ChatGPT

韬睿量化平台八步投资流程 - 结构与功能重构设计

1. 页面总览结构图

- **根容器** – 整个插件页面的根 `<div>` 容器，包含顶部导航、主工作区等。
- **顶部导航栏** – 固定在页面顶端，用于显示流程进度及全局操作。
  - 平台标识/标题 – 显示"韬睿量化平台"字样或Logo（可选）。
  - 流程进度指示 – 八个步骤的导航（可用进度条或步骤导航条表示1~8步骤），当前步骤高亮，其余步骤可点击跳转查看 [1]。
  - 全流程控制区域 – 包含"一键执行全部流程"按钮、暂停/恢复按钮（可选）以及当前状态指示（如当前执行到第几步，或成功/失败标记）。
- **主工作区** – 页面主体区域，用于展示各模块的界面和交互内容。
  - 模块面板容器 – 包含八个模块面板的区域。一次显示一个模块面板（支持按步骤切换），也可以采用选项卡或手风琴式切换方式。
  - 信息获取模块面板（步骤1）
  - 市场趋势模块面板（步骤2）
  - 投资主线模块面板（步骤3）
  - 候选池模块面板（步骤4）
  - 因子构建模块面板（步骤5）
  - 策略开发模块面板（步骤6）
  - 回测验证模块面板（步骤7）
  - 实盘交易模块面板（步骤8）
  - 模块面板布局 – **各模块面板**内部一般划分为左右两部分：左侧为配置区，右侧为结果展示区。也可根据需要拆分为上下或分区卡片布局，确保配置项和结果视图清晰分离。每个模块面板应有自身的标题（模块名称）和操作按钮（如"运行"、"下一步"等）。
- **底部状态栏（可选）** – 显示系统提示、错误消息或日志摘要，便于用户追踪操作过程（实现全流程的可追溯性）。

结构层级描述:

```
Root <div id="quant-platform">
├─ Header (顶部导航栏)
│   ├─ Logo/Title
│   ├─ Step Progress Indicator (8步流程导航)
│   └─ Global Controls (Run All 等全局按钮)
├─ Main Container (主工作区)
│   └─ Module Panels Container
│       ├─ Module 1: 信息获取 (配置区 + 展示区)
│       ├─ Module 2: 市场趋势 (配置区 + 展示区)
│       ├─ ... (其余模块同理)
```

```
│       └─ Module 8: 实盘交易 (配置区 + 展示区)
└─ Footer (状态栏或日志区域)
```

2. 每个模块的职责、输入输出定义、触发机制

各模块在八步量化流程中各司其职，模块之间通过输入/输出数据衔接，并通过事件或函数触发实现联动。下面详细描述每个模块：

信息获取模块（步骤1）

- **职责**：从各类数据源获取金融市场信息，包括行情数据、财务指标、新闻资讯等。为后续分析提供原始材料。
- **输入**：用户选择的数据范围和来源配置，例如时间区间、标的资产列表、数据源类型（行情、基本面、新闻等）。
- **输出**：获取的原始数据集（如股票价格序列、指标表、新闻摘要等），存储到全局数据状态中，如 `state.rawData`。该输出将提供给"市场趋势"和后续模块使用。
- **触发机制**：用户点击"获取数据"按钮手动触发。成功完成后会发布一个事件（如 `"dataFetched"`）通知系统数据已准备好。若启用了全流程自动模式，则该事件会自动触发下一模块执行。

市场趋势模块（步骤2）

- **职责**：分析市场宏观和微观趋势。从获取的数据中计算总体市场走势指标（如指数趋势、行业轮动、情绪指标等），为制定投资主线提供依据。
- **输入**：来自信息获取模块的市场原始数据（例如大盘指数走势、行业数据）。
- **输出**：市场趋势分析结果，如趋势判断（上升/下降/震荡）、热点行业列表、市场风险指标等，保存为 `state.marketTrend`。这些结果将作为下一步选择投资主线的参考。
- **触发机制**：手动触发（点击"分析趋势"）或在全流程模式下自动被"dataFetched"事件触发启动。如果没有前置数据，会提示用户先完成信息获取。完成分析后，触发事件（如 `"trendAnalyzed"`）以通知投资主线模块。

