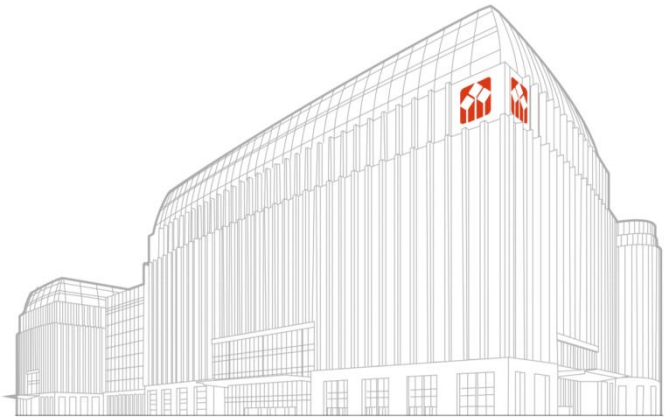
华泰证券 MQUANT 量化交易

用户使用手册

V3.1



文档版本记录

版本号	发布日期	修订内容	修订人
V2.0	20200425	1、 新增期权程序化交易部分内容； 2、 新增 python 编辑器支持代码错误实时检查； 3、 支持 python 策略调试； 4、 支持 C++版本策略。	邱小铭
V2.1	20200616	1、 新增 C++版本介绍 2、 新增批量固化、批量启动策略介绍 3、 调整文档结构	邱小铭
V2.2	20200804	1、 新增手工干预介绍 2、 新增逐笔数据线程池回调 3、 实例参数修改窗口常驻 4、 新增支持 TWAP、VWAP 算法 5、 新增播放 wav 音频文件	邱小铭
V2.3	20201017	1、新增设置实例备注接口 2、新增统一报单接口 3、新增历史 tick 查询接口 4、Level-2 tick 数据新增字段 5、交易时段放开回测，每个时间切片限速 1s 6、K 线查询精确到分钟	邱小铭
V3.0	20210107	1、 新增手工干预、资金推送、持仓推送、实时资金流向、策略准备停止等回调 2、 补充实时资金流向等数据结构 3、 删除模拟交易相关说明，模拟交易下线 4、 新增 API 接口，包括：策略声音提示、弹框提示、统一报单接口、可转债转股接口、查询历史 tick 接口、查询 ETF 信息、创建拆单算法等	邱小铭
V3.1	20210127	1、 新增高速模式配置 2、 新增支持多线程配置 3、 新增报撤单异步响应推送（默认不推送，需要调用接口打开开关才会推送） 4、 部分接口返回值修改（兼容） 5、 部分遗漏接口补充 6、 两种打开 MQuant 菜单崩溃问题的	邱小铭

		自助解决方案 7、 新增参数定义说明	

目 录

1 MATIC 机构交易平台简介

华泰证券 MATIC 机构交易平台为专业客户提供涵盖极速行情、极速交易、算法交易、量化平台、合规风控在内的一篮子量化交易解决方案，通过金融科技赋能，提升交易体验，助力客户捕捉投资机会，赢得先机。

MATIC 集基础交易、两融交易、组合交易、ETF 套利、策略交易、算法交易、量化交易、风控管理、运营管理、投后分析等功能服务于一体，为您提供投资全流程服务。

华泰证券为您提供一对一的客户服务。如有疑问或需要技术支持，请随时联系我们。

技术支持电话　025-83387904 ，　025-83387903

技术支持 QQ　2159615762，3152798201，2119948171，2835459142

2 运行环境

MATIC 支持的操作系统包括：Win 7 及以上版本，包括 32 位、64 位版本，推荐使用 win7 64 位操作系统。

3 安装、更新、登录、登出、设置

3.1 安装

MATIC 采用绿色免安装方式，下载 MATIC 客户端压缩包后，请解压，然后双击文件夹中的 matic.exe 文件，即可启动客户端程序。

客户端下载地址：https://huatech.htsc.com.cn/documents?navId=2&treeId=7

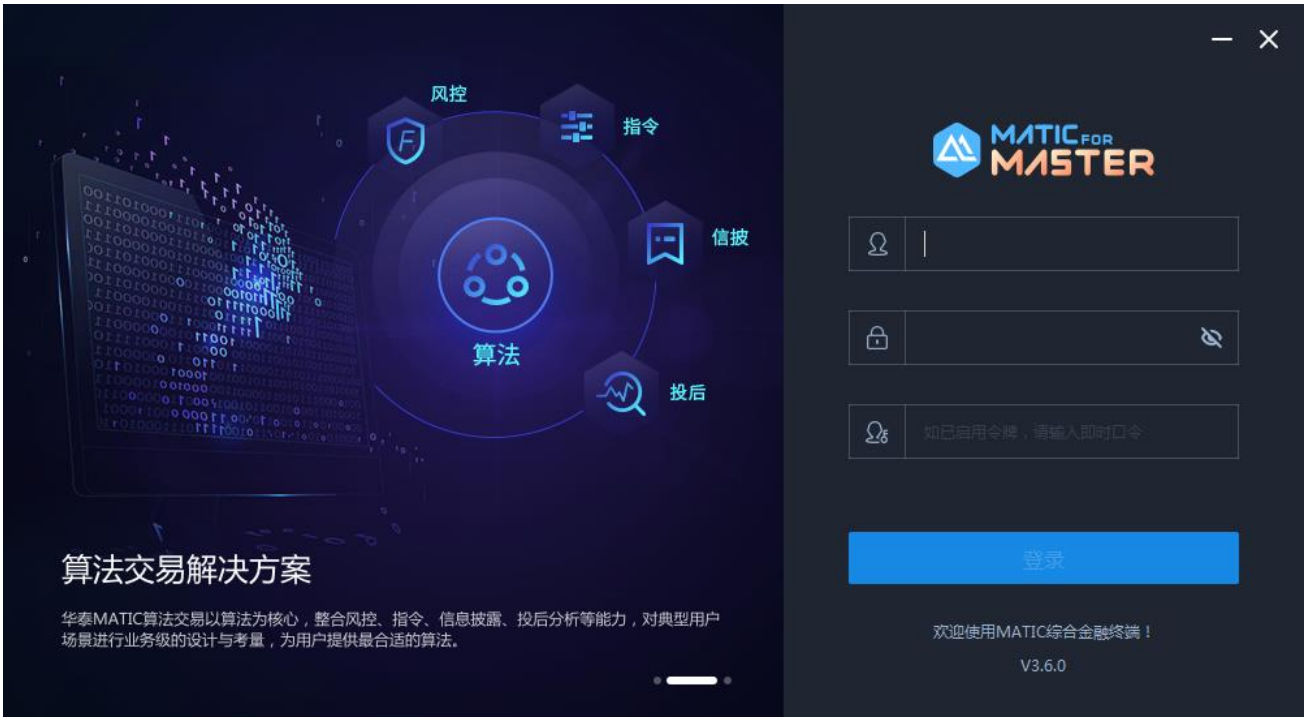
3.2 更新

MATIC 采用增量更新方式，打开客户端后系统将自动检查客户端版本信息，如有新版本将直接升级，升级界面如下：

若自动更新失败，请手动下载最新版客户端，下载地址：
https://huatech.htsc.com.cn/documents?navId=2&treeId=7

3.3 登录

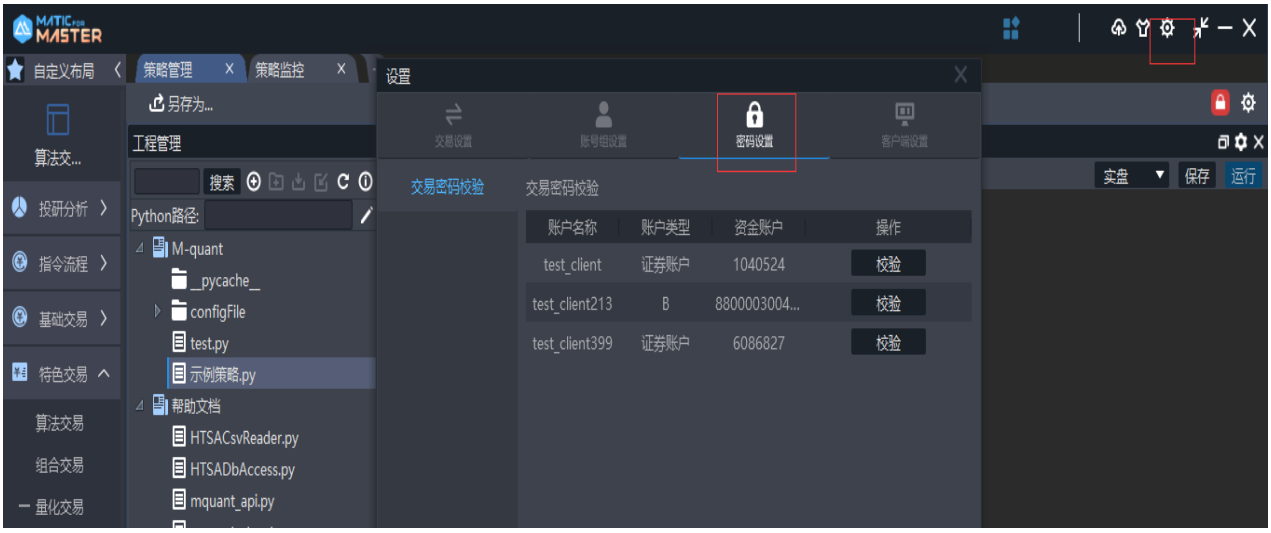
双击 matic.exe 文件即会弹出如下所示的登录界面：

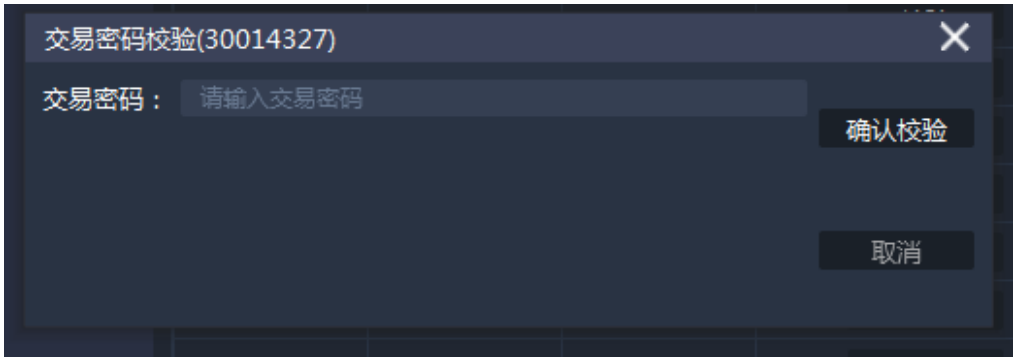


输入华泰证券分配给您的账号和密码，点击“登录”进入客户端主界面，默认进入基础交易界面。

初次登录需要修改密码。

登录进来之后，如未校验资金账号密码，首先需要点击最顶端的设置图标→密码设置，校验资金账号密码并重新登录，如已经校验过资金账号密码则不需重复验证，如下图所示：





3.4 登出

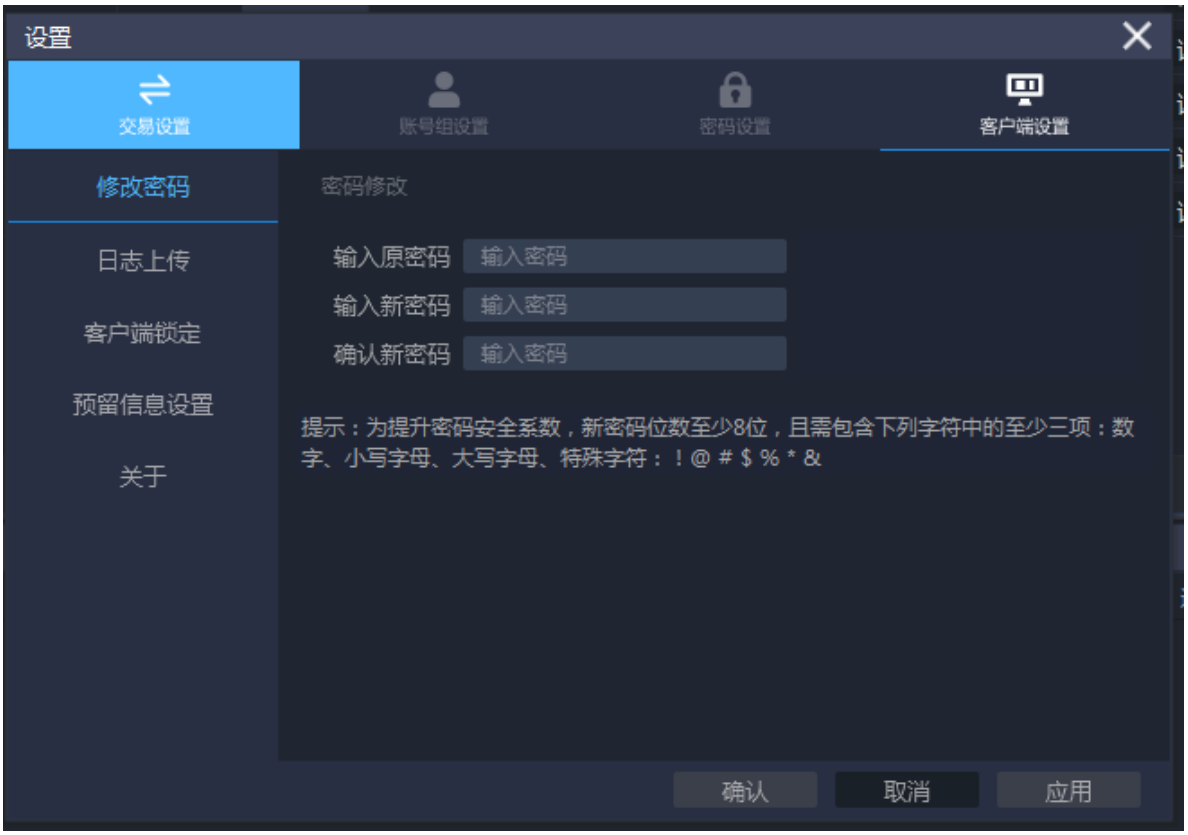
在登录时点击客户端右上角的关闭按钮即退出系统。

3.5 交易设置

MATIC 为平台上的所有交易功能提供了交易设置的功能，您可根据自己的使用习惯和需求对相关功能进行设置，提升交易过程中的使用体验。具体设置界面如图：

3.6 客户端设置

MATIC 为使用者提供了客户端密码修改、日志上传、客户端锁定等功能设置。具体设置界面如图：



4 MQuant 量化交易简介

4.1 简介

MQuant 是 MATIC 系统提供的客户端量化交易模块，其主要特点是多语言、简单、高效、本地化和可扩展。

多语言：针对不同用户的编程能力、以及对策略执行性能、并发处理能力的不同要求，MQuant 提供了对 C++和 python 两种主流量化编程语言的支持。如果客户对交易性能和并发处理能力要求不高，可以选择简单的 python 语言，很快搭建自己的量化交易系统；如果客户具备专业编程能力，且策略对执行性能和并发处理能力都有很高的要求，则可以使用 C++来搭建量化交易系统。

简单：MQuant 采用事件驱动模式，提供非常简单的 api 接口，同时提供丰富的策略编写教学素材，用户很容易就可以掌握。对使用 python 语言的用户而言，MQuant 集成了一个仿 pycharm 的 python 编辑器，支持代码联想、自动补全、快捷操作、错误实时提示、调试等功能，用户无需单独安装 python 环境即可使用。对于使用 C++语言的用户，只需要在创建策略时关联一个本地任意位置的动态库即可正常启动策略，而且策略可以即时更新生效。

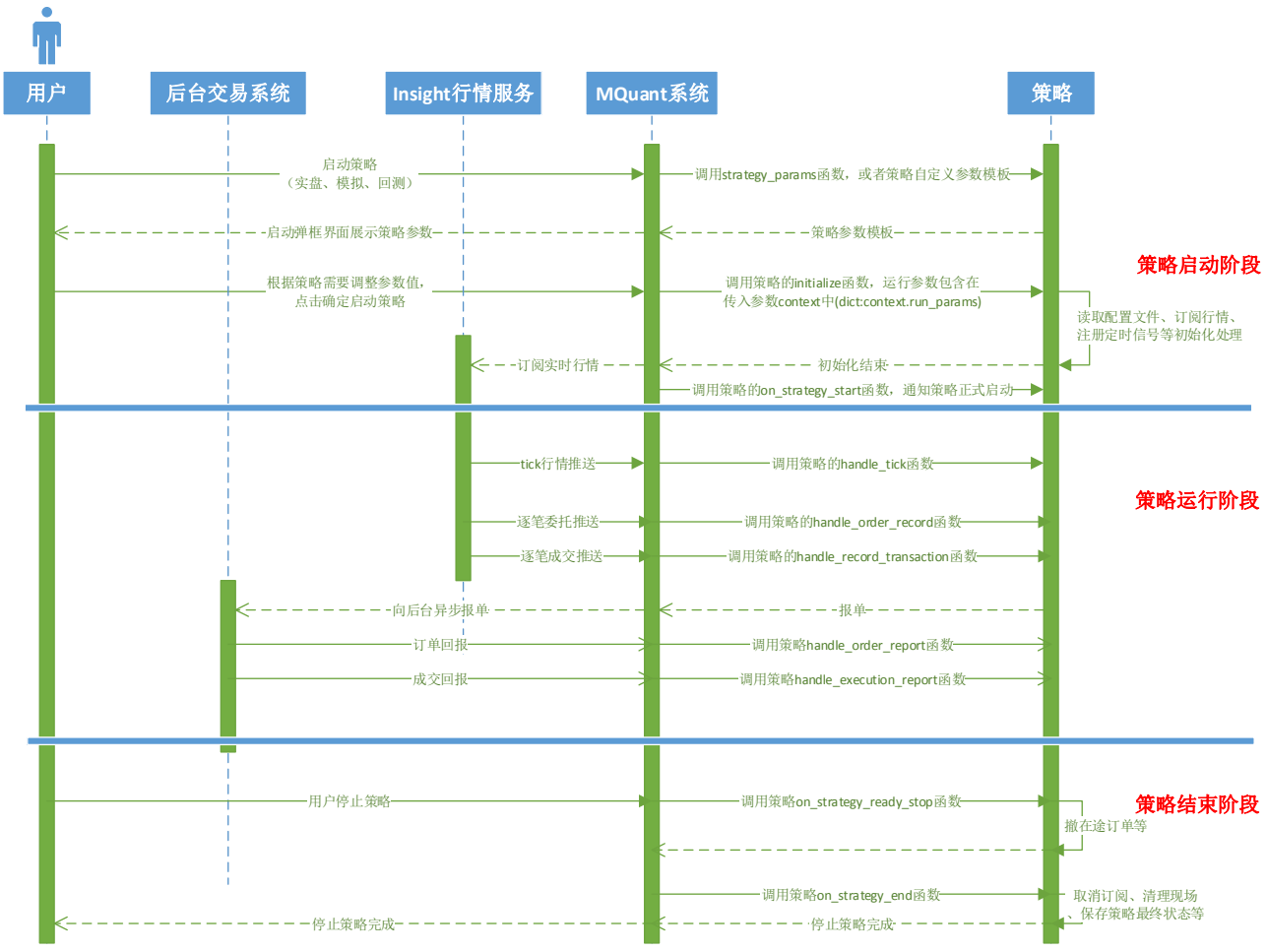
本地化：MQuant 的策略在用户本地运行，避免了策略泄漏的风险，具有极强的保密性。

可扩展：MQuant 能够利用 python 的量化生态或者 C++的任意第三方库扩展系统能力，例如，如果需要计算 MA 均线，可以使用 python 的 talib 库进行计算；用户从互联网下载的数据也可以通过 pandas 加载到策略实例中，为策略实例的运行提供更丰富的环境。

MQuant 支持的业务品种包括股票（包含科创板）、基金、债券（包含可转债）、ETF 申赎、两融、股指期货、沪深期权以及 MATIC 提供的各种算法；行情数据支持 Level-2 的实时行情和逐笔委托、逐笔成交以及多粒度的分钟 k 线、日 k 线数据。

功能列表		
序号	**功能名称**	**功能说明**
1	策略编辑	策略的编写、调试、实盘运行、回测、模拟交易等功能
2	策略执行监控	策略管理、策略实例列表、策略运行日志、策略委托查询、策略成交查询、持仓查询等功能
3	策略复盘	记录策略运行过程中的每个细节（包括订单中间状态等），并在操作上以时间为基准进行联动，方便复盘策略运行过程。
4	回测绩效分析	提供策略回测的绩效分析能力，包括绩效概览、交易详情和每日持仓

4.2 策略执行流程



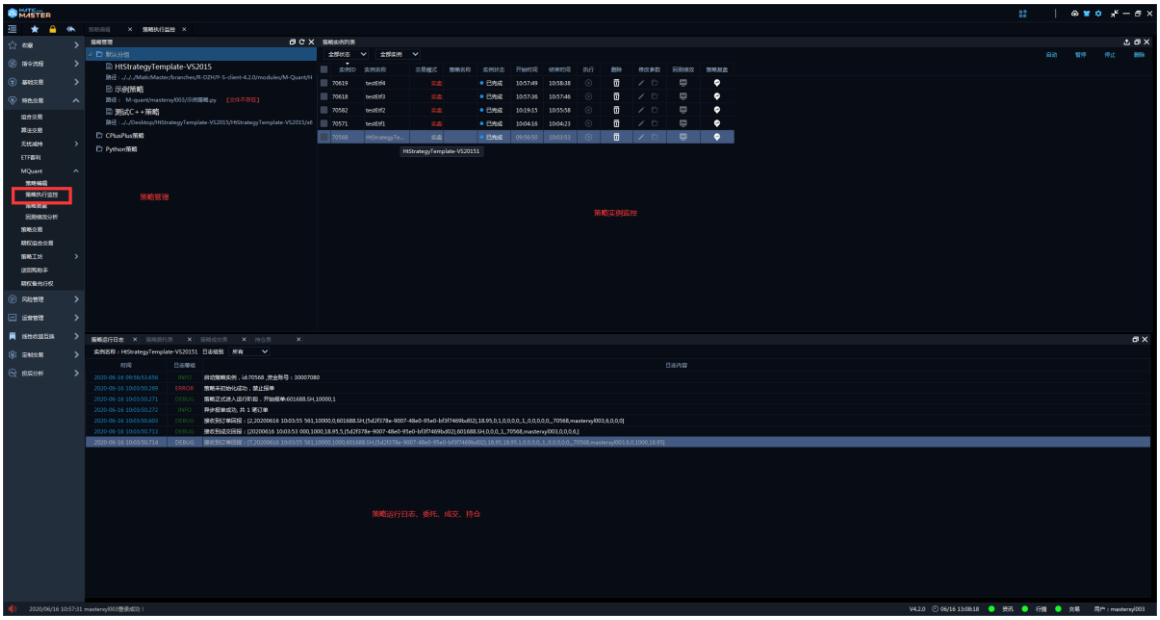
MQuant 是一个事件驱动策略引擎，策略的执行是通过 MQuant 系统内部的事件来驱动的，事件包括：

（1） 控制事件：策略启动、停止、暂停、恢复
（2） 行情事件：tick、逐笔委托（仅深市标的支持）、逐笔成交（沪市逐笔成交不包含撤单成交）、分钟 k
（3） 交易事件：订单回报（订单状态有任何变化都会推送）、成交回报（订单产生成交时推送）
（4） 时钟事件：周期定时信号、单次定时信号
（5） 其他事件：策略正式启动、参数修改、ETF 实时 IOPV、开盘信号（仅回测）、收盘信号（仅回测）

对于行情信号、定时信号和 ETF 实时 IOPV 而言，客户需要主动调用 subscribe 订阅才能接收到推送，其他事件客户只需要实现相应的回调函数即可正常接收到对应的事件。

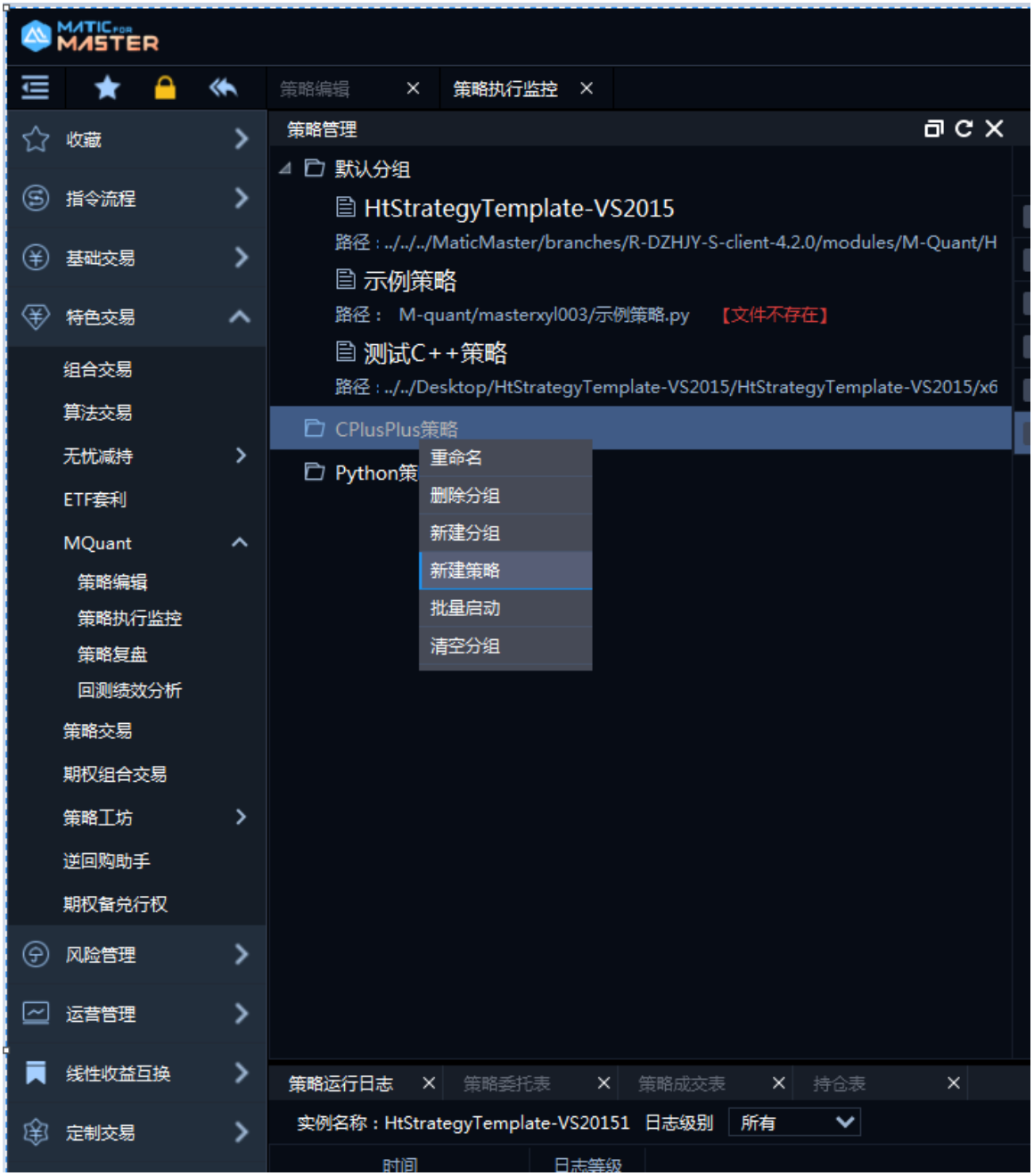
5 策略执行监控

点击“特色交易→量化交易→策略监控”，打开策略监控页面，能看到策略管理页面大概包含左侧的菜单栏、策略管理、策略实例监控、策略运行日志和策略委托表、策略成交表和持仓表几个部分，如下图所示：



