FinSage v2.0 技术文档

Multi-Agent Financial Portfolio Management System

多智能体金融组合管理系统

文档版本: 2.0
最后更新: 2024年12月
适用版本: FinSage v2.0+

目录

1. 系统架构总览

1.1 架构图

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                          FinSage v2.0 系统架构                              │
├─────────────────────────────────────────────────────────────────────────────┤
│                                                                             │
│  ┌───────────────────────────────────────────────────────────────────────┐  │
│  │              第一层: 专家智能体层 (Expert Layer)                      │  │
│  │                                                                       │  │
│  │  ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐             │  │
│  │  │ Stock  │ │ Bond   │ │Commodity│ │ REITs  │ │ Crypto │             │  │
│  │  │ Expert │ │ Expert │ │ Expert │ │ Expert │ │ Expert │             │  │
│  │  │(87K+)  │ │(2.1K+) │ │(985+)  │ │(540+)  │ │(4.7K+) │             │  │
│  │  └───┬────┘ └───┬────┘ └───┬────┘ └───┬────┘ └───┬────┘             │  │
│  │      └──────────┴────┬─────┴──────────┴──────────┘                  │  │
│  │                      ▼                                               │  │
│  │         ┌──────────────────────────────────────┐                     │  │
│  │         │  FactorEnhancedExpertMixin           │                     │  │
│  │         │  LLM分析(60%) + 因子评分(40%)        │                     │  │
│  │         └──────────────────────────────────────┘                     │  │
│  └───────────────────────────────────────────────────────────────────────┘  │
│                                  │                                          │
│                                  ▼                                          │
│  ┌───────────────────────────────────────────────────────────────────────┐  │
│  │              第二层: 管理层 (Manager Layer)                           │  │
│  │                                                                       │  │
│  │     ┌────────────────┐   ┌────────────────┐   ┌────────────────┐     │  │
│  │     │   Portfolio    │   │   Position     │   │    Hedging     │     │  │
│  │     │   Manager      │   │  Sizing Agent  │   │     Agent      │     │  │
│  │     │  (资产配置)    │   │  (仓位规模)    │   │  (对冲策略)    │     │  │
│  │     └───────┬────────┘   └───────┬────────┘   └───────┬────────┘     │  │
│  │             └────────────────────┼────────────────────┘              │  │
│  │                                  ▼                                    │  │
│  │                       Manager Coordinator                             │  │
│  │                       (多轮讨论达成共识)                              │  │
│  └───────────────────────────────────────────────────────────────────────┘  │
│                                  │                                          │
│                                  ▼                                          │
│  ┌───────────────────────────────────────────────────────────────────────┐  │
│  │              第三层: 风控层 (Risk Layer)                    │    │
│  │  硬性约束(Veto) → 软性约束(Warning) → 盘中监控(Alert)        │    │
│  └──────────────────────────────────────────────────────────────┘    │
│                                  │                                   │
│                                  ▼                                   │
│  ┌──────────────────────────────────────────────────────────────┐    │
│  │              第四层: 执行层 (Execution Layer)                │    │
│  │  订单生成 → 成本计算(佣金+滑点) → 组合状态更新               │    │
│  └──────────────────────────────────────────────────────────────┘    │
│                                                                       │
│  ┌──────────────────────────────────────────────────────────────┐    │
│  │                    支撑层 (Infrastructure)                    │    │
│  │  ┌──────────┐  ┌───────────┐  ┌─────────────┐  ┌─────────────┐  │    │
│  │  │   LLM    │  │ FMP Ultra │  │   Hedging   │  │  Checkpoint │  │    │
│  │  │ Provider │  │  数据层   │  │   Toolkit   │  │   System    │  │    │
│  │  │ OpenAI/  │  │ 96,000+   │  │  11种工具   │  │  断点续传   │  │    │
│  │  │ Claude   │  │ 全市场资产│  │ CVaR/风险平价│  │ 状态恢复    │  │    │
│  │  └──────────┘  └───────────┘  └─────────────┘  └─────────────┘  │    │
│  └──────────────────────────────────────────────────────────────┘    │
└──────────────────────────────────────────────────────────────────────┘
```

1.2 数据流图

```
┌─────────────────────────────────────────────────────────────────────────┐
│                              数据处理流程                                 │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│   ┌─────────────────────────────────────────────────────────────────┐   │
│   │  1. 数据获取 (FMP Ultra API)                                      │   │
│   ├─────────────────────────────────────────────────────────────────┤   │
│   │  FMPClient.get_full_asset_universe()                            │   │
│   │    ├── /stock-list (87,761+)                                    │   │
│   │    ├── /cryptocurrency-list (4,786+)                            │   │
│   │    ├── /commodities-list (40+) + /etf-list [commodity] (945+)   │   │
│   │    ├── /company-screener [sector=Real Estate] (540+)            │   │
│   │    └── /etf-list [bond] (2,165+)                                │   │
│   └─────────────────────────────────────────────────────────────────┘   │
│                                  │                                      │
│                                  ▼                                      │
│   ┌─────────────────────────────────────────────────────────────────┐   │
│   │  2. 并行专家分析                                                   │   │
│   ├─────────────────────────────────────────────────────────────────┤   │
│   │  ThreadPoolExecutor (max_workers=5)                             │   │
│   │    ├── StockExpert.analyze()     → ExpertReport                 │   │
│   │    ├── BondExpert.analyze()      → ExpertReport                 │   │
│   │    ├── CommodityExpert.analyze() → ExpertReport                 │   │
│   │    ├── REITsExpert.analyze()     → ExpertReport                 │   │
│   │    └── CryptoExpert.analyze()    → ExpertReport                 │   │
│   └─────────────────────────────────────────────────────────────────┘   │
│                                  │                                      │
│                                  ▼                                      │
│   ┌─────────────────────────────────────────────────────────────────┐   │
│   │  3. 管理层协调 (2轮讨论)                                            │   │
│   ├─────────────────────────────────────────────────────────────────┤   │
│   │  ManagerCoordinator.coordinate()                                │   │
│   │    Phase 1: 并行分析                                              │   │
│   │      ├── PortfolioManager.decide()   → PortfolioDecision        │   │
│   │      ├── PositionSizingAgent.analyze() → PositionSizingDecision │   │
│   │      └── HedgingAgent.analyze()      → HedgingDecision          │   │
│   │    Phase 2: 讨论修正 (max 2 rounds)                                │   │
│   │      ├── revise_based_on_feedback()                             │   │
│   │      └── consensus check: sizing_diff<5% && hedge_diff<2%       │   │
│   │    Phase 3: 整合输出                                              │   │
│    │       └── IntegratedDecision                                     │    │
│    └──────────────────────────────────────────────────────────────────┘    │
│                                    │                                       │
│                                    ▼                                       │
│    ┌──────────────────────────────────────────────────────────────────┐    │
│    │  4. 风控检查 & 执行                                              │    │
│    ├──────────────────────────────────────────────────────────────────┤    │
│    │  RiskController.check()                                          │    │
│    │     ├── 硬性约束: max_single_asset=15%, max_drawdown=15%         │    │
│    │     └── 软性约束: target_volatility=12%                          │    │
│    │  ExecutionEngine.execute()                                       │    │
│    │     ├── 订单生成                                                 │    │
│    │     ├── 成本计算 (佣金0.1% + 滑点0.05%)                          │    │
│    │     └── 组合状态更新                                             │    │
│    └──────────────────────────────────────────────────────────────────┘    │
│                                                                            │
└────────────────────────────────────────────────────────────────────────────┘
```

1.3 核心设计原则

原则	描述	实现方式
分层解耦	各层职责明确	专家→管理→风控→执行
LLM驱动	核心决策由LLM推理	GPT-4o-mini + 结构化提示词
多智能体协作	并行讨论达成共识	ThreadPoolExecutor + 多轮修正
风险优先	三层风控机制	硬性约束 + 软性优化 + 盘中监控
全市场覆盖	96,000+资产自由选择	FMP Ultra API + 因子筛选
因子增强	学术因子补充LLM判断	LLM 60% + 因子 40%

1.4 项目目录结构