投资主线模块（步骤3）

- **职责**：确定投资主线（主要投资策略方向或主题）。根据市场趋势结果，建议若干投资主线方案（例如热点板块、策略风格如动量或价值等）。用户可从中选择主要方向。
- **输入**：市场趋势分析结果（热点板块、宏观趋势）以及用户偏好（可能通过配置选择关注的主题类型）。
- **输出**：选定的投资主线方案（如"科技成长板块动量策略"），保存至 `state.investTheme` 或类似结构，作为后续筛选候选池的依据。
- **触发机制**：用户在界面上选择或确认某一主线方案，点击"确定主线"触发。也可在全流程模式下由 `"trendAnalyzed"` 事件自动触发推荐默认主线并继续。模块完成后发布事件（如 `"themeSelected"`）通知候选池模块。

候选池模块（步骤4）

- **职责**：基于投资主线，构建候选投资标的池。按照选定主线的条件，从全市场筛选出符合主题的资产列表（如股票池）。可以提供筛选条件配置，如市值、行业、增长率等，以进一步缩小范围。

- **输入**：选定的投资主线定义（主题/板块）以及用户配置的筛选参数（例如基本面和技术面筛选条件）。所需的基础数据来自信息获取模块已存储的数据或额外数据调用。
- **输出**：候选资产池列表（如股票代码清单及其关键指标），保存为 `state.candidatePool`。这将用于因子构建和策略开发。
- **触发机制**：用户点击“生成候选池”按钮手动运行筛选逻辑，或在全流程下由 `"themeSelected"` 事件自动触发。如果输入条件不完整（如未选择主线或数据缺失），模块会警示用户。完成后发布事件（如 `"poolGenerated"`）给下游模块。

因子构建模块（步骤5）

- **职责**：制定量化因子，并计算候选池中各资产的因子值。用户可在此配置选择哪些因子（例如动量、价值、质量等）或定义自定义因子公式。模块负责从数据中计算这些因子值，以备策略使用。
- **输入**：候选池资产列表及其相关数据（行情和财务数据），用户选择的因子类型或定义，以及可能的参数（窗口期等）。
- **输出**：计算得到的因子数据集（每个候选资产的因子评分/信号），保存到 `state.factors`。还包括所选因子配置描述，供策略模块参考记录。
- **触发机制**：用户点击“计算因子”运行，或通过 `"poolGenerated"` 事件自动触发全流程计算。执行时模块应显示因子计算进度（特别是数据量大时需要进度提示 [1]）。完成后发布事件（如 `"factorsReady"`）通知策略开发模块。

策略开发模块（步骤6）

- **职责**：根据前述产出开发具体的交易策略。提供代码编辑器（集成CodeMirror）或图形化界面以编写/配置策略逻辑 [2]。用户可选择策略模板（如均值回归策略、多因子选股策略等）并加以参数调整。该模块汇总主线、因子和候选池信息生成最终的策略定义。
- **输入**：因子构建的结果数据和配置、候选池列表、选定的投资主线，以及用户在编辑器中编写或选择的策略逻辑（包括买卖信号生成、仓位管理、风险控制参数等）。
- **输出**：完整的策略模型/脚本（例如一段可执行的策略代码或配置JSON），保存为 `state.strategy`. 这包含策略所需的所有参数，在回测和实盘中使用。
- **触发机制**：用户手动点击“保存策略”或“运行策略”以完成开发步骤。也可以在全流程中由 `"factorsReady"` 事件自动触发加载默认策略模板并前往下一步（但通常策略开发需要人工干预调整，不建议全自动跳过）。当策略准备就绪且保存后，发布事件（如 `"strategyReady"`）推动回测验证模块运行。

回测验证模块（步骤7）

- **职责**：对上一步的策略进行历史回测和绩效验证。根据用户设定的回测窗口和参数，模拟策略在历史数据上的交易表现。计算关键指标（收益率、夏普比率、最大回撤等），并可视化收益曲线、交易信号等用于验证策略有效性。
- **输入**：策略开发输出的策略模型（代码）、全局数据中心内的历史行情数据（可能由信息获取模块提供）、用户配置的回测参数（如起止日期、初始资金、交易成本假设等）。
- **输出**：回测结果，包括交易明细、资金曲线、业绩指标报告，存入 `state.backtestResult`。同時計算出对策略的分析评价（可能包含策略强弱项、风险指标），供用户参考和进一步优化。
- **触发机制**：用户点击“开始回测”运行，或在全流程模式下由 `"strategyReady"` 事件自动触发。运行期间应显示进度条和剩余时间提示 [1]。回测完成后触发事件（如 `"backtestDone"`）。如果设置了自动优化，则该事件会进一步触发优化过程，或提示用户查看结果并进入优化调整阶段。