5.1 策略管理

策略管理界面位于策略执行监控的左上角，每个策略与一个可执行的策略文件（python 脚本或者 dll）关联。Python 策略可以在策略编辑界面通过固化策略的方式产生一个策略（参考 4.1.4 锁定和固化策略），此外，Python 和 C++的策略都可以直接在策略管理界面的树状结构上右键点击-》新建策略，通过弹出的新建策略弹框来关联策略和对应的策略文件，如下图所示：


MATIC FOR MASTER
策略编辑
策略执行监控
收藏
指令流程
基础交易
特色交易
组合交易
算法交易
无忧减持
ETF套利
MQuant
策略编辑
策略执行监控
策略复盘
回测绩效分析
策略交易
期权组合交易
策略工坊
逆回购助手
期权备兑行权
风险管理
运营管理
线性收益互换
定制交易
策略管理
默认分组
HtStrategyTemplate-VS2015
路径：../../../MaticMaster/branches/R-DZHJY-S-client-4.2.0/modules/M-Quant/H
示例策略
路径： M-quant/masterxyl003/示例策略.py 【文件不存在】
测试C++策略
路径：../../Desktop/HtStrategyTemplate-VS2015/HtStrategyTemplate-VS2015/x6
CPlusPlus策略
Python策
重命名
删除分组
新建分组
新建策略
批量启动
清空分组
策略运行日志
策略委托表
策略成交表
持仓表
实例名称：HtStrategyTemplate-VS20151
日志级别
所有
时间
日志等级

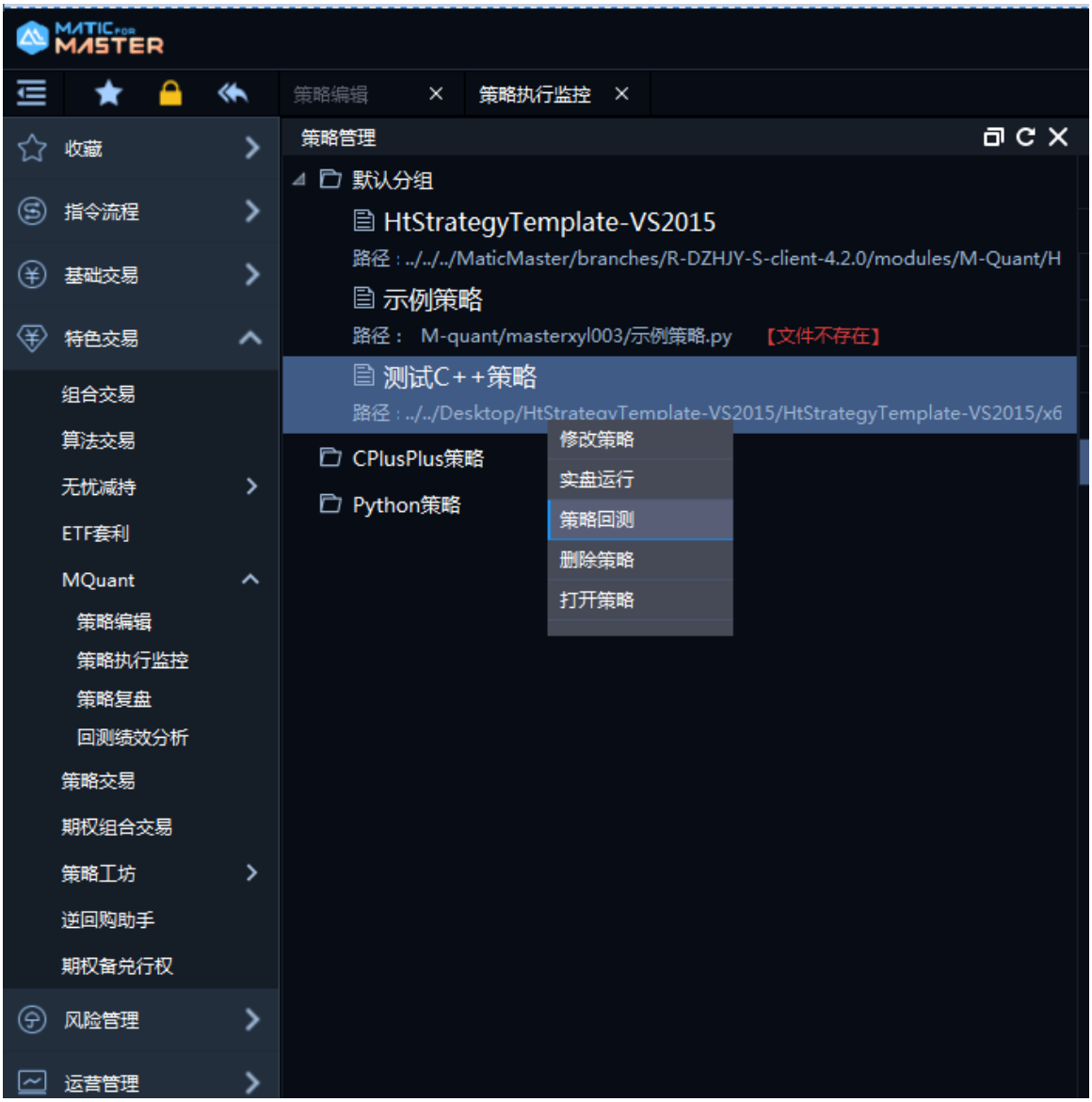
支持基于同一个策略文件批量创建一批策略，同一批次策略的批次序号从 0 开始，依次加 1 递增，同时在上下文变量 context 中可以获取到当前策略的批次序号。

5.2 策略执行

双击策略管理界面上的策略，会弹出策略运行框，如下图所示：

用户可以在界面上设置策略的资金账号，修改运行参数，设置策略窗口的显隐，以及可以选择在策略中接收所勾选资金账号的所有委托、成交推送。点击确定，策略正式启动。

策略回测的功能入口在策略的右键菜单上，如下图所示：



点击策略回测，弹出策略回测的参数弹框，如下图所示：

用户可以在此界面上设置回测参数，并进行回测。

5.3 策略实例列表

如果在策略管理中运行了策略，则可以在策略列表中看到策略实例以及策略实例的状态，可以通过筛选查看不同状态的实例，也可以通过点击实例列表顶部的启动、暂停、停止、删除按钮来操作实例，也可以通过点击“执行”下的按钮来启动或暂停策略，如下图所示：

5.4 策略运行参数

5.4.1 参数定义

MQuant 采用用户自定义参数的模式，用户只需要在策略脚本中定义 def strategy_params()函数，然后函数返回符合格式要求的 dict 或 json string，系统即可解析参数并展示在策略启动前的参数界面上，如下图所示：

```python
def strategy_params():
    """
    策略可自定义运行参数，启动策略时会写入到context对象的run_params字段内
    :return:dict对象，key为参数名，value为一个包含参数默认值、参数描述（选填）的字典
    :remark:可选实现
    """
    # 示例如下：
    dict_params = {
        '证券代码': {'value': '000001.SZ/000002.SZ', 'desc': '交易标的'},  # 'desc'字段可填写，也可不填写
        '买入价格': {'value': '17.50/27.50'},
        '卖出价格': {'value': '18.50/28.00'},
        '撤单时间间隔':{'value': 10}
    }
    return dict_params
```

dict 模式

```python
def strategy_params():
    dict_params = {
        '自定义字段': {'value': ['支持', '不支持'], 'desc': '下拉列表，是否支持自定义字段，默认支持'},
        '策略类型': {'value': ['主策略', '子策略'], 'desc': '主策略负责接收消息，发现不是当前策略资金账号的，则通过消息分发出去，子策略接收到消息后再进行处理'},
        '备份文件数量': {'value': 0, 'desc': '备份文件数量，0表示不留备份'},
        '废单字段补全': {'value': True},
        '仅下载MQuant订单': {'value': True, 'desc': '如果为False，表示下载勾选资金账号的所有订单，如果为True，则只下载inst_id不为空的订单'},
        '操作记录': {'value': False, 'desc': '如果为False，表示不单独记录用户操作和废单信息，设为True可以记录'},
        '算法实例下载时间间隔': {'value': 0, 'desc': '设置算法实例下载的时间间隔，单位为秒，0表示不下载'},
        '仅更新订单回报': {'value': False, 'desc': '勾选此参数时，盘中[9:00-15:00]仅会下载order_xxxx.csv文件'},
        '全量下载所有数据时间': {'value': '15:00:05', 'desc': '设置全量下载所有委托、成交、资金、持仓数据时间'}
    }
    return json.dumps(dict_params)
```

json string 模式

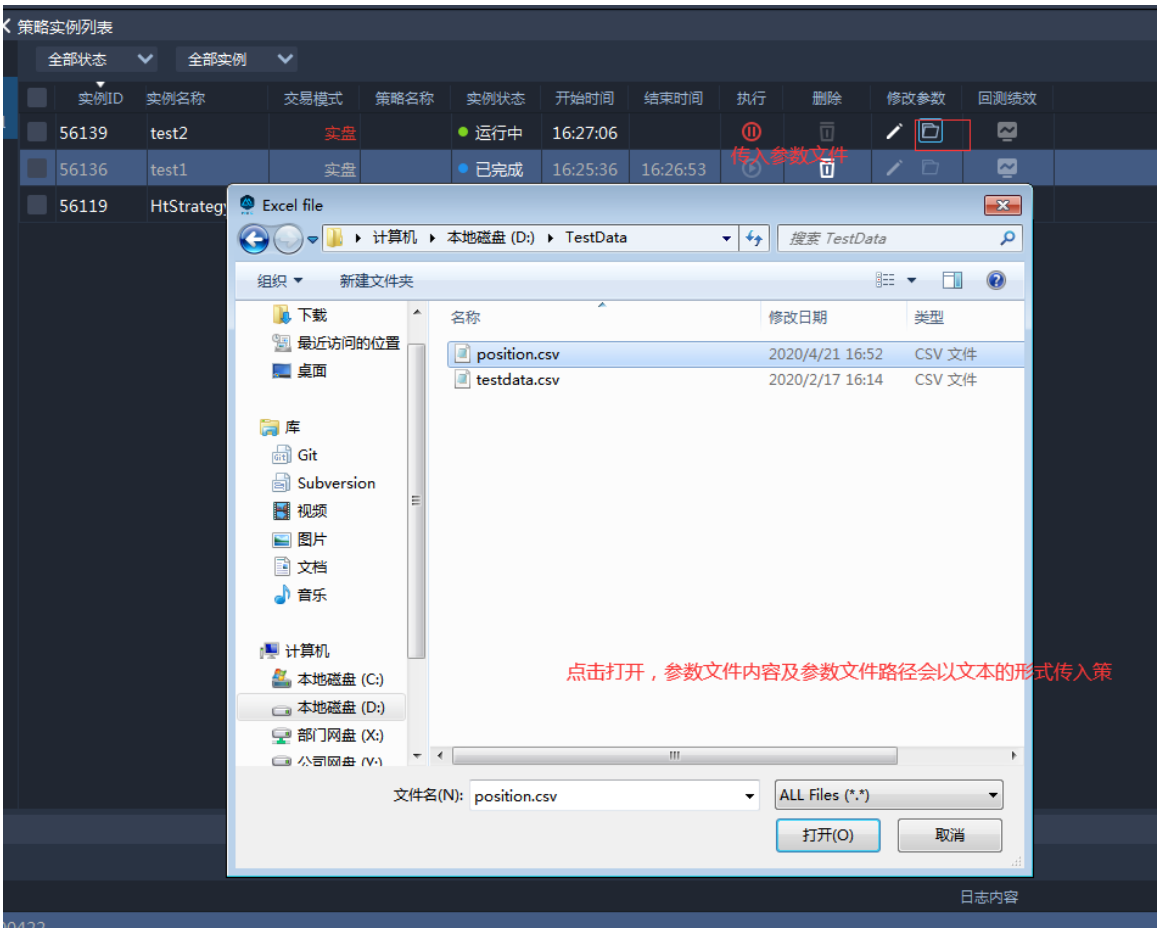
目前 MQuant 支持四种类型的参数：编辑框（type:int、float、string，可不填写）、单选下拉选择框（type:list，可不填写）、勾选框（type:bool，可不填写）、表格（type:table，必填，否则会当成普通 string 参数处理），其中，表格类型的参数比较特殊，定义方式如下：

表格参数的文件路径支持相对路径和绝对路径，要注意相对路径是相对于 matic 软件根目录的路径。

C++版本的参数定义较为简单，可以直接参考 API 编写即可。

5.4.2 参数修改

支持策略运行时修改参数，MQuant 支持两种参数修改方式，一种是修改策略运行参数，另一种是传入外部参数文件，操作界面如下图所示：



策略在 on_strategy_params_change 回调函数中接收参数修改信息，并根据策略实际情况作出响应。如果是修改策略运行参数，在调用 on_strategy_params_change 之前，context.run_params 中的内容会更新为修改后的内容；如果是传入参数文件，context.run_params 中的内容保持不变。

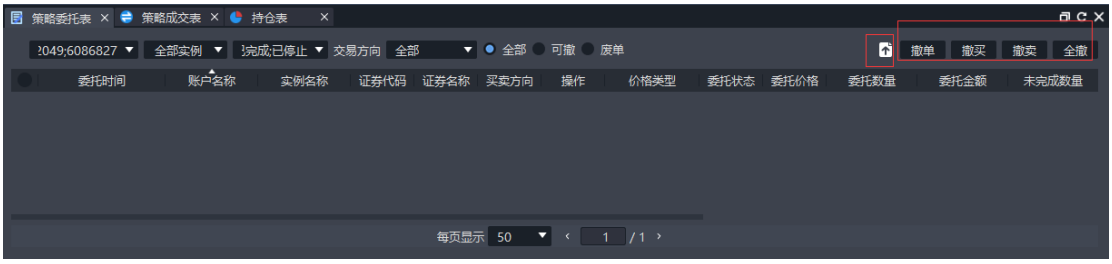
5.5 策略运行日志

在策略实例列表的右侧窗口，显示了策略运行的日志，包括运行的时间、日志的等级、日志的内容，目前系统还不支持关闭日志窗口。

5.6 策略委托表、策略成交表、持仓查询

在策略实例列表的下侧窗口，显示了策略委托表、策略成交表、持仓表。

策略委托表展示了委托时间、账户名称、实例名称、证券代码、买卖方向、价格等具体信息，首先选中某几条或全部状态是未完成的委托，可以通过委托列表右上边的“撤单”、“撤买”、“撤卖”、“全撤”对订单进行相应的交易操作，通过向上的箭头标志可以导出委托信息，如下图所示：



策略成交表展示了所有已经成交的订单的成交时间、账户名称、实例名称、证券代码、买卖方向、价格等具体信息，选中成交信息后，通过右顶方 向上的箭头标志可以导出成交信息，如下图所示：

持仓表展示了资金账号下可用的持仓，包括证券代码、证券名称、持仓数量、可用数量等信息，通过右顶方向上的箭头标志可以导出持仓信息。持仓的查看分两种模式，即汇总和明细，汇总模式是把所有资产账户上的同一支股票上的所有持仓数量相加，在列表中同一只股票只有一行信息；明细模式即把每个资产账号的持仓列出来，如下图所示：

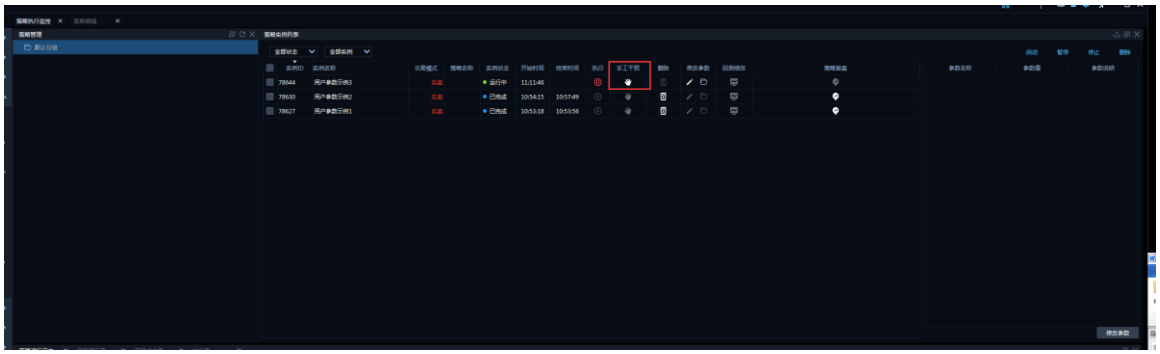
5.7 人工干预

为了方便用户对策略执行过程进行人工干预，MQuant 提供了一种很灵活的方式，允许策略在执行过程中以表格的形式弹出弹窗，用户可以对表格中的行进行勾选/去掉勾选、新增、删除等操作，然后将操作结果传给策略，策略再根据用户修改后的结果执行相应的操作。如下图所示：

```
param_template = {"left_layout":{"name":"操作表格","ui_type":"table","columns":[{"name":"","ui_type":"check_box"},
{"name":"证券代码","ui_type":"code_edit"},{"name":"持仓数量","ui_type":"line_edit","editable":"0"},
{"name":"可用数量","ui_type":"line_edit","editable":"0"},{"name":"持仓均价","ui_type":"line_edit","editable":"0"},{"name":"止损价","ui_type":"line_edit"}],
"rows":[[0,"601688.SH",5400,1800,29.15,29.15],[0,"000001.SZ",3800,3000,18.15,18.15]]}}
get_user_params(json.dumps(param_template), "测试参数")
```

如果用户需要在策略执行过程中主动进行干预，可以点击策略实例列表的实例手工干预按钮，此时，系统将会回调策略的 on_request_user_params_template 回调函数，策略可在该回调中对用户的干预行为作出响应，如下图所示：

```
def on_request_user_params_template(context, params):
    """
    category:事件回调
    brief:触发弹出用户参数设置窗口
    desc:用户在策略执行监控实例列表界面点击手工干预按钮，将会触发此回调函数，策略可以在此回调函数中获取用户参数，也可不实现
    :param context:
    :param params: 保留字段
    :return:
    :remark:可选实现
    """
    print('on request params:', params)
    param_template = {"left_layout":{"name":"操作表格","ui_type":"table","check_box":True,"columns":[{"name":"","ui_type":"check_box"},
    {"name":"证券代码","ui_type":"code_edit"},{"name":"持仓数量","ui_type":"line_edit","editable":"0"},
    {"name":"可用数量","ui_type":"line_edit","editable":"0"},{"name":"持仓均价","ui_type":"line_edit","editable":"0"},{"name":"止损价","ui_type":"line_edit"}],
    "rows":[[0,"600000.SH",5400,1800,29.15,29.15],[0,"000001.SZ",3800,3000,18.15,18.15]]}}
    strParams = get_user_params(json.dumps(param_template), "test111")
    print(strParams)
```



相比较其他量化平台的手工干预方式，MQuant 的手工干预模式使用场景更多，表达的内容也可以更丰富。而且，策略可以在任何时候弹出这样的交互界面，搭配参数修改，可以做出更复杂的交互行为（类似传参驱动策略弹出不同的交互界面）。

5.8 声音提示

在策略自动执行的过程中，如果遇到一些策略难以处理的小概率事件，如废单、资金不足、持仓不足等，用户需要策略能主动提醒用户对这些异常情况进行处理，为此，MQuant 提供了播放音频文件的能力。MQuant 提供了两个 API 接口：play_sound 和 stop_sound，分别用于播放 wav 音频文件和停止播放 wav 音频文件，系统提供了默认的声音提示，同时也支持用户传入自定义的 wav 文件。

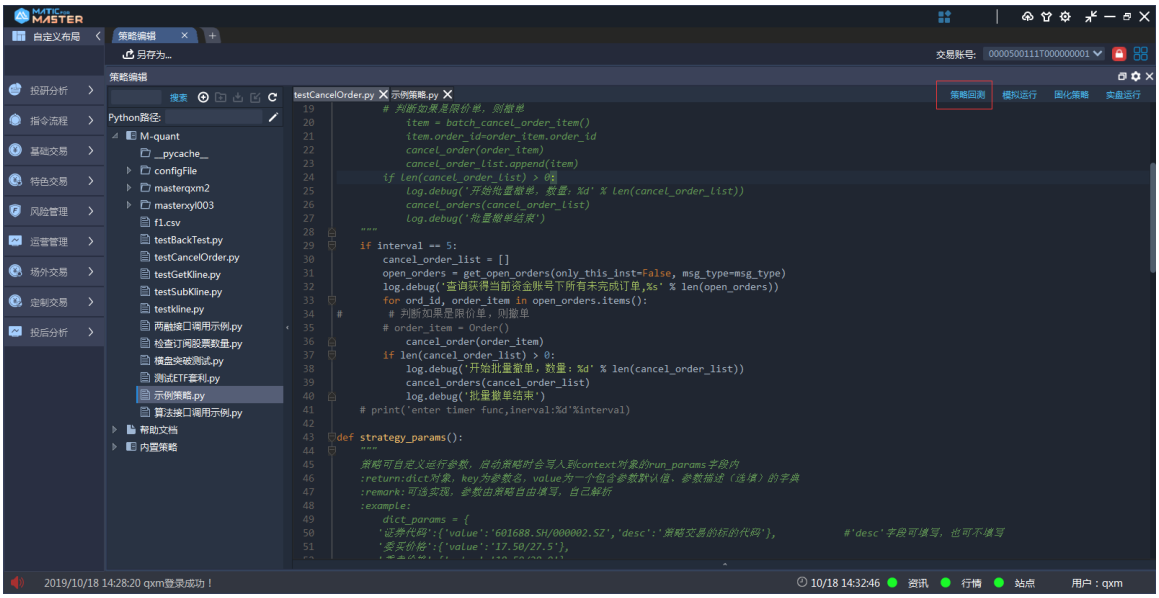
5.9 执行算法

鉴于部分客户有通过算法进行自动拆单的需求，MQuant 提供了 TWAP、VWAP 两种执行算法，支持算法的启动、停止、暂停、恢复、算法实例查询、算法订单推送等，目前使用算法需要单独授权，如有需要，可联系 Matic 技术支持。

6 策略回测及回测绩效分析

为了方便用户测试验证策略，MQuant 提供了策略回测和回测绩效分析的能力。策略回测支持 tick、分钟 k 线和日 k 线的回测。

策略回测入口在策略编辑界面，如下图所示：



点击策略回测，弹出回测参数设置框，如下图所示：

在此界面上，用户可以选择回测时间范围、行情粒度、时间粒度、初始资金、印花税和手续费。 tick 级回测数据量很大，建议选择时间范围短一点；时间粒度参数决定间隔多长时间（行情时钟），策略平台向用户策略推送一次时钟信号，建议统一使用默认值。

回测过程中的时钟采用行情时钟，即使用回放行情信号产生时钟信号推送给策略，如果用户在策略中需要获取当前时间，请使用 get_cur_time 接口替代 datetime.datetime.now()。

回测暂不支持初始底仓，如需底仓，建议在回测第一个交易日买入底仓标的，回测第二个交易日即可使用。

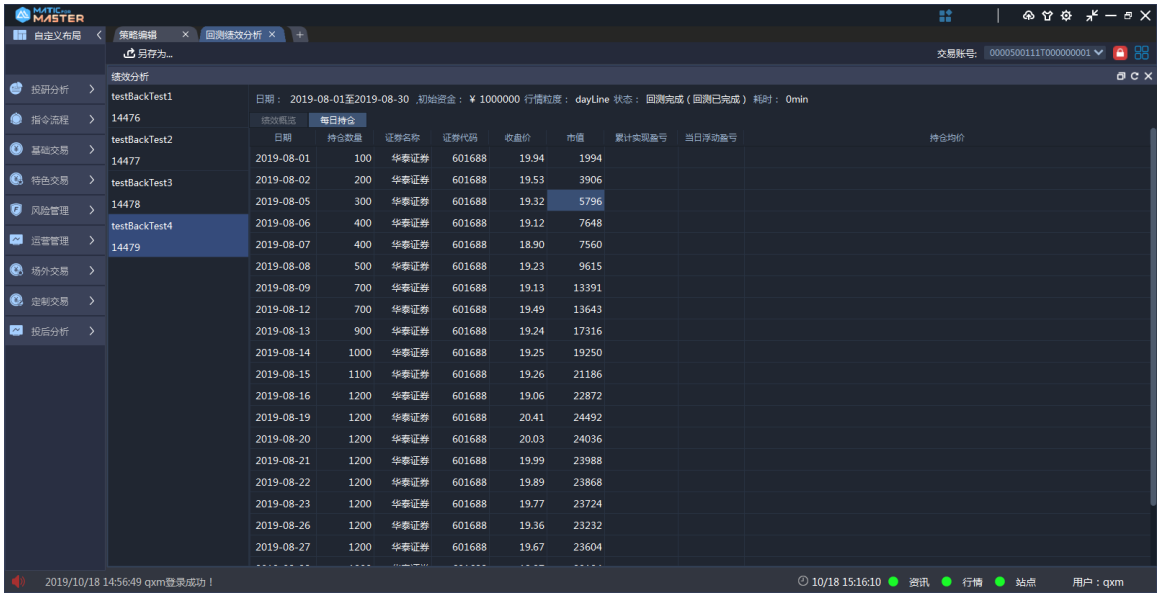
回测标的范围需要策略脚本调用 register_backtest_symbol 接口注册，可参考内置策略中的“回测示例.py”示例策略。

回测系统采用全同步方式，即每一个信号的回调函数处理完成后，才会发送下一个信号，即使在回调函数中使用了 time.sleep(secs)这样的函数，也不会影响回测的正常执行。

回测中查询 k 线，如果是查询当前日期前 N 日的 k 线数据，会查询行情回放当前日期的 k 线，非实际当前日期前 N 日的 k 线。

为了方便用户在回测过程中，每日加载初始数据，回测系统提供了每日开市（market_open）和闭市（market_close）信号，用户可以在这两个回调中重新初始化每日数据。