```
FinSage/
├── main.py                     # 主入口
├── finsage/
│   ├── config.py               # 配置类定义
│   ├── agents/                 # 智能体模块
│   │   ├── base_expert.py      # 专家基类 (Action, ExpertReport)
│   │   ├── portfolio_manager.py # 组合经理 (PortfolioDecision)
│   │   ├── position_sizing_agent.py  # 仓位规模 (PositionSizingDecision)
│   │   ├── hedging_agent.py    # 对冲策略 (HedgingDecision)
│   │   ├── manager_coordinator.py  # 管理协调器 (IntegratedDecision)
│   │   ├── factor_enhanced_expert.py  # 因子增强混入类
│   │   └── experts/            # 5位专家
│   │       ├── stock_expert.py
│   │       ├── bond_expert.py
│   │       ├── commodity_expert.py
│   │       ├── reits_expert.py
│   │       └── crypto_expert.py
│   ├── factors/                # 因子评分模块
│   │   ├── base_factor.py      # 因子基类
│   │   ├── stock_factors.py    # 股票FF-5因子
│   │   ├── bond_factors.py     # 债券因子
│   │   ├── commodity_factors.py # 商品因子
│   │   ├── reits_factors.py    # REITs因子
│   │   └── crypto_factors.py   # 加密货币因子
│   ├── data/                   # 数据层
│   │   ├── fmp_client.py       # FMP Ultra API客户端
│   │   ├── data_loader.py      # 数据加载器
│   │   ├── enhanced_data_loader.py  # 增强数据加载器
│   │   └── market_data.py      # 市场数据提供者
│   ├── hedging/                # 对冲工具包
│   │   ├── toolkit.py          # 对冲工具箱
│   │   ├── dynamic_selector.py # 动态对冲选择器
│   │   └── tools/              # 11种对冲工具
│   │       ├── risk_parity.py
│   │       ├── black_litterman.py
│   │       ├── cvar_optimization.py
│   │       └── ...
│   ├── risk/                   # 风控模块
│   │   └── risk_controller.py
│   └── core/
│       └── orchestrator.py      # 核心协调器
├── .env                         # 环境变量 (API Keys)
└── results/                     # 输出结果
```

2. 核心配置系统

文件: `finsage/config.py`

2.1 配置类层次

```
FinSageConfig (主配置)
├── LLMConfig          # LLM配置
├── TradingConfig      # 交易配置
├── RiskConfig         # 风控配置
├── AssetConfig        # 资产配置
└── DataConfig         # 数据配置
    └── FMPConfig      # FMP Ultra API配置
```

2.2 LLMConfig

```
@dataclass
class LLMConfig:
    """LLM配置"""
    provider: str = "openai"      # "openai", "anthropic", "local"
    model: str = "gpt-4o-mini"
    api_key: Optional[str] = None
    temperature: float = 0.7
    max_tokens: int = 2000
    timeout: int = 60

    def __post_init__(self):
        if self.api_key is None:
            self.api_key = os.environ.get("OA_OPENAI_KEY") or os.environ.get("OPENAI_AP
```

参数	默认值	说明
provider	"openai"	LLM提供商 (openai/anthropic/local)
model	"gpt-4o-mini"	模型名称
temperature	0.7	温度参数 (0-1)
max_tokens	2000	最大输出token
timeout	60	超时(秒)

2.3 TradingConfig

```
@dataclass
class TradingConfig:
    """交易配置"""
    initial_capital: float = 1_000_000.0
    transaction_cost: float = 0.001     # 0.1%
    slippage: float = 0.0005            # 0.05%
    min_trade_value: float = 100.0
    rebalance_frequency: str = "daily"  # "daily", "weekly", "monthly"
    rebalance_threshold: float = 0.02   # 2%触发再平衡
```

参数	默认值	说明
initial_capital	1,000,000	初始资金
transaction_cost	0.001	交易成本(0.1%)
slippage	0.0005	滑点(0.05%)
rebalance_frequency	"daily"	再平衡频率
rebalance_threshold	0.02	再平衡阈值(2%)

2.4 RiskConfig

```python
@dataclass
class RiskConfig:
    """风控配置"""
    # 硬性约束 (Veto - 违反则拒绝)
    max_single_asset: float = 0.15       # 单资产上限15%
    max_asset_class: float = 0.50        # 资产类别上限50%
    max_drawdown_trigger: float = 0.15   # 最大回撤触发15%
    max_portfolio_var_95: float = 0.03   # 组合VaR上限3%

    # 软性约束 (Warning - 违反则警告)
    target_volatility: float = 0.12      # 目标波动率12%
    max_correlation_cluster: float = 0.60  # 相关性聚集上限
    min_diversification_ratio: float = 1.2  # 最小分散化比率
```

类型	参数	默认值	说明
硬性	max_single_asset	0.15	单资产上限(15%)
硬性	max_asset_class	0.50	资产类别上限(50%)
硬性	max_drawdown_trigger	0.15	回撤触发(15%)
硬性	max_portfolio_var_95	0.03	VaR上限(3%)
软性	target_volatility	0.12	目标波动率(12%)
软性	max_correlation_cluster	0.60	相关性上限(60%)
软性	min_diversification_ratio	1.2	分散化比率

2.5 AssetConfig

```
@dataclass
class AssetConfig:
    """资产配置"""
    allocation_bounds: Dict[str, Dict[str, float]] = field(default_factory=lambda: {
        "stocks": {"min": 0.30, "max": 0.50, "default": 0.40},
        "bonds": {"min": 0.15, "max": 0.35, "default": 0.25},
        "commodities": {"min": 0.10, "max": 0.25, "default": 0.15},
        "reits": {"min": 0.05, "max": 0.15, "default": 0.10},
        "crypto": {"min": 0.00, "max": 0.10, "default": 0.05},
        "cash": {"min": 0.02, "max": 0.15, "default": 0.05},
    })

    default_universe: Dict[str, List[str]] = field(default_factory=lambda: {
        "stocks": ["SPY", "QQQ", "IWM", "AAPL", "MSFT", "GOOGL", "AMZN", "NVDA"],
        "bonds": ["TLT", "IEF", "SHY", "LQD", "HYG", "AGG"],
        "commodities": ["GLD", "SLV", "USO", "DBA", "COPX"],
        "reits": ["VNQ", "IYR", "DLR", "EQIX"],
        "crypto": ["BTC-USD", "ETH-USD"],
    })
```

类别	最小	最大	默认	可选数量 (FMP)
stocks	30%	50%	40%	87,761+
bonds	15%	35%	25%	2,165+ ETF
commodities	10%	25%	15%	985+ (期货+ETF)
reits	5%	15%	10%	540+
crypto	0%	10%	5%	4,786+
cash	2%	15%	5%	-

2.6 FMPConfig (数据层核心)

```python
@dataclass
class FMPConfig:
    """FMP Ultra API 配置"""
    api_key: Optional[str] = None
    base_url: str = "https://financialmodelingprep.com/stable"
    tier: str = "ultra"

    endpoints: Dict[str, str] = field(default_factory=lambda: {
        # Stock Screener
        "company_screener": "/company-screener",

