ChatGPT

基于 Cursor 的 PTrade 策略模板自动生成指南

在 **Cursor** 编程环境中调用大型语言模型（LLM）插件工具链，可以自动生成符合 **PTrade** 回测框架的策略模板代码，并确保代码结构与韬睿量化系统的投资逻辑兼容。下面将从框架定义、选股逻辑模块化、自动生成文件、可用工具和输出结构等方面进行详细说明。

1. PTrade 策略代码标准框架定义

PTrade 量化策略采用事件驱动框架，要求用户编写特定的回调函数，包括初始化函数和行情处理函数等 [1]。最基本的策略结构一般包含：

- **初始化函数**（`initialize(context)` 或称 `init(context)`）：策略开始时调用，用于设置初始状态（如设定股票池、基准、全局变量等）。这是必需的步骤 [1]。
- **行情处理函数**（`handle_data(context, data)` 或有时称 `handle_bar`）：每个事件周期调用，是策略的主体逻辑，用于根据每个时间步的市场数据进行交易决策，也是必需实现的函数 [1]。
- **可选的辅助回调**：包括盘前函数 `before_trading_start(context, data)`（每天开盘前执行，用于每日初始化，默认9:10触发）和盘后函数 `after_trading_end(context, data)`（每天收盘后执行，用于收盘后处理，默认15:30触发），根据需要添加 [2]。对于更高频策略，还有 `tick_data` 或 `run_interval` 等函数用于更细粒度的事件驱动 [3]。但 `initialize` **和** `handle_data` **是策略的最小必需结构** [1]。

以上函数组成了 PTrade 策略的标准框架。一个简单的模板代码例如：

```
# 策略模板示例
def initialize(context):
    """策略初始化：设定股票池、初始参数等"""
    # 设定基准和股票池
    set_benchmark('000300.SS')             # 例如设定沪深300指数为基准
    g.stocks = ['600519.SS', '000858.SZ']  # 设定观测股票池（示例为茅台和五粮液）
    set_universe(g.stocks)                 # 将股票池注册为策略关注标的
    # 其他初始化设置
    set_commission(commission_type=0, commission_list=[0, 0.001, 0, 0.001, 0,
5])  # 示例：设置手续费率
    log.info("Initialized strategy with universe: %s" % g.stocks)

def handle_data(context, data):
    """逐行情/逐Bar处理：实现买卖逻辑"""
    for stock in g.stocks:
        price = data[stock]['close']
        # 简单示例：均线策略
        hist = get_history(20, '1d', 'close', stock)
```

```
        ma5 = hist['close'][-5:].mean()
        ma20 = hist['close'][-20:].mean()
        if ma5 > ma20 and stock not in context.portfolio.positions:
            order_value(stock, context.portfolio.cash * 0.5)  # 金叉买入一半仓位
        elif ma5 < ma20 and stock in context.portfolio.positions:
            order_target(stock, 0)  # 死叉卖出清仓
```

上述代码演示了 PTrade 策略的基本结构，包括初始化设定股票池和参数，以及在 `handle_data` 中根据简单均线交叉策略进行买卖下单。实际编写时需要确保只使用 PTrade 平台支持的 API，否则会被视为违规调用 [4]。遵循这一标准框架能够保证策略代码与 **韬睿量化系统** 的投资逻辑和执行架构兼容。

2. 模块化封装韬睿选股逻辑

为了提高策略的可读性和复用性，可以将韬睿量化系统中的选股逻辑模块化，拆分为独立的筛选函数，并在主策略中调用。**模块化设计**可使代码结构清晰，方便扩展和定制 [5]。针对所提到的几类选股思路，例如"强主线"、 "技术突破"、 "ETF轮动"等，我们可以分别封装如下：

- **强主线选股**：强主线通常指当前市场中最强的主题或板块。可通过追踪行业/概念指数的涨幅或资金流入判断主线板块，然后选取其中的龙头股。例如，可以编写函数筛选最近若干日行业指数涨幅最大的行业，然后获取该行业内市值或成交额最大的几只股票作为候选。

- **技术突破选股**：技术突破筛选关注价格形态的突破点，比如股票突破重要阻力位、创出新高或放量上涨等信号。可实现一个函数检查股票是否突破前高、突破某均线压力等条件，返回满足条件的股票列表。

- **ETF 轮动选股**：ETF轮动通过比较多只 ETF 的表现，轮动持有表现最强的一只。典型方法是计算各ETF的动量或涨幅指标，选择动量最强的标的。如果当前持有的ETF不再最强，则调仓换入更强的ETF。例如有名的"二八轮动"策略，即在大盘股指数ETF与小盘股指数ETF之间轮动：当动量指标为正且无持仓时，买入动量较大的一个；动量转负时平仓，每日收盘判断是否调仓 [6]。

