpperLimitPrice;        ///< 涨停价
double LowerLimitPrice;        ///< 跌停价
double PreDelta;               ///< 昨虚实度（期权）
double CurrDelta;              ///< 今虚实度（期权）
double BandingUpperPrice;       ///< 上带价（能源中心）
double BandingLowerPrice;      ///< 下带价（能源中心）

// 五档买卖盘
double BidPrice1;
int BidVolume1;
double AskPrice1;
int AskVolume1;
double BidPrice2;
int BidVolume2;
double AskPrice2;
int AskVolume2;
double BidPrice3;
int BidVolume3;
double AskPrice3;
int AskVolume3;
double BidPrice4;
int BidVolume4;
double AskPrice4;
int AskVolume4;
double BidPrice5;
int BidVolume5;
double AskPrice5;
int AskVolume5;
};

```

7.2.3.2 K 线数据 (KlineData)

```

C++
struct KlineData {
    uint64_t    timestamp;           ///< K 线周期结束 Unix 秒时间戳
    double      openPrice;           ///< 开盘价
    double      highestPrice;        ///< 最高价
    double      lowestPrice;         ///< 最低价
    double      closePrice;          ///< 收盘价
    uint64_t    volume;              ///< 周期成交量
    double      turnover;            ///< 周期成交额
    int         positionChange;      ///< 周期持仓变化
    int         tradeCount;          ///< 周期成交次数
};

```

7.2.3.3 Tick 数据 (TickData)

```

C++
struct TickData {
    int64_t timestampSec;            ///< Unix 秒级时间戳
    uint16_t millisec;              ///< 毫秒 0~999
    double lastPrice;               ///< 最新价
    double openPrice;               ///< 开盘价
    double highestPrice;            ///< 最高价
    double lowestPrice;             ///< 最低价
    double PreClosePrice;           ///< 昨收价
    double averagePrice;            ///< 当日均价
    uint32_t volume;                ///< 现量 (增量成交量)
    uint64_t openInterest;          ///< 当前持仓量
    uint64_t totalVolume;           ///< 累计总成交量
    double totalTurnover;           ///< 累计总成交额
    double bidPrice1;               ///< 买一价
    int32_t bidVolume1;             ///< 买一量
    double askPrice1;               ///< 卖一价
    int32_t askVolume1;             ///< 卖一量
};

```

7.2.3.4 分时数据 (MinData)

```

C++
struct MinData {
    uint64_t    timestamp;           ///< Unix 秒时间戳

```

```

double      NewPrice;          ///< 当前价格
int         positionChange;    ///< 持仓变化
uint64_t    volume;           ///< 周期成交量
double      turnover;         ///< 周期成交额
};

```

7.2.3.5 交易所信息 (CThostFtdcExchangeField)

```

C++
///交易所
struct CThostFtdcExchangeField
{
char      ExchangeID[9];      ///< 交易所代码
char      ExchangeName[61];   ///< 交易所名称
char      ExchangeProperty;   ///< 交易所属性
};

```

7.2.3.6 品种信息 (ProductInfoData)

```

C++
struct ProductInfoData
{
char      reserve1[31];        ///< 预留无效填充字段
char      ProductName[21];     ///< 品种中文名称
char      ExchangeID[9];       ///< 交易所代码
CZCE/SHFE/DCE/CFFEX/INE/GFEX
char      ProductClass;        ///< 品种类型：期货、期权、
组合等
int        VolumeMultiple;     ///< 合约乘数
double     PriceTick;          ///< 最小变动价位
int        MaxMarketOrderVolume; ///< 市价单最大下单手数
int        MinMarketOrderVolume; ///< 市价单最小下单手数
int        MaxLimitOrderVolume; ///< 限价单最大下单手数
int        MinLimitOrderVolume; ///< 限价单最小下单手数
char      PositionType;        ///< 持仓类型
char      PositionDateType;    ///< 持仓日期类型（今仓/昨
仓）
char      CloseDealType;       ///< 平仓处理方式
char      TradeCurrencyID[4];  ///< 交易币种
char      MortgageFundUseRange; ///< 质押资金可用范围
};

```