        # Market Data
        "quote": "/quote",
        "batch_quote": "/batch-quote",
        "stock_list": "/stock-list",
        "etf_list": "/etf-list",
        "profile": "/profile",

        # Multi-Asset
        "cryptocurrency_list": "/cryptocurrency-list",
        "commodities_list": "/commodities-list",

        # Financial Statements
        "income_statement": "/income-statement",
        "balance_sheet": "/balance-sheet-statement",
        "cash_flow": "/cash-flow-statement",

        # Key Metrics & Ratios
        "key_metrics_ttm": "/key-metrics-ttm",
        "ratios_ttm": "/ratios-ttm",
        "financial_scores": "/financial-scores",

        # Historical Data
        "historical_price": "/historical-price-eod/full",

        # News
        "stock_news": "/stock-news",
    })

    rate_limit: Dict[str, int] = field(default_factory=lambda: {
        "requests_per_minute": 750,  # Ultra tier
        "batch_size": 100,
        "delay_between_batches": 0.1,
    })

    def get_url(self, endpoint: str, **params) -> str:
        """构建完整的 API URL"""
        url = f"{self.base_url}{self.endpoints[endpoint]}"
        params["apikey"] = self.api_key
        query_string = "&".join(f"{k}={v}" for k, v in params.items() if v is not None)
        return f"{url}?{query_string}"
```

2.7 预定义配置模板

```python
# 默认配置
config = FinSageConfig()

# 保守配置 - 更低波动率，更高债券比例
CONSERVATIVE_CONFIG = FinSageConfig(
    risk=RiskConfig(
        max_single_asset=0.10,
        max_asset_class=0.40,
        target_volatility=0.08,
    ),
    assets=AssetConfig(
        allocation_bounds={
            "stocks": {"min": 0.20, "max": 0.40, "default": 0.30},
            "bonds": {"min": 0.30, "max": 0.50, "default": 0.40},
            "commodities": {"min": 0.05, "max": 0.15, "default": 0.10},
            "reits": {"min": 0.05, "max": 0.10, "default": 0.08},
            "crypto": {"min": 0.00, "max": 0.02, "default": 0.02},
            "cash": {"min": 0.05, "max": 0.20, "default": 0.10},
        }
    ),
)

# 激进配置 - 更高股票和加密货币比例
AGGRESSIVE_CONFIG = FinSageConfig(
    risk=RiskConfig(
        max_single_asset=0.20,
        max_asset_class=0.60,
        target_volatility=0.18,
    ),
    assets=AssetConfig(
        allocation_bounds={
            "stocks": {"min": 0.40, "max": 0.70, "default": 0.55},
            "bonds": {"min": 0.05, "max": 0.20, "default": 0.10},
            "commodities": {"min": 0.10, "max": 0.25, "default": 0.15},
            "reits": {"min": 0.05, "max": 0.15, "default": 0.10},
            "crypto": {"min": 0.00, "max": 0.15, "default": 0.07},
            "cash": {"min": 0.00, "max": 0.05, "default": 0.03},
        }
    ),
)
```

3. 专家智能体层

3.1 核心数据结构

3.1.1 Action 枚举 (9级动作空间)

```
class Action(Enum):
    """9-Action Trading Space"""
    SELL_100 = "SELL_100%"
    SELL_75 = "SELL_75%"
    SELL_50 = "SELL_50%"
    SELL_25 = "SELL_25%"
    HOLD = "HOLD"
    BUY_25 = "BUY_25%"
    BUY_50 = "BUY_50%"
    BUY_75 = "BUY_75%"
    BUY_100 = "BUY_100%"
```

动作	含义	权重调整
SELL_100	全部卖出	-100%
SELL_75	卖出75%	-75%
SELL_50	卖出50%	-50%
SELL_25	卖出25%	-25%
HOLD	持有	0%
BUY_25	买入25%	+25%
BUY_50	买入50%	+50%
BUY_75	买入75%	+75%
BUY_100	全仓买入	+100%

3.1.2 ExpertRecommendation

```
@dataclass
class ExpertRecommendation:
    """专家建议数据结构"""
    asset_class: str                     # 资产类别
    symbol: str                          # 资产代码
    action: Action                       # 建议动作
    confidence: float                    # 置信度 [0, 1]
    target_weight: float                 # 建议权重
    reasoning: str                       # 决策理由
    market_view: Dict[str, Any]          # 市场观点
    risk_assessment: Dict[str, float]    # 风险评估

    def to_dict(self) -> Dict:
        return {
            "asset_class": self.asset_class,
            "symbol": self.symbol,
            "action": self.action.value,
            "confidence": self.confidence,
            "target_weight": self.target_weight,
            "reasoning": self.reasoning,
            "market_view": self.market_view,
            "risk_assessment": self.risk_assessment,
        }
```

3.1.3 ExpertReport

```
@dataclass
class ExpertReport:
    """专家完整报告"""
    expert_name: str                         # 专家名称
    asset_class: str                         # 资产类别
    timestamp: str                           # 时间戳
    recommendations: List[ExpertRecommendation]   # 资产推荐列表
    overall_view: str                        # 整体观点 (bullish/bearish/neutral)
    sector_allocation: Dict[str, float]      # 细分配置建议
    key_factors: List[str]                   # 关键影响因素
```

3.2 BaseExpert 基类

文件: `finsage/agents/base_expert.py`

3.2.1 类定义

```
class BaseExpert(ABC):
    """
    专家Agent基类

    每个专家负责特定资产类别的分析和建议:
    - Stock Expert: 股票
    - Bond Expert: 债券
    - Commodity Expert: 大宗商品
    - REITs Expert: 房地产投资信托
    - Crypto Expert: 加密货币
    """

    def __init__(
        self,
        llm_provider: Any,
        asset_class: str,
        symbols: List[str],
        config: Optional[Dict] = None
    ):
        self.llm = llm_provider
        self.asset_class = asset_class
        self.symbols = symbols
        self.config = config or {}
        self.max_single_weight = self.config.get("max_single_weight", 0.15)
        self.min_confidence = self.config.get("min_confidence", 0.5)

    @property
    @abstractmethod
    def name(self) -> str:
        """专家名称"""
        pass

    @property
    @abstractmethod
    def expertise(self) -> List[str]:
        """专业领域列表"""
        pass

    @abstractmethod
    def _build_analysis_prompt(
        self,
        market_data: Dict[str, Any],
        news_data: List[Dict],
        technical_indicators: Dict[str, Any],
    ) -> str:
        """构建分析Prompt"""
        pass
```

3.2.2 核心方法

```python
def analyze(
    self,
    market_data: Dict[str, Any],
    news_data: Optional[List[Dict]] = None,
    technical_indicators: Optional[Dict[str, Any]] = None,
    macro_data: Optional[Dict[str, Any]] = None,
) -> ExpertReport:
    """执行分析并生成报告"""
    from datetime import datetime

    # 1. 构建Prompt
    prompt = self._build_analysis_prompt(
        market_data=market_data,
        news_data=news_data or [],
        technical_indicators=technical_indicators or {},
    )

    # 2. 调用LLM
    response = self.llm.create_completion(
        messages=[
            {"role": "system", "content": self._get_system_prompt()},
            {"role": "user", "content": prompt}
        ],
        temperature=0.7,
        max_tokens=2000,
    )

    # 3. 解析响应
    recommendations = self._parse_llm_response(response)

    # 4. 构建报告
    report = ExpertReport(
        expert_name=self.name,
        asset_class=self.asset_class,
        timestamp=datetime.now().isoformat(),
        recommendations=recommendations,
        overall_view=self._determine_overall_view(recommendations),
        sector_allocation=self._calculate_sector_allocation(recommendations),
        key_factors=self._extract_key_factors(response),
    )
    return report
```

3.2.3 系统提示词模板