以上逻辑可以分别实现为函数。例如，下面给出**ETF轮动**逻辑的一个简化代码示例，将两只指数ETF基于20日动量进行轮动：

```
# 示例：二八轮动策略的选股/调仓逻辑封装
def select_top_index(context, data):
    """基于20日动量指标选择当前最强指数ETF"""
    etfs = ['510050.SS', '510300.SS']  # 上证50ETF 和 沪深300ETF
    # 获取20日收盘价历史
    hist = get_history(20, '1d', 'close', security_list=etfs)
    # 计算动量指标 = (今日收盘价 - 20日前收盘价) / 今日收盘价
    momentum = {}
    for code in etfs:
        price_today = data[code]['close']
        price_20d_ago = hist['close'][code][0]
        momentum[code] = (price_today - price_20d_ago) / price_today
    # 找出动量最大的ETF
```

```
    top_etf = max(momentum, key=momentum.get)
    return top_etf

def handle_data(context, data):
    # 调用选股逻辑获取当前应持有的ETF
    top_etf = select_top_index(context, data)
    # 获取当前持仓情况
    holdings = context.portfolio.positions
    # 卖出不再是最强主线的ETF
    for etf in ['510050.SS', '510300.SS']:
        if etf != top_etf and holdings.get(etf, {}).get('amount', 0) > 0:
            order_target(etf, 0)  # 将非最强ETF清仓
    # 买入当前最强ETF（如尚未持有）
    if holdings.get(top_etf, {}).get('amount', 0) == 0:
        order_target_value(top_etf, context.portfolio.cash)  # 全仓买入最强ETF
```

上例中，我们定义了 `select_top_index` 函数来计算并返回动量最强的ETF，然后在 `handle_data` 中调用它，实现简单的指数轮动策略。这一思路与常见的动量轮动模型一致 [6] 。类似地，我们可以实现：

- `filter_strong_mainline(context, data)` 函数：返回当前市场**主线板块**中的强势股名单。例如通过行业指数涨幅排名筛选领涨板块，再选其中龙头个股。
- `filter_breakout_stocks(context, data)` 函数：返回近期出现**技术突破**信号的股票清单。比如筛选突破前高且成交量放大的个股。
- `filter_factor_stocks(context, data)` 函数：实现**因子选股**策略，根据基本面或技术面因子打分筛股。例如多因子模型综合价值（PE/PB）、质量（ROE/利润增长）和动量等因子，对股票评分并选出前N名。 [7]

将选股逻辑拆分成独立函数后，主策略（`handle_data`）中就能灵活调用这些筛选模块。例如，可以先用 `filter_strong_mainline` 得到强主线股票池，再在此基础上调用 `filter_breakout_stocks` 进一步筛选出技术形态突破的个股，最后对结果下单交易。通过模块化，**策略逻辑更加清晰**，也方便根据需要替换或组合不同的选股模块。

3. 自动生成策略代码、文档及回测配置

借助 LLM 插件，可以自动生成完整的策略项目，包括策略代码（`.py` 文件）、策略说明文档（Markdown）以及回测配置参数文件（JSON/YAML）。在 Cursor 中，这可以通过与 AI 的对话一步完成，而不需要手动逐个复制粘贴到不同文件 [8] 。实现流程如下：

- **策略代码生成**：在 Cursor 聊天面板（快捷键 Ctrl+K）输入指令，请求 AI 基于给定策略思路编写 PTrade 策略代码。例如：“根据均线交叉策略，生成包含 *init* 和 *handle_data* 函数的 *PTrade* 策略代码”。由于我们希望生成的是完整代码文件，可以明确告诉 AI 输出Python代码片段。LLM 将参考我们提供的描述，输出符合 PTrade API 规范的代码框架（包含上述标准函数和逻辑）。

- **策略说明文档**：可以让 AI 根据策略代码自动生成配套的说明文档，格式为 Markdown。例如："为上述策略代码撰写一份策略说明文档，包含策略原理、选股逻辑、买卖规则等。" 文档应简要说明策略意图和参数设置，方便日后查阅或与他人交流。

- **回测配置文件**：为了方便调整和记录回测参数，我们可以使用 JSON 或 YAML 文件存储回测配置。可请 AI 生成这样一个配置模板。例如："生成该策略的回测配置文件模板，格式为*JSON*，包含起始日期、结束日期、初始资金、基准、滑点、手续费等参数。"LLM 输出的结果可能如下（JSON 示例）：

```
{
    "start_date": "2020-01-01",
    "end_date": "2023-12-31",
    "benchmark": "000300.SS",
    "initial_cash": 1000000,
    "slippage": 0.001,
    "commission_rate": 0.0003,
    "frequency": "1d"
}
```