回测完成后，会提示是否查询策略绩效，点击确定后进入回测绩效分析界面，如下：

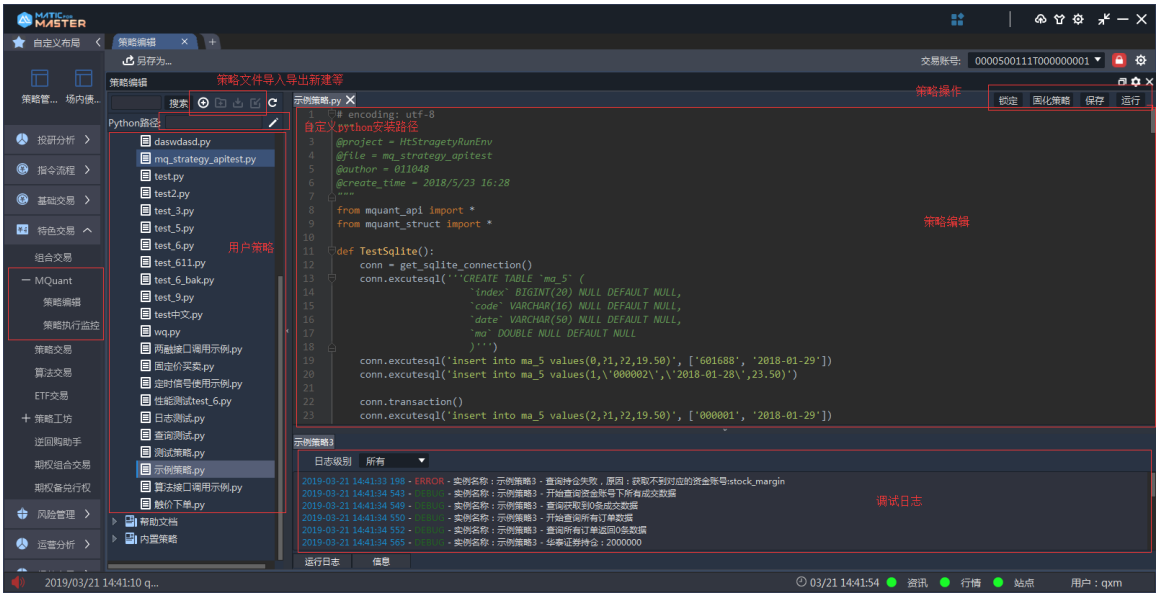


界面左侧是正常回测完成的策略实例，点击实例，在右侧会展示实例的绩效数据，包含绩效概览和每日持仓，如上图所示。

7 Python 版本

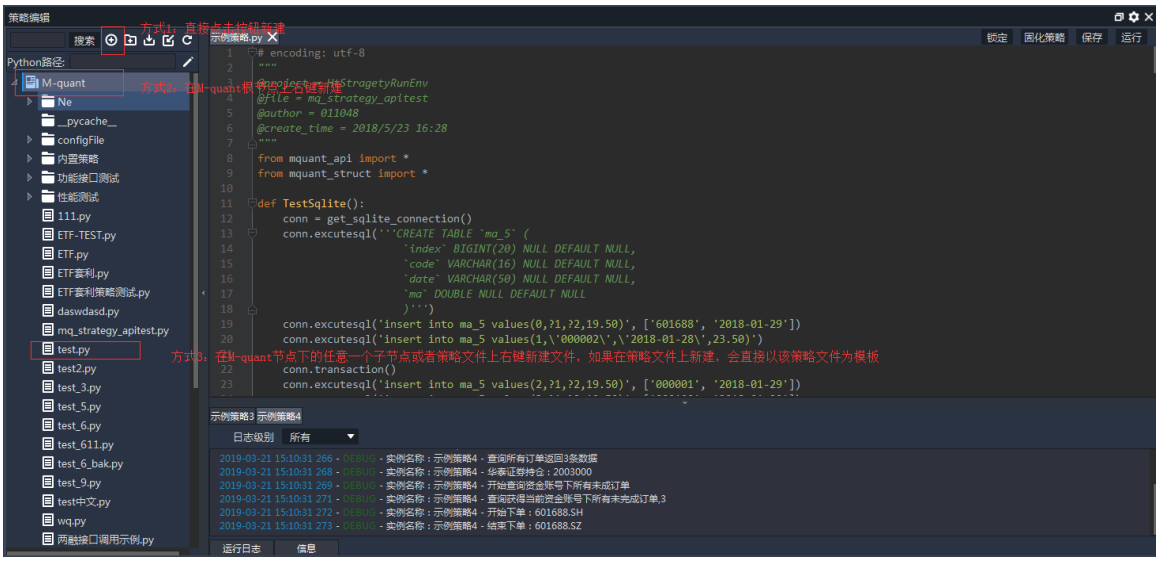
7.1 策略编辑

登录 MATIC 客户端，在左侧导航栏选择“特色交易→量化交易→策略编辑”，点击进入策略编辑页面。大体上可以分为左侧的菜单栏、中间的策略文件管理、右上的策略编辑和右下的策略调试日志输出四个区域，此外还包括策略文件管理的工具条、策略操作工具条等。如下图所示：

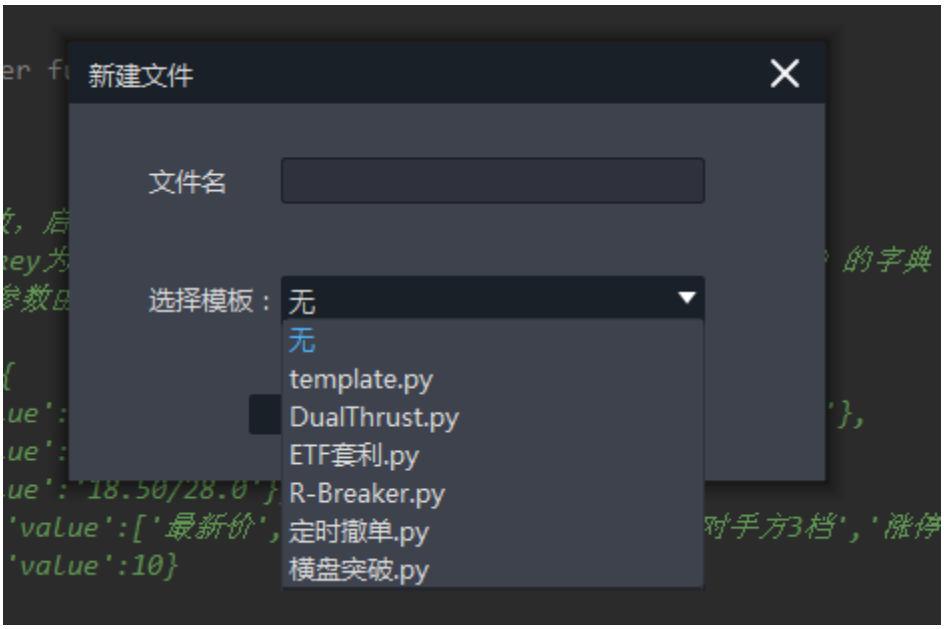


7.1.1 新建策略

MQuant 新建策略有三种方式，如下图所示：

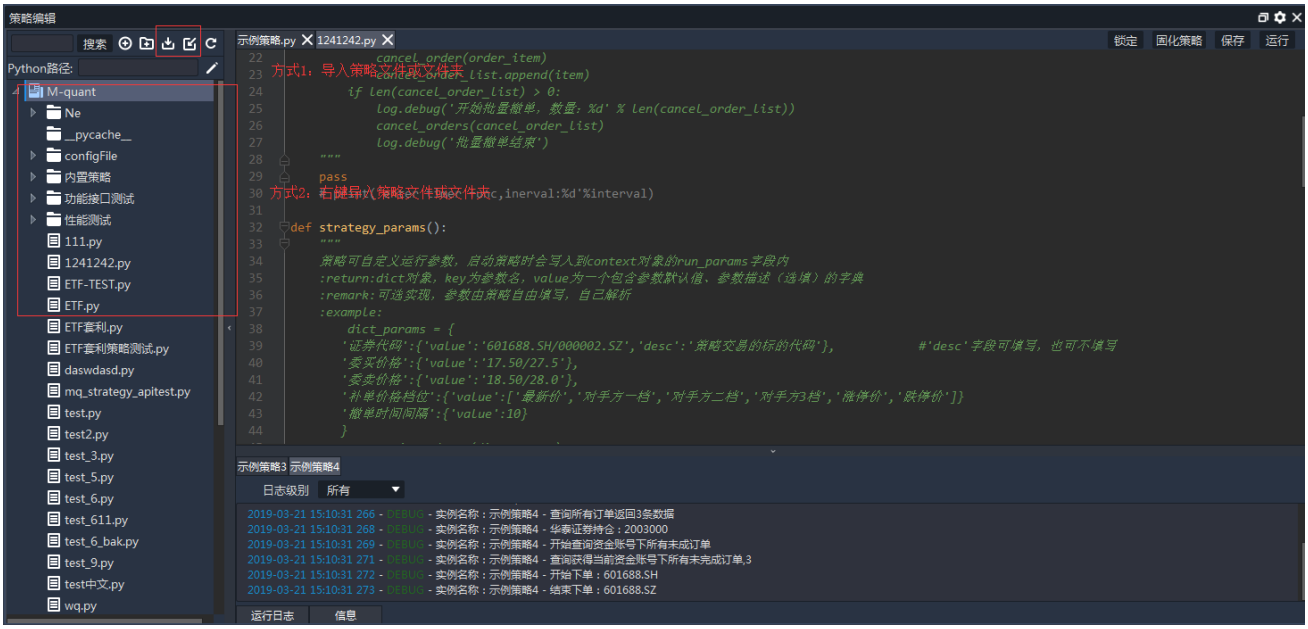


新建策略时，可以选择策略模板文件，策略模板包括基础策略模板 template.py 和 MQuant 的所有内置策略，如下图所示。选中用户策略文件右键新建策略时，会直接以被选中文件为模板，如上图方式 3 所示。



7.1.2 导入策略

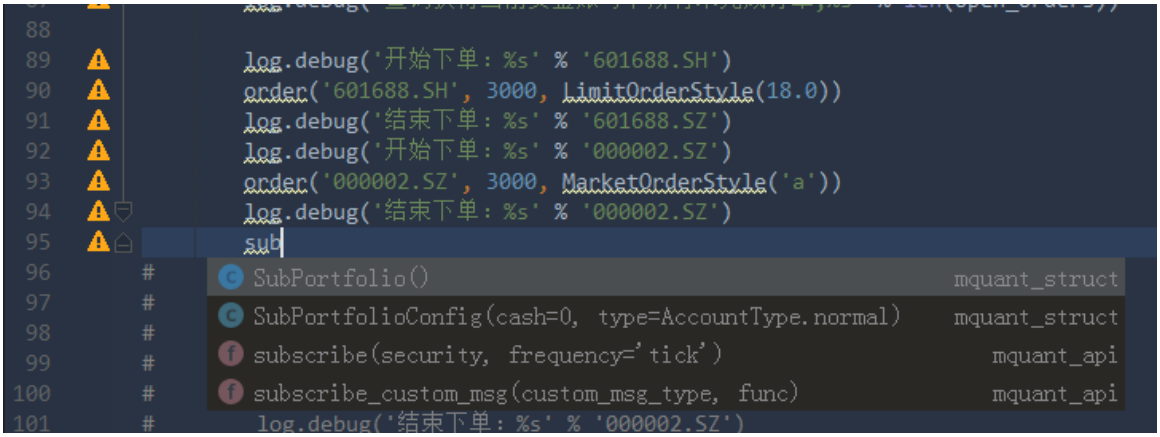
MQuant 提供两种策略导入方式，如下图所示。导入文件支持的文件类型包含 csv、txt、ini、cfg、py、pyc、pyd，导入文件夹时，如果文件夹中包含上述类型之外的文件，这些文件将不会被导入。



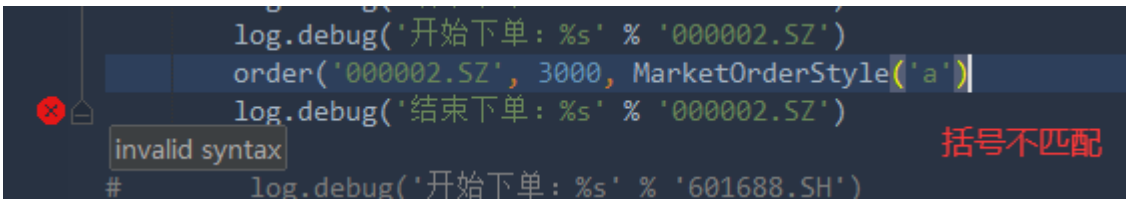
7.1.3 编写策略

MQuant 提供了一个基础的 python 编辑器，编辑器提供了基础的快捷键（如复制、粘贴、字体放大缩小、注释/反注释等，完整的快捷键列表详见附录：快捷键列表）、代码联想、自动补全、错误实时提示、策略调试功能，方便您在此编辑器中编写策略。

策略编写联想区分大小写，如下图所示。

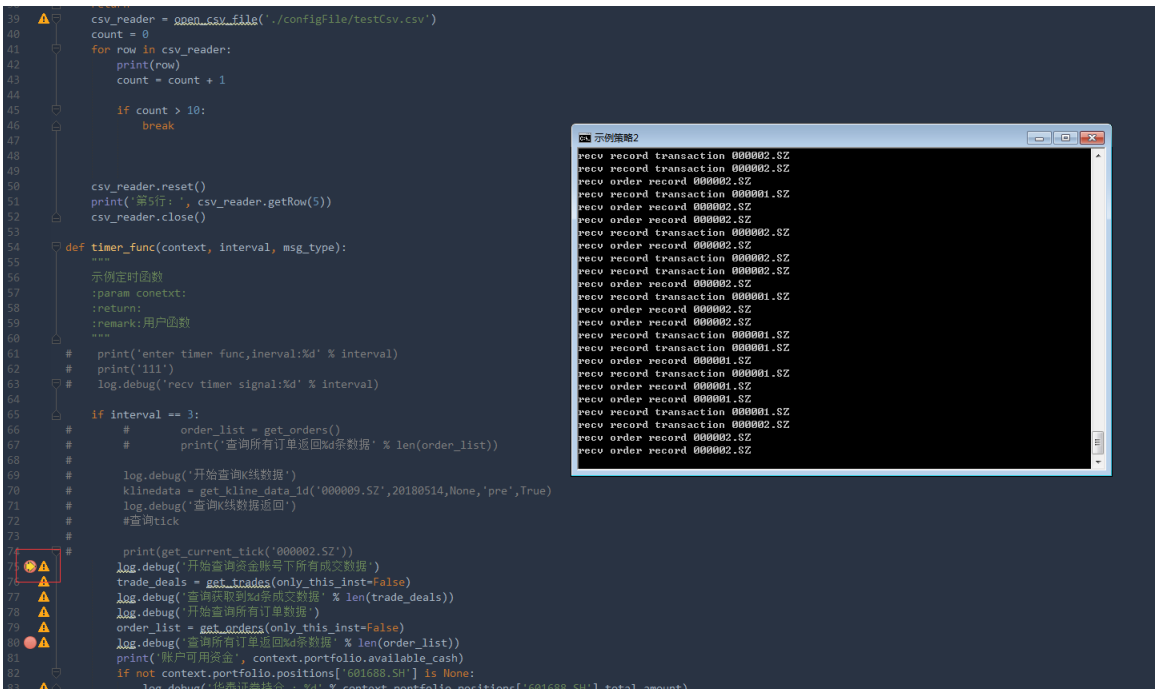


策略错误实时提示如下：



点击左侧的错误或告警图标，会弹出错误或告警信息。

支持断点调试，如下图所示：



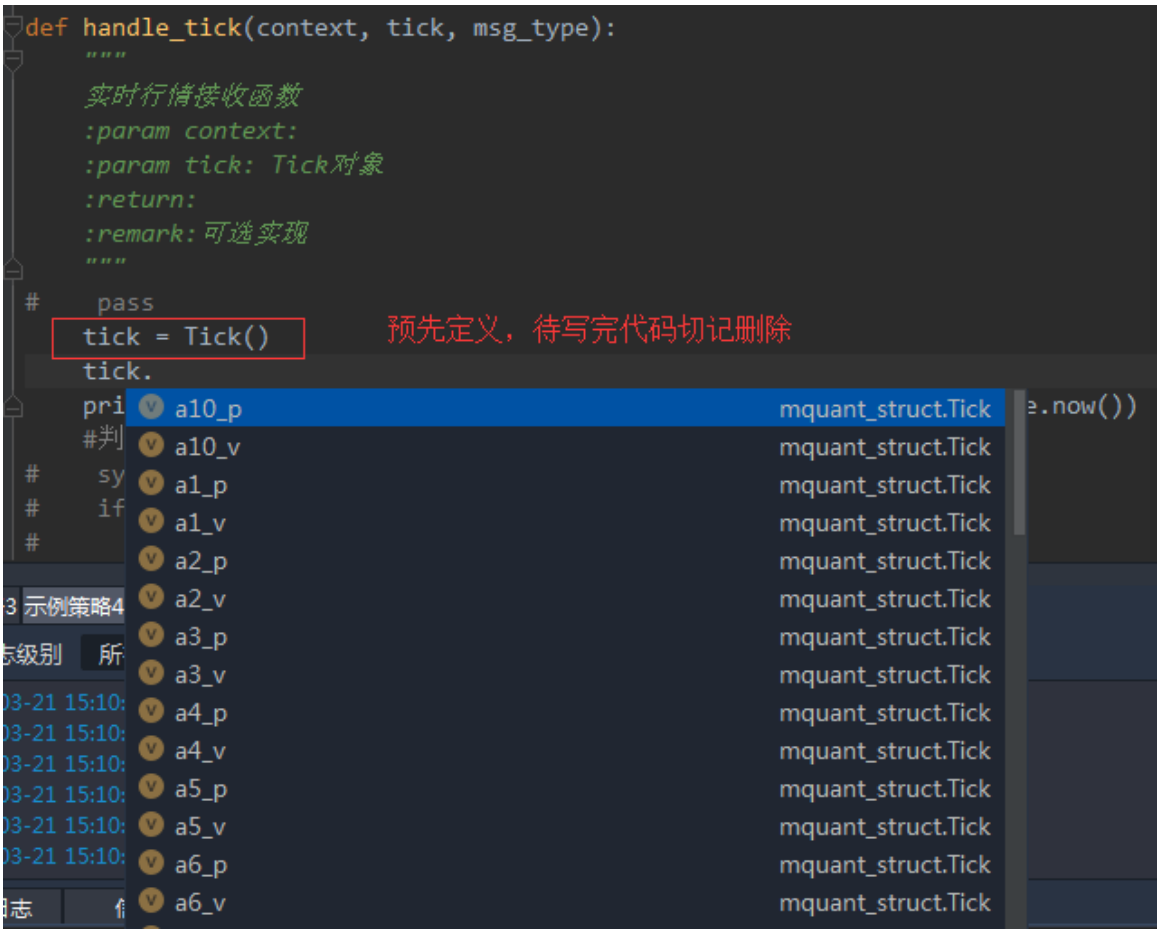
此外，为了方便客户编写策略，尽量避免通过查询手册来编写策略，MQuant 在策略编写界面上提供了 python 的 API 目录，如下图所示：

用户可以通过 API 目录了解 MQuant 具备的能力，同时在编写策略时，找到相应的 API，双击即可定位到接口文件中相应位置。

介绍一些策略编写的小技巧：

1） 通过 ctrl+f、ctrl+r 快捷键配合 enter 键查找或替换文本，将会使您的策略编写效率提高很多。

2） 由于 python 本身是弱类型的语言，如果一个变量未在本文档内当前位置前定义，编辑器是无法识别其类型的，可以在编写策略时，对通过参数传入的变量先进行类型定义，待写完策略后，再删除定义代码，如下图所示：



7.1.4 自定义策略运行参数

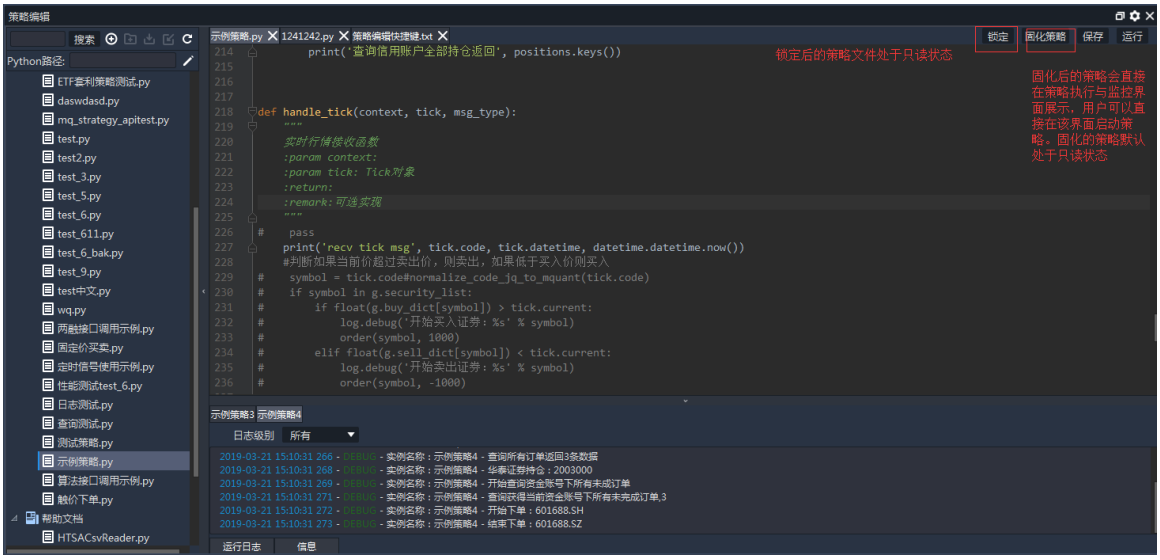
MQuant 支持用户自定义策略运行参数，参数类型（type 属性）包括 int、float、string、bool、list、table。用户只需要在策略的 strategy_params 回调函数中定义策略需要的运行参数，即可在启动策略时，从界面上查看并修改策略运行参数，如下：

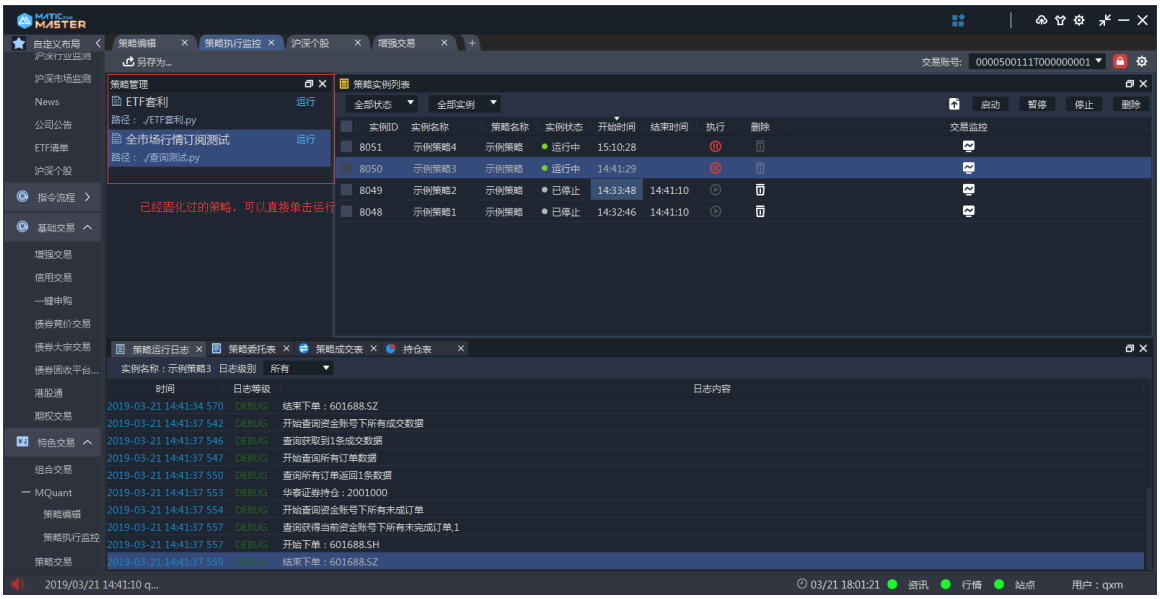
```python
def strategy_params():
    """
    category:事件回调
    brief:策略运行参数定义
    desc:
        策略运行参数定义，可选实现。策略可自定义运行参数，启动策略时，会在启动弹框中显示策略自定义的参数，客户在界面修改参数值，修改后的参数会写入
        、bool类型的参数。

    :return:dict对象，key为参数名，value为一个包含参数默认值、参数描述（选填）的字典
    :remark:可选实现，参数由策略自由填写，由策略平台解析显示在界面上，支持编辑框、下拉列表、勾选框三种形式的参数，后续可根据需求进一步丰富
    :example:
        dict_params = {
       '证券代码':{'value':'601688.SH/000002.SZ','desc':'策略交易的标的代码'},                    #'desc'字段可填写，也可不填写，编辑框类
       '委买价格':{'value':'17.50/27.5'},                                                  #编辑框类型参数
       '委卖价格':{'value':'18.50/28.0'},                                                  #编辑框类型参数
       '补单价格档位':{'value':['最新价','对手方一档','对手方二档','对手方3档','涨停价','跌停价']},  #下拉列表格式参数
       '使用持仓':{'value': True, 'desc':'买入篮子时是否使用持仓中已有的成分券额度'}              #bool类型参数
       '撤单时间间隔':{'value':10}                                                         #编辑框类型参数
       }
        return json.dumps(dict_params)
    """
    dict_params = {
        'ETF基金代码': {'value': '510050.SH', 'desc': '交易标的'},  # 'desc'字段可填写，也可不填写
        #        'ETF基金代码': {'value': '159901.SZ', 'desc': '交易标的'},  # 'desc'字段可填写，也可不填写
        '报单价格档位': {'value':['最新价', '对手方一档', '对手方二档', '对手方三档', '涨/跌停价'],'type':'list'},
        '补单价格档位': {'value': ['最新价', '对手方一档', '对手方二档', '对手方三档', '涨/跌停价']},
        '套利预估金额': {'value': '1000'},  # 超过该金额，则进行套利操作
        '交易单位份数': {'value': 1, 'desc': '如50ETF，一个交易单位是900000份基金'},
        '撤单时间间隔': {'value': 10},
        '使用持仓':{'value': True, 'desc':'买入篮子时是否使用持仓中已有的成分券额度'},
        '持仓文件':{'value': 'D:\\TestData\\position.csv', 'type':'table'}
    }
    return json.dumps(dict_params)
```

启动策略时的效果如下：

7.1.5 锁定和固化策略

为了避免您编写好的策略被不小心修改，MQuant 提供了锁定和固化策略的能力。锁定策略可以让策略文件处于只读状态；固化策略用于策略已经测试成熟时，将策略文件固化，方便您直接在策略执行监控界面启动策略实例。如下图所示：





7.2 接口说明

MQuant 接口在“工程管理→MQuant→帮助文档”下的“MQuant_api.py”中，数据结构在“工程管理→MQuant→帮助文档”下的“MQuant_struct.py”中。

MQuant 接口分为行情、交易、查询、日志、定时信号和回调函数这六类基础的接口，此外，新增加的品种或者能力会以单独的类接口的方式提供。

7.2.1 行情接口

1) 行情订阅

def subscribe(security, frequency='tick'):