```
def _get_system_prompt(self) -> str:
    """获取系统Prompt"""
    return f"""你是一位专业的{self.asset_class}投资专家。

## 你的专业领域
{chr(10).join(f'- {e}' for e in self.expertise)}

## 你的职责
1. 分析提供的市场数据和新闻
2. 评估各资产的投资价值
3. 给出具体的交易建议和置信度
4. 解释你的决策逻辑

## 输出格式
请以JSON格式输出你的分析结果:
{{
    "overall_view": "bullish/bearish/neutral",
    "recommendations": [
        {{
            "symbol": "资产代码",
            "action": "BUY_25%/BUY_50%/BUY_75%/BUY_100%/HOLD/SELL_25%/SELL_50%/SELL_75%
            "confidence": 0.0-1.0,
            "target_weight": 0.0-1.0,
            "reasoning": "决策理由",
            "risk_level": "low/medium/high"
        }}
    ],
    "key_factors": ["关键因素1", "关键因素2"],
    "market_analysis": "市场分析总结"
}}
"""
```

3.2.4 整体观点判定

```python
def _determine_overall_view(self, recommendations: List[ExpertRecommendation]) -> str:
    """根据建议确定整体观点"""
    if not recommendations:
        return "neutral"

    buy_weight = sum(
        r.confidence for r in recommendations
        if "BUY" in r.action.value
    )
    sell_weight = sum(
        r.confidence for r in recommendations
        if "SELL" in r.action.value
    )

    if buy_weight > sell_weight * 1.5:
        return "bullish"
    elif sell_weight > buy_weight * 1.5:
        return "bearish"
    return "neutral"
```

3.3 五位专家实现

专家	资产类别	特殊因子	文件
StockExpert	股票	FF-5因子(盈利/价值/规模/投资/动量)	`stock_expert.py`
BondExpert	债券	Carry/Value/Low-Risk/Momentum/Duration	`bond_expert.py`
CommodityExpert	商品	期限结构/动量/基差/Carry	`commodity_expert.py`
REITsExpert	REITs	NAV折溢价/行业前景/特质风险	`reits_expert.py`
CryptoExpert	加密货币	网络效应/采纳度/崩盘风险	`crypto_expert.py`

3.3.1 StockExpert 示例

```python
class StockExpert(BaseExpert):
    @property
    def name(self) -> str:
        return "Stock Expert"

    @property
    def expertise(self) -> List[str]:
        return [
            "股票基本面分析",
            "Fama-French 5因子模型",
            "行业轮动策略",
            "估值分析 (P/E, P/B, EV/EBITDA)",
            "盈利质量评估",
        ]

    def _build_analysis_prompt(
        self,
        market_data: Dict[str, Any],
        news_data: List[Dict],
        technical_indicators: Dict[str, Any],
    ) -> str:
        prompt = "## 股票分析任务\n\n"
        prompt += "### 候选股票\n"
        for symbol in self.symbols:
            if symbol in market_data:
                data = market_data[symbol]
                prompt += f"\n{symbol}:\n"
                prompt += f"  价格: ${data.get('price', 'N/A')}\n"
                prompt += f"  市盈率: {data.get('pe_ratio', 'N/A')}\n"
                prompt += f"  市净率: {data.get('pb_ratio', 'N/A')}\n"
                prompt += f"  ROE: {data.get('roe', 'N/A')}\n"

                # 因子评分 (如果有)
                if 'factor_score' in data:
                    prompt += f"  因子评分: {data['factor_score']:.2f}\n"
                    prompt += f"  因子信号: {data.get('factor_signal', 'N/A')}\n"

        prompt += "\n### 任务\n"
        prompt += "请分析以上股票并给出投资建议。\n"
        return prompt
```

3.4 专家协作流程

```
┌─────────────────────────────────────────────────────────────┐
│                    Expert Collaboration                       │
├─────────────────────────────────────────────────────────────┤
│  1. 数据准备                                                  │
│     MarketDataProvider.get_data() → market_data              │
│                                                              │
│  2. 并行分析 (ThreadPoolExecutor)                              │
│                                                              │
│     ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐   │
│     │ Stock  │ │  Bond  │ │Commodity│ │ REITs  │ │ Crypto │   │
│     │analyze │ │analyze │ │analyze │ │analyze │ │analyze │   │
│     └───┬────┘ └───┬────┘ └───┬────┘ └───┬────┘ └───┬────┘   │
│         └──────────┴─────┬────┴──────────┴──────────┘        │
│                          ▼                                    │
│  3. 报告汇总: Dict[asset_class, ExpertReport]                   │
│                                                              │
│  4. 送入管理层 → ManagerCoordinator.coordinate()                │
└─────────────────────────────────────────────────────────────┘
```

4. 管理层智能体

4.1 核心数据结构

4.1.1 PortfolioDecision

```python
@dataclass
class PortfolioDecision:
    """组合决策结果"""
    timestamp: str
    target_allocation: Dict[str, float]       # 目标配置 {asset: weight}
    trades: List[Dict[str, Any]]              # 交易指令
    hedging_tool_used: str                    # 使用的对冲工具
    reasoning: str                            # 决策理由
    risk_metrics: Dict[str, float]            # 风险指标
    expert_summary: Dict[str, str]            # 各专家意见摘要
```

4.1.2 PositionSizingDecision

```python
@dataclass
class PositionSizingDecision:
    """仓位决策结果"""
    timestamp: str
    position_sizes: Dict[str, float]     # {asset: position_size}
    sizing_method: str                   # 使用的仓位方法
    reasoning: str                       # 决策理由
    risk_contribution: Dict[str, float]  # 每个资产的风险贡献
```

4.1.3 HedgingDecision

```python
@dataclass
class HedgingDecision:
    """对冲决策结果"""
    timestamp: str
    hedging_strategy: str              # 对冲策略名称
    hedge_ratio: float                 # 对冲比例
    hedge_instruments: List[Dict]      # 对冲工具列表
    expected_cost: float               # 预期成本
    expected_protection: float         # 预期保护水平
    reasoning: str                     # 决策理由
    tail_risk_metrics: Dict[str, float]  # 尾部风险指标
    dynamic_recommendation: Optional[Dict] = None  # 动态选择推荐
```

4.1.4 IntegratedDecision

```python
@dataclass
class IntegratedDecision:
    """整合后的最终决策"""
    timestamp: str

    # 资产配置
    target_allocation: Dict[str, float]
    position_sizes: Dict[str, float]

    # 对冲
    hedging_strategy: str
    hedge_ratio: float
    hedge_instruments: List[Dict]

    # 交易
    trades: List[Dict[str, Any]]

    # 风险
    risk_metrics: Dict[str, float]
    tail_risk_metrics: Dict[str, float]

    # 决策过程
    discussion_rounds: int
    consensus_reached: bool
    individual_decisions: Dict[str, Any]
    final_reasoning: str
```

4.2 Manager Coordinator

文件: `finsage/agents/manager_coordinator.py`

4.2.1 三阶段协调流程

```
Phase 1: 并行独立分析

┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐
│ Portfolio Mgr   │  │ Position Sizing│  │ Hedging Agent   │
│  - 解读报告      │  │  - Kelly公式     │  │  - 尾部风险      │
│  - 动态配置      │  │  - 风险平价      │  │ - 选择策略      │
│  - 目标分配      │  │  - 波动率调整    │  │ - 选择工具      │
└────────┬────────┘  └────────┬────────┘  └────────┬────────┘
         │                    │                    │
         ▼                    ▼                    ▼
Phase 2: 多轮讨论 (max_rounds=2)