这个配置文件涵盖了常见的回测设定，例如时间范围、基准指数、初始资金、滑点和佣金费率等。实际使用时，可将其解析后应用于回测引擎，或者直接在策略代码的 `initialize` 中通过相应 API（如 `context.capital`、`set_slippage` 等）设置等效的参数 [9]。

在 Cursor 中，生成上述多个文件非常高效。得益于 Cursor 的 **Composer** 功能，我们可以在一次对话中让 AI 同时输出多个文件的内容 [8]。例如，可以要求："请创建一个名为 `my_strategy.py` 的文件，写入策略代码；创建 `README.md` 写入策略说明；创建 `config.json` 写入回测参数。" AI 会给出包含各文件内容的答复，我们只需确认并保存相应文件即可。这种**一键多文件生成**能力显著提升了策略开发的效率。

4. 可用插件、开源模板和教程

为实现上述自动化流程，可以利用一些现有的插件和资源：

- **DeepSeek 策略生成**：DeepSeek 是一套专为量化交易开发的 AI Agent 工具。在量化社区有不少使用 DeepSeek 辅助 PTrade 策略开发的案例 [10]。其方法通常是先将 **PTrade API文档** 等资料作为知识库提供给模型，然后通过精心设计的提示词让模型依据模板要求输出代码 [11]。比如前述的策略框架规范（包含初始化、事件函数的说明）就常用于提示模型严格遵守 PTrade 策略结构 [12]。DeepSeek 能结合 **提示词工程**、**检索增强生成（RAG）**等技术，自动生成较为完整的策略代码，并迭代优化。这与我们在 Cursor 中手动提供规范和需求让 GPT 模型生成代码的思路类似。有关 DeepSeek 生成代码的详细教程，社区中有不少分享（知乎、CSDN 等都有**保姆级**教程介绍如何使用 DeepSeek 辅助编程）。

- **Cursor 内置 AI / Copilot**：Cursor 本身内置对 GPT-4、Claude 等模型的支持，并提供类似 GitHub Copilot 的智能补全功能 [13]。利用 Cursor，我们在编写策略时可以实时获得代码片段建议，或者通过对话要求 AI 完成特定功能。例如，在编写 `handle_data` 时让 Copilot 自动补全下单逻辑。在 Cursor 环境中，由于它能索引整个项目代码，可以在上下文中理解策略结构，从而给出更准确的修改建议 [14]。如果已经有 PTrade 策略的开源模板或过往示例代码，放入 Cursor 工作区中，AI 可以参考这些文件风格，生成兼

容的代码。GitHub 上也有一些类似 PTrade 的**本地回测框架**（如 *SimTradeLab* [15]），包含了PTrade API的实现和示例策略，我们可以将这些示例作为模板。使用 Cursor 的 “@文件名” 引用功能，还可以在对话中调用这些模板片段，让 AI 在此基础上修改生成新策略。

- **开源策略模板**：社区中存在一些共享的 PTrade/QMT 策略模板工程。例如，有开发者整理了常用策略如均线交叉、海龟交易、行业轮动、多因子选股等的代码，实现了在不同平台间转换的脚本 [16]。这些模板通常包含规范的策略框架和注释说明。我们可以将此类模板项目导入 Cursor，然后使用 AI 的 **重构/编辑** 指令，根据我们的需求调整策略逻辑。例如，把已有的“双均线策略模板”修改为“RSI指标策略”。这样借助现有模板和 AI的代码生成能力，可以快速开发出定制策略。

在运用上述工具时，要注意模型可能引入的错误。例如 PTrade API 的函数名、用法细节，需要通过**对照官方文档**或**调试**来验证。为减少大模型的幻觉和错误，可在生成后让 AI 执行**规范核查**——例如让其检查代码中是否有未定义变量、未使用的库等，并根据 PTrade API 列表移除不支持的调用 [4] [17]。一些教程中建议了这样的多轮对话流程，以提高最终代码的正确性 [18]。

5. 策略项目结构及部署建议

为了方便管理和后续部署，我们可以将生成的策略及相关文件组织成规范的项目结构。例如：

```
my_strategy/                # 策略项目文件夹
├── strategy.py            # 策略代码（含 initialize、handle_data 等函数）
├── config.json            # 回测参数配置文件
└── README.md              # 策略说明文档
```

（当然，文件格式可按需调整，如配置文件使用 YAML，说明文档使用 Markdown 或 HTML 等。）

在将策略投入 **PTrade 回测**或实盘部署之前，请注意以下几点：

- **策略结构约束**：确保策略代码包含 PTrade 要求的所有必要函数，并遵循平台的生命周期调用顺序。如前所述，`initialize` 和 `handle_data` 是必需的，若使用盘前/盘后函数则要符合其调用时机 [1]。切勿在不允许的阶段调用特定 API（比如避免在 `handle_data` 中做订阅推送类操作），以免逻辑出错。