```

char        reserve2[31];          ///< 预留无效填充字段 2
double      UnderlyingMultiple;    ///< 标的乘数（期权使用）
char        ProductID[81];         ///< 品种代码 rb/au/cu
char        ExchangeProductID[81]; ///< 交易所内品种代码
char        OpenLimitControlLevel; ///< 开仓限额控制粒度
char        OrderFreqControlLevel; ///< 报单频率限制粒度
char        IsNightTradingFlag;    ///< 【自定义字段】夜盘标记 0=
无夜盘 1=有夜盘
char        ProductTradStatus;     ///< 【自定义字段】品种交易
状态
};

```

7.2.3.7 合约信息 (InstrumentData)

```

C++
struct InstrumentData
{
    char        reserve1[31];          ///< 预留无效填充字段
    char        ExchangeID[9];         ///< 交易所代码
    char        InstrumentName[21];     ///< 合约中文名称
    char        reserve2[31];          ///< 预留无效填充字段
    char        reserve3[31];          ///< 预留无效填充字段
    char        ProductClass;          ///< 品种类型
    int         DeliveryYear;           ///< 交割年份
    int         DeliveryMonth;          ///< 交割月份
    int         MaxMarketOrderVolume;  ///< 市价单最大下单量
    int         MinMarketOrderVolume;  ///< 市价单最小下单量
    int         MaxLimitOrderVolume;   ///< 限价单最大下单量
    int         MinLimitOrderVolume;   ///< 限价单最小下单量
    int         VolumeMultiple;        ///< 合约乘数
    double      PriceTick;              ///< 最小变动价位
    char        CreateDate[9];          ///< 合约创建日期 YYYYMMDD
    char        OpenDate[9];            ///< 上市日期 YYYYMMDD
    char        ExpireDate[9];          ///< 到期日 YYYYMMDD
    char        StartDelivDate[9];      ///< 开始交割日
    char        EndDelivDate[9];        ///< 结束交割日
    char        InstLifePhase;          ///< 合约生命周期状态
    char        IsTrading;              ///< 是否可交易 (0=否 1=是)
    char        PositionType;           ///< 持仓类型
    char        PositionDateType;      ///< 持仓日期类型
};

```

```

double      LongMarginRatio;        ///< 多头保证金率
double      ShortMarginRatio;       ///< 空头保证金率
char        MaxMarginSideAlgorithm; ///< 是否启用大额单边保证金算
法
char        reserve4[31];            ///< 预留无效填充字段
double      StrikePrice;             ///< 期权行权价（期权合约有
效）
char        OptionsType;             ///< 期权类型（认购/认沽）
double      UnderlyingMultiple;     ///< 标的乘数
char        CombinationType;        ///< 组合合约类型
char        InstrumentID[81];       ///< 合约代码 au2608/rb2610
char        ExchangeInstID[81];    ///< 交易所内部合约代码
char        ProductID[81];         ///< 所属品种代码
char        UnderlyingInstrID[81]; ///< 期权标的合约代码
char        IsMainSecond;           ///< 【自定义字段】主力次主力标
记：1 主力、2 次主力、0 普通合约
};

```

7.3 模式三：TCP 连接类（长连接服务）

7.3.1 使用场景

适用于多进程共享数据、局域网/外网多设备远程读取数据、需要稳定长连接实时查询、高频连续请求的业务场景，相比 HTTP 短连接，TCP 长连接无需重复建立连接，延迟更低、稳定性更强。

7.3.2 使用方式

- 启动 FDAQI 服务端，开启 TCP 监听服务；
- 客户端通过 Socket 连接服务端指定 IP（本机 127.0.0.1/局域网 IP/公网 IP）+固定端口 7445；
- 客户端按照固定数据包结构构造请求包并发送；
- 服务端接收请求、解析参数并查询本地数据，返回标准化响应数据包；
- 客户端解析响应包，读取二进制数据或状态信息。

7.3.3 TCP 连接查询数据类型枚举

C++