┌─────────────────────────────────────────────────────────┐
│ Round 1: 反馈交换                                          │
│   PM Decision → Sizing Agent (调整仓位)                   │
│   PM Decision → Hedging Agent (调整对冲)                  │
│                                                         │
│ 共识检查: sizing_diff < 5% && hedge_diff < 2%           │
│   → 达成共识: 跳出                                        │
│   → 未达成: 进入 Round 2                                  │
└─────────────────────────────────────────────────────────┘
         │
         ▼
Phase 3: 整合输出 → IntegratedDecision
```

4.2.2 共识检查机制

```python
def _check_consensus(
    self,
    old_sizing: PositionSizingDecision,
    new_sizing: PositionSizingDecision,
    old_hedging: HedgingDecision,
    new_hedging: HedgingDecision,
) -> bool:
    """检查是否达成共识"""
    # 检查仓位变化
    sizing_diff = 0.0
    for asset in old_sizing.position_sizes:
        old_size = old_sizing.position_sizes.get(asset, 0)
        new_size = new_sizing.position_sizes.get(asset, 0)
        sizing_diff += abs(old_size - new_size)

    # 检查对冲比例变化
    hedge_diff = abs(old_hedging.hedge_ratio - new_hedging.hedge_ratio)

    # 如果变化都很小，认为达成共识
    consensus = (sizing_diff < 0.05) and (hedge_diff < 0.02)
    return consensus
```

4.2.3 决策整合逻辑

```
def _integrate_decisions(self, pm, sizing, hedging, current_portfolio, rounds):
    """整合三个智能体的决策"""

    # 最终配置 = PM配置(60%) + Sizing调整(40%)
    final_allocation = {}
    for asset_class, pm_weight in pm.target_allocation.items():
        if asset_class in sizing.position_sizes:
            sizing_weight = sizing.position_sizes.get(asset_class, pm_weight)
            final_allocation[asset_class] = pm_weight * 0.6 + sizing_weight * 0.4
        else:
            final_allocation[asset_class] = pm_weight

    # 归一化
    total = sum(final_allocation.values())
    if total > 0:
        final_allocation = {k: v / total for k, v in final_allocation.items()}

    # 应用对冲调整
    if hedging.hedge_ratio > 0:
        hedge_ratio = hedging.hedge_ratio
        if "cash" in final_allocation:
            final_allocation["cash"] += hedge_ratio * 0.5
        if "stocks" in final_allocation:
            final_allocation["stocks"] = max(0.1, final_allocation["stocks"] - hedge_ra
        # 重新归一化
        total = sum(final_allocation.values())
        final_allocation = {k: v / total for k, v in final_allocation.items()}

    return IntegratedDecision(...)
```

4.3 Portfolio Manager

文件: `finsage/agents/portfolio_manager.py`

4.3.1 11种优化工具

工具	说明	适用场景
mean_variance	均值-方差优化	追求风险收益平衡

工具	说明	适用场景
min_variance	最小方差	保守投资者
max_sharpe	最大夏普比	追求风险调整收益
risk_parity	风险平价	风险均衡配置
black_litterman	Black-Litterman模型	结合主观观点
max_diversification	最大分散化	分散化投资
equal_weight	等权重	简单配置
inverse_volatility	反波动率	低波动策略
cvar_optimization	CVaR优化	控制尾部风险
hierarchical_risk_parity	层次风险平价	资产聚类配置
robust_optimization	稳健优化	参数不确定性

4.3.2 LLM动态配置

```
def _build_dynamic_allocation_prompt(self, expert_summary, market_volatility, vix):
    """构建动态配置 Prompt"""
    experts_text = ""
    for exp in expert_summary:
        experts_text += f"""
- {exp['asset_class'].upper()} 专家:
  观点: {exp['view']} (信心: {exp['confidence']:.0%})
  推荐: {', '.join(exp['top_picks'])}
  权重范围: {exp['bounds']['min']:.0%} - {exp['bounds']['max']:.0%}
"""

    return f"""## 动态资产配置任务

### 市场环境
- VIX: {vix:.1f} ({market_volatility} volatility)

### 五位专家观点
{experts_text}

### 任务
根据专家观点和市场环境，决定各资产类别的目标权重。

规则:
1. 看多(bullish)的资产类别应增加权重 (接近max)
2. 看空(bearish)的资产类别应减少权重 (接近min)
3. 中性(neutral)的资产类别保持默认权重
4. 高波动市场应增加bonds和cash
5. 所有权重之和必须等于1.0

输出格式 (JSON):
{{
  "allocation": {{
    "stocks": 0.40,
    "bonds": 0.25,
    "commodities": 0.15,
    "reits": 0.10,
    "crypto": 0.05,
    "cash": 0.05
  }},
  "reasoning": "简要说明配置理由"
}}
"""
```

4.4 Position Sizing Agent

文件: `finsage/agents/position_sizing_agent.py`

4.4.1 5种仓位方法

方法	说明	公式	适用场景
equal_weight	等权配置	$w_i = 1/n$	无明确偏好
risk_parity	风险平价	$w_i \propto 1/\sigma_i$	风险均衡
volatility_target	波动率目标	$w *= \sigma_{target}/\sigma_{portfolio}$	控制波动
kelly	Kelly准则	$f = \mu/\sigma^2 \times 0.5$	高置信度信号
max_sharpe	最大夏普	$\max(\mu'w/\sqrt{(w'\Sigma w)})$	追求效率

4.4.2 风险平价实现

```python
def _risk_parity_sizing(self, target_allocation, market_data):
    """风险平价配置: 每个资产贡献相等的风险"""
    returns_df = pd.DataFrame(market_data.get("returns", {}))

    # 计算各资产波动率
    volatilities = {}
    for asset in target_allocation:
        if asset in returns_df.columns:
            vol = returns_df[asset].std() * np.sqrt(252)
            volatilities[asset] = max(vol, 0.01)
        else:
            volatilities[asset] = 0.15  # 默认波动率

    # 风险平价: 权重与波动率成反比
    inv_vols = {asset: 1.0 / vol for asset, vol in volatilities.items()}
    total_inv_vol = sum(inv_vols.values())
    weights = {asset: inv_vol / total_inv_vol for asset, inv_vol in inv_vols.items()}
    return weights
```

4.5 Hedging Agent

文件: `finsage/agents/hedging_agent.py`

4.5.1 7种对冲策略

策略	说明	触发条件
put_protection	买入看跌期权保护	VIX适中，需要下行保护
collar	领口策略 (买看跌卖看涨)	降低期权成本
tail_hedge	尾部风险对冲	负偏度/高峰度
dynamic_hedge	动态对冲	需要实时调整
diversification	分散化对冲	一般市场环境
safe_haven	避险资产对冲	VIX>25高波动
none	无需对冲	风险在可接受范围

4.5.2 尾部风险评估

```
def _assess_tail_risk(self, allocation, market_data) -> Dict[str, float]:
    """评估组合的尾部风险"""
    tail_risk = {
        "vix": market_data.get("macro", {}).get("vix", 20.0),
        "var_95": -0.02,
        "var_99": -0.04,
        "cvar_95": -0.03,
        "max_drawdown": -0.15,
        "skewness": 0.0,
        "kurtosis": 3.0,
    }

    returns_df = pd.DataFrame(market_data.get("returns", {}))
    if returns_df.empty:
        return tail_risk

    # 构建组合收益
    weights = np.array([allocation.get(a, 0) for a in allocation if a in returns_df.col
    portfolio_returns = returns_df[list(allocation.keys())].dot(weights / weights.sum()