实盘交易模块（步骤8）

- **职责**：将最终策略部署到实盘环境执行（真实或模拟交易）。连接券商交易接口或模拟撮合系统，下单交易并监控策略实时表现。提供界面查看当前持仓、交易日志、收益曲线以及策略信号的实时状况。
- **输入**：经过验证的最终策略模型、实盘所需账户信息和交易接口配置（如API密钥、交易市场等），以及可能的风控参数（实盘资金上限、单笔交易限制等）。
- **输出**：实时交易状态和结果数据，例如当前持仓和盈亏、交易执行确认。关键数据写入 `state.liveTrade`（包含当前策略运行状态、实时绩效）。同时可以将交易数据和结果汇总回写到数据中台，供日后分析。
- **触发机制**：用户明确点击“启动实盘”执行（出于安全考虑，不会在全流程自动触发实盘交易）。此模块持续运行，非瞬时完成，没有下一个模块。实盘过程中可手动停止策略，或根据风险规则自动停止。模块会持续发布事件（如 `"tradeUpdate"`）以更新界面上的状态，并在交易停止时发布 `"tradeStopped"` 事件。

3. 页面组件划分建议（基于原生 JS/CSS）

为实现上述功能和布局，在不使用前端框架的情况下进行组件划分，提升代码组织和复用性：

- **顶部导航/进度组件**：独立为一个组件（例如 `StepNavbar`）。负责绘制步骤进度指示条和全局按钮。可以用纯 HTML 列表或进度条表示8个步骤，每个步骤项可绑定点击事件（跳转模块面板）。“一键执行”按钮也在此组件内，实现全局控制逻辑。样式用CSS固定顶部定位，采用高亮当前步骤的视觉区分。
- **模块面板组件**：每个模块对应一个独立DOM容器，封装其内部元素和逻辑。可设计一个通用的模块面板结构，如 `<div class="module-panel" id="module1">`，里面包含：
- 配置区子组件 – 采用如 `<form>` 或 `<div class="config-section">` 划分，用于放置该模块的所有输入控件（下拉框、输入框、复选框等）。例如，因子构建模块的配置区包含因子选择复选框列表，回测模块的配置区包含日期选择和参数输入。
- 展示区子组件 – 如 `<div class="output-section">`，用于展示模块输出结果（表格、图表、文本报告等）。例如，市场趋势模块展示区显示趋势图表，候选池模块展示股票列表表格，回测模块展示收益曲线和指标表格。
- 模块操作按钮 – 每个模块面板内部有自己的操作按钮，如“运行分析”、“下一步”等。这些按钮也是模块组件的一部分。可以通过添加 `<button class="run-btn">` 等实现，并绑定相应事件处理。
- **组件样式**：使用原生CSS或Sass组织各模块样式，采用统一的BEM命名或前缀（例如 `.module-panel .config-section`）避免冲突。布局上建议每个模块面板采用flex或grid布局，将配置区和展示区并列（默认左侧配置右侧结果，或者根据需要调整）。
- **图表/表格子组件**：对于复杂的结果展示，如趋势图、回测资金曲线、因子IC值图表，可以封装绘图逻辑为可重用模块（例如 `ChartComponent`）。虽然不使用前端框架，可借助轻量级库 (如Chart.js、Plotly等) 在需要时绘图，但其调用可视为子组件。表格展示可以用原生 `<table>`，并通过JS生成行列。将通用的表格渲染逻辑封装，以便候选池列表、回测绩效表等重复使用。
- **消息提示/对话框组件**：提供统一的消息提示框，用于错误警告（如数据缺失）、进度提示或结果概要。可实现一个轻量弹窗或提示条组件，在各模块需要时调用。例如在因子计算时如果用户未选资产，会通过此组件显示“未找到候选资产，请先完成候选池步骤”的提示。
- **代码编辑器组件**：策略开发模块内集成CodeMirror编辑器，可封装初始化代码编辑器的逻辑为组件（例如 `StrategyEditor`）。该组件负责在一个 `<textarea>` 或 `<div>` 容器内加载CodeMirror，应用基本配置（语言模式、主题等），并提供获取用户编辑代码的方法给模块使用。