```

typedef enum
{
    QueryType_None = 0,          ///< 无效/未知查询类型

    QueryType_HeartBeat = 1,     ///< 客户端心跳包，维持长连
接
    QueryType_ServerMessage = 2, ///< 服务端主动推送文本消息
    QueryType_Connect = 3,       ///< 客户端连接建立通知（预
留）
    QueryType_Disconnect = 4,    ///< 服务端下发断开连接通知

    QueryType_ExchangeQuery = 10, ///< 查询交易所基础信息
    QueryType_ProductQuery = 11,  ///< 查询品种基础信息
    QueryType_InstrumentQuery = 12, ///< 查询合约基础信息

    QueryType_MarketDataCache = 20, ///< 查询实时行情缓存快照
    QueryType_TickData = 21,        ///< 查询逐笔 Tick 历史数据

    QueryType_TimeShare = 30,      ///< 分时线（1 分钟 K 合成）
    QueryType_Sec5Kline = 31,     ///< 5 秒 K 线
    QueryType_Min1Kline = 32,     ///< 1 分钟 K 线
    QueryType_Min5Kline = 33,     ///< 5 分钟 K 线
    QueryType_DayKline = 34,      ///< 日 K 线
} QueryTypeEnum;

```

7.3.4 TCP 请求数据包结构（客户端→服务端）

```

C++
typedef struct _SocketReqPacket
{
    DWORD    sessionTime;          ///< 会话时间戳，客户端自行填充
    DWORD    sessionId;           ///< 客户端会话唯一标识，响应包
原样带回
    int      queryType;           ///< 查询类型，参照
QueryTypeEnum

    char     instIds[4096];        ///< 合约 ID 列表，多个使用逗号
分隔
    char     exchanges[64];       ///< 交易所筛选，逗号分隔
    char     product[512];        ///< 品种代码筛选

```

```
bool    isOption;                ///< true=同时返回期货+期权; false=仅期货合约
bool    sortAsc;                ///< true=时间升序; false=时间降序
int     count;                  ///< 请求最大返回条数, <=0 代表读取全部

char     tradingDay[9];          ///< 交易日 YYYYMMDD, 固定 8 字符 + 字符串结束符
char     reserve[128];           ///< 预留扩展缓冲区
} SocketReqPacket;
```

7.3.5 TCP 响应数据包结构（服务端→客户端）

```
C++
typedef struct _SocketRespPacket
{
    DWORD    sessionTime;        ///< 客户端请求原始会话时间戳
    DWORD    sessionId;          ///< 客户端唯一会话 ID, 原样回传
    int      queryType;           ///< 查询类型 QueryTypeEnum

    char     clientIp[32];        ///< 客户端 IP 字符串
    int      resultCode;          ///< 执行结果状态码
    char     serverMsg[512];      ///< 提示文本/错误描述
    char     reserve[128];        ///< 预留扩展字段