    # 计算 VaR
    tail_risk["var_95"] = float(np.percentile(portfolio_returns, 5))
    tail_risk["var_99"] = float(np.percentile(portfolio_returns, 1))

    # 计算 CVaR (Expected Shortfall)
    var_95_mask = portfolio_returns <= tail_risk["var_95"]
    tail_risk["cvar_95"] = float(portfolio_returns[var_95_mask].mean())

    # 计算偏度和峰度
    from scipy import stats
    tail_risk["skewness"] = float(stats.skew(portfolio_returns))
    tail_risk["kurtosis"] = float(stats.kurtosis(portfolio_returns) + 3)

    return tail_risk
```

4.5.3 对冲策略选择Prompt

```
prompt = f"""## 对冲策略选择任务

### 尾部风险评估
- VIX: {vix:.1f} ({tail_risk.get('vix_level', 'moderate')})
- 日VaR (95%): {var_95:.2%}
- 日CVaR (95%): {cvar_95:.2%}
- 历史最大回撤: {max_dd:.2%}
- 偏度: {skewness:.2f} (负值=左偏/下行风险大)
- 峰度: {kurtosis:.2f} (>3=肥尾)

### 风控约束
- 最大回撤容忍: {risk_constraints.get('max_drawdown', 0.15):.1%}
- 目标波动率: {risk_constraints.get('target_volatility', 0.12):.1%}
- 最大对冲成本: {self.max_hedge_cost:.1%}

### 可用策略
- put_protection: 买入看跌期权保护
- collar: 领口策略
- tail_hedge: 尾部风险对冲
- dynamic_hedge: 动态对冲
- diversification: 分散化对冲
- safe_haven: 避险资产对冲
- none: 无需对冲

### 任务
选择最适合当前风险状况的对冲策略。

考虑因素:
1. VIX高时期权成本高，避免买入期权
2. 负偏度和高峰度表示需要更多尾部保护
3. 如果风险指标在可接受范围内，可以选择"none"

输出格式 (JSON):
{{"strategy": "策略名", "reasoning": "选择理由"}}
"""
```

5. 因子增强系统

文件: `finsage/agents/factor_enhanced_expert.py`

5.1 核心架构

```
┌─────────────────────────────────────────────────────────────┐
│                  FactorEnhancedExpertMixin                  │
├─────────────────────────────────────────────────────────────┤
│  最终评分 = LLM评分 × 0.6 + 因子评分 × 0.4                  │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  ┌─────────────────────┐    ┌─────────────────────┐         │
│  │   LLM分析 (60%)     │    │  因子评分 (40%)     │         │
│  │   - 市场数据        │    │  - 学术因子         │         │
│  │   - 新闻情绪        │    │  - 量化信号         │         │
│  │   - 技术指标        │    │  - 历史收益         │         │
│  └─────────────────────┘    └─────────────────────┘         │
│              │                        │                     │
│              └────────────┬───────────┘                     │
│                           ▼                                 │
│                 ┌────────────────┐                          │
│                 │ 综合调整       │                          │
│                 │ - confidence   │                          │
│                 │ - action       │                          │
│                 │ - target_weight│                          │
│                 └────────────────┘                          │
│                           │                                 │
│                           ▼                                 │
│                EnhancedRecommendation                       │
└─────────────────────────────────────────────────────────────┘
```

5.2 EnhancedRecommendation

```python
@dataclass
class EnhancedRecommendation(ExpertRecommendation):
    """增强版建议 - 包含因子评分"""
    factor_score: Optional[FactorScore] = None
    factor_signal: Optional[str] = None    # "STRONG_BUY", "BUY", "HOLD", "SELL", "STRO
    factor_alpha: Optional[float] = None   # 预期Alpha
```

5.3 因子融合逻辑

5.3.1 置信度调整

```python
def _adjust_confidence(self, rec, factor_score) -> float:
    """综合调整置信度"""
    llm_conf = rec.confidence
    factor_conf = factor_score.composite_score

    # 加权平均
    combined = (
        llm_conf * (1 - self.factor_weight) +
        factor_conf * self.factor_weight
    )

    # 如果信号一致，提高置信度（+10%）
    llm_bullish = "BUY" in rec.action.value
    factor_bullish = factor_score.signal in ["STRONG_BUY", "BUY"]

    if llm_bullish == factor_bullish:
        combined = min(combined * 1.1, 1.0)
    else:
        combined = max(combined * 0.9, 0.1)

    return round(combined, 3)
```

5.3.2 动作调整

```python
def _adjust_action(self, rec, factor_score) -> Action:
    """综合调整动作"""
    factor_action_map = {
        "STRONG_BUY": 2, "BUY": 1, "HOLD": 0, "SELL": -1, "STRONG_SELL": -2,
    }
    llm_action_map = {
        Action.BUY_100: 2, Action.BUY_75: 1.5, Action.BUY_50: 1, Action.BUY_25: 0.5,
        Action.HOLD: 0,
        Action.SELL_25: -0.5, Action.SELL_50: -1, Action.SELL_75: -1.5, Action.SELL_100
    }

    llm_strength = llm_action_map.get(rec.action, 0)
    factor_strength = factor_action_map.get(factor_score.signal, 0)

    # 加权平均: LLM 60% + Factor 40%
    combined_strength = (
        llm_strength * (1 - self.factor_weight) +
        factor_strength * self.factor_weight
    )

    # 映射回动作
    if combined_strength >= 1.5:
        return Action.BUY_75
    elif combined_strength >= 1.0:
        return Action.BUY_50
    elif combined_strength >= 0.5:
        return Action.BUY_25
    elif combined_strength > -0.5:
        return Action.HOLD
    elif combined_strength > -1.0:
        return Action.SELL_25
    elif combined_strength > -1.5:
        return Action.SELL_50
    else:
        return Action.SELL_75
```

5.4 各资产类别因子

5.4.1 股票 - Fama-French 5因子

因子	指标	方向	说明
profitability	ROE, 毛利率	越高越好	盈利能力
value	P/E, P/B	越低越好	价值因子
size	市值	中小盘溢价	规模因子
investment	资产增长率	越低越好	保守投资
momentum	12个月收益	越高越好	动量效应

5.4.2 债券4因子

因子	说明	计算方式
carry	到期收益率	YTM - 短期利率
value	信用利差	信用利差 vs 历史均值
low_risk	久期风险	久期归一化
momentum	价格动量	过去12个月收益

5.4.3 商品3因子

因子	说明	信号
term_structure	期限结构	Backwardation=买入, Contango=卖出
momentum	价格动量	正动量=买入
basis	基差	正基差=买入

5.4.4 REITs因子

因子	说明
nav_discount	NAV折溢价

因子	说明
sector_outlook	行业前景 (办公/零售/物流等)
idiosyncratic_risk	特质风险

5.4.5 加密货币因子

因子	说明
network_effect	网络效应 (活跃地址/交易量)
adoption	机构采纳度
crash_risk	崩盘风险 (波动率/偏度)

5.5 创建因子增强专家

```python
from finsage.agents.factor_enhanced_expert import create_factor_enhanced_expert
from finsage.agents.experts.stock_expert import StockExpert
from finsage.factors import StockFactorScorer

# 创建因子增强版股票专家
enhanced_expert = create_factor_enhanced_expert(
    base_expert_class=StockExpert,
    factor_scorer=StockFactorScorer(),
    llm_provider=my_llm,
    symbols=["AAPL", "MSFT", "GOOGL"],
    factor_weight=0.4  # 因子权重40%
)