- **全局变量使用**：PTrade 提供了全局变量 `g` 来在多个回调间共享状态。使用 `g` 时需留意持久化行为——回测/实盘过程中，`g` 中可序列化的变量会在策略暂停后保存并恢复。不支持序列化的对象（如打开的文件句柄、自定义类实例等）不要存入 `g`，或者以 `__` 双下划线开头命名以避免序列化 [19]。合理利用全局变量可以在 **initialize** 中传递参数、在盘中累积信号等，但也要防止滥用导致状态难以跟踪。

- **配置参数校验**：如果使用了独立的配置文件，部署时应加载并验证这些参数。例如，检查起始日期格式是否正确，手续费设置是否合理等。PTrade 回测系统本身也允许在界面上设定这些参数，如果代码中也设置了，要确保两者一致或不会冲突 [20] [21]。

- **逐步测试与优化**：生成的策略模板往往是一个起点，建议先用少量数据在本地或模拟环境下测试，确保选股函数和交易逻辑按预期运行，再运行完整回测以评估绩效。Cursor 提供了 **Interpreter** 模式，可以在对话

中执行 Python 代码片段做快速验证 [22]。利用这一功能，可以在不离开编辑器的情况下，测试某段选股函数对给定数据的输出，或模拟一次买卖流程，以便快速修正潜在的问题。

最后，借助 **Cursor + LLM** 实现 “一键生成选股逻辑+策略代码+回测配置+文档” 是非常具有前景的开发范式。通过清晰地定义策略逻辑，并充分利用插件与模板，我们可以大幅缩短策略开发周期。从**策略思路**到**代码落地**，再到**回测验证**，整个流程高度自动化，使得量化投资策略的开发和迭代进入一个高效的新阶段。 [11] [8]

参考资料：

- PTrade 量化交易 API 官方文档 [1] [1]
- SimTradeLab 开源项目文档（PTrade 本地回测框架） [5] [15]
- CSDN 博客：《二八轮动策略_PTrade 策略代码》 [6]
- CSDN 博客：《AI量化学习资料 - 用 DeepSeek 玩转 PTrade 策略开发》 [11] [12]
- 知乎专栏：有关 Cursor AI 编程和多文件代码生成的使用教程 [8] [14]

---

[1] [19] Ptrade 量化交易 API接口文档
https://ptradeapi.com/

[2] [3] [4] [11] [12] [17] 1、AI量化学习资料 - 用DEEPSEEK玩转PTrade策略开发.zip\AI量化学习资料 - 1、PTrade策略开发提示词（参考模板）.md_ptrade所有api函数接口清单-CSDN博客
https://blog.csdn.net/weixin_36595077/article/details/145715207

[5] [15] GitHub - kay-ou/SimTradeLab: SimTradeLab is an open-source backtesting framework inspired by PTrade's event-driven architecture. It features a lightweight, modular design and full syntax compatibility, enabling seamless strategy development and validation.
https://github.com/kay-ou/SimTradeLab

[6] 股市轮动策略-CSDN博客
https://blog.csdn.net/u012724887/article/details/105413427

[7] PTrade量化策略示例——三因子策略（附源码） - 知乎专栏
https://zhuanlan.zhihu.com/p/1921527574166769689

[8] 终于有人讲清楚了！Cursor保姆级教程，错过后悔
https://zhuanlan.zhihu.com/p/722268566

[9] [20] [21] iQuant平台用python编写简单投资策略常用函数_iquant csdn-CSDN博客
https://blog.csdn.net/qq_62939934/article/details/130324521

[10] 王炸组合：用Deepseek零基础玩转PTrade策略开发（附 ... - 知乎专栏
https://zhuanlan.zhihu.com/p/24196110828

[13] 全网最全面详细的Cursor使用教程，让开发变成聊天一样容易原创
https://blog.csdn.net/m0_68116052/article/details/142832657

[14] 编程神器 Cursor：怎么用 AI 写一个完整的项目？出了城，吃着火锅还唱着歌，突然就把项目写完了！_cursor写项目-CSDN博客
https://blog.csdn.net/qq_41739364/article/details/144270234

[16] 使用Claude Code 一键转换聚宽策略到QMT/Ptrade，轻松实现多平台 ...
https://zhuanlan.zhihu.com/p/1960076831534330575

[18] Deepseek生成的代码怎么使用？PTrade到底怎么结合 ... - Bilibili
http://www.bilibili.com/read/cv41196481/

[22] 程序员的AI 神器：Cursor - 巨人肩膀
https://www.atbigapp.com/blog/1875512627412787200