    int      recordCount;         ///< 返回有效记录条数
    int      dataLen;             ///< 后方柔性数组二进制总字节长度
    BYTE     dataBuf[1];          ///< 柔性数组, 承载结构体数组原始二进制
} SocketRespPacket;
```

7.3.6 包构造与解析说明

包构造规则：客户端需完整初始化请求结构体，填充会话时间戳、唯一会话 ID、对应查询类型及筛选参数，未使用字段默认置空，保证结构体内存完整对齐后发送。心跳包仅需填充 sessionTime、sessionId、queryType=1，其余参数置空，用于维持

长连接不中断。

包解析规则：客户端接收固定大小响应包头后，优先读取 dataLen 字段，根据数据长度截取后续二进制数据流，结合对应业务结构体（KlineData/TickData/合约信息等）批量解析数据；通过 resultCode 判断请求结果，非 0 为查询异常，可读取 serverMsg 获取报错详情。

7.4 模式四：HTTP 访问类（网络接口）

7.4.1 使用场景

适用于网页读取、Python 脚本快速调用、第三方系统对接、无依赖快速查询、跨设备简易数据获取场景，无需复杂 SDK 和 DLL 依赖，仅通过 HTTP GET 请求即可获取标准化 JSON 数据，接入门槛极低。

7.4.2 访问方式

启动 FDAQI HTTP 服务后，默认监听本机 9713 端口，支持局域网、公网访问，全部接口采用 GET 请求，统一返回 **UTF-8 标准 JSON 格式数据**，底层数据读取逻辑与 DLL 调用一致，读取 SQLite 静态基础信息与本地二进制时序数据。

7.4.3 完整 HTTP 接口访问范例

1. 交易所信息查询

http://127.0.0.1:9713/?type=Exchanges&Key=fdapi_testkey

http://127.0.0.1:9713/?type=Exchanges&Key=fdapi_testkey&Exchange=CZCE,DCE

2. 品种信息查询

http://127.0.0.1:9713/?type=Products&Key=fdapi_testkey&Exchange=DCE

http://127.0.0.1:9713/?type=Products&Key=fdapi_testkey&Product=FG

http://127.0.0.1:9713/?type=Products&Key=fdapi_testkey

3. 合约信息查询

http://127.0.0.1:9713/?type=Instruments&Key=fdapi_testkey&Product=FG

http://127.0.0.1:9713/?type=Instruments&Key=fdapi_testkey&Exchange=DCE

http://127.0.0.1:9713/?type=Instruments&Key=fdapi_testkey&InstrumentID=FG609,jm2609,SH609

4. 实时行情缓存查询

http://127.0.0.1:9713/?type=MarketCache&Key=fdapi_testkey&InstrumentID=FG609,jm2609,SH609

http://127.0.0.1:9713/?type=MarketCache&Key=fdapi_testkey

5. Tick 逐笔数据查询

http://127.0.0.1:9713/?type=Ticks&Key=fdapi_testkey&InstrumentID=jm2609

http://127.0.0.1:9713/?type=Ticks&Key=fdapi_testkey&InstrumentID=jm2609&TradingDay=20260630&Count=20

6. 各周期 K 线数据查询

http://127.0.0.1:9713/?type=Sec5Kline&Key=fdapi_testkey&InstrumentID=jm2609&Count=200

http://127.0.0.1:9713/?type=Min1Kline&Key=fdapi_testkey&InstrumentID=jm2609&Count=100

http://127.0.0.1:9713/?type=Min5Kline&Key=fdapi_testkey&InstrumentID=jm2609&Count=80

http://127.0.0.1:9713/?type=DayKline&Key=fdapi_testkey&InstrumentID=jm2609

7. 分时线数据查询

http://127.0.0.1:9713/?type=TimeMinLine&Key=fdapi_testkey&InstrumentID=jm2609

八、其他注意事项

1. 合约代码严格区分大小写，全程遵循 CTP 原始编码规范使用。
2. 实盘、仿真运行目录自动物理隔离，对应的二进制行情文件互不覆盖、互不干扰。
3. SQLite 数据库仅支持单进程写入静态基础数据表，支持多进程只读访问；二进制数据文件仅允许服务端单线程写入，所有外部程序仅可只读调用，禁止写入操作。
4. Fdapi 查询接口返回数据为 outDataPtr 指针，仅在当前线程本次查询有效；
5. K 线聚合切分规则：仅当 Tick 数据时间秒数大于 K 线桶结束秒数时，才会完成周期切分，23:00 收盘后的延迟 Tick 数据，不会生成 23:01、23:05 等无效伪 K 线。
6. 数据文件类型区分规则：Tick、Sec5 数据为单交易日独立文件，Min1、Min5 数据为跨交易日环形存储文件。
7. TDX 通达信 K 线导入为增量追加模式，重复执行只增加增量 K 线数据，支持一分钟、五分钟 and 日 K 线数据导入。
8. 系统性能核心调控规则：同步调整订阅合约数量与数据落地类型组合，可有效降低程序内存占用、减少磁盘 IO 负载，优化整体运行性能。