# 执行因子增强分析
report = enhanced_expert.analyze_with_factors(
    market_data=market_data,
    returns=returns_df,
    market_regime="bull"
)
```

6. 对冲工具包

文件: `finsage/hedging/toolkit.py`

6.1 HedgingToolkit 架构

```
┌──────────────────────────────────────────────────────────────┐
│                      HedgingToolkit                          │
├──────────────────────────────────────────────────────────────┤
│  tools: Dict[str, HedgingTool]                               │
│                                                              │
│  ┌──────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐        │
│  │RiskParity│ │BlackLit  │ │   CVaR   │ │ Regime   │ ...    │
│  └──────────┘ └──────────┘ └──────────┘ └──────────┘        │
│                                                              │
│  get_tool(name) → HedgingTool                                │
│  list_tools() → List[Dict]                                   │
│  call(tool_name, returns, expert_views, constraints)         │
│  compare_tools(portfolio) → Dict[str, HedgeResult]          │
└──────────────────────────────────────────────────────────────┘
```

6.2 可用对冲工具

工具	说明	主要用途
risk_parity	风险平价	风险均衡配置
black_litterman	Black-Litterman	结合主观观点
cvar_optimization	CVaR优化	控制尾部风险
regime_switching	体制切换	适应市场状态
copula_hedging	Copula对冲	极端事件保护
factor_neutralization	因子中性化	消除因子暴露
dynamic_hedging	动态对冲	实时调整
mean_variance	均值方差	经典优化

工具	说明	主要用途
min_variance	最小方差	保守配置
max_diversification	最大分散化	分散投资
robust_optimization	稳健优化	参数不确定性

6.3 对冲资产库 (70+资产)

6.3.1 波动率相关

代码	名称	用途
VXX	iPath S&P 500 VIX Short-Term	VIX多头对冲
UVXY	ProShares Ultra VIX Short-Term	2倍VIX多头
SVXY	ProShares Short VIX Short-Term	VIX空头
VIXY	ProShares VIX Short-Term Futures	VIX期货追踪

6.3.2 反向ETF

代码	名称	用途
SH	ProShares Short S&P 500	-1倍标普
SDS	ProShares UltraShort S&P 500	-2倍标普
SQQQ	ProShares UltraPro Short QQQ	-3倍纳指
PSQ	ProShares Short QQQ	-1倍纳指

6.3.3 避险资产

代码	名称	用途
GLD	SPDR Gold Shares	黄金ETF
IAU	iShares Gold Trust	黄金ETF
SLV	iShares Silver Trust	白银ETF

代码	名称	用途
TLT	iShares 20+ Year Treasury	长期国债
IEF	iShares 7-10 Year Treasury	中期国债
SHY	iShares 1-3 Year Treasury	短期国债

6.3.4 尾部风险专用

代码	名称	用途
TAIL	Cambria Tail Risk ETF	黑天鹅保护
SWAN	Amplify BlackSwan Growth & Treasury	保本增长

6.3.5 货币对冲

代码	名称	用途
UUP	Invesco DB US Dollar Index Bullish	美元多头
FXE	Invesco CurrencyShares Euro Trust	欧元
FXY	Invesco CurrencyShares Japanese Yen	日元

6.4 动态对冲选择器

文件: `finsage/hedging/dynamic_selector.py`

```
class DynamicHedgeSelector:
    """动态对冲资产选择器"""

    def recommend(
        self,
        portfolio_weights: Dict[str, float],
        returns_data: pd.DataFrame,
        hedge_strategy: str,
        hedge_ratio: float,
        market_data: Dict,
        risk_constraints: Dict,
    ) -> HedgeRecommendation:
        """
        从70+资产中筛选最优对冲工具

        评分因素:
        - 与组合的负相关性
        - 流动性 (交易量)
        - 成本 (费率)
        - 历史对冲效果
        """
        pass
```

7. 数据层架构

文件: `finsage/data/fmp_client.py`

7.1 FMP Ultra API 概览

FinSage使用 **FMP Ultra API** 作为主数据源，支持96,000+全市场资产。

7.1.1 资产覆盖

资产类别	数量	端点
股票	87,761+	`/stock-list` , `/company-screener`

资产类别	数量	端点
加密货币	4,786+	`/cryptocurrency-list`
商品期货	40+	`/commodities-list`
商品ETF	945+	`/etf-list`
REITs	540+	`/company-screener` (sector=Real Estate)
债券ETF	2,165+	`/etf-list`

7.1.2 API端点分类

```
Market Data (市场数据)
├── /quote              - 股票报价
├── /batch-quote        - 批量报价
├── /stock-list         - 全球股票列表 (87,761+)
└── /etf-list           - ETF列表 (13,299+)

Multi-Asset (多资产)
├── /cryptocurrency-list  - 加密货币列表 (4,786+)
└── /commodities-list     - 商品期货列表 (40+)

Financial Statements (财务报表)
├── /income-statement     - 利润表
├── /balance-sheet-statement  - 资产负债表
└── /cash-flow-statement  - 现金流量表

Key Metrics (关键指标)
├── /key-metrics-ttm      - 关键指标 (TTM)
├── /ratios-ttm           - 财务比率 (TTM)
└── /financial-scores     - 财务评分

Historical Data (历史数据)
├── /historical-price-eod/full   - 历史日线数据
├── /historical-chart/1min       - 分钟级数据
└── /historical-chart/1hour      - 小时级数据

Screening (筛选)
└── /company-screener     - 全市场筛选
```

7.2 FMPClient 核心类

```python
class FMPClient:
    """FMP Ultra API 客户端"""

    def __init__(self, config: Optional[FMPConfig] = None):
        self.config = config or FMPConfig()
        self.session = requests.Session()
        self._cache = {}

    def screen_stocks(
        self,
        market_cap_min: Optional[float] = None,
        market_cap_max: Optional[float] = None,
        price_min: Optional[float] = None,
        price_max: Optional[float] = None,
        beta_min: Optional[float] = None,
        beta_max: Optional[float] = None,
        sector: Optional[str] = None,
        industry: Optional[str] = None,
        country: str = "US",
        exchange: str = "NYSE,NASDAQ,AMEX",
        is_actively_trading: bool = True,
        limit: int = 1000,
    ) -> pd.DataFrame:
        """股票筛选"""
        pass

    def get_cryptocurrency_list(self) -> pd.DataFrame:
        """获取加密货币列表 (4,786+)"""
        pass

    def get_commodities_list(self) -> pd.DataFrame:
        """获取商品期货列表 (40+)"""
        pass

    def get_commodity_etfs(self) -> pd.DataFrame:
        """获取商品ETF (945+)"""
        pass

    def get_bond_etfs(self) -> pd.DataFrame:
        """获取债券ETF (2,165+)"""
        pass

    def get_reit_etfs(self) -> pd.DataFrame:
        """获取REIT ETF"""
        pass

    def get_reits(self) -> pd.DataFrame:
        """获取REITs个股 (540+)"""
        pass

    def get_full_asset_universe(
        self,
        include_stocks: bool = True,
        include_crypto: bool = True,
        include_commodities: bool = True,
        include_reits: bool = True,
        include_bonds: bool = True,
        stock_market_cap_min: float = 1e9,
        stock_limit: int = 1000,
    ) -> Dict[str, pd.DataFrame]:
        """
        一键获取全资产范围