接口功能：订阅标的相关推送消息，如 tick 行情，逐笔委托、逐笔成交。
参数说明：

security: 支持股票代码及股票代码列表，如’601688.SH’或者[‘601688.SH’,’000002.SZ’]股票代码格式支持聚宽和 MQuant 的股票代码格式

frequency:目前支持 MarketDataType 类中的'tick'、'kline_1m'(一分钟 k 线)、'record_order'(逐笔委托)、'record_transaction'(逐笔成交)，以及 ‘etf_estimate_info’(ETF 预估信息) ,'fund_flow'实时资金流向信息

返回值：None

2) 取消订阅

def unsubscribe(security, frequency='tick'):

接口功能：取消订阅标的相关推送消息，如 tick 行情，逐笔委托、逐笔成交。

参数说明：

security:支持股票代码及股票代码列表，股票代码格式支持聚宽和 MQuant 的股票代码格式

frequency: 目前支持 MarketDataType 类中的'tick'、'record_order'(逐笔委托)、'record_transaction'(逐笔成交)，以及 ‘etf_estimate_info’(ETF 预估信息)

返回值：None

def unsubscribe_all():

接口功能：取消订阅所有推送事件

返回值：None

3) 获取最新 tick

def get_current_tick(security):

接口功能：从缓存中获取最新的 tick 数据，您必须先订阅标的实时行情，待接收到行情推送后，才能获取到

参数说明：

security:标的代码， 支持股票、商品期货和股指期货。 不可以使用主力合约和指数合约代码

返回值：

security 对应的最新 Tick 对象，需要判断 tick 里面的代码是否=security，如果不相等，说明获取失败

4) 查询历史 tick

def get_tick(symbol, start_time, end_time):

接口功能：

查询历史 tick 数据，交易时段查询限速，5s 查询一次，非交易日和非交易时段不限速。单次查询时间范围不可跨日。

参数说明：

symbol: 标的

start_time: 开始时间，datetime.datetime 类型

end_time:结束时间，datetime.datetime 类型

返回值：

返回列表，list<Tick>

7.2.2 普通交易接口

1) 统一报单接口

def order_normal(order_request, account_type=AccountType.normal,batch_no=-1, last_batch_flag=0):

接口功能：

提供统一报单接口，支持 A 股、两融、期货、期权报单，不包含 ETF 申赎、直接还款、备兑锁定解锁等

参数说明：

last_batch_flag: 当前委托是否为 batch_no 对应的最后一笔委托, 0:不是, 1:是，目前保留不用

batch_no:可以自定义批次号，如果不需要，则不要传入此参数即可

account_type:账户类型，默认为普通 A 股账户

order_request: 报单请求，单笔报单为 OrderRequest 类型，批量报单为 list<OrderRequest>类型

返回值：

Order 对象或者 None，如果异步报单请求发送成功，则返回 Order 对象，失败则返回 None，批量报单返回 list<Order>或 None

2) 单笔报单

def order(security, amount, style=None, side='long', pindex=0, msg_type=-1):

接口功能：单笔下单接口，目前仅支持 A 股、场内基金、债券

参数说明：

security: 标的代码,支持 MQuant 代码格式和聚宽代码格式

amount:交易数量，正数表示买入，负数表示卖出

style:订单类型，支持市价单和限价单，参数类型为 OrderStyle 子类型，None 代表 MarketOrder。MQuant 在聚宽基础上扩展了市价单类型，默认为最优五档即时成交剩余撤销(沪市深市均可)，您也可选择其他类型

side:买卖方向，在 OrderSide 类中定义，也可以直接输入 long’/’short’，分别表示买/卖。默认为买入，股票、基金暂不支持开空单。**该字段暂不可用**

pindex:在使用 set_subportfolios 创建了多个仓位时，指定 subportfolio 的序号，从 0 开始，比如 0 指定第一个 subportfolio, 1 指定第二个 subportfolio，默认为 0，目前 MQuant 只支持默认值 0

msg_type 透传字段，传入可获得协程并发能力，暂不可用

返回值：

Order 对象或者 None，如果创建订单成功，则返回 Order 对象，失败则返回 None

3) 批量报单

def orders(orders, batch_no=-1, last_batch_flag=0, msg_type=-1):

接口功能：批量下单接口，支持自定义批次号，支持一个批次里面支持不同买卖方向，不同订单类型的订单，目前仅支持 A 股、场内基金、债券

参数说明：

orders:list<batch_order_item>

batch_no:可以自定义批次号，但需要保证资金账号下全局唯一，如果不需要，则不传入此参数即可

last_batch_flag:当前委托是否为 batch_no 对应的最后一笔委托, 0:不是, 1:是，目前保留不用

msg_type 透传字段，传入可获得协程并发能力

返回值：

list<Order>，列表数量一定与订单数量一致且顺序对应，但是列表元素可能为 None

4) 单笔撤单

def cancel_order(order, batch_flag=0, account_type = AccountType.normal, msg_type=-1):

接口功能：单笔撤单接口

参数说明：

order:Order 对象或者 order_id 或者 batch_no，按照 order_id 还是 batch_no 撤单取决于 batch_flag 字段

batch_flag: 0 表示按照 order_id 撤单，1 表示按照 batch_no 撤单，默认按照 order_id 撤单

account_type:账号类型，默认为 A 股账号，AccountType 类型

msg_type 透传字段，传入可获得协程并发能力，暂不可用

返回值：

成功返回 Order 对象，失败返回 None

5) 批量撤单

def cancel_orders(orders , account_type = AccountType.normal, msg_type=-1):

接口功能：批量撤单接口

参数说明：

orders: list<batch_cancel_order_item>

account_type:账号类型，默认为 A 股账号，AccountType 类型

msg_type:透传字段，传入可获得协程并发能力

返回值：

List<Order>，列表数量一定与订单数量一致且顺序对应，但是列表元素可能为 None

7.2.3 期货交易接口

def order_future(security, amount, style=None, action=OrderAction.UNKNOWN,
invest_type=HedgeFlag.SPECULATION, close_direction=CloseDirection.DEFAULT):

接口功能：期货开平仓

参数说明：

security: 标的代码,支持 MQuant 代码格式和聚宽代码格式

amount:交易数量，正数表示买入，负数表示卖出

style:订单类型，支持市价单和限价单，参数类型为 OrderStyle 子类型，None 代表 MarketOrder。MQuant 在聚宽基础上扩展了市价单类型，默认为最优五档即时成交剩余撤销(沪市深市均可)，您也可选择其他类型

action:开平方向，OrderAction 类型

invest_type 投资类型,HedgeFlag 类型，分为投机、套保、套利，默认为投机，期货有效

close_direction 平仓方式 CloseDirection 类型，默认为优先平老仓

返回值：

Order 对象或者 None，如果创建订单成功，则返回 Order 对象，失败则返回 None

def orders_future(orders, batch_no=-1, last_batch_flag=0):

接口功能：期货批量下单接口，暂不支持自定义批次号，也不支持按批次号撤单

参数说明：

orders:list<batch_order_item>

batch_no:可以自定义批次号，但需要保证资金账号下全局唯一，如果不需要，则不传入此参数即可，暂不支持

last_batch_flag:当前委托是否为 batch_no 对应的最后一笔委托, 0:不是, 1:是，目前保留不用

返回值：

list<Order>，列表数量一定与订单数量一致且顺序对应，但是列表元素可能为 None

期货撤单、委托、成交、资金、持仓查询接口与 A 股共用

def get_future_contract_info(symbol, hedge_flag):

接口功能：获取期货合约信息

参数说明：

symbol: 期货合约代码

hedge_flag 投保标记,在 HedgeFlag 中定义

返回值：成功返回 FutureContractInfo 对象，失败返回 None

7.2.4 期权交易接口（期权程序化必须进行报备，否则将会提示无权限）

1）期权报单（包括开、平、备兑、行权）

def order_option(security, amount, style=None, action=OrderAction.UNKNOWN, covered_flag=OptionCoveredFlag.UNKNOWN):

接口功能：期权买开、卖开、买平、卖平、备兑开仓、备兑平仓、行权

参数说明：

security: 标的代码,支持 MQuant 代码格式和聚宽代码格式

amount:交易数量，正数表示买入，负数表示卖出

style:订单类型，支持市价单和限价单，参数类型为 OrderStyle 子类型，None 代表 MarketOrder。MQuant 在聚宽基础上扩展了市价单类型，默认为最优五档即时成交剩余撤销(沪市深市均可)，您也可选择其他类型

action:开平方向，OrderAction 类型

covered_flag 备兑标志，OptionCoveredFlag 类型

返回值：

Order 对象或者 None，如果创建订单成功，则返回 Order 对象，失败则返回 None

期权撤单、委托、成交、资金、持仓查询接口与 A 股共用

2）备兑证券锁定/解锁

def option_trans(security, amount):

接口功能：上交所期权备兑开平仓时，标的证券的锁定、解锁，深市期权或在进行备兑开平仓交易时自动锁定/解锁标的券，无需用户调用接口锁定/解锁

参数说明：

security:标的券代码

: amount:锁定/解锁数量，amount>0 表示锁定，amount<0 表示解锁

返回值：

Order 对象或者 None，如果锁定/解锁接口调用成功，则返回 Order 对象，失败则返回 None

7.2.5 统一报单接口

def order_normal(order_request, account_type=AccountType.normal, batch_no=-1, last_batch_flag=0):

接口功能：统一报单接口，支持 A 股、两融、期货、期权报单，不包含 ETF 申赎、直接还款、备兑锁定解锁等

参数说明：

order_request: 报单请求，单笔报单为 OrderRequest 类型，批量报单为 list<OrderRequest>类型

account_type:账户类型，AccountType 类型，默认为普通 A 股账户

batch_no:可以自定义批次号，如果不需要，则不要传入此参数即可

last_batch_flag: 当前委托是否为 batch_no 对应的最后一笔委托, 0:不是, 1:是，目前保留不用

7.2.6 债转股接口

def bond_convert_to_stock(convert_symbol, qty, account_type = AccountType.normal):

接口功能：可转债转股

参数说明：

convert_symbol: 转股代码

qty: 数量，单位为张

返回值：

Order 对象或者 None，如果转股接口调用成功，则返回 Order 对象，失败则返回 None

7.2.7 查询接口

7.2.7.1 订单查询

（1）获取所有未完成订单

def get_open_orders(page_no=1, page_size=1000, only_this_inst=True, account_type = AccountType.normal, msg_type=-1,inst_id='')

接口功能：获取未完成订单

参数说明：

page_no:分页查询，页码

page_size:每页数量，默认为 1000，建议在 1000 以内,-1 表示查询符合条件的全部订单

only_this_inst:是否只查询当前实例的订单,默认为 True

account_type:账号类型，默认为 A 股账号，AccountType 类型

msg_type 透传字段，传入可获得协程并发能力，暂不可用

inst_id:如果 only_this_inst=False，可以查询指定 inst_id 的订单信息，only_this_inst=True 时无效

返回值：dict<order_id, Order>

（2）获取所有未完成订单（扩展接口）

def get_open_orders_ex(page_no=1, page_size=1000, only_this_inst=True, account_type = AccountType.normal, msg_type=-1,inst_id='')

接口功能：获取未完成订单

参数说明：

page_no:分页查询，页码

page_size:每页数量，默认为 1000，建议在 1000 以内,-1 表示查询符合条件的全部订单

only_this_inst:是否只查询当前实例的订单,默认为 True

account_type:账号类型，默认为 A 股账号，AccountType 类型

msg_type 透传字段，传入可获得协程并发能力，暂不可用

inst_id:如果 only_this_inst=False，可以查询指定 inst_id 的订单信息，only_this_inst=True 时无效

返回值：tuple, [total_count, is_last, dict<order_id,Order>]

total_count 是当前账号下所有未成订单数量，不代表满足所有传入过滤条件的订单总数；

is_last 表示本次查询返回的是否为最后的一页，即使本次查询返回 0 条数据，也不代表分页查询已经查询完所有满足条件的数据，需要根据 is_last 来判断是否确实已经查询完所有数据。

dict<order_id,Order> 返回的数据列表。

（3）查询所有订单

def get_orders(order_id='', security='', status=None, page_no=1, page_size=1000, only_this_inst=True,

msg_type=-1 , account_type = AccountType.normal, inst_id=’’):

接口功能：查询订单

参数说明：

order_id: 订单 id

security:标的代码，可以用来查询指定标的的所有订单

status: OrderStatus， 查询特定订单状态的所有订单

page_no:分页查询，页码

page_size:每页数量，默认为 1000， 建议在 1000 以内,-1 表示查询符合条件的全部订单

only_this_inst:是否只查询当前实例的订单，默认为 True

msg_type 透传字段，传入可获得协程并发能力

account_type:账号类型，默认为 A 股账号，AccountType 类型

inst_id:如果 only_this_inst=False，可以查询指定 inst_id 的订单信息，only_this_inst=True 时无效

返回值：dict<order_id, Order>

（4）查询所有订单（扩展接口）

def get_orders_ex(order_id='', security='', status=None, page_no=1, page_size=1000, only_this_inst=True, msg_type=-1 , account_type = AccountType.normal, inst_id=’’):

接口功能：查询订单

参数说明：

order_id: 订单 id

security:标的代码，可以用来查询指定标的的所有订单

status: OrderStatus， 查询特定订单状态的所有订单

page_no:分页查询，页码

page_size:每页数量，默认为 1000， 建议在 1000 以内,-1 表示查询符合条件的全部订单

only_this_inst:是否只查询当前实例的订单，默认为 True

msg_type 透传字段，传入可获得协程并发能力

account_type:账号类型，默认为 A 股账号，AccountType 类型

inst_id:如果 only_this_inst=False，可以查询指定 inst_id 的订单信息，only_this_inst=True 时无效

返回值：tuple, [total_count, is_last, dict<order_id,Order>]

total_count 是当前账号下所有满足状态条件的订单数量，不代表满足所有传入过滤条件的订单总数；

is_last 表示本次查询返回的是否为最后的一页，即使本次查询返回 0 条数据，也不代表分页查询已经查询完所有满足条件的数据，需要根据 is_last 来判断是否确实已经查询完所有数据。

dict<order_id,Order> 返回的数据列表。

（5）查询成交

def get_trades(order_id='', security='', page_no=1, page_size=1000, account_type = AccountType.normal,include_rejected_orders=False, include_withdraw_orders=True, only_this_inst=True, msg_type=-1, inst_id=''):

接口功能：获取当天的所有成交记录，一个订单可能分多次成交

参数说明：

order_id:订单 id，默认为空，填写订单 id 查询指定成交

security:标的代码，默认为空，填写标的代码查询标的的所有成交

page_no:分页查询，页码

page_size:每页数量，默认为 1000，建议在 1000 以内,-1 表示查询符合条件的全部订单

account_type:账号类型，默认为 A 股账号，AccountType 类型

include_rejected_orders: 是否包含废单成交，默认不包含

include_withdraw_order: 是否包含撤单成交，默认包含

only_this_inst:是否只查询当前实例的成交，默认为 True

msg_type 透传字段，传入可获得协程并发能力

inst_id:如果 only_this_inst=False，可以查询指定 inst_id 的订单信息，only_this_inst=True 时无效

返回值：dict<trade_id, Trade>

（6）查询成交（扩展接口）

def get_trades_ex(order_id='', security='', page_no=1, page_size=1000, account_type = AccountType.normal,include_rejected_orders=False, include_withdraw_orders=True, only_this_inst=True, msg_type=-1, inst_id=''):

接口功能：获取当天的所有成交记录，一个订单可能分多次成交

参数说明：

order_id:订单 id，默认为空，填写订单 id 查询指定成交

security:标的代码，默认为空，填写标的代码查询标的的所有成交

page_no:分页查询，页码

page_size:每页数量，默认为 1000，建议在 1000 以内,-1 表示查询符合条件的全部订单

account_type:账号类型，默认为 A 股账号，AccountType 类型

include_rejected_orders: 是否包含废单成交，默认不包含

include_withdraw_order: 是否包含撤单成交，默认包含

only_this_inst:是否只查询当前实例的成交，默认为 True

msg_type 透传字段，传入可获得协程并发能力

inst_id:如果 only_this_inst=False，可以查询指定 inst_id 的订单信息，only_this_inst=True 时无效

返回值：tuple, [total_count, is_last, list<Trade>]

total_count 是当前账号下所有成交数量，不代表满足所有传入过滤条件的成交总数；

is_last 表示本次查询返回的是否为最后的一页，即使本次查询返回 0 条数据，也不代表分页查询已经查询完所有满足条件的数据，需要根据 is_last 来判断是否确实已经查询完所有数据。

list<Trade> 返回的数据列表。

7.2.7.2 持仓查询

def get_positions(account_type=AccountType.normal, symbol=''):

接口功能：查询持仓，不支持期货、期权

参数说明：

account_type:账户类型，默认为 A 股账户，在 AccountType 中定义

:param symbol:标的，如果标的为空，返回 dict<symbol, Position>对象，否则，返回 Position 或者 None

返回值：

失败返回 None,成功返回 dict<symbol, Position>

def get_positions_ex(account_type=AccountType.normal, symbol=''):

接口功能：查询持仓，支持所有品种

参数说明：

account_type:账户类型，默认为 A 股账户，在 AccountType 中定义

:param symbol:标的，如果标的为空，返回 dict<symbol, Position>对象，否则，返回 Position 或者 None

返回值：

失败返回 None,成功返回 list< Position>

单只证券的持仓还可以通过 context.portfolio.positions[symbol]的方式获取，这种方式使用了运行时获取的技术，只能输入标的代码获取单标的持仓，不能直接获取到所有持仓。对于期货和期权，同一只标的可能有多个持仓，此时，通过这种方式获取到的持仓是 list<Position>对象

7.2.7.3 资金查询

（1）方式一

资金通过 context.portfolio 结构下的相关成员变量获取，参考 context 定义。

示例：

获取账户可用资金：context.portfolio.available_cash

（2）方式二

def get_fund_info(account_type=AccountType.normal):

接口功能：查询资金信息

参数说明：

account_type:账号类型，默认为普通 A 股账号

返回值：

资金信息，成功返回 FundUpdateInfo 对象，失败返回 None

7.2.7.4 K 线数据查询

MQuant 提供了目前提供查询日 k 和分钟 k 的接口，获取方式有两种，一种是获取从开始时间到结束时间范围内的所有 k 线数据；另一种是获取从某日开始前/后指定天数的 k 线数据。

def get_kline_data(symbol, start_date, end_date=None, kline_type = KLineDataType.KLINEData_1M, fq='pre', sync=True, msg_type=-1, include_end_date = False)

接口功能：查询 k 线数据，如果 k 线数据已经准备好，则返回 KLineData 对象列表，否则返回 None，目前支持日 k 和分钟 k

参数说明：

symbol:证券代码，支持聚宽和 MQuant 两种格式

start_date:开始日期,支持 datetime 对象格式和 int 格式，int 示例格式：20180702

end_date:结束日期，默认为当前时间，支持 datetime 对象格式和 int 格式，int 示例格式：20180702

kline_type:k 线数据类型，支持 KLineDataType 对象格式和 str，str 示例格式：kline_1m

fq:复权方式，默认为前复权，可选 pre(前复权),None(不复权),post(后复权)，目前只支持前复权

sync:是否同步查询，默认为同步查询，异步查询不保证本次会返回数据

msg_type:透传字段，传入可获得协程并发能力

include_end_date:时间范围是否包含结束时间，废弃，实际效果是始终包含

返回值：

成功返回 KLineData 对象,失败返回 None

def get_kline_data_from_init_date(symbol, days, init_date=None,kline_type = KLineDataType.KLINEData_1M, fq=None, sync=True, msg_type=-1 , date_type=DateType.NORMAL_DATE, security_exchange_type=SecurityExchangeType.UNKNOWN,include_init_date = False)

接口功能：获取初始日期开始的 days 天的 k 数据，目前支持日 k 和分钟 k

参数说明：

symbol:证券代码，支持聚宽和 MQuant 两种格式

days:int 类型，大于 0 表示获取 init_date 之后 days 天的日 k 数据(包括 init_date),小于 0 表示获取 init_date 之前 days 天的日 k 数据(不包括 init_date)

init_date:初始日期，支持 datetime 对象格式和 int，int 示例格式：20180702

kline_type:k 线数据类型，支持 KLineDataType 对象格式和 str，str 示例格式：kline_1m

fq:复权方式，默认为前复权，可选 pre(前复权),None(不复权),post(后复权)，目前只支持前复权

sync:是否同步查询，默认为同步查询，异步查询不保证本次会返回数据

msg_type 透传字段，传入可获得协程并发能力

date_type:日期类型，DateType 类型，包括自然日和交易日，默认为自然日

security_exchange_type: 代码交易市场，主要是区分港股代码，需要用户传入区分是港股市场还是沪深港

通,SecurityExchangeType 类型，如果传入 SecurityExchangeType.UNKNOWN，取默认市场

include_init_date:　时间范围是否包含初始时间，废弃，实际效果是始终包含

返回值：

成功返回 KLineData 对象,失败返回 None

以下两个接口是上面两个接口的日 k 线特例。

def get_kline_data_1d(symbol, start_date, end_date=None, fq='pre', sync=True, msg_type=-1, include_end_date = False):

接口功能：查询指定时间范围内的日 k 线数据，如果 k 线数据已经准备好，则返回 KLineData 对象，否则返回 None

参数说明：

symbol:证券代码，支持聚宽和 MQuant 两种格式

start_date:开始日期,支持 datetime 对象格式和 int 格式，int 示例格式：20180702

end_date:结束日期，默认为当前时间（取到的数据不包含当天），支持 datetime 对象格式和 int 格式，int 示例格式：20180702

fq:复权方式，默认为前复权，可选 pre(前复权),None(不复权),post(后复权)，目前只支持前复权

sync:是否同步查询，默认为同步查询，异步查询不保证本次会返回数据

msg_type 透传字段，传入可获得协程并发能力

include_end_date:时间范围是否包含结束时间，废弃，实际效果是始终包含

返回值：

成功返回 KLineData 对象,失败返回 None

def get_kline_data_1d_from_init_date(symbol, days, init_date=None, fq=None, sync=True, msg_type=-1, date_type=DateType.NORMAL_DATE,
security_exchange_type=SecurityExchangeType.UNKNOWN,include_init_date = False):

接口功能：获取初始日期开始的 days 天的日 k 数据

参数说明：

symbol:证券代码，支持聚宽和 MQuant 两种格式

days:int 类型，大于 0 表示获取 init_date 之后 days 天的日 k 数据(包括 init_date),小于 0 表示获取 init_date 之前 days 天的日 k 数据(不包括 init_date)，此处的 days 是指交易日

init_date:初始日期，支持 datetime 对象格式和 int，int 示例格式：20180702

fq:复权方式，默认为前复权，可选 pre(前复权),None(不复权),post(后复权)，目前只支持前复权

sync:是否同步查询，默认为同步查询，异步查询不保证本次会返回数据

msg_type 透传字段，传入可获得协程并发能力

date_type:日期类型，DateType 类型，包括自然日和交易日，默认为自然日

security_exchange_type: 代码交易市场，主要是区分港股代码，需要用户传入区分是港股市场还是沪深港通, SecurityExchangeType 类型，如果传入 SecurityExchangeType.UNKNOWN，取默认市场

include_init_date: 时间范围是否包含初始时间，废弃，实际效果是始终包含

返回值：

成功返回 KLineData 对象,失败返回 None

7.2.7.5 证券代码查询

def get_symbol_list(exchange_type, security_type, security_sub_type=SecuritySubType.Ashares):

接口功能：获取证券代码列表

参数说明：

exchange_type:市场，类型为 ExchangeType 对象，仅支持一次订阅一个市场的行情

security_type: 标的类型，类型为 SecurityType 对象,填空字符串表示不受该条件约束

security_sub_type：标的子类型,类型为 SecuritySubType 对象,默认类型为 A 股，填空字符串表示不受该条件约束

返回值：

成功返回证券代码列表[601688.SH,600000.SH,...],失败返回 None

典型使用场景：

您可以调用此接口获取证券代码列表，然后调用 subscribe 函数订阅某个市场某类证券的全部行情

def get_symbol_detial(symbol):

接口功能：获取单只证券代码详情

参数说明：

symbol: 证券代码

返回值：成功返回 SecurityDetial 对象，失败返回 None

7.2.7.6 指数成分券查询

def get_index_components (index_symbol):

接口功能：获取证券代码列表

参数说明：

index_symbol:指数代码

返回值：

list<IndexComponent>，即指数成分券列表。

典型使用场景：

您可以调用此接口获取证券代码列表，然后调用 subscribe 函数订阅某个市场某类证券的全部行情

def get_symbol_detial(symbol):

接口功能：获取单只证券代码详情

参数说明：

symbol: 证券代码

返回值：成功返回 SecurityDetial 对象，失败返回 None

7.2.8 回调函数

（1） 初始化函数

函数定义：def initialize(context)

调用时间：策略初始化阶段调用

是否必须：是

特别说明：策略在初始化函数中，一般进行一些参数存储、信号订阅等初始化操作，允许读取文件，不允许报单、撤单等交易行为和访问网络

示例：

```
def initialize(context):
    """
    策略初始化，启动策略时调用，用户可在初始化函数中订阅行情、设置标的、设置定时处理函数等
    该函数中允许读取文件，除此之外的其他函数禁止读取文件
    :param context:
    :return:
    :remark:必须实现
    """
    run_timely(timer_func, 3)  # 注册一个3s的定时函数，用于下单
    subscribe(context.run_params['证券代码'].strip('/').split('/'))
    subscribe(context.run_params['证券代码'].strip('/').split('/'),'record_order')
    subscribe(context.run_params['证券代码'].strip('/').split('/'),'record_transaction')
    g.security_list=context.run_params['证券代码'].strip('/').split('/')
    lst_buy_price = context.run_params['买入价格'].strip('/').split('/')
    lst_sell_price = context.run_params['卖出价格'].strip('/').split('/')
    g.buy_dict = dict(zip(g.security_list,lst_buy_price))
    g.sell_dict = dict(zip(g.security_list,lst_sell_price))
```

（2） 策略开始启动回调

函数定义：def on_strategy_start(context):

调用时间：策略初始化（initialize）执行完毕后立即调用

是否必须：否

特别说明：该回调中允许交易，不允许与外部程序进行任何交互

（3） 自定义参数模板函数

函数定义：def strategy_params()