组件划分小结: 我们通过**顶部导航**、**各模块面板**、**图表表格子组件**、**消息提示**以及**代码编辑器**等划分前端组件。每个模块面板内再细分配置和展示子区域。这种划分结合原生JS开发特点，实现模块独立、功能清晰的界面结构 [2]。模块化设计使得每个功能相对独立，可随时替换或扩展，比如数据获取模块的实现可按需更换，而不影响其他部分 [2]。

4. 数据流 / 状态流机制建议（无框架实现）

在无前端框架情况下，应设计**统一的数据状态管理**和**模块间通信**机制，以确保各模块联动和状态可追溯：

- **全局数据中台 (State Store)**：建立一个全局的`DataCenter`（数据中心对象）或称"数据中台"，用于集中存储各模块输出的数据和全局状态 [3] [4]。例如:

```
window.AppState = {
    rawData: null,
    marketTrend: null,
    investTheme: null,
    candidatePool: null,
    factors: null,
    strategy: null,
    backtestResult: null,
    liveTrade: null,
    currentStep: 1,
    // 以及其他需要的状态标记
};
```

  每个模块在完成时，将结果写入对应的`AppState`属性。这样后续模块可直接访问所需数据，而不需紧耦合调用。**数据中台**还可以包含方法，如获取某模块数据快照、清除数据等，以管理状态生命周期。采用集中存储使整个流程的数据可以追踪和统一管理 [5]。

- **消息总线 (Event Bus)**：实现一个全局**事件发布-订阅**系统，用于模块之间解耦通信 [6]。可自行编写一个简单的事件总线类（如上文引用的EventBus示例 [6]）或使用轻量库（如 mitt）。通过发布/订阅模式，各模块无需直接调用彼此的方法，而是监听特定事件：

- 事件定义：为流程中的重要节点定义事件名称，例如`"dataFetched"`, `"trendAnalyzed"`, `"themeSelected"`, `"poolGenerated"`, `"factorsReady"`, `"strategyReady"`, `"backtestDone"`, `"tradeUpdate"`等。
- 订阅事件：模块在初始化时向事件总线订阅相关事件。如回测模块订阅`"strategyReady"`事件，策略开发模块订阅`"factorsReady"`事件等。一旦事件触发，模块的回调函数就会被调用，实现相应操作 [7]。
- 发布事件：当模块完成其任务时，利用事件总线发布事件，附带需要传递的数据或状态标识。例如信息获取模块完成后执行`EventBus.publish("dataFetched", { data: ... })`，候选池模块完成后发布`"poolGenerated"`，并附加候选池列表（当然，由于我们有全局状态，也可以只传递简要信号让订阅方自行从AppState取数据）。

- **参数传递**：通过事件总线发布事件时可以传递参数，使订阅者能够获取到发布方的一些结果或标识 [8] [9]。例如发布 `"backtestDone"` 时附带回测结果概要，订阅的优化模块可直接使用。

- **模块联动逻辑**：借助数据中台和事件总线，实现模块自动联动及状态感知：

- 自动联动: 在"全流程执行"或用户开启自动模式时，使上一步完成事件直接触发下一步。例如，当收到 `"strategyReady"` 时，回测模块自动开始执行（读取 AppState.strategy）。这通过事件订阅实现，无需模块直接调用模块 [10]。
- 手动触发: 用户也可以手动逐步执行。即使未用事件触发，模块运行时仍从全局状态读取所需输入。例如用户直接跳到回测模块，也可以通过 AppState.strategy 获取已有策略数据（如果存在）进行回测。

- 状态感知: 每个模块在加载或运行前，可以检查所依赖的 AppState 是否有相应数据，没有则提示用户或灰掉运行按钮。这确保用户了解流程依赖，并提高可追溯性（知道缺少什么前置步骤）。例如如果用户尝试运行候选池模块但 AppState.investTheme 为空，则系统提示需要先完成投资主线选择。

- **数据追溯**：通过集中状态和事件日志，可实现操作过程的追溯。建议记录事件流或操作日志（可存在 AppState.logs 数组），记录每个模块何时运行、输出摘要和事件触发。这种日志可用于在界面底部状态栏回放，帮助用户查看流程中每步发生的情况，提升透明度和可调试性。

该数据流机制结合"数据中台+消息总线"实现了解耦和统一管理。一方面**消息总线**实现了跨模块通信和解耦 [6]，另一方面**数据中台**保证了数据的一致存储和获取 [5]。这种架构类似典型量化系统中的管理者模块和数据中心协同工作：管理模块负责调度指令，各功能模块独立运行，数据中心存储所有结果供后续使用 [11] [3]。