        Returns:
            {
                "stocks": DataFrame,      # 1000只
                "crypto": DataFrame,      # 4786种
                "commodities": DataFrame, # 985只
                "reits": DataFrame,       # 540只
                "bonds": DataFrame,       # 2165只
            }
        """
        pass

    def get_key_metrics_ttm(self, symbol: str) -> Dict:
        """获取关键指标 (TTM)"""
        pass

    def get_ratios_ttm(self, symbol: str) -> Dict:
        """获取财务比率 (TTM)"""
        pass

    def get_historical_prices(
        self,
        symbol: str,
        start_date: Optional[str] = None,
        end_date: Optional[str] = None,
    ) -> pd.DataFrame:
        """获取历史价格"""
        pass
```

7.3 FactorScreener 因子筛选器

```python
class FactorScreener:
    """多因子筛选器"""

    def __init__(self, client: FMPClient):
        self.client = client

    def screen_by_factors(
        self,
        symbols: List[str],
        factors: List[str],
        weights: Dict[str, float],
        top_n: int = 50,
    ) -> pd.DataFrame:
        """
        多因子筛选 Top N

        Args:
            symbols: 候选股票列表
            factors: 使用的因子 ["value", "quality", "momentum"]
            weights: 因子权重 {"value": 0.3, "quality": 0.4, "momentum": 0.3}
            top_n: 返回数量

        Returns:
            筛选后的DataFrame，按综合评分排序
        """
        pass

    def compute_value_score(self, metrics: Dict) -> float:
        """计算价值因子评分"""
        pe = metrics.get("peRatioTTM", float("inf"))
        pb = metrics.get("pbRatioTTM", float("inf"))
        ev_ebitda = metrics.get("evToEbitdaTTM", float("inf"))

        # 归一化并取反（低估值=高分）
        score = 1.0 / (1 + pe/20) * 0.4 + 1.0 / (1 + pb/3) * 0.3 + 1.0 / (1 + ev_ebitda
        return score

    def compute_quality_score(self, metrics: Dict) -> float:
        """计算质量因子评分"""
        roe = metrics.get("roeTTM", 0)
        roa = metrics.get("roaTTM", 0)
        gross_margin = metrics.get("grossProfitMarginTTM", 0)

        score = min(roe/0.20, 1.0) * 0.4 + min(roa/0.10, 1.0) * 0.3 + min(gross_margin/
        return score

    def compute_momentum_score(self, returns: pd.Series) -> float:
        """计算动量因子评分"""
        if len(returns) < 252:
            return 0.5

        mom_12m = (1 + returns).prod() - 1
        mom_1m = (1 + returns.iloc[-21:]).prod() - 1

        # 12-1月动量（排除最近1月）
        mom_12_1 = mom_12m - mom_1m

        # 归一化到 [0, 1]
        score = min(max((mom_12_1 + 0.5) / 1.0, 0), 1)
        return score
```

7.4 环境变量配置

```
# .env 文件

# FMP API（必需）
FMP_API_KEY=your_fmp_api_key
FMP_API_TIER=ultra
FMP_BASE_URL=https://financialmodelingprep.com/stable

# OpenAI API（必需）
OPENAI_API_KEY=your_openai_key

# 备用数据源（可选）
POLYGON_API_KEY=your_polygon_key
ALPHA_VANTAGE_KEY=your_av_key
```

7.5 使用示例

```python
from finsage.data import FMPClient, FactorScreener

# 初始化客户端
client = FMPClient()

# 一键获取全资产范围
universe = client.get_full_asset_universe(
    include_stocks=True,
    include_crypto=True,
    include_commodities=True,
    include_reits=True,
    include_bonds=True,
    stock_market_cap_min=1e9,  # 10亿美元以上
    stock_limit=1000
)

# 输出统计
for asset_class, df in universe.items():
    print(f"{asset_class}: {len(df)} 只")
# stocks: 1000 只
# crypto: 4786 种
# commodities: 985 只
# reits: 540 只
# bonds: 2165 只

# 多因子筛选
screener = FactorScreener(client)
top_stocks = screener.screen_by_factors(
    symbols=universe["stocks"]["symbol"].tolist(),
    factors=["value", "quality", "momentum"],
    weights={"value": 0.3, "quality": 0.4, "momentum": 0.3},
    top_n=50
)
print(top_stocks.head())
```

8. 附录

A. 快速启动

```
# 1. 克隆项目
git clone https://github.com/your-repo/finsage.git
cd finsage

# 2. 安装依赖
pip install -r requirements.txt

# 3. 配置环境变量
cp .env.example .env
# 编辑 .env 填入 API Keys

# 4. 运行回测
python main.py \
  --start 2024-01-01 \
  --end 2024-12-01 \
  --frequency weekly \
  --capital 1000000 \
  --model gpt-4o-mini

# 5. 查看结果
ls results/
```

B. 命令行参数

参数	说明	默认值
`--start`	开始日期	6个月前
`--end`	结束日期	今天
`--frequency`	再平衡频率 (daily/weekly/monthly)	daily
`--capital`	初始资金	1000000
`--model`	LLM模型	gpt-4o-mini

参数	说明	默认值
`--log-level`	日志级别	INFO
`--config`	配置模板 (default/conservative/aggressive)	default

C. 关键数据结构速查

数据结构	位置	用途
`Action`	`base_expert.py`	9级交易动作枚举
`ExpertRecommendation`	`base_expert.py`	单资产推荐
`ExpertReport`	`base_expert.py`	专家分析报告
`EnhancedRecommendation`	`factor_enhanced_expert.py`	因子增强推荐
`PortfolioDecision`	`portfolio_manager.py`	组合决策
`PositionSizingDecision`	`position_sizing_agent.py`	仓位决策
`HedgingDecision`	`hedging_agent.py`	对冲决策
`IntegratedDecision`	`manager_coordinator.py`	整合决策
`FactorScore`	`base_factor.py`	因子评分
`FactorExposure`	`base_factor.py`	因子暴露

D. API限制 (FMP Ultra)

参数	值
请求/分钟	750
批量大小	100
缓存有效期	24小时
历史数据	30年+
实时数据	15分钟延迟

E. 风控规则速查

E.1 硬性约束 (Veto)

规则	阈值	处理
单资产上限	15%	拒绝交易
资产类别上限	50%	拒绝交易
最大回撤触发	15%	全面减仓
组合VaR上限	3%	拒绝高风险配置

E.2 软性约束 (Warning)

规则	阈值	处理
目标波动率	12%	警告并调整
相关性聚集	60%	建议分散
分散化比率	1.2	建议增加资产

F. LLM提示词模板索引

模块	方法	用途
BaseExpert	`_get_system_prompt()`	专家角色定义
PortfolioManager	`_build_tool_selection_prompt()`	对冲工具选择
PortfolioManager	`_build_dynamic_allocation_prompt()`	动态资产配置
PositionSizingAgent	`_select_sizing_method()`	仓位方法选择
HedgingAgent	`_select_hedging_strategy()`	对冲策略选择

G. 常见问题

Q1: 如何添加新的资产类别?

1. 在 `AssetConfig.allocation_bounds` 中添加新类别
2. 创建新的专家类继承 `BaseExpert`
3. 创建对应的因子评分器
4. 在 `ManagerCoordinator` 中注册

Q2: 如何调整因子权重?

```
enhanced_expert = create_factor_enhanced_expert(
    StockExpert,
    StockFactorScorer(),
    factor_weight=0.5  # 50% 因子权重 (默认40%)
)
```

Q3: 如何禁用动态对冲选择?

```
hedging_agent = HedgingAgent(
    llm_provider=llm,
    use_dynamic_selection=False  # 禁用动态选择
)
```

Q4: 如何使用保守配置?

```
from finsage.config import CONSERVATIVE_CONFIG

orchestrator = Orchestrator(config=CONSERVATIVE_CONFIG)
```

文档结束

版本: 2.0 | 更新日期: 2024年12月