调用时间：策略启动前调用

是否必须：否

特别说明：自定义参数模板包含两种模板形式，一种是 dict，另一种是 json。目前支持普通编辑框、单选下拉列表和 bool 型勾选框

示例：

dict 模式

```
def strategy_params():
    """
    策略可自定义运行参数，启动策略时会写入到context对象的run_params字段内
    :return:dict对象，key为参数名，value为一个包含参数默认值、参数描述（选填）的字典
    :remark:可选实现
    """
    # 示例如下：
    dict_params = {
        '证券代码': {'value': '000001.SZ/000002.SZ', 'desc': '交易标的'},  # 'desc'字段可填写，也可不填写
        '买入价格': {'value': '17.50/27.50'},
        '卖出价格': {'value': '18.50/28.00'},
        '撤单时间间隔':{'value': 10}
    }
    return dict_params
```

json 模式：

```
def strategy_params():
    """
    策略可自定义运行参数，启动策略时会写入到context对象的run_params字段内
    :return:dict对象，key为参数名，value为一个包含参数默认值、参数描述（选填）的字典
    :remark:可选实现
    """
    # 示例如下：
    dict_params = {
        'ETF基金代码': {'value': '510050.SH', 'desc': '交易标的'},  # 'desc'字段可填写，也可不填写
        #       'ETF基金代码': {'value': '159901.SZ', 'desc': '交易标的'},  # 'desc'字段可填写，也可不填写
        '报单价格档位': {'value':['最新价', '对手方一档', '对手方二档', '对手方三档', '涨/跌停价']},
        '补单价格档位': {'value': ['最新价', '对手方一档', '对手方二档', '对手方三档', '涨/跌停价']},
        '套利预估金额': {'value': '1000'},  # 超过该金额，则进行套利操作
        '交易单位份数': {'value': 1, 'desc': '如50ETF，一个交易单位是900000份基金'},
        '撤单时间间隔': {'value': 10},
        '使用持仓': {'value': True, 'desc':'买入篮子时是否使用持仓中已有的成分券额度'}
    }
    return json.dumps(dict_params)
```

下拉列表

勾选框

（4） Tick 数据接收函数

函数定义：def handle_tick(context, tick, msg_type)

参数说明：

context：全局上下文 Context 对象

tick：推送的 Tick 对象

msg_type：保留字段

调用时间：接收到 tick 数据推送时

是否必须：否

特别说明：1）MQuant 支持 Level-2 行情，最快 3s 一个 tick，2）某些不活跃的标的 tick 间隔时间稍长，最长不超过 15s，3）收市期间 1 分钟一个 tick。4）订阅 tick 数据对带宽占用较大，不建议订阅大量标的的 tick 数据

（5） 逐笔委托接收函数

函数定义：def handle_order_record(context, order_record, msg_type)

参数说明：

context：全局上下文 Context 对象

order_record：逐笔委托数据，RecordOrder 对象

msg_type：保留字段

调用时间：接收到逐笔委托数据推送时

是否必须：否

特别说明：1）深市代码有逐笔委托，沪市没有。2）订阅逐笔委托对带宽占用较大，不建议订阅大量标的的逐笔委托数据

（6） 逐笔成交接收函数

函数定义：def handle_record_transaction(context, record_transaction, msg_type)

参数说明：

context：全局上下文 Context 对象

record_transaction ：逐笔成交数据，RecordTransaction 对象

msg_type：保留字段

调用时间：接收到逐笔成交数据推送时

是否必须：否

特别说明：订阅逐笔成交对带宽占用较大，不建议订阅大量标的的逐笔成交数据。

（7） 实时资金流向推送

函数定义：def handle_fund_flow(context, fund_flow):

参数说明：

context：全局上下文 Context 对象

fund_flow：资金流向数据，FundFlow 类型

调用时间：通过 subscribe 接口订阅资金流向数据后，有资金流向数据更新时调用

是否必须：否

（8） 订单回报接收函数

函数定义：def handle_order_report(context, ord, msg_type)

参数说明：

context：全局上下文 Context 对象

ord ：订单数据，Order 对象

msg_type：保留字段

调用时间：接收订单状态变化时推送

是否必须：否

特别说明：无。

（9） 成交回报接收函数

函数定义：def handle_execution_report(context, execution, msg_type):

参数说明：

context：全局上下文 Context 对象

execution：订单数据，Trade 对象

msg_type：保留字段

调用时间：订单有成交时推送

是否必须：否

特别说明：

成交回报函数，可选实现。策略报单后，只要订单产生成交（可能分多笔成交），都会进入成交回报函数，策略可以在此函数中获取本次成交信息（成交价格、数量、金额等），只要当前策略报单的成交回报会通过此函数通知给策略。回测模式下，成交回报也在此函数中处理。

（10） 资金更新回调

函数定义：def on_fund_update(context, fund_info):

参数说明：

context：全局上下文 Context 对象

fund_info：资金更新数据，FundUpdateInfo 对象

调用时间：策略的资金账号有资金信息变动时推送

是否必须：否

特别说明：实时行情更新导致的市值变动不会触发资金推送

（11） 持仓更新回调

函数定义：def on_position_update(context, pos_info):

参数说明：

context：全局上下文 Context 对象

pos_info：持仓更新信息，Position 类对象

调用时间：持仓更新时调用

是否必须：否

特别说明：实时行情更新导致的市值变动不会触发持仓推送

（12） 请求用户参数异步响应

函数定义：def on_rsp_user_params(context, json_params, wnd_title):

参数说明：

context：全局上下文 Context 对象

json_params：与 get_user_params 传入参数模板格式相同，但参数值会和用户通过界面修改后的值保持一致

wnd_title:界面弹窗的窗口名称

调用时间：异步模式调用 get_user_params 接口弹出弹窗，用户设置弹窗参数后，点击确定，会调用此回调

是否必须：否

特别说明：策略调用 get_user_params 在界面弹出一个参数框（异步模式），用户修改参数后，点击确定，修改后的参数将通过此回调函数回传给策略

（13） 用户触发弹出参数设置窗口请求处理函数

函数定义：def on_request_user_params_template(context, params):

参数说明：

context：全局上下文 Context 对象

params：用户输入参数，保留不用

调用时间：用户在策略执行监控实例列表界面点击手工干预按钮，将会触发此回调函数

是否必须：否

特别说明：用户在策略执行监控实例列表界面点击手工干预按钮，将会触发此回调函数，策略可以在此回调函数中获取用户参数，也可不实现

（14） ETF 预估信息接收函数

函数定义：def handle_etf_estimate_info(context, etf_estimate_info, msg_type)

参数说明：

context：全局上下文 Context 对象

etf_estimate_info：ETF 预估信息，EtfEstimateInfo 对象

msg_type：保留字段

调用时间：接收 ETF 预估信息变化时推送

是否必须：否

特别说明：ETF 预估信息定时 1s 计算一次。

（15） 策略准备停止通知函数

函数定义：def on_strategy_ready_stop(context):

参数说明：

context：全局上下文 Context 对象

调用时间：策略终止前调用

是否必须：否

特别说明：策略准备结束回调，可选实现。停止策略时，会先调用此回调，调用结束后，再调用 on_strategy_end 回调。该回调中允许交易，不允许与外部系统交互，策略可选择在此回调函数中撤掉在途订单。

（16） 策略终止通知函数

函数定义：def on_strategy_end(context)

参数说明：

context：全局上下文 Context 对象

调用时间：策略终止时调用

是否必须：否

特别说明：该函数中允许读写文件，可以在此函数中进行环境清理工作，例如取消订阅等。

（17） 策略运行过程中参数变更通知函数

函数定义：def on_strategy_params_change(params, path)

参数说明：

params：策略参数，可以是任意形式，由您自行传入，并在策略中自行解析

path: 如果传入参数文件，path 为参数文件路径，否则为空字符串

调用时间：通过 MQuant 执行监控界面设置参数时调用

是否必须：否

特别说明：支持两种设置参数的方式，一种是修改策略的运行参数，另一种是策略运行时传入参数文件，如

（18） K 线数据接收函数

函数定义：def handle_data (context, data)

参数说明：

context：全局上下文 Context 对象

data：推送的 KLineDataPush 对象

msg_type：保留字段

调用时间：接收到 tick 数据推送时

是否必须：否

特别说明：分钟 k 数据可能会有秒级延迟，特别是测试环境

（19） 开盘信号接收函数（回测专用）

函数定义：def market_open (context, trade_date)

参数说明：

context：全局上下文 Context 对象

trade_date 交易日

调用时间：回测过程中每个交易日开始时

是否必须：否

特别说明：仅回测模式下会被调用

（20） 收盘信号接收函数（回测专用）

函数定义：def market_close (context, trade_date)

参数说明：

context：全局上下文 Context 对象

trade_date 交易日

调用时间：回测过程中每个交易日结束时

是否必须：否

特别说明：仅回测模式下会被调用

（21） 异步报撤单回调

函数定义：def on_recv_order_reply(context, order_reply):

参数说明：

context：全局上下文 Context 对象

order_reply 报撤单异步响应，OrderReply 对象

调用时间：客户端收到报撤单异步响应时（必须首选开启全局开关，默认不接收异步响应）

是否必须：否

特别说明：

报撤单异步响应，可选实现。必须调用 set_enable_order_reply(True)开启响应消息推送后，才可收到响应。set_enable_order_reply 是全局开关，任何一个实例调用此接口设置后，其他实例都会受影响。

7.2.9 定时信号

MQuant 提供两种类型的定时信号，都是通过 run_timely 接口订阅的，一种是订阅周期性的定时信号，例如每 3s 推送一次的定时信号；第二种是订阅某个时刻触发一次的定时信号，例如 14:23:57 秒触发。函数定义如下：

def run_timely(func, interval, start_time=None, stop_time=None , custom_params=''):

接口功能：订阅日内定时信号，精度为秒

参数说明：

func: 回调函数，形式为 func(context, interval,msg_type),多个定时信号可共用一个回调函数

interval: 间隔时间，单位为秒，interval=-1 时，表示只在起始时间 start_time 发出一次定时信号(start_time=None

判定为非法），interval>0 表示订阅周期性的定时信号

start_time: 起始时间，格式为:"hh:mm:ss",比如 "10:00:01", "01:00:05"(精确到秒)，None 表示以订阅请求发起时间为开始时间

stop_time: 结束时间，格式为:"hh:mm:ss",比如 "10:00:01", "01:00:05"(精确到秒)，None 表示没有结束时间

返回值：None

custom_params:用户透传字段，会在定时回调函数中传给策略

接口调用示例：

```
def period_timer_func(context, interval, custom_params):
    #周期性定时信号
    pass

def single_timer_func(context, time, custom_params):
    #单次触发的定时信号
    pass

run_timely(period_timer_func,3,'period_timer_3')          #订阅一个 3s 的周期性定时信号
run_timely(single_timer_func,-1,'10:01:01','single_timer_10:01:01')      #订阅 10:01:01 触发的单次触发定时信号
```

特别说明：

设置了起始时间的周期定时信号，会从设置的起始时间开始，首先触发一次定时回调函数，然后每隔设置的时间间隔触发一次回调函数。

7.2.10 日志接口

MQuant 日志操作是通过 log 类的一系列静态函数来支持的，包括 debug、info、warn、error 四种级别的日志，支持设置日志级别，支持设置日志是否在控制台窗口显示。此外，为了支持用户在策略运行过程中记录自定义日志，系统还提供了注册用户自定义日志文件、写入用户自定义日志文件和清空用户自定义日志文件的接口。接口如下：

```
class log(object):
    """
    日志模块，与聚宽日志模块及 python 的 logging 模块用法基本一致，需要使用 MQuant 统一日志格式，不支持自定义日志格式
    """
    @staticmethod
    def error(content, label='')
        参数说明：
            content:日志内容
            label:日志标签，不同标签的日志会记录到对应的日志文件中，默认记录到系统日志文件中
```

@staticmethod

def warn(content, label='')

参数说明：

content:日志内容

label:日志标签，不同标签的日志会记录到对应的日志文件中，默认记录到系统日志文件中

@staticmethod

def info(content, label='')

参数说明：

content:日志内容

label:日志标签，不同标签的日志会记录到对应的日志文件中，默认记录到系统日志文件中

@staticmethod

def debug(content, label='')

参数说明：

content:日志内容

label:日志标签，不同标签的日志会记录到对应的日志文件中

@staticmethod

def set_log_level(log_level, lable='')

接口功能：设置日志文件记录日志级别，仅影响策略端日志记录，日志默认级别都是 DEBUG

参数说明：

log_level:LOG_LEVEL 类型

label:日志标签，不同标签的日志级别设置互不影响，默认设置系统日志级别

@staticmethod

def set_console_log_level(log_level, lable='')

接口功能：设置控制台日志级别，仅影响策略端日志记录，日志的默认级别都是 DEBUG，调用此接口设置用户自定义日志的级别会自动开启控制台输出

参数说明：

log_level:LOG_LEVEL 类型

lable:日志标签，不同标签的日志级别设置互不影响

@staticmethod

def enable_console_output(enable, label='')

接口功能：设置是否允许 log 模块打印的日志在控制台输出,默认系统日志允许输出到控制台，用户自定义日志不输出到控制台

参数说明：

enable: bool 类型

label:日志标签，不同标签的日志级别设置互不影响

@staticmethod

def register_strategy_report_file(file_path, label)

接口功能：注册用户自定义日志文件，注册的日志文件策略运行过程中允许写入，不允许读取；该接口必须在初始化阶段调用

参数说明：

file_path: 文件路径，支持相对路径和绝对路径，相对路径为相对于软件根目录的路径，注意，一定要保证文件所在目录存在，否则写入文件会失败

label: 标签，相当于 file_path 的别名

@staticmethod

def clean_log_file(label)

接口功能：清空日志文件

参数说明：

label: 日志文件别名

7.2.11 其他接口

7.2.11.1 策略停止接口

MQuant 支持策略主动停止，调用函数定义如下：

```
def stop_stragety():
    """
    停止策略：策略端向 MATIC 请求
    """
```

7.2.11.2 启动新策略接口

MQuant 支持从运行中的策略实例 A 启动策略实例 B，间隔时间目前是 10 分钟，函数定义如下：

def start_strategy(strategy_file, run_params='', instance_name='', show_params=False):

接口功能：创建策略实例，允许带参启动，参数会直接传给新启动的实例，不会做任何额外处理， 新启动的实例会继承父实例的资金账号

参数说明：

strategy_file 策略文件路径，支持相对路径，相对路径为相对用户目录下的文件路径（界面上能看到的相对于 MQuant 根节点的路径）

run_params 启动参数，最长 4096 字节,可以为空

instance_name 可选参数，如果未填写默认生成

show_params 启动时是否弹出参数修改弹框，可选参数，如果未填写默认不弹出

返回值：

成功返回实例 ID，失败返回空字符串

备注：

如下图，策略实例 A（实例 ID：26119）创建了实例 B（实例 ID：26120，调用 start_strategy

传入的 instance_name='测试策略')，则策略实例 B 的实例名称会显示为“测试策略_26119”

7.2.11.3 文件访问接口

MQuant 内置了访问 ini 和 csv 的接口，您也可以使用 python 自带的文件访问接口在初始化函数中读取任意参数文件。如果需要安装第三方的文件读取包，需要设置自定义策略运行环境。

（1） 读取 ini 文件

def read_ini_config(path, section, key, buf_len=1024):

接口功能：读取 ini 配置文件,仅允许在 initialize 函数中使用

参数说明：

path:ini 配置文件路径，支持相对路径和绝对路径

section:ini 文件 section

key:ini 文件 key

buf_len:缓冲区长度

返回值：ini 文件相应 section 中 key 对应的 value

示例：

```
configValue = read_ini_config('./configFile/usrCfg.ini', 'TEST', 'key1')
```

（2） 读取 csv 文件

def open_csv_file(path, encodeing='utf-8'):

接口功能：获取 csv 读取对象，目前只支持 utf-8 编码的 csv 文件

参数说明：

path:csv 文件路径，支持相对路径和绝对路径

encodeing:编码方式

返回值：CsvReader 对象，在 HTSACsvReader.py 中定义

示例：

```
csv_reader = open_csv_file('./configFile/testCsv.csv')
    count = 0
    for row in csv_reader:
```

```
        print(row)
        count = count + 1

        if count > 10:
            break



csv_reader.reset()
print('第 5 行：', csv_reader.getRow(5))
csv_reader.close()
```

7.2.11.4 获取当前时间

MQuant 提供获取当前时间接口的主要目的是为了在行情回放及回测场景下为策略提供准确的行情时钟，兼容实时行情场景下的时间获取。函数定义如下：

def get_cur_time():

接口功能：获取当前时间，实盘模式下等同于 datetime.datetime.now()，回测及行情回放模式下，是根据行情计算出的当前时间

返回值：datetime.datetime 对象

7.2.11.5 Sqlite 内存数据库访问接口

MQuant 为每个策略实例提供了一个独立的 sqlite 内存数据库，方便用户在策略运行过程中进行数据存取。获取 sqlite 连接函数如下：

def get_sqlite_connection():

接口功能：获取 MQuant 内置的 sqlite 内存数据库连接

返回值：HtDbConnection 对象，在 HTSADbAccess 中定义

示例：

```
conn = get_sqlite_connection()
conn.excutesql('''CREATE TABLE `ma_5` (
                        `index` BIGINT(20) NULL DEFAULT NULL,
                        `code` VARCHAR(16) NULL DEFAULT NULL,
                        `date` VARCHAR(50) NULL DEFAULT NULL,
                        `ma` DOUBLE NULL DEFAULT NULL
                        )''')
conn.excutesql('insert into ma_5 values(0,?1,?2,19.50)', ['601688', '2018-01-29'])
conn.excutesql('insert into ma_5 values(1,\'000002\',\'2018-01-28\',23.50)')
```

```
conn.transaction()
conn.excutesql('insert into ma_5 values(2,?1,?2,19.50)', ['000001', '2018-01-29'])
conn.excutesql('insert into ma_5 values(3,?1,?2,19.50)', ['000100', '2018-01-29'])
conn.excutesql('insert into ma_5 values(4,?1,?2,19.50)', ['601600', '2018-01-29'])
conn.rollback()

conn.transaction()
conn.excutesql('insert into ma_5 values(5,?1,?2,19.50)', ['000001', '2018-01-29'])
conn.excutesql('insert into ma_5 values(6,?1,?2,19.50)', ['000100', '2018-01-29'])
conn.excutesql('insert into ma_5 values(7,?1,?2,19.50)', ['601600', '2018-01-29'])
conn.commit()

conn.excutesql('select * from ma_5')
print(conn.fetchall())
```

7.2.11.6 执行外部 python 脚本

def exec_py(py_file_path, params='', python_interpreter_path='', cmd_run=False):

接口功能：

同步执行外部 python 脚本,支持使用用户本地的 python 环境，典型的应用场景是通过第三方财经信息库获取数据并写入文件，在 MQuant 策略脚本中调用此接口执行外部脚本后，立即读取外部脚本落在本地的文件，仅支持初始化阶段调用

参数说明：

py_file_path:python 文件路径，建议全路径

params: 脚本执行参数

python_interpreter_path:python 解释器路径，要带上 python.exe，空字符串表示 mquant 内置解释器，也可以用用户本地安装的解释器

cmd_run: 是否直接通过 cmd 执行。注意，cmd_run=True 时，函数返回值为空字符串。

返回值：

执行 python 脚本的返回值

7.2.11.7 人工干预

def get_user_params(json_params_template, wnd_title, sync=False):

接口功能：

请求用户参数

参数说明：

json_params_template:参数模板，体现在界面上是一个弹窗，目前弹窗中仅支持表格。参数模板格式示

例如下：

{"left_layout":{"name":"操作表格","ui_type":"table","columns":[{"name":"","ui_type":"check_box"},{"name":"证券代码","ui_type":"code_edit"},{"name":"持仓数量","ui_type":"line_edit","editable":"0"},{"name":"可用数量","ui_type":"line_edit","editable":"0"},{"name":"持仓均价","ui_type":"line_edit","editable":"0"},{"name":"止损价","ui_type":"line_edit"}],"rows":[[0,"601688.SH",5400,1800,29.15,29.15],[0,"000001.SZ",3800,3000,18.15,18.15]]}}

wnd_title:窗口标题

sync：是否同步请求，如果采用同步模式，策略会等用户对弹框进行确定或者取消后，才会继续向下执行，由于策略的回调是单线程的，这样会导致所有通知消息都阻塞；默认为异步模式。

返回值：

同步模式返回用户确认后的内容，格式与参数模板一致；异步模式返回空字符串。

7.2.11.8 声音提示

（1）播放音频文件

def play_sound(wav_file=''):

接口功能：

播放音频文件，目前仅支持播放无损 wav 文件，如果传入为空，则播放系统默认的音频文件

参数说明：

wav_file:声音文件，如果为空，则使用系统自带的声音

返回值：

成功，返回当前音频的唯一 ID，停止音频时需要传入，失败返回空字符串

（2）停止播放音频文件

def stop_sound(sound_id):

接口功能：

停止播放指定 sound_id 的音频，sound_id 为 play_sound 接口的返回值

参数说明：

sound_id:声音文件，如果为空，则使用系统自带的声音

返回值：

None

7.2.11.9 设置实例备注

def set_instance_remark(remark):

接口功能：

设置策略实例的备注信息，展示在策略实例列表界面上。

参数说明：

remark: 实例备注

返回值：None

7.2.11.10 设置报撤单异步响应全局开关

def set_enable_order_reply(enable=False):

接口功能：设置报撤单异步响应开关

参数说明：

enable: True，开启，此时所有策略都能接收到异步响应；False，关闭，此时所有策略都不能接收到异步响应

返回值：

0：成功，-1：失败

7.2.12 两融接口类

7.2.12.1 交易接口

（1）融资买入

def margincash_open(security, amount, style=None, pindex=0, batch_no=-1, msg_type=-1):

接口功能： 融资买入

参数说明：

security: 标的代码，支持单标的和多标的，多标的用 list 传入

amount: 数量，支持单标的和多标的，多标的用 list 传入

style:订单类型，支持市价单和限价单，参数类型为 OrderStyle 子类型，None 代表 MarketOrder，支持单标的和多标的，多标的用 list 传入

pindex:无效字段，仅保留用作聚宽兼容

batch_no:批次号

msg_type 透传字段，传入可获得协程并发能力，暂不可用

返回值：

list<Order>对象或者 None，如果创建委托成功，则返回 list<Order>对象，失败则返回 None

（2）卖券还款

def margincash_close(security, amount, contract_no='', style=None, pindex=0, batch_no=-1, msg_type=-1):

接口功能： 卖券还款

参数说明：

contract_no: 合约编号，如果为空，表示按照默认顺序归还

security: 标的代码，支持单标的和多标的，多标的用 list 传入

amount: 数量，支持单标的和多标的，多标的用 list 传入

style:订单类型，支持市价单和限价单，参数类型为 OrderStyle 子类型 ，None 代表 MarketOrder，支持单标的和多标的，多标的用 list 传入

pindex:无效字段，仅保留用作聚宽兼容

batch_no:批次号

msg_type 透传字段，传入可获得协程并发能力，暂不可用

返回值：

list<Order>对象或者 None，如果创建委托成功，则返回 list<Order>对象，失败则返回 None

（3）融券卖出

def marginsec_open(security, amount, position_type=PositionType.normal, style=None, pindex=0, batch_no=-1, msg_type=-1):

接口功能： 融券卖出,不支持市价单

参数说明：

position_type:头寸类型 PositionType，支持普通头寸和专项头寸，默认为普通头寸

security: 标的代码，支持单标的和多标的，多标的用 list 传入

amount: 数量，支持单标的和多标的，多标的用 list 传入

style: 参数类型为 OrderStyle 子类型， None 代表 MarketOrder，支持单标的和多标的，多标的用 list 传入

pindex: 无效字段，仅保留用作聚宽兼容

batch_no: 批次号

msg_type 透传字段，传入可获得协程并发能力，暂不可用

返回值：

list<Order>对象或者 None，如果创建委托成功，则返回 list<Order>对象，失败则返回 None

（4）买券还券

def marginsec_close(security, amount, contract_no='', position_type=PositionType.normal, style=None, pindex=0, batch_no=-1, msg_type=-1):

接口功能： 买券还券

参数说明：

contract_no:合约编号，如果为空，表示按照默认顺序归还

position_type:头寸类型 PositionType，支持普通头寸和专项头寸，默认为普通头寸

security: 标的代码，支持单标的和多标的，多标的用 list 传入

amount: 数量，支持单标的和多标的，多标的用 list 传入

style:参数类型为 OrderStyle 子类型, None 代表 MarketOrder，支持单标的和多标的，多标的用 list 传入

pindex:无效字段，仅保留用作聚宽兼容

batch_no:批次号

msg_type 透传字段，传入可获得协程并发能力

返回值：

list<Order>对象或者 None，如果创建委托成功，则返回 list<Order>对象，失败则返回 None

（5）直接还券

def marginsec_direct_refund(security, amount, position_type=PositionType.normal, contract_no='', pindex=0, batch_no=-1, msg_type=-1):

接口功能： 直接还券

参数说明：

contract_no:合约编号，如果为空，表示按照默认顺序归还

position_type:头寸类型 PositionType，支持普通头寸和专项头寸，默认为普通头寸

security: 标的代码，支持单标的和多标的，多标的用 list 传入

amount: 数量，支持单标的和多标的，多标的用 list 传入

style:参数类型为 OrderStyle 子类型, None 代表 MarketOrder，支持单标的和多标的，多标的用 list 传入

pindex:无效字段，仅保留用作聚宽兼容

msg_type :透传字段，传入可获得协程并发能力

返回值：

list<Order>对象或者 None，如果创建委托成功，则返回 list<Order>对象，失败则返回 None

（6）直接还款

def margincash_direct_refund(value, contract_no='', pindex=0, msg_type=-1):

接口功能： 直接还款

参数说明：

contract_no:合约编号，如果为空，表示按照默默认顺序归还

value: 还款金额

pindex: 无效字段，仅保留用作聚宽兼容

msg_type :透传字段，传入可获得协程并发能力

返回值：None

（7）担保品买卖

def margin_trade(security, amount, style=None, pindex=0, batch_no=-1, msg_type=-1):

接口功能： 普通信用交易，即担保品买卖

参数说明：

security: 标的代码，支持单标的和多标的，多标的用 list 传入

amount: 数量，支持单标的和多标的，多标的用 list 传入,数量<0 表示卖出

style:参数类型为 OrderStyle 子类型, None 代表 MarketOrder，支持单标的和多标的，多标的用 list 传入

index: 无效字段，仅保留用作聚宽兼容

batch_no: 批次号

msg_type 透传字段，传入可获得协程并发能力

返回值：

list<Order>对象或者 None，如果创建委托成功，则返回 list<Order>对象，失败则返回 None

7.2.12.2 查询接口

（1）获取融资标的列表

def get_margincash_stocks():

接口功能：获取融资标的列表

返回值：返回上交所、深交所最近一次披露的的可融资标的列表的 list

（2）获取融资标的列表

def get_marginsec_stocks():

接口功能：获取融券标的列表

返回值：返回上交所、深交所最近一次披露的的可融券标的列表的 list。

（3）查询信用合约

def get_margin_contract(contract_no=''):

接口功能：查询信用合约

返回值：

查询成功,如果 contract_no 为空，返回 dict<contract_no,MarginContract>,如果 contract_no 不为空，返回 MarginContract 对象

查询不成功，返回 None

（4）查询信用资产

def get_margin_assert():

接口功能：查询信用资产

返回值：

查询成功，返回 MarginAssert 类对象，

失败返回 None,信用资产非实时更新，会定时从柜台同步，目前定时时间间隔 1 分钟一次

（5）从柜台查询信用资产

def get_margin_asset_from_counter():

接口功能：从柜台查询信用资产

返回值：返回 MarginAssetRsp 对象

备注：此接口目前限频为 10s 一次

（6）查询担保券

def get_assure_security_list():