5. 模块联动示例：策略开发 → 回测验证 → 优化回写

以下以"策略开发→回测验证→优化调整"的联动过程为例，说明模块如何协同工作，实现自动化流程迭代：

- **策略开发完成触发回测**：假设用户在策略开发模块完成了策略配置，并点击"保存策略"。模块会将策略模型保存到 AppState.strategy，并通过 EventBus 发布事件 `"strategyReady"`，附带策略标识或名称。回测验证模块订阅了 `"strategyReady"` 事件，因此立即收到通知，提取最新策略数据。此时如果用户开启了自动联动模式或使用了一键全流程，回测模块会自动开始执行历史回测。界面上可自动切换到回测模块面板，让用户实时观察回测过程进度。

- **回测结果触发优化**：回测验证模块运行完毕后，计算出绩效指标和报告，保存到 AppState.backtestResult，并发布 `"backtestDone"` 事件（其中包含主要绩效指标，如收益率、回撤等）。假如系统设计了自动优化流程（例如针对策略参数的调优），优化过程模块（可以是策略开发模块内的一个功能或一个独立服务）已订阅 `"backtestDone"`。当该事件触发时，优化模块开始工作：读取回测结果中指标判断策略弱点，并调整策略参数（如修改因子权重、调节买卖阈值等）。优化计算完成后，将调整后的参数/策略**回写**到 AppState.strategy 中，并触发 `"strategyOptimized"` 或 `"strategyReady"` 再次通知。此时策略开发模块可以监听到新的策略更新事件，更新其界面显示（例如在代码编辑器中高亮修改的参数或在配置界面提示"参数已优化"）。

- **迭代循环**：上述过程可以根据需要迭代多次。例如系统可以在一次回测后自动尝试不同参数多轮回测（此时回测模块本身可以被优化模块多次调用）。每次新的回测完成都会发布 `"backtestDone"`，触发新的优

化，反复循环，直到达到某个停止条件（如性能指标达标或达到最大迭代次数）。**模块联动**确保了这一切均可自动完成，每个阶段输出自动成为下阶段输入，无需人工干预。

- 在UI上，可以让用户设置是否启用“自动优化”开关。如果启用，则订阅链激活；如果关闭，则在回测完成后仅提示结果，由用户自行决定下一步是否优化。
- 一旦用户对优化后的策略满意，便可手动触发进入实盘交易模块。优化模块在最终确定策略时也会发布最终的 `"strategyReady"`（或一个专门的 `"finalStrategyReady"`）事件给实盘模块，但通常仍需用户确认才真正执行实盘。
- **状态追踪**：在整个策略→回测→优化链路中，全局状态会保存每轮迭代的结果。例如 AppState 可以记录 `strategy.history` 列表，存放每次优化调整前后的参数集，以及对应的回测绩效。这使用户可以追溯每次调整如何影响结果，满足可追溯性需求。界面上也可提供查看历史优化记录的UI，帮助用户理解优化过程。

联动示例小结：通过事件总线的发布/订阅和全局状态共享，上述模块实现了自动衔接：策略开发完成触发回测，回测完成触发优化，优化更新后又反馈到策略模块，形成闭环的改进流程。这体现了插件设计对模块联动的支持和对流程的自动化处理。用户可以通过配置开关选择全自动迭代或手动逐步调整，以保持对流程的掌控。