接口功能：查询担保券

返回值：担保券列表 dict<symbol,discount_ratio>

（7）查询融券标的信息

def get_margin_security_info(symbol='', position_type=PositionType.undefine):

接口功能：查询融券标的信息

参数说明：

symbol:标的

position_type:头寸性质，在 PositionType 中定义

返回值：

查询成功，返回 list<MarginSecurityInfo>，查询失败返回 None

（8）查询大约可交易数量

def get_avaliable_qty_for_trade(symbol, entrust_type, side, style=None, position_prop=PositionType.normal):

接口功能：信用交易查询大约可交易数量

参数说明：

symbol:标的，必传

entrust_type:委托类型，EntrustType 类型，必传

side:交易方向，OrderSide 类型，

style:价格类型，MarketOrderStyle 或者 LimitOrderStyle 类型

position_prop:头寸性质，在 PositionType 中定义

返回值：

返回指定条件的最大可交易数量

7.2.13 ETF 接口类

7.2.13.1 交易接口

（1）篮子下单接口（废弃）

def basket_order(self, amount, style=None, side='long', pindex=0, msg_type=-1):

接口功能：篮子下单口

参数说明：

security: ETF 基金代码

amount:交易篮子份数，正数表示买入，负数表示卖出

style:订单类型，支持市价单和限价单，参数类型为 OrderStyle 子类型，None 代表 MarketOrder。MQuant

在聚宽基础上扩展了市价单类型，默认为最优五档即时成交剩余撤销(沪市深市均可)，您也可选择其他类型

side:买卖方向，在 OrderSide 类中定义，也可以直接输入 long’/’short’，分别表示买/卖。默认为买入，股票、基金暂不支持开空单。**该字段暂不可用**

pindex:在使用 set_subportfolios 创建了多个仓位时，指定 subportfolio 的序号, 从 0 开始, 比如 0 指定第一个 subportfolio, 1 指定第二个 subportfolio，默认为 0，目前 MQuant 只支持默认值 0

msg_type 透传字段，传入可获得协程并发能力，暂不可用

返回值：

Order 对象或者 None, 如果创建订单成功, 则返回 Order 对象, 失败则返回 None

（2）ETF 基金申赎接口

def etf_purchase_redemption(self,amount, style=None, side='long', pindex=0, msg_type=-1):

接口功能：ETF 基金申赎接口

参数说明：

security: ETF 基金代码

amount:基金申赎数量, 正数表示申购, 负数表示赎回

style:订单类型，支持市价单和限价单，参数类型为 OrderStyle 子类型，None 代表 MarketOrder。MQuant 在聚宽基础上扩展了市价单类型，默认为最优五档即时成交剩余撤销(沪市深市均可)，您也可选择其他类型

side:买卖方向，在 OrderSide 类中定义，也可以直接输入 long’/’short’，分别表示买/卖。默认为买入，股票、基金暂不支持开空单。**该字段暂不可用**

pindex:在使用 set_subportfolios 创建了多个仓位时，指定 subportfolio 的序号, 从 0 开始, 比如 0 指定第一个 subportfolio, 1 指定第二个 subportfolio，默认为 0，目前 MQuant 只支持默认值 0

msg_type 透传字段，传入可获得协程并发能力，暂不可用

返回值：

Order 对象或者 None, 如果创建订单成功, 则返回 Order 对象, 失败则返回 None

7.2.13.2 查询接口

（1）获取 ETF 基本信息

def getEtfInfo(self):

接口功能： 获取 ETF 基本信息

返回值：HtEtfInfo 对象，失败返回 None

（2）获取 ETF 成分券列表信息

def getEtfConstituentList(self):

接口功能：获取 ETF 成分券列表信息，返回 list<HtEtfConstituentInfo>

返回值：list<HtEtfConstituentInfo>

（3）获取 ETF 成分券标的列表信息

def getEtfConstituentSymbols(self):

接口功能：获取 ETF 成分券的证券代码列表，返回 list<str>

返回值：list<str>

（4）获取 ETF 单只成分券信息

def getEftConstituentInfo(self, constituentSymbol):

接口功能：获取 ETF 单只成分券信息，返回 HtEtfConstituentInfo

参数说明：

constituentSymbol:成分券代码

返回值： HtEtfConstituentInfo

7.2.14 算法接口类

7.2.14.1 运行接口

（1）创建算法实例（废弃）

def create_instance(algo_type: AlgoType, start_time: datetime.datetime, end_time: datetime.datetime,side: OrderSide, order_list,
account_type=AccountType.unknown, extra_params={}):

接口功能：创建算法实例

参数说明：

account_type:默认（AccountType.unknown）取当前账号

algo_type:算法类型 AlgoType

start_time:支持字符串，datetime.datetime 对象和 datetime.time 对象，字符串格式为"09:30:00"

end_time:支持字符串，datetime.datetime 对象和 datetime.time 对象，字符串格式为"14:57:00"

side:OrderSide 对象

order_list:[{},{}]，列表中的每个 item 至少包含证券代码和委托数量两个参数，例如 [{AlgoParamKeys.symbol:'601688.SH',AlgoParamKeys.amount:3000}]

extra_params: 额外参数，字典格式，key 在 AlgoParamKeys 中定义,例如 {AlgoParamKeys.withdraw_interval:20,AlgoParamKeys.min_qty:100}

返回值：

成功返回 instanceId，失败返回 None

（2）创建拆单算法实例（仅 TWAP、VWAP 等常规拆单算法，需要单独申请开通权限）

def start_split_order_algo_instance(account_type, algo_params):

接口功能：

创建拆单算法实例，包括 AITWAP 和 AIVWAP，返回 CreateAlgoInstanceRsp 类型，创建后立即启动

参数说明：

account_type: 账户类型

algo_params: 算法参数,SplitOrderAlgoParam 类型

返回值：

CreateAlgoInstanceRsp 类型，如果成功，返回实例 ID，否则会返回错误信息

特别说明：

注意，目前通过 MQuant 创建算法实例需要单独申请授权，且同时在线的标的数量有限制。

（3）停止算法实例

def stop_instance(instId):

接口功能：停止算法实例

参数说明：

instId: 实例 id

返回值：None

（4）暂停实例

def pause_instance(instId):

接口功能：暂停实例

参数说明：

instId: 实例 id

返回值：None

（5）恢复实例

def resume_instance(instId):

接口功能：恢复实例

参数说明：

instId: 实例 id

返回值：None

7.2.14.2 查询接口

（1） 查询获取实例信息

def get_instance_info(instId):

接口功能： 查询获取实例信息，包含实例状态、进度、标的进度等信息

仅提供查询接口，不提供推送接口

参数说明：

instId: 实例 id

返回值：

成功，返回 AlgoInstanceInfo 对象，失败返回 None

（2） 查询当前用户的所有实例列表

def get_instance_id_list():

接口功能： 查询当前用户的所有实例列表

返回值：

成功，[inst_id1,inst_id2,...],失败返回 None

7.2.15 回测专用接口

def register_backtest_symbols(symbols)

接口功能：注册回测标的列表，必须在初始化阶段调用，实盘脚本调用无影响

参数说明：

symbols 回测标的列表，支持单只标的'601688.SH',也支持列表格式['601688.SH','000002.SZ']

返回值：None

备注：

仅通过该接口注册的标的列表才会收到对应的行情数据，系统才会接受其报单请求

7.2.16 发布订阅自定义信号

def publish_custom_msg(custom_msg_type, msg_data):

接口功能：发布自定义信号

参数说明：

custom_msg_type:自定义信号，int 类型，值应该在 10001-65536 之间

msg_data: 自定义信号参数，str 类型

返回值：None

def subscribe _custom_msg(custom_msg_type, func):

接口功能：订阅自定义信号

参数说明：

custom_msg_type:自定义信号，int 类型，值应该在 10001-65536 之间

func:回调函数，回调函数形式为 func(context, msg_type, msg_data)

返回值：None

def unsubscribe _custom_msg(custom_msg_type):

接口功能：取消订阅自定义信号

参数说明：

custom_msg_type:自定义信号，int 类型，值应该在 10001-65536 之间

返回值：None

7.3 自定义策略运行环境

MQuant 系统自带了一个绿色版的 python 解释器，包含了 python 解释器所有内置库，如果您需要用到第三方库，如 pandas、talib 等，可以有两种方式：

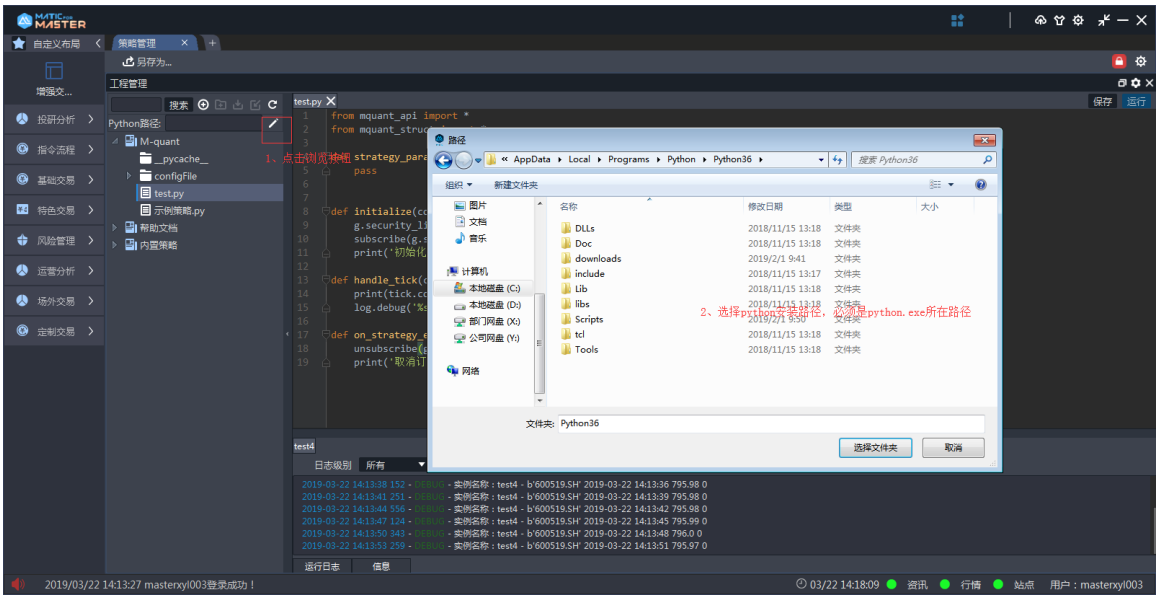
7.3.1 设置本机 python 运行环境

设置本机 python 运行环境的步骤如下：

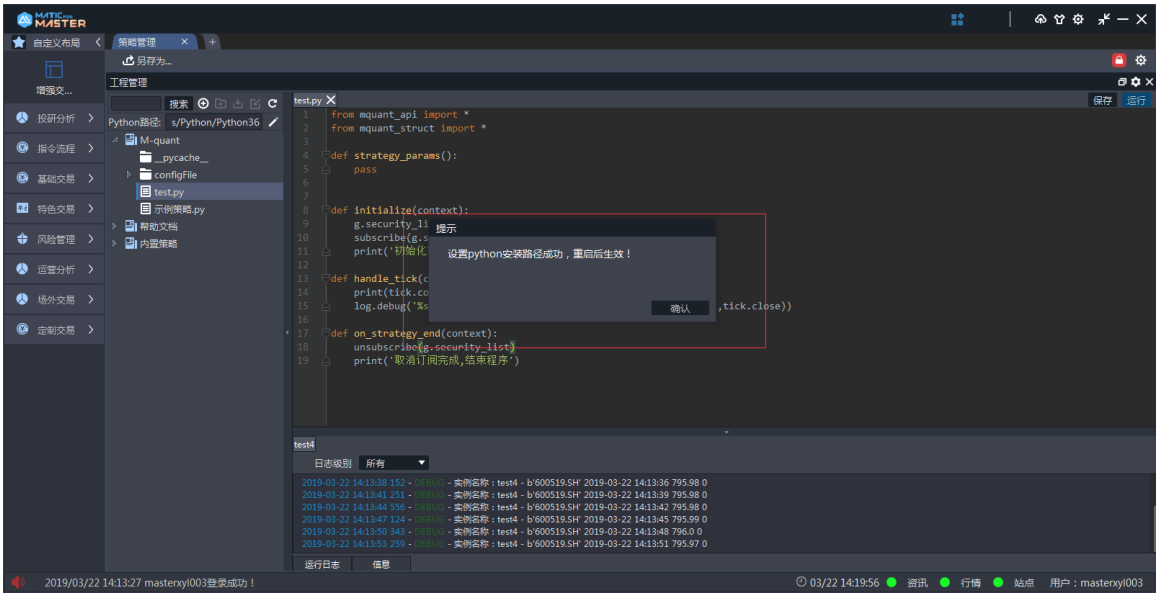
（1） 安装 64 位 python 解释器，python 版本为 3.6.2

（2） 安装所需的第三方包，如果您机器上安装了多个版本的 python，建议用绝对路径的 pip.exe 安装，避免安装到其他版本的 python 环境中。

（3） 在 MQuant 策略编辑界面上设置 python 安装路径，如下图所示：



（4） 按照界面提示重启软件，即可正常使用



7.3.2 在 MQuant 自带的绿色 python 解释器中安装第三方包（推荐）

在 cmd 窗口中执行 “<你的 MATIC 路径>\python.exe <你的 MATIC 路径>\Scripts\pip.exe install <第三方 python 包>”即可安装，如下图所示：

```
C:\Users\011048>D:\MaticMaster\common\release\qt5.8.0\win32-msvc2015\x64\python.
exe D:\MaticMaster\common\release\qt5.8.0\win32-msvc2015\x64\Scripts\pip.exe ins
tall D:\011048\Downloads\pika-1.1.0-py2.py3-none-any.whl
Processing d:\011048\downloads\pika-1.1.0-py2.py3-none-any.whl
Installing collected packages: pika
Successfully installed pika-1.1.0
```

7.4 附录

数据结构定义

7.4.1.1 行情类型定义

```
class MarketDataType(object):
    """
    行情类型定义
    """
    TICK = 'tick'  # tick 行情
KLINE_1M = 'kline_1m'  # 1 分钟 k 线
    RECORD_ORDER = 'record_order'  # 逐笔委托
    RECORD_TRANSACTION = 'record_transaction'  # 逐笔成交
    FUND_FLOW = 'fund_flow'  # 资金流向
```

7.4.1.2 行情 Tick 数据结构

```
class Tick(object):
    """
    实时行情数据，里面的金额、价格单位都是元，数量都是股/张
    """
    def __init__(self):
        self.code = ''  # 标的的代码，实时传入,MQuant 格式
        self.datetime = None  # tick 时间，实时传入
        self.current = 0  # 最新价，实时传入
        self.open = 0  # 开盘价，实时传入
        self.close = 0  # 收盘价
        self.pre_close = 0  # 昨收价，实时传入
        self.high = 0  # 最高价，实时传入
        self.low = 0  # 最低价，实时传入
```

```
self.volume = 0  # 截至到当前时刻的成交量，实时传入
self.amount = 0  # 截至到当前时刻的成交额，实时传入
self.position = 0  # 截至到当前时刻的持仓量，目前暂时不用，后续支持期货可用
self.bid_volume = 0  # 当前时刻的委买挂单总量
self.ask_volume = 0  # 当前时刻的委卖挂单总量
self.w_bid_price = 0  # 加权平均委买价格
self.w_ask_price = 0  # 加权平均委卖价格
# 涨跌停价
self.high_limit = 0
self.low_limit = 0
self.trading_phase_code = TradingPhaseCode.StartBeforeOpen  # 行情状态
self.num_trades = 0  # 成交笔数，2020.2.16 新增
self.total_bid_number = 0  # 买入总笔数，2020.2.16 新增
self.total_ask_number = 0  # 卖出总笔数，2020.2.16 新增
# 十档盘口，获取时查询
# 卖盘，挂单量
self.a1_v = 0
self.a1_amount_list = []  # 卖一挂单数量列表，最多 50
self.a2_v = 0
self.a3_v = 0
self.a4_v = 0
self.a5_v = 0
self.a6_v = 0
self.a7_v = 0
self.a8_v = 0
self.a9_v = 0
self.a10_v = 0
# 卖盘，挂单价
self.a1_p = 0
self.a2_p = 0
self.a3_p = 0
self.a4_p = 0
self.a5_p = 0
self.a6_p = 0
self.a7_p = 0
self.a8_p = 0
self.a9_p = 0
self.a10_p = 0

# 买盘，挂单量
self.b1_v = 0
self.b1_amount_list = []  # 买一挂单数量列表，最多 50
self.b2_v = 0
self.b3_v = 0
```

```
self.b4_v = 0
self.b5_v = 0
self.b6_v = 0
self.b7_v = 0
self.b8_v = 0
self.b9_v = 0
self.b10_v = 0
# 买盘，挂单价
self.b1_p = 0
self.b2_p = 0
self.b3_p = 0
self.b4_p = 0
self.b5_p = 0
self.b6_p = 0
self.b7_p = 0
self.b8_p = 0
self.b9_p = 0
self.b10_p = 0

# 买盘，委托笔数
self.b1_nums = 0
self.b2_nums = 0
self.b3_nums = 0
self.b4_nums = 0
self.b5_nums = 0
self.b6_nums = 0
self.b7_nums = 0
self.b8_nums = 0
self.b9_nums = 0
self.b10_nums = 0
# 卖盘，委托笔数
self.a1_nums = 0
self.a2_nums = 0
self.a3_nums = 0
self.a4_nums = 0
self.a5_nums = 0
self.a6_nums = 0
self.a7_nums = 0
self.a8_nums = 0
self.a9_nums = 0
self.a10_nums = 0

# 期货期权专用
self.pre_open_interest = 0    # 昨持仓
```

```
        self.open_interest = 0   # 持仓总量
        self.pre_settle_price = 0   # 昨结算价
        self.settle_price = 0   # 今结算价
        self.pre_delta = 0   # 昨虚实度
        self.curr_delta = 0   # 今虚实度

        # 股债基
        self.withdraw_buy_num = 0   # 买入撤单笔数
        self.withdraw_buy_amount = 0   # 买入撤单数量
        self.withdraw_buy_money = 0   # 买入撤单金额
        self.withdraw_sell_num = 0   # 卖出撤单笔数
        self.withdraw_sell_amount = 0   # 卖出撤单数量
        self.withdraw_sell_money = 0   # 卖出撤单金额
        self.bid_trade_max_duration = 0   # 买入委托成交最大等待时间
        self.ask_trade_max_duration = 0   # 卖出委托成交最大等待时间
        self.num_bid_orders = 0   # 买方委托价位数
        self.num_ask_orders = 0   # 卖方委托价位数
        self.IOPV = 0
```

7.4.1.3 推送 K 线数据结构

```
class KLineDataPush(object):
    """
    k 线数据，推送时的 k 线数据结构
    """

    def __init__(self):
        self.datetime = None   # 时间，datetime.datetime 类型
        self.open = 0   # k 线周期内的开盘价
        self.close = 0   # k 线周期内的收盘价
        self.high = 0   # k 线周期内的最高价
        self.low = 0   # k 线周期内的最低价
        self.total_volume = 0   # k 线周期内的成交总量
        self.total_money = 0   # k 线周期内的总成交金额
        self.num_trades = 0   # k 线周期内的总成交笔数
        self.symbol = ''            #证券代码
        self.data_type = KLineDataType.KLINEData_1M #k 线数据类型
```

7.4.1.4 查询 k 线数据结构

```
class KLineData(object):
```

```
"""
k 线数据
"""

def __init__(self):   # 数据，key 包含'open', 'close', 'high', 'low', 'volume', 'money'，对应的 value 都是 list
    self.data = {
        'date': [],
        'open': [],
        'close': [],
        'high': [],
        'low': [],
        'volume': [],
        'money': []
    }
    self.symbol = ''   # 证券代码,支持 MQuant 和聚宽格式

def is_ready(self):
    """
    数据是否准备好
    :return:
    """
    if len(self.security) > 6:
        return True
    return False

def highest_price(self):
    """
    最高价
    :return:
    """
    return max(self.data['high'])

def lowest_price(self):
    """
    最低价
    :return:
    """
    return min(self.data['low'])

def avg_money(self):
    """
    平均成交金额
    :return:
    """
```

```
        if len(self.data['money']) == 0:
            return 0
        return sum(self.data['money']) / len(self.data['money'])

    def avg_volume(self):
        """
        平均成交量
        :return:
        """
        if len(self.data['volume']) == 0:
            return 0
        return sum(self.data['volume']) / len(self.data['volume'])

    def avg_close_price(self):
        """
        平均收盘价
        :return:
        """
        if len(self.data['close']) == 0:
            return 0
        return sum(self.data['close']) / len(self.data['close'])
```

7.4.1.5 订单类型

```
class OrderStyle(metaclass=ABCMeta):
    """
    订单类型
    """

    @abstractmethod
    def get_order_style(self):
        pass

    @abstractmethod
    def get_limited_price(self):
        pass
```

市价单类型

```
class MarketOrderStyle(OrderStyle):
    """
    市价单,默认为五档即成剩撤
    """
```

```
def __init__(self, type='a' , limit_price=1.0):
    """
    科创板的保护限价通过 limit_price 指定
    :param type:
        "2";最新价,暂不提供使用
        "3";涨停价,暂不提供使用
        "4";跌停价,暂不提供使用
        "5";买 1,暂不提供使用
        "6"; ///买 2,暂不提供使用
        "7"; ///买 3,暂不提供使用
        "8"; ///买 4,暂不提供使用
        "9"; ///买 5,暂不提供使用
        "10"; ///买 6,暂不提供使用
        "11"; ///买 7,暂不提供使用
        "12"; ///买 8,暂不提供使用
        "13"; ///买 9,暂不提供使用
        "14"; ///买 10,暂不提供使用
        "15"; ///卖 1,暂不提供使用
        "16"; ///卖 2,暂不提供使用
        "17"; ///卖 3,暂不提供使用
        "18"; ///卖 4,暂不提供使用
        "19"; ///卖 5,暂不提供使用
        "20"; ///卖 6,暂不提供使用
        "21"; ///卖 7,暂不提供使用
        "22"; ///卖 8,暂不提供使用
        "23"; ///卖 9,暂不提供使用
        "24"; ///卖 10,暂不提供使用
        "25"; ///"买:卖 1/卖:买 1",暂不提供使用
        "26"; ///"买:卖 2/卖:买 2",暂不提供使用
        "27"; ///"买:卖 3/卖:买 3",暂不提供使用
        "28"; ///"买:卖 4/卖:买 4",暂不提供使用
        "29"; ///"买:卖 5/卖:买 5",暂不提供使用
        "30"; ///"买:买 1/卖:买 1",暂不提供使用
        "31"; ///"买:买 2/卖:买 2",暂不提供使用
        "32"; ///"买:买 3/卖:买 3",暂不提供使用
        "33"; ///"买:买 4/卖:买 4",暂不提供使用
        "34"; ///"买:买 5/卖:买 5",暂不提供使用
        "35"; ///买:卖 1/卖:卖 1,暂不提供使用
        "36"; ///买:卖 2/卖:卖 2,暂不提供使用
        "37"; ///买:卖 3/卖:卖 3,暂不提供使用
        "38"; ///买:卖 4/卖:卖 4,暂不提供使用
        "39"; ///买:卖 5/卖:卖 5,暂不提供使用
        "40"; ///买/卖:最新价,暂不提供使用
```

"a"; ///上交所五档即成剩撤
"b"; ///上交所五档即成剩转限价
"a"; ///深交所五档即成剩撤
"c"; ///深交所即成剩撤
"d"; ///深交所对手方最优
"e"; ///深交所本方最优
"f"; ///深交所全额成交或撤
"l"; ///中金所五档即成剩撤
"PFP"#/盘后固定价（科创板）
"0" #/增强限价盘（港股通）
"2" #/竞价限价盘（港股通）
"""

```
self.__type = type
self.limit_price = limit_price
def get_order_style(self):
return self.__type

def get_limited_price(self):
return self.limit_price
```

限价单类型

```
class LimitOrderStyle(OrderStyle):
    """
    限价单
    """

    def __init__(self, limit_price):
        self.limit_price = limit_price

    def get_order_style(self):
        return '1'

    def get_limited_price(self):
        return self.limit_price
```

7.4.1.6 订单状态

```
class OrderStatus(Enum):
```

```
    # 订单新创建未委托，用于盘前/隔夜单，订单在开盘时变为 open 状态开始撮合(待报)
    new = 8

    # 订单未完成, 无任何成交（已报）
    open = 0

    # 订单未完成, 部分成交（部成）
    filled = 1

    # 订单完成, 已撤销, 可能有成交, 需要看 Order.filled 字段（已撤/部撤）
    canceled = 2

    # 订单完成, 交易所已拒绝（废单）
    rejected = 3

    # 订单完成, 全部成交, Order.filled 等于 Order.amount（已成）
    held = 4

    # 订单取消中，只有实盘会出现，回测/模拟不会出现这个状态（已报待撤/部成待撤）
    pending_cancel = 9
```

7.4.1.7 买卖方向

```
class OrderSide(object):
    """
    交易方向定义
    """
    BUY = 'long'   # 买入
    SELL = 'short'   # 卖出
    UNKNOWN = ''      #未知
```

7.4.1.8 订单数据结构

```
class Order(object):
    """
    订单对象
    """

    def __init__(self):
        """
        以下字段除标注'下单立即返回'字样之外，均需要调用查询订单接口（get_open_orders、get_orders等）查询才会填充
        """
```

self.status = OrderStatus.new # 状态，一个 OrderStatus 值，随着订单执行状态改变
self.add_time = None # 订单创建时间, datetime.datetime 对象，为后台时间
# self.is_buy = False # bool 值，买还是卖，对于期货: 开多/平空 → 买，开空/平多→卖，下单立即返回
self.amount = 0 # 下单数量, 不管是买还是卖, 都是正数，下单立即返回
self.filled = 0 # 已经成交的股票数量, 正数
self.security = '' # 股票代码，同下单请求的 security 参数，下单立即返回
self.order_id = '' # 订单 ID，为 MQuant 系统内部生成的订单 id，非委托编号，下单/撤单立即返回
self.price = 0.0 # 平均成交价格, 已经成交的股票的平均成交价格(一个订单可能分多次成交)
self.avg_cost = 0.0 # 卖出时表示下卖单前的此股票的持仓成本, 用来计算此次卖出的收益.
# 买入时表示此次买入的均价(等同于 price).
self.side = '' # 买卖方向，多(买)/空（卖），OrderSide 类型，下单立即返回
self.action = OrderAction.UNKNOWN # 开平行为，OrderAction 类型，期货有效
self.invest_type = HedgeFlag.UNKNOWN # 投资类型，HedgeFlag 类型，报单立即返回，期货有效
self.close_direction = CloseDirection.DEFAULT # 平仓类型，CloseDirection 类型，报单立即返回，期货有效
self.batch_no = -1 # 批次号
self.orig_order_id = -1 # 原始订单号，撤单订单有效，目前保留字段
self.symbol = '' # 证券代码，MQuant 格式
self.entrust_price = 0 # 委托价格，下单立即返回
self.style = None # 订单类型，下单立即返回
self.cancel_info = '' # 废单原因，废单有效
self.withdraw_amount = 0 # 撤单数量
self.business_balance = 0 # 已成交金额
self.clear_balance = 0 # 成交费用，目前保留字段
self.create_time = datetime.datetime.now()
self.entrust_prop = '0' #委托类型，在 EntrustProp 中定义
# "0"# 买卖
# "N"# ETF 申赎
# "f": // /债券质押
self.entrust_type = EntrustType.entrust # 委托类型，包含在订单执行回报中，在 EntrustType 中定义
self.algo_inst_id = None #算法实例 ID，由 MQuant 启动的算法实例相关订单中有此字段，包含在订单执行回报中
self.error_code = None #错误码
self.inst_id = None #MQuant 实例 ID，仅查询订单返回
self.trader_name = None #交易员名
self.entrust_no = '' #柜台委托编号，2019.11.19 添加
self.covered_flag = OptionCoveredFlag.UNKNOWN #期权备兑标志，2019.11.19 添加，期权有效
self.last_amount = 0 # 本次成交数量，仅推送有效，2020.3.2 添加

```
        self.last_price = 0.0   # 本地成交价格，仅推送有效，2020.3.2 添加
        self.fund_account = ''   # 资金账号，2020.05.24 新增
        self.security_exchange = SecurityExchangeType.UNKNOWN   # 市场，2020.10.26 新增，港股通
```

7.4.1.9 成交数据结构

```
class Trade(object):
    """
    成交订单对象
    """

    def __init__(self):
        self.time = None   # 成交时间
        self.amount = 0   # 成交数量
        self.price = 0.0   # 成交价格
        self.trade_id = ''   # 成交编号
        self.order_id = ''   # 订单 id
        """以下为 MQuant 新增字段"""
        self.security = ''   # 股票代码，聚宽格式，下单立即返回
        self.symbol = ''   # 证券代码，MQuant 格式
        self.real_type = 0   # 成交类型，0：买卖，2：撤单
        self.business_balance = 0   # 成交金额
        self.cost_balance = 0   # 成交费用，目前保留字段
        self.orig_order_id = -1   # 原始订单号，撤单订单有效，目前保留字段
        self.side = ''   # 买/卖方向,OrderSide 类型
        self.algo_inst_id = None   # 算法实例 ID，由 MQuant 启动的算法实例相关订单中有此字段，包含在订单执行回报中
        self.inst_id = None   # Mquant 实例 ID，仅查询成交返回
        self.trader_name = None   # 交易员
        self.action = OrderAction.UNKNOWN   # 开平行为，OrderAction 类型，期货有效
        self.invest_type = HedgeFlag.UNKNOWN   # 投资类型，HedgeFlag 类型，报单立即返回，期货有效
        self.close_direction = CloseDirection.DEFAULT   # 平仓类型，CloseDirection 类型，报单立即返回，期货有效
        self.entrust_no = ''            #柜台委托编号，  2019.11.19 新增
        self.style = None     #订单类型
        self.fund_account = ''   # 资金账号，2020.05.24 新增
        self.security_exchange = SecurityExchangeType.UNKNOWN   # 市场，2020.10.26 新增，港股通
        self.entrust_prop = EntrustProp.trade   # 委托属性，包含在订单执行回报中
        self.entrust_type = EntrustType.entrust
```

7.4.1.10 委托属性

```
class EntrustProp(object):
    """
    委托类型
    """
    trade = '0'   # 买卖
    redemption = 'N'   # ETF 申赎
    pledge = 'f'   # 债券质押
    covered_transfer = '9'   #备兑划转
    option_exercise = 'a'   #期权行权
```

7.4.1.11 委托类型

```
class EntrustType(object):
    """
    委托类型，成交类型
    """
    entrust = '0'   # 委托
    cancelOrder = '2'   # 撤单
    creditFinancing = '6'   # 信用融资       卖券还款、融资买入
    creditMargin = '7'   # 信用融券
    creditOpen = '8'   # 信用平仓#
    creditTransactions = '9'   # 信用交易
```

7.4.1.12 批量报单订单结构

```
class batch_order_item(object):
    """
    批次下单的单个订单对象
    批次下单允许一个批次里面存在不同订单类型，不同买卖方向的订单
    """

    def __init__(self):
        """
        :param security:  标的代码,支持 MQuant 代码格式和聚宽代码格式
        :param amount:交易数量，正数表示买入，负数表示卖出
        :param style:参见 order styles, None 代表 MarketOrder,
                        MQuant 在聚宽基础上扩展了市价单类型，默认为最优五档即时成交剩余撤销(沪市深市均可)，您也可选择其他类型
        :param side:long'/'short'，开空单还是多单。默认为多单，股票、基金暂不支持开空单。，或者 OrderSide 类型
```

```
        :param action: 开平标志，OrderAction 类型，默认为开仓
        :param invest_type: 投资类型，投机、套保、套利
        :param close_direction: 平仓类型
        :param pindex:在使用 set_subportfolios 创建了多个仓位时，指定 subportfolio 的序号，从 0 开始，比
    如 0 指定第一个 subportfolio,1 指定第二个 subportfolio，默认为 0，目前 MQuant 只支持默认值 0
        :return:Order 对象或者 None，如果创建订单成功，则返回 Order 对象，失败则返回 None
        """
        self.security = ''
        self.amount = 0
        self.style = None
        self.side = 'long'
        self.action = OrderAction.UNKNOWN
        self.invest_type = HedgeFlag.SPECULATION
        self.close_direction = CloseDirection.DEFAULT
        self.pindex = 0
```

7.4.1.13 批量撤单订单结构

```
class batch_cancel_order_item(object):
    """
    批量撤单的单个订单信息
    """

    def __init__(self):
        """
        :param order_id 原始订单的 order_id,batch_flag=0 有效
        :param batch_no 批次号，batch_flag=1 有效
        :param batch_flag 0 表示按照 order_id 撤单，1 表示按照批次号撤单，默认按照 order_id 撤单
        """
        self.order_id = ''
        self.batch_no = ''
        self.batch_flag = 0
```

7.4.1.14 账户类型

```
class AccountType(object):
    """
    账号属性，分为 A 股和信用
    """
    normal = 'stock'    # A 股账号
    margin = 'stock_margin'    # 信用账号
    futures = 'futures'    # 期货，暂不支持
```

```
    index_futures = 'index_futures'   # 指数期货，暂不支持
    open_fund = 'open_fund'   # 场外基金，暂不支持
    option = 'option'            #期权
    unknown = 'unknown'   # 未知类型
```

7.4.1.15 持仓数据结构

```
class Position(object):
    """
    单只标的的持仓信息
    """

    def __init__(self):
        super(Position, self).__init__()
        self.security = ''   # 标的代码
        self.price = 0   # 最新行情价格
        self.avg_cost = 0   # """ 该字段目前保留
        # 开仓均价，买入标的的加权平均价, 计算方法是:
        # (buy_volume1 * buy_price1 + buy_volume2 * buy_price2 + ...) / (buy_volume1 + buy_volume2 + ...)
        # 每次买入后会调整 avg_cost, 卖出时 avg_cost 不变. 这个值也会被用来计算浮动盈亏.
        # """
        self.hold_cost = 0   # 持仓成本
        self.init_time = 0   # 建仓时间，格式为 datetime.datetime,该字段目前保留
        self.transact_time = 0   # 最后交易时间，格式为 datetime.datetime,该字段目前保留
        self.total_amount = 0   # 总仓位
        self.closeable_amount = 0   # 可卖出的仓位
        self.today_amount = 0   # 今天开的仓位
        self.locked_amount = 0   # 冻结仓位
        self.value = 0   # 标的价值，计算方法是: price * (total_amount + locked_amount) ，仅支持 A 股
        self.side = 0   # OrderSide.UNKNOWN   # 废弃
        self.redemption_num = 0   # 可申赎数
        self.pindex = 0   # 仓位索引，subportfolio index
        self.position_prop = PositionProp.LONG   # 仓位类型，股、债、基为多仓，期货分多仓和空仓
        ###以下字段期货专用
        self.open_cost = 0   # 开仓均价，股票和 hold_cost 相同，期货=开仓成本/（总持仓*合约乘数）
        self.today_amount = 0   # 今天开的仓位
        self.old_amount = 0   # 昨持仓
        self.occupy_margin = 0   # 占用保证金
        # 开仓：交易前已占用保证金+开仓价*开仓数量*合约乘数*保证金比例；
        # 平仓：交易前已占用保证金—开仓价*平今仓数量*合约乘数*保证金比例—昨结算价*平昨仓数量*合约乘数*保证金比例
        self.close_pos_profit = 0   # 平仓盈亏（盯市），只有平仓的合约才会计入平仓盈亏,老仓采用昨结算价计算
```

```
        self.close_profit_by_trade = 0  # 平仓盈亏（逐笔）,只有平仓的合约才会计入平仓盈亏,老仓采用开仓价计算
        self.commission = 0  # 手续费
        self.contract_multiplier = 0  # 合约乘数
        self.hedge_flag = HedgeFlag.UNKNOWN
        self.option_hold_type = OptionHoldType.DEFAULT              #期权持仓类型
        self.fund_account = ''  # 资金账号
        self.stock_account = ''  # 股东账号，可用于区分多股东账号、沪港通、深港通场景
        self.security_exchange = SecurityExchangeType.UNKNOWN  # 市场，沪港通、深港通可通过市场区分
```

7.4.1.16 全局上下文 context

```
class Context(object):
    def __init__(self):
        super(Context, self).__init__()
        self.subportfolios = []
        self.portfolio = None  # 当前账户信息，单个操作仓位时，portfolio 指向 subportfolios[0]，多账户时通过设置指向不同账户，参数类型为 Portfolio 类型
        self.current_dt = None  # 当前单位时间的开始时间, datetime.datetime 对象，暂不可用
        self.previous_date = None  # 前一个交易日, datetime.date 对象，暂不可用
        self.universe = []  # 查询 set_universe()设定的股票池,默认为空的 list，如果用户调用 set_universe 设置，则立即进行同步
        self.run_params = {}  # 表示此次运行的参数，字典类型，在 MQuant 中为提交策略时的参数字典
        self.cur_account_type = None  # 当前账户类型，在 AccountType 中定义
```

7.4.1.17 仓位信息

```
class Portfolio(object):
    """
    仓位信息
    """

    def __init__(self):
        super(Portfolio, self).__init__()
        self.inout_cash = 0  # 累计出入金，当天银证转入/转出导致变化，证券买卖不会引发变化
        self.available_cash = 0  # 可用资金，可用来购买证券的资金
        self.transferable_cash = 0  # 可取资金，即可以提现的资金，不包括今日卖出证券所得资金，暂时不可用
        self.locked_cash = 0  # 挂单锁住资金，暂时不可用
        self.type = AccountType.normal  # 账户所属类型
        self.margin = 0  # 保证金，股票、基金保证金都为 100%
        self.positions = {}  # 等同于 long_positions
```

```
self.long_positions = {}  # 多单的仓位, dict<symbol, Position>
self.short_positions = {}  # 空单的仓位, dict<symbol, Position>,暂不可用
self.total_value = 0  # 总资产, 包括现金, 保证金, 仓位的总价值, 可用来计算收益
self.returns = 0  # 总权益的累计收益,暂时不可用
self.starting_cash = 0  # 初始资金, 暂时不可用
self.total_liability = 0  #: 总负债, 等于融资负债、融券负债、利息总负债的总和,暂时不可用
self.positions_value = 0  # 持仓价值, 股票基金才有持仓价值, 期货为 0，暂时不可用
self.locked_cash_by_purchase = 0  # 基金申购未完成所冻结的金额,暂时不可用
self.locked_cash_by_redeem = 0  # 基金赎回未到账的金额,暂时不可用
self.locked_amount_by_redeem = 0  # 基金赎回时，冻结的份额,暂时不可用
self.net_value = 0  #: 净资产, 等于总资产减去总负债,暂时不可用
self.cash_liability = 0  #: 融资负债,暂时不可用
self.sec_liability = 0  #: 融券负债,暂时不可用
self.interest = 0  #: 利息总负债,暂时不可用
self.maintenance_margin_rate = 0  #: 维持担保比例,暂时不可用
self.available_margin = 0  #: 融资融券可用保证金,暂时不可用
self.frozen_cash = 0  # 新增字段，包含所有冻结的资金
self.total_cash = 0  # 新增字段，资金总量，暂时不可用
self.settled_cash = 0  # 新增字段，期初资金
# 以下字段为期货专用
self.available_margin = 0  #: 可用保证金,期货可用
self.occupied_margin = 0  # 占用保证金，期货可用
self.pre_balance = 0  # 期初权益，期货可用
self.current_balance = 0  # 客户权益，期货可用
self.risk_level = 0  # 占用保证金/客户权益
self.holding_profit = 0  # 盯市盈亏，以持仓成本计算的浮动盈亏
self.close_pos_profit = 0  # 平仓盈亏（盯市），只有平仓的合约才会计入平仓盈亏,老仓采用昨结算价计算
self.close_profit_by_trade = 0  # 平仓盈亏（逐笔）,只有平仓的合约才会计入平仓盈亏,老仓采用开仓价计算
self.commission = 0  # 手续费
self.frozen_commission = 0  # 冻结手续费
self.freezed_deposit = 0  # 委托冻结保证金
```

7.4.1.18 证券市场

```
class ExchangeType(object):
    """
    交易所定义
    """
    SH = 'SH'  # 上海

    SZ = 'SZ'  # 深圳
```

```
    SHF = 'SHF'   # 上期

    CFFEX = 'CF'   # 中金

    ZCE = 'ZCE'   # 郑商

    DCE = 'DCE'   # 大商

    HK = 'HK'   # 港交所

    SI = 'SI'   # 申万
```

7.4.1.19 证券类型

```
class SecurityType(object):
    """
    标的类型定义
    """
    IndexType = "1"   # 指数

    StockType = "2"   # 股票

    FundType = "3"   # 基金

    BondType = "4"   # 债券

    RepoType = "5"   # 回购

    WarrantType = "6"   # 权证，暂不支持

    OptionType = "7"   # 期权，暂不支持

    FuturesType = "8"   # 期货，暂不支持

    ForexType = "9"   # 外汇，暂不支持

    RateType = "10"   # 利率，暂不支持

    NmetalType = "11"   # 贵金属，暂不支持
```

7.4.1.20 证券子类型

```
class SecuritySubType(object):
```

"""
标的子类型
"""
AsiaIndex = "01002" # 亚洲指数

InternationalIndex = "01003" # 国际指数

Systemclassificationindex = "01004" # 系统分类指数

Userclassificationindex = "01005" # 用户分类指数

Futuresindex = "01006" # 期货指数

Indexspot = "01007" # 指数现货

Ashares = "02001" # A 股（主板）

ScienTechInnovateBoard = '02200' #科创板

ScienTechInnovateBoard = '02200' #科创板

Smallandmediumstock = "02002" # 中小板股

Gemstock = "02003" # 创业板股

Strategicemergingboard = "02006" # 战略新兴板

Newthreeboard = "02007" # 新三板

MainboardofHongKongshares = "02008" # 港股主板

HongKongEquitygem = "02009" # 港股创业板

HongkonglistedNASDstock = "02010" # 香港上市 NASD 股票

Hongkongextendedplatestock = "02011" # 香港扩展板块股票

Preferredstock = "02100" # 优先股

Fund = "03001" # 基金（封闭式）

ListedopenfundLOF = "03003" # 上市开放基金 LOF

TradingopenindexfundETF = "03004" # 交易型开放式指数基金 ETF

Classificationsubfund = "03005" # 分级子基金

Extendedplatefund = "03006" # 扩展板块基金(港)

Redemptionfundonly = "03007" # 仅申赎基金

Governmentbonds = "04001" # 政府债券（国债）

Corporatebond = "04002" # 企业债券

Financialbond = "04003" # 金融债券

Corporatedebt = "04004" # 公司债

Convertiblebond = "04005" # 可转债券

Privatedebt = "04006" # 私募债

Exchangeableprivatedebt = "04007" # 可交换私募债

Securitiescompanysubordinateddebt = "04008" # 证券公司次级债

Securitiescompanyshorttermdebt = "04009" # 证券公司短期债

Exchangeablecorporatedebt = "04010" # 可交换公司债

Bondpreissue = "04011" # 债券预发行

Pledgetypetreasurybondrepurchase = "05001" # 质押式国债回购

Thecorporatebondpledgedrepo = "05002" # 质押式企债回购

Buybackofbuyoutbond = "05003" # 买断式债券回购

Bidrepurchase = "05004" # 报价回购

Stockoption = "07001" # 个股期权

ETFoption = "07002" # ETF 期权

Indexfutures = "08001" # 指数期货

Commodityfutures = "08002" # 商品期货

```
    Stockfutures = "08003"   # 股票期货

    Bondfutures = "08004"   # 债券期货

    Interbankinterestratefutures = "08005"   # 同业拆借利率期货

    ExchangeFundNoteFuturesExchangeFundpaperfutures = "08006"   # 外汇基金票据期货

    ExchangeForPhysicalsfuturesturntospot = "08007"   # 期货转现货
```

7.4.1.21 标的 ST 状态

```
class STStatus(object):
    """
    股票的 ST 标识
    """
    notst = 0           #非 ST 标的
    special = 1         #ST
    serious = 2         #*ST
```

7.4.1.22 标的状态

```
class TradingPhaseCode(object):
    StartBeforeOpen = "0"# 开盘前，启动
    OpenAggregateAuction = "1"# 开盘集合竞价
    AfterAggregateAuction = "2"# 开盘集合竞价阶段结束到连续竞价阶段开始之前
    ContinuousAuction = "3"# 连续竞价
    CloseAtNoon = "4"# 中午休市
    CloseAggregateAuction = "5"# 收盘集合竞价
    Closed = "6"# 已闭市
    PostTrading = "7"# 盘后交易
    TemporarilySuspended = "8"# 临时停牌
    VolatilityInterrupted = "9"# 波动性中断
```

7.4.1.23 证券详情

```
class SecurityDetial(object):
    """
    证券详情
    """
    def __init__(self):
       self.symbol = ''
```

```
self.display_name = ''   #中文名
self.spell_name = ''   # 拼音简称
self.start_date = None   #上市日期, [datetime.date] 类型，暂不提供
self.end_date = None      #退市日期， [datetime.date] 类型, 如果没有退市则为 2200-01-01，暂不提供
self.security_type = None            #股票、基金、金融期货、期货、债券基金、股票基金、QDII 基金、货币基金、混合基金、场外基金，SecurityType 中定义
self.security_sub_type = None     #子类型，在 SecuritySubType 中定义
self.STFlag = STStatus.notst           #ST 标志，在 STStatus 中定义，暂不提供
self.TradingPhaseCode = 0                  #股票状态，在 TradingPhaseCode 中定义
self.RoundLot = 100                   #单手股数
self.LocalTotalShare = 0            #本市总股本
self.LocalListedShare = 0           #本市流通股本
self.TickSize = 0                     #价格精度，暂不可用
self.HighLimitPrice = 0 #涨停价
self.LowLimitPrice = 0   #跌停价
self.PreClosePrice = 0   #昨收盘价
self.MarginTradingFlag = 0   #融资融券标识，1 表示支持融资融券，0 表示不支持融资融券，待确认
#以下字段期权专用
self.StrikePrice = 0      #行权价格，期权有效
self.StrikeDate = None   #行权日期， 期权有效，yyyyMMdd 格式
self.PutOrCall = ''        #认购('C')/认沽('')
self.UnderlyingSymbol = ''   #标的证券代码
# 以下字段可转债专用
self.ConvertSymbol = ''   # 转股代码
self.ConvertPrice = 0.0   # 转股价格
self.ConvertStartDate = ''   # 转股开始日期,yyyyMMdd
self.ConvertEndDate = ''   # 转股结束日期,yyyyMMdd
```

7.4.1.24 逐笔委托数据结构

```
class RecordOrder(object):
    def __init__(self):
        self.code = ''   # 标的,MQuant 格式
        self.datetime = None   # 行情时间
        self.side = OrderSide.BUY   # 买卖方向，开空单还是多单。默认为多单，股票、基金暂不支持开空单。OrderSide 类型
        self.order_type = -1   # -1 无效，1 市价 2 限价 3 本方最优
        self.price = 0.0   # 委托价格
        self.amount = 0   # 委托数量
```

```
        self.order_index = 0    # 委托编号
```

7.4.1.25 逐笔成交数据结构

```
class RecordTransaction(object):
    def __init__(self):
        self.code = ''    # 标的,MQuant 格式
        self.datetime = None    # 行情时间
        self.trade_type = 0    # 成交类别，
        # 0       交易业务成交记录
        # 1  交易业务撤单回报记录
        # 2       即时成交剩余撤销委托”未能成交部分或其他原因的自动撤单回报记录
        # 3 ETF 基金申购/赎回成功回报记录或 ETF 基金申购/赎回成功证券给付明细回报记录  4
ETF 基金申购/赎回撤单报记录
        # 5  最优五档即时成交剩余撤销委托”未能成交部分的自动撤单或其他原因的自动撤单回报
记录
        # 6  全额成交或撤销委托”未能成交时的自动撤单或其他原因自动撤单回报记
        # 7  本方最优价格委托的撤单回报记录
        # 8       对手方最优价格委托的撤单回报记录
        # 9       ETF 基金申购/赎回成功允许/必须现金替代明细回报记录
        self.side = 0    # 0  成交方向不明    1    买方成交      2       卖方成交
        self.price = 0.0    # 成交价格
        self.amount = 0    # 成交数量
        self.business_balance = 0.0    # 成交金额
        self.trade_buy_no = 0    # 买方委托序号
        self.trade_sell_no = 0    # 卖方委托序号
```

7.4.1.26 ETF 预估信息

```
class EtfEstimateInfo(object):
    def __init__(self):
        self.code = ''    # 标的的代码，实时传入,MQuant 格式
        self.datetime = None    # ETF 时间，实时传入
        self.IOPV = 0    # 最新价计算的 IOPV
        self.DIOPVS1 = 0    # 卖一价计算的 IOPV
        self.DIOPVB1 = 0    # 买一价计算的 IOPV
        self.premium = 0    # 溢价套利预估利润
        self.discount = 0    # 折价套利预估利润
```

7.4.1.27 CSV 文件读取类

```
class CsvReader(metaclass=ABCMeta):
    """
    csv 文件读取类
    """

    def __init__(self, file_path, encodeing='utf-8'):
        """
        :param file_path:
        :param encodeing:
        """
        super(CsvReader, self).__init__()
        self.reader = CsvReaderImpl(file_path, encodeing)

    # 支持 iterable
    def __iter__(self):
        return self.reader

    def __del__(self):
        if self.reader is not None:
            self.reader.__del__()
            self.reader = None

    # 获取一行数据，返回 list
    # 默认获取下一行数据（rowNum=0）
    # 可以获取指定行数据，但是效率较低
    # 建议通过 for row in CsvReader:的方式获取数据
    def getRow(self, rowNum=0):
        if self.reader is None:
            return []
        else:
            self.reset()
            return self.reader.getRow(rowNum)

    # 关闭 csv 文件读取对象，可以不调用，但是如果需要在同一作用域内打开多个 csv 文件，建议读取完成就调用关闭函数释放资源
    def close(self):
        self.__del__()

    def reset(self):
        if self.reader is not None:
            return self.reader.reset()
```

7.4.1.28 sqlite 内存数据库连接类

```
class HtDbConnection(metaclass=ABCMeta):
    # __metaclass__ = ABCMeta
    """docstring for ClassName"""

    # def __init__(self):
    # super(HtSqliteConnection, self).__init__()

    ###执行 sql 语句，支持使用?作为替代符，替代的参数依次放入 parameters 列表中
    @abstractmethod
    def excutesql(self, sql, parameters=[]):
        # print(sql)
        pass

    # raise AttributeError('子类必须实现')

    ###开始事务
    @abstractmethod
    def transaction(self):
        pass

    ###提交事务
    @abstractmethod
    def commit(self):
        pass

    ###回滚到上次提交之前
    @abstractmethod
    def rollback(self):
        pass

    ###获取查询结果
    @abstractmethod
    def fetchall():
        pass

    @abstractmethod
    def fetchone():
        pass

    @abstractmethod
    def fetchmany(self, num):
        pass
```

7.4.1.29 快捷键列表

Down: 光标移动到下一行
Down+Shift: 光标移动到下一行并选中经过区域
Down+Ctrl: 编辑视图向下滚动一行，如果当前光标所在行不可见，则将光标移动到可见的首行
Up: 光标移动到上一行
Up+Shift: 标移动到上一行并选中经过区域
Up+Ctrl: 编辑视图向上滚动一行，如果当前光标所在行不可见，则将光标移动到可见的最后一行
[+Ctrl: 段落向上移动
]+Ctrl: 段落向下移动
Left: 光标左移
Left+Shift: 光标左移并选中经过区域
Left+Ctrl: 光标以单词为单位左移
Left+Shift+Ctrl: 光标以单词为单位左移并选中经过区域
Left+Alt+Shift: 光标左移，支持选中多行的同一位置
Right: 光标右移
Right+Shift: 光标右移并选中经过区域
Right+Ctrl: 光标以单词为单位右移
Right+Shift+Ctrl: 光标以单词为单位右移并选中经过区域
Right+Alt+Shift: 光标右移，支持选中多行的同一位置
/+Ctrl: 注释/解注释切换
Home: 光标移动到行首有效字符位置（不包含缩进的空格）
Home+Shift: 光标移动到行首有效字符位置并选中经过的区域
Ctrl+Home: 光标移动到文档开始处
Home+Alt: 光标移动到行首
Home+Alt+Shift: 光标移动到行首并选中经过的区域
End: 光标移动到行尾
End+Shift: 光标移动到行尾并选中经过的区域
Ctrl+End: 光标移动到文档结束处
Ctrl+Shift+End: 光标移动到文档结尾并选中经过的区域
End+Alt: 光标移动到行尾
End+Alt+Shift: 光标移动到行尾并选中经过的区域
PageUp: 向上翻页
Shift+PageUp: 向上翻页并选中经过的区域
PageDown: 向下翻页
Shift+PageDown: 向下翻页并选中经过的区域