6. 全流程“一键执行”和单模块独立运行机制建议

为兼顾新手快速体验和高级用户灵活控制，平台应同时支持**一键执行全部流程**和**各模块独立运行**两种机制：

- **一键执行全部流程**：
- **触发方式**：用户点击顶部导航栏的“运行全流程”按钮。一键执行会按照预定的模块顺序，从步骤1自动执行到步骤8。【实现上】可以在点击后，程序依次调用每个模块的运行函数，也可以设置全局标志并依赖事件链自动推进。推荐采用**事件链**：即设置一个 `runAll` 标志为true，然后触发执行信息获取模块，后续每个模块完成后如果 `runAll` 为true则自动触发下一个模块事件。这样模块间解耦的同时实现顺序执行。
- **默认配置**：为了全流程顺畅进行，每个模块在未单独配置时可以使用系统默认参数。例如信息获取默认获取预设市场数据范围，候选池默认筛选大盘股，因子构建默认选取常用因子模板等。用户依然可以在流程启动前调整这些参数；若未调整就开始全流程，将使用默认值以避免中途停顿。
- **执行过程**：一键运行后，UI应锁定当前流程不被手动干扰（或提供“暂停”按钮）。顶部进度指示应动态更新，显highlight当前执行模块。当某一步出现错误或需要用户决策（如策略开发步需要用户编辑策略逻辑），系统应暂停自动流程并提示用户介入，待处理完再继续。在自动模式下顺利跑完全流程后，可以给出总结提示（例如“全流程执行完毕，请查看结果”）。
- **错误处理**：若某一步失败（如数据源获取失败、回测出错等），系统应中止后续步骤，并在顶部或状态栏给出错误信息。此时“runAll”标志重置为false，用户可以修正问题后重新执行剩余流程或重新开始。
- **单模块独立运行**:
- **自由选择步骤**：允许用户点击任一步骤导航，进入对应模块界面，并手动运行该模块功能。这样用户可以针对某一步骤反复调试，例如多次调整因子参数重新计算，或多次回测优化策略，而不必每次重跑整个流程。

- **前置依赖检查**：独立运行时，模块需要检查自身运行所需的数据是否准备齐全。如果缺少前置步骤输出，则有几种处理策略：
    1. **禁用**：直接禁用“运行”按钮，并在模块区域显示提示“请先完成第X步”。例如用户直接跳到回测模块但还没有策略，则提示需要先完成策略开发。
    2. **请求使用默认/历史数据**：如果系统有上一次流程的数据缓存或默认示例数据，可询问用户是否使用这些数据继续。例如没有候选池时，可以提供一个默认候选池列表作为演示，让用户可以单独体验后续模块。
    3. **引导执行前置**：检测到依赖未满足时，引导触发前置步骤执行。比如在回测模块点击“运行”时如果没有策略，弹出对话框问是否先跳转到策略开发模块。用户同意则切换过去，引导其完成策略定义，再返回回测模块继续。
- **数据隔离**：独立运行某模块时，应保证不会无意修改其他模块状态，除非用户明确保存更改。例如用户单独重新运行候选池筛选，可以允许其更新 AppState.candidatePool（因为这本就是流程产出的一部分）。但是如果用户跳过策略模块直接更改某些全局参数，这些更改应在适当时机反馈。总的原则是模块独立运行时就像局部的小流程，其输出依然写入全局状态，但用户应知晓其他部分仍是之前的状态，避免混淆。

- **结果查看**：每个模块独立运行后，用户可以在该模块的展示区看到结果，并决定下一步。例如用户手动运行完因子构建，看到因子有效性后，可能重新回到候选池调整筛选条件，再次运行因子计算。系统应允许这种跳转反复，使用全局状态串联数据。**可追溯性**在此体现为：用户知道当前 AppState 中哪些数据是最新的（例如在UI上可标记“候选池(已更新)”），哪些可能已过期（如之前的趋势分析可能相对于最新数据已过期）。可以通过在导航步骤上增加状态指示（绿色对勾表示已完成，红色表示需要重跑）来提示。

- **模式切换**：提供清晰的模式区分机制：当“一键执行全部流程”按钮按下时，系统进入自动流程模式，UI上可能显示“批处理模式运行中...”；当用户手动逐步操作时，则处于交互模式。用户可在任何时刻取消自动模式（如点击暂停），切换回手动模式，以便插入人工决策。

机制综述：全流程按钮满足一键完成端到端流程的需求，适合在有默认策略模板或快速测试时使用，而单模块运行机制确保专业用户可以在任意环节进行微调和重复实验。两者结合提高了交互效率和灵活性，单人操作时既可一气呵成又能细节打磨，所有这些都在统一的状态管理和事件驱动框架下运作，保证流程各环节高度联通，同时保持必要的解耦和可控性 [2] 。通过这样的设计，整个八步投资流程插件将具备高效的交互体验、完善的可追溯数据链路以及可靠的模块协同能力。

---

[1] [2] 从拍脑袋到算出来：开源量化神器 FactorHub 深度分析 | 人人都是产品经理
https://www.woshipm.com/ai/6276681.html

[3] [4] [5] [11] 19 量化投资：典型的量化投资系统都包含哪些模块？ - 启航黑珍珠号 - 博客园
https://www.cnblogs.com/Linzj5950/articles/18509891

[6] [7] [8] [9] [10] 如何在 JavaScript 中实现事件总线 （Event Bus） - 独书先生 - 博客园
https://www.cnblogs.com/DuShuSir/articles/15997598.html