Delete: 删除光标后的字符

Delete+Shift: 剪切（同 Ctrl+X）

Ctrl+Delete: 删除光标后的单词

Ctrl+Shift+BackSpace: 删除当前行光标之前的所有内容

Insert: 切换编辑和重写模式

Insert+Shift: 粘贴（同 Ctrl+V）

Insert+Ctrl: 复制（同 Ctrl+S）

Escape: 取消

Backspace: 删除光标之前的字符

Ctrl+BackSpace: 删除光标之前的单词

Backspace+Alt: 撤销编辑（同 Ctrl+Z）

Ctrl+Z: 撤销编辑

Ctrl+Y: 恢复上一次删除/撤销的内容

Ctrl+X: 剪切

Ctrl+C: 复制

Ctrl+V: 粘贴

Ctrl+A: 全选

Tab: 缩进

Shift+Tab: 取消缩进

Add+Ctrl: 字号放大（同 Ctrl+鼠标滚轮向上滚动）

Ctrl+鼠标滚轮 Up： 字号放大

Ctrl+鼠标滚轮 Down： 字号缩小

Ctrl+L: 剪切当前行

Ctrl+Shift+L: 删除当前行

Ctrl+Shift+T: 复制当前行

Ctrl+T: 当前行与上一行交换位置

Ctrl+D: 复制当前行并粘贴到下一行

Ctrl+S: 保存文档

Ctrl+F: 查找

Ctrl+R: 替换

7.4.1.30 头寸类型

```
class PositionType(Enum):
    """
    头寸类型
    """
    normal = 1   # 普通头寸
vip = 2   # 专项头寸
```

7.4.1.31 信用交易合约

```
class MarginContract(object):
    """
    信用交易合约,包含融资负债和融券负债
    """

    def __init__(self, contract_type=MarginContractType.unknown):
        self.contract_type = contract_type   # 负债类型
        self.open_date = None   # 开仓日期，datetime.date 类型
        self.symbol = ''   # 标的
        self.position_type = PositionType.normal   # 头寸类型（融券合约专用）
        self.unpaid_amount = 0   # 未还金额
        self.unpaid_qty = 0   # 未还数量
        self.interest = 0   # 利息
        self.cost = 0   # 费用
        self.deadline = None   # 归还截止日期,datetime.date 类型
        self.contract_no = ''   # 合同编号
```

7.4.1.32 信用资产信息

```
class MarginAssert(object):
    """
    信用资产信息, 后台系统定时从柜台同步，不保证实时性
    """

    def __init__(self):
        self.cash_asset = 0   # 现金资产
        self.security_market_value = 0   # 证券市值
        self.assure_asset = 0   # 担保资产
```

```
        self.total_liability = 0   # 总负债
        self.maintain_value = 0   # 个人维持担保比例
        self.available_margin = 0   # 可用保证金
        self.occupy_margin = 0   # 占用保证金
        self.collateral_available_money = 0   # 买担保品可用资金
        self.finance_available_money = 0   # 买融资标的可用资金
        self.security_available_fund = 0   # 买券还券可用资金
        self.cash_available_money = 0   # 现金还款可用资金
        self.finance_quota_capacity = 0   # 融资额度上限
        self.finance_available_quota = 0   # 融资可用额度
        self.finance_occupy_quota = 0   # 融资占用额度
        self.finance_occupy_margin = 0   # 融资占用保证金
        self.finance_compact_quota = 0   # 融资合约金额
        self.finance_compact_commission = 0   # 融资合约费用
        self.finance_compact_interest = 0   # 融资合约利息
        self.finance_market_value = 0   # 融资市值,暂不提供
        self.finance_compact_revenue = 0   # 融资合约收益
        self.security_loan_quota_capacity = 0   # 融券额度上限
        self.security_loan_available_quota = 0   # 融券可用额度
        self.security_loan_occupy_quota = 0   # 融券占用额度
        self.security_loan_occupy_margin = 0   # 融券占用保证金
        self.security_loan_compact_quota = 0   # 融券合约金额
        self.security_loan_compact_commission = 0   # 融券合约费用
        self.security_loan_compact_interest = 0   # 融券合约利息
        self.security_loan_market_value = 0   # 融券市值，暂不提供
        self.security_loan_compact_revenue = 0   # 融券合约收益
        self.transfer_asset = 0   # 可转出资产
        self.compact_total_interest = 0   # 合约总利息
        self.net_asset = 0   # 净资产
        self.withdraw_quota = 0   # 可取金额
        # self.other_fare = 0                                #其他费用
        self.security_loan_sell_balance = 0   # 融资卖出所得金
```

7.4.1.33 ETF 成分券信息

```
class HtEtfConstituentInfo:
    """
    ETF 成分券信息
```

```
    """

    def __init__(self):
        self.symbol = ''   # 成分券代码，MQuant 格式
        self.sample_size = 0   # 样本数量
        self.cash_replace_flag = 0   # 现金替代标志，
        # 0 禁止替代
        # 1 允许替代
        # 2 必须替代
        # 3 非沪市退补现金替代
        # 4 非沪市必须现金替代
        # 5 非沪深退补现金替代
        # 6 非沪深必须现金替代
        self.deposit_ratio = 0   # 保证金率（ 溢价比率），允许现金替代标的此字段有效
        self.replace_balance = 0   # 替代金额,必须现金替代标的此字段有效
        self.pre_close_px = 0   # 昨收价格
        self.high_limit_px = 0   # 涨停
        self.low_limit_px = 0   # 跌停价
        self.suspend_flag = False   # 停牌标志
```

7.4.1.34 ETF 信息

```
class HtEtfInfo(object):
    """
    ETF 信息
    """

    def __init__(self):
        super(HtEtfInfo, self).__init__()
        self.etf_fund_symbol = ''   # etf 基金代码
        self.etf_symbol = ''   # etf 申赎代码
        self.report_unit = 0   # 一个申赎单位的基金份额
        self.cash_balance = 0   # 预估现金差额
        self.max_cash_ratio = 0   # 最大现金替代比例
        self.pre_cash_componet = 0   # T-1 日申购基准单位现金余额
        self.nav_percu = 0   # T-1 日申购基准单位净值
        self.nav_pre = 0   # T-1 日基金单位净值
```

```
        self.allot_max = 0    #  申购份额上限
        self.redeem_max = 0    #  赎回份额上限
        self.etf_type = EtfType.unknown    # ETF 类型
        self.etf_status = EtfStatus.unknown    # ETF 基金状态
```

7.4.1.35 算法类型

```
class AlgoType(Enum):
    """
    算法类型
    """
    """
    AIVWAP = 6    # AIVWAP 算法
    """
    基于深度学习的 VWAP 执行算法
    """
    AIVWAP = 6    # AIVWAP 算法
    """
    基于深度学习的智能执行算法 AIVWAP
    """
    SMARTTWAP = 15
    """
    从 AES 迁移的 TWAP 算法
    """


    UNKNOWN = 64
```

7.4.1.36 算法委托属性

```
class AlgoEntrustProp(object):
    """
    算法委托属性
    """
    normal = '0'    #  普通 A 股交易
    credit = '9'    #  信用交易
    credit_financing = '6'    #  融资买入
```

7.4.1.37 算法参数定义

```
class AlgoParamKeys(object):
    """
    算法参数定义，填写参数时请务必使用类中定义的参数，否则不保证参数证券
    特别注意：参数中所有 key 和 value 都必须是字符串，不允许数值型
    """
    """
    以下参数适用于所有算法
    """
    symbol = '证券代码'  # 标的代码，必填
    amount = '委托数量'  # 报单数量，只支持正整数，必填
    limit_price = '限价'  # 限价，非必填，不填写默认不限价
    remark = '备注'  # 备注
    """
    以下参数只适用于非 AI 的部分算法，具体请参考 MATIC 的算法交易菜单下的各算法参数
    """
    withdraw_interval = '撤单时间'  # 撤单时间，非必填，不填写使用算法默认值
    min_qty = '最小委托单位'  # 最小委托单位，单位为股（债券为张），非必填，不填写使用算法默认值
    price_level = '委托档位'  # 委托价格档位，非必填，不填写使用算法默认值，委托档位默认 0 当前价格，
    # 1-5 买入对应卖出价 1-5 档卖出对应买入价 1-5 档，买入对应涨停价卖出对应跌停价档位无价格的用涨跌停价
    withdraw_mode = '撤单模式'  # 撤单模式，非必填，不填写使用算法默认值，撤单模式 0 为普通， 1 为自动，默认 0
    fliter_limit_price = '是否过滤涨跌停'  # 是否过滤涨跌停，是否过滤涨跌停 0 为不过滤， 1 为过滤，默认 0。
    max_single_amount = '单笔最大'  # 单笔最大
    min_single_amount = '单笔最小'  # 单笔最小
    scavenging_ratio = '扫盘比例'  # 扫盘比例
    order_interval = '委托间隔'  # 委托间隔时间，单位为秒
    deal_participation_ratio = '成交参与比例'  # 成交参与比例
    single_max_order = '单笔最大模式'  # 单笔最大模式 0 普通， 1 增强，普通模式按照单笔最大拆分订单，
    # 增强模式强制使用单笔最大为当前委托数不拆单，该模式会减少当前委托数量。默认 0
    strategy_style = '策略风格'  # 策略的交易风格默认平衡型保守型委托价格偏向成本低的一方，激进型委托价格可能会高于市场平均，平衡型介于两者之间
    # 保守型 0，平衡型 1, 激进型 2
    market_participation_ratio = '市场占比'  # 策略运行区间的策略成交的市场占比阈值，大于该阈值暂停交易至占比低于该阈值恢复交易，设置为 0，则不进行该阈值的限制
```

```
    # 如 60% 应该填写 60
    entrust_prop = '委托属性'  # 委托属性，可以不填写，如果不填写，使用 A 股账号时默认为普通 AlgoEntrustProp.normal
    # 使用信用账号时默认为普通 AlgoEntrustProp.credit，融资买入时必须填写 AlgoEntrustProp.credit_financing
    max_order_qty = '封单数量'  #封单数量，某些单标的算法，如极速抢涨停，界面上的封单金额需要换算成封单数量传给算法平台

    """
    以下参数适用于 AI 算法
    """
    max_market_participation_ratio = '最大市场占比'  # 当前总委托量不大于当前市场总成交额*最大市场占比，如 60%应填写 0.6
buy_sell_progress_diff = '买卖进度差'      #abs(买金额进度-卖金额进度)<买卖进度差
order_side = '交易方向'                    #交易方向 key
order_side_buy = 'Buy'                     #买入
order_side_sell = 'Sell'                   #卖出
```

7.4.1.38 算法实例状态

```
class AlgoInstanceStatus(object):
    """
    算法实例状态
    """
    RUNNING = '0'  # 运行中
    PAUSE = '1'  # 暂停
    STOP = '2'  # 停止
    INIT = '3'  # 初始状态
    UNKNOWN = '4'  # 未知状态
    FINISH = '5'  # 已完成
```

7.4.1.39 标的的进度信息

```
class AlgoSymbolInfo(object):
    """
    标的的进度信息
    """
```

```
    def __init__(self):
        self.cum_qty = 0    # 累计成交总量
        self.order_qty = 0    # 委托总量
        self.avg_px = 0    # 成交均价
        self.cum_amount = 0    # 成交金额
        self.target_qty = 0    # 目标总量
        self.progress = 0    # 标的成交进度
        self.symbol = ''
```

7.4.1.40 算法实例信息

```
class AlgoInstanceInfo(object):
    """
    算法实例信息
    """

    def __init__(self):
        self.inst_id = ''    # 实例 ID
        self.status = AlgoInstanceStatus.UNKNOWN    # 实例状态
        self.symbol_info = []    # 标的的进度信息列表，元素类型为 AlgoSymbolInfo 对象
        self.progress = 0    # 实例成交进度
```

7.4.1.41 日志级别

```
class LOG_LEVEL(Enum):
    DEBUG = 0
    INFO = 1
    WARNING = 2
    ERROR = 3
    MAX_LEVEL = 4
```

7.4.1.42 头寸性质

```
class PositionType(Enum):
    头寸类型
    undefine = -1    # 未知
    normal = 1    # 普通头寸
    vip = 2    # 专项头寸
```

7.4.1.43 融券标的信息

```
class MarginSecurityInfo(object):
```

```
    """
    融券标的信息
    """
    def __init__(self):
        self.loan_available_qty = 0   # 融券可用额度
        self.loan_ratio = 0   # 融券保证金比例
        self.position_type = PositionType.undefine   # 头寸性质
        self.symbol = ''   # 标的
```

7.4.1.44 日期类型

```
class DateType(object):
    """
    日期类型
    """
    NORMAL_DATE = 0   # 自然日
    TRADE_DATE = 1   # 交易日
```

7.4.1.45 证券交易市场

```
class SecurityExchangeType(object):
    """
    证券交易市场
    """
    UNKNOWN = '-1'    #未知

    SH = '1'   ##上交所

    SZ = '2'   # 深交所

    TZA = '9'   # 特转 A

    TZB = 'A'   # 特转 B

    YHJ = 'B'   # 银行间

    JJ = 'C'   # 基金

    SHB = 'D'   # 上海 B

    HHK = 'G'   # 沪 HK（沪港通南向）

    SZB = 'H'   # 深圳 B
```

```
    HK = 'HK'    # 香港

    CFFEX = 'M'    # 中金所

    SHF = 'N'    # 上期所

    CZCE = 'O'    # 郑商所

    DCE = 'P'    # 大商所

    SZHK = 'S'    # 深 HK（深港通南向）

    SSC = 'SSC'    # 沪股通（沪港通北向）

    SZC = 'SZC'    # 深股通（深港通北向）
```

7.4.1.46 仓位属性

```
class PositionProp(object):
    """
    仓位属性
    """
    SHORT_FOR_SHORT = '0'     #备兑持仓
    LONG = '1'    # 多仓
    SHORT = '2'    # 空仓
    UNKNOWN = ''    # 未知
```

7.4.1.47 开平行为

```
class OrderAction(object):
    """
    订单行为
    """
    OPEN = '1'    # 开
    CLOSE = '2'    # 平
    EXERCISE = '3'    # 行权
    AUTO_EXERCISE = '4'    # 自动行权
    UNKNOWN = ''
```

7.4.1.48 期货投保标记

```
class HedgeFlag(object):
    """
```

```
    期货投保标记
    """
    SPECULATION = '1'   # 投机
    HEDGING = '2'   # 套保
    ARBITRAGE = '3'   # 套利
    UNKNOWN = ''   # 未知
```

7.4.1.49平仓类型

```
class CloseDirection(object):
    """
    平仓类型
    """
    DEFAULT = '0'   # 默认
    CLOSE_TODAY = '1'   # 平今仓
    CLOSE_OLD = '2'   # 平老仓
    FORCE_CLOSE = '3'   # 强平
    FORCE_OFF = '4'   # 强减
    LOCAL_FORCE_CLOSE = '5'   # 本地强平
```

7.4.1.50期货合约信息

```
class FutureContractInfo(object):
    """
    期货合约信息
    """
    def __init__(self):
        self.symbol=''
        self.price_step=0            #最小价格变动单位
        self.contract_multiplier = 0       #合约乘数
        self.start_deliv_date = 0     #开始交割日
        self.end_deliv_date = 0         #结束交割日
        self.long_margin_ratio=0       #多头保证金率（按金额）
        self.short_margin_ratio = 0 #空头保证金率（按金额）
        self.long_margin_ratio_amount=0       #多头保证金率（按手数）
        self.short_margin_ratio_amount = 0 #空头保证金率（按手数）
        self.margin_ratio_flag=1             #1 按金额     2 按手数
        self.market_max_buy_qty = 0        # 市价最大买入数量
        self.market_max_sell_qty = 0       # 市价最大卖出数量
        self.market_min_buy_qty = 0        # 市价最小买入数量
        self.market_min_sell_qty = 0       # 市价最小卖出数量
        self.max_buy_qty = 0                 # 限价最大买入数量
        self.max_sell_qty=0                  # 限价最大卖出数量
        self.min_buy_qty = 0                 # 限价最小买入数量
```

```
        self.min_sell_qty = 0                # 限价最小卖出数量
```

7.4.1.51期权备兑标志

```
class OptionCoveredFlag(object):
    """
    期权备兑信息
    """
    UNCOVERED = '1'   #非备兑
    COVERED = '2'      #备兑
    UNKNOWN = ''      #未知
```

7.4.1.52期权持仓类型

```
class OptionHoldType(object):
    """
    期权持仓类型
    """
    DEFAULT = '0'
    RIGHT = '1' #权利仓
    COMPULSORY = '2'     #义务仓
    COVERED = '3'    #备兑仓
```

7.4.1.53K 线数据类型

```
class KLineDataType(object):
    """
    k 线类型定义
    """
    KLINEData_1M = "kline_1m"   # 1 分钟 k 线
    KLINEData_5M = "kline_5m"   # 5 分钟线
    KLINEData_15M = "kline_15m"   # 15 分钟线
    KLINEData_30M = "kline_30m"   # 30 分钟线
    KLINEData_60M = "kline_60m"   # 60 分钟线
    KLINEData_120M = "kline_120m"   # 120 分钟线,暂不支持
    KLINEData_1D = "kline_1d"   # 日 k 线
    KLINEData_1W = "kline_1w"   # 周线,暂不支持
    KLINEData_15D = "kline_15d"   # 15 日线，暂不支持
    KLINEData_1MON = "kline_1mon"   # 月线，暂不支持
    KLINEData_3MON = "kline_3mon"   # 3 月线，暂不支持
    KLINEData_4MON = "kline_4mon"   # 4 月线，暂不支持
    KLINEData_6MON = "kline_6mon"   # 半年线，暂不支持
    KLINEData_1Y = "kline_1y"   # 年线，暂不支持
```

7.4.1.54指数成分券

```
class IndexComponent(object):
    def __init__(self):
        self.symbol = ''
        self.weight = 0.0   # 权重
        self.include_date = 0   # 进入日期
        self.exclude_date = 0   # 退出日期
        self.status = 0   # 最新状态
```

7.4.1.55拆单算法参数

```
class SplitOrderAlgoParam(object):
    """
    拆单算法参数
    """
    def __init__(self):
        self.algo_type = AlgoType.UNKNOWN          # 算法类型，仅支持 AITWAP 和 AIVWAP 两种算法
        self.start_time = None                     # 开始时间,datetime.datetime 类型
        self.end_time = None                       # 结束时间，datetime.datetime 类型
        self.order_side = OrderSide.BUY            # 交易方向，默认为买入
        self.entrust_type = EntrustType.entrust # 委托类型，A 股账号该字段无效，会被强制填入 EntrustType.entrust，信用账号用户需要填入 EntrustType.creditTransactions 担保品买卖，EntrustType.creditFinancing 信用融资, EntrustType.creditMargin 信用融券
        self.max_market_ratio = 0.1                # 最大市场占比，默认 0.1，当前总委托量不大于市场总成交金额*最大市场占比
        self.call_auction_ratio = 0                # 参与集合竞价比例，0 表示不参与
        self.style = 1                             # 0 保守 1 平稳 2 激进，保守型以降低成本为目标，平稳型以降低成本、提高成交进度、降低撤单比为目标，激进型以提高成交进度、降低撤单比为目标，默认为平稳型
        self.forbidden_limit = False               # 是否涨停不卖跌停不买，默认为否
        self.call_auction_offset_ratio = 0   # 集合竞价价格偏离比例
        self.order_list=[]                         # 标的信息列表，列表元素为 AlgoOrderInfo 类型
        self.remark = ''                           # 备注


        # 以下字段仅 SMARTTWAP 支持
        self.min_order_qty = 100   # 单笔最小量
        self.max_order_qty = 1000000   # 单笔最大委托量
        self.withdraw_type = 0   # 0 普通 1 自动
        self.entrust_interval = 30   # 委托间隔（s）
        self.withdraw_interval = 12   # 撤单时间间隔（s）
```

self.entrust_price_level = 0 # 0 当前价格，1-5 买入对应卖出价 1-5 档，卖出对应买入价 1-5 档，买入对应涨停价卖出对应跌停价档位无价格的用涨跌停价

self.limit_price_entrust_forbidden = True # 该参数生效时，不允许以涨停价买入，不允许以跌停价卖出；该参数不生效时，不做涨跌停价格控制

7.4.1.56创建算法响应

```
class CreateAlgoInstanceRsp(object):
    """
    创建算法的响应
    """
    def __init__(self):
        self.inst_id = ''          # 实例 ID
        self.err_code = 0          # 错误码，0 表示成功，1 表示超时，-1 表示失败，超时时可以通过告警接口播放提示音，然后通过算法交易界面人工检查算法实例是否创建成功，或者联系 Matic 技术支持
        self.err_info = ''         # 错误信息
```

7.4.1.57报单请求信息

```
class OrderRequest(object):
    """
    报单请求
    """
    def __init__(self):
        self.symbol = ''   # 代码
        self.amount = 0    # 报单数量
        self.side = OrderSide.UNKNOWN   # 报单方向，OrderSide 中定义
        self.style = None   # 订单类型，限价单用 LimitOrderStyle，市价单用 MarketOrderStyle
        self.entrust_type = EntrustType.entrust   # 委托类型，默认为买卖委托，信用交易需要特别关注委托类型
        self.action = OrderAction.UNKNOWN   # 开平方向，期权、期货需要填写
        self.invest_type = HedgeFlag.UNKNOWN   # 投资类型，期货填写，包含投机、套保、套利
        self.close_direction = CloseDirection.DEFAULT   # 平仓类型，期货、期权平仓填写
        self.covered_flag = OptionCoveredFlag.UNKNOWN   # 备兑类型，期权备兑填写，其他不填
        self.position_type = PositionType.normal   # 头寸性质，融券卖出、买券还券、直接还券必填，其他不用填写，PositionType 命名空间中定义
        self.contract_no = ''   # 信用合约编号，信用交易归还合约时选填，不填按默认顺序归还
        self.channel_type = 1   # ETF 申赎通道标志，深市跨市 ETF 申赎有效，1：现金申赎，2：实物申赎，目前仅支持现金申赎，暂时保留该字段
        self.security_exchange = SecurityExchangeType.UNKNOWN   # 证券市场，港股通必须填写，其他报单类型无效
```

7.4.1.58 资金流向数据

```
class FundFlow(object):
    """
    资金流向数据
    """

    def __init__(self):
        self.symbol = ''
        self.datetime = None
        self.super_large_order = None  # 超大单 FundFlowDetial 对象
        self.large_order = None  # 大单 FundFlowDetial 对象
        self.middle_order = None  # 中单 FundFlowDetial 对象
        self.small_order = None  # 小单 FundFlowDetial 对象
        self.main_order = None  # 主力 FundFlowDetial 对象
```

7.4.1.59 资金流向详情

```
class FundFlowDetial(object):
    """
    资金流向详情
    """

    def __init__(self):
        self.in_flow_value = 0.0  # 流入金额
        self.in_flow_qty = 0  # 流入股数
        self.out_flow_value = 0.0  # 流出金额
        self.out_flow_qty = 0  # 流出股数
```

7.4.1.60 资金更新信息

```
class FundUpdateInfo(object):
    """
    资金推送信息
    """

    def __init__(self):
        self.available_cash = 0  # 可用资金
        self.frozen_cash = 0  # 冻结资金
        self.total_value = 0  # 总资产
```

```
        self.settled_cash = 0   # 期初资金
        self.hold_cash = 0   # 当前资金余额
        self.market_value = 0   # 证券市值
        self.transferable_cash = 0   # 可取资金
        self.hk_available_cash = 0   # 港股可用资金
        # 以下字段为期货专用
        self.available_margin = 0   # 保证金余额
        self.occupied_margin = 0   # 占用保证金
        self.pre_balance = 0   # 期初权益
        self.current_balance = 0   # 客户权益
        self.risk_level = 0   # 风险度
        self.close_pos_profit = 0   # 平仓盈亏（盯市）
        self.close_profit_by_trade = 0   # 平仓盈亏（逐笔）
        self.commission = 0   # 手续费
        self.frozen_commission = 0   # 冻结手续费
        self.freezed_deposit = 0   # 委托冻结保证金
        self.fund_account = ''   # 资金账号
```

7.4.1.61 ETF 状态

```
class EtfStatus(object):
    """
    ETF 基金状态
    """
    forbidden = "0"   # 全部禁止
    allow = "1"   # 全部允许
    onlyAllowApplyBuy = "2"   # 仅允许申购
    onlyAllowRedemption = "3"   # 仅允许赎回
    unknown = "-1"   # 未知
```

7.4.1.62 ETF 类型

```
class EtfType(object):
    """
    ETF 基金类型
    """
    singleMarket = "1"   # 单市场 ETF
    crossBorder = "2"   # 跨境 ETF
    crossMarket = "3"   # 跨市 ETF
    currency = "4"   # 货币 ETF
    bond = "5"   # 实物债券 ETF
    commodity = "6"   # 商品 ETF
```

```
    cashBond = "7"   # 现金债券 ETF
    unknown = "-1"   # 未知
```

7.4.1.63报撤单响应

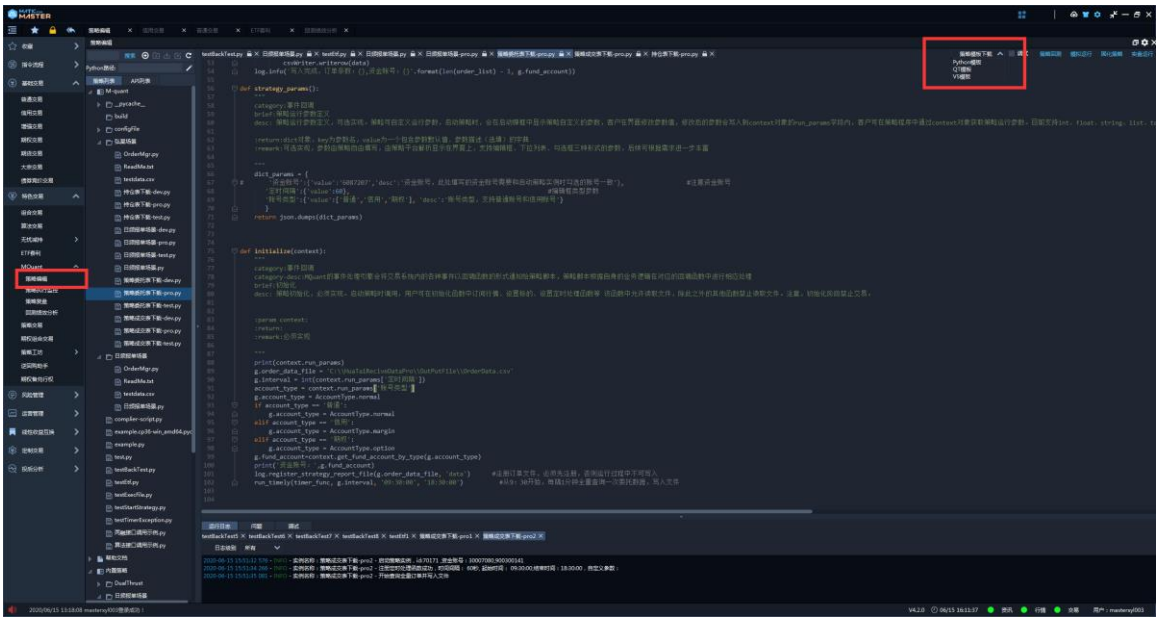
```
class OrderReply(object):
    """
    下单响应数据
    """

    def __init__(self):
        self.entrust_type = ''   # 类型
        self.error_code = ''   # 错误码
        self.fund_account = ''   # 资金账号
        self.order_id = ''   # 订单编号
        self.Origin_order_id = ''   # 原订单编号
        self.batch_no = -1   # 批次号
        self.algo_inst_id = ''   # 算法实例 ID 注
        self.inst_id = ''   # 实例 ID
        self.error_info = ''   # 错误信息
```

8 C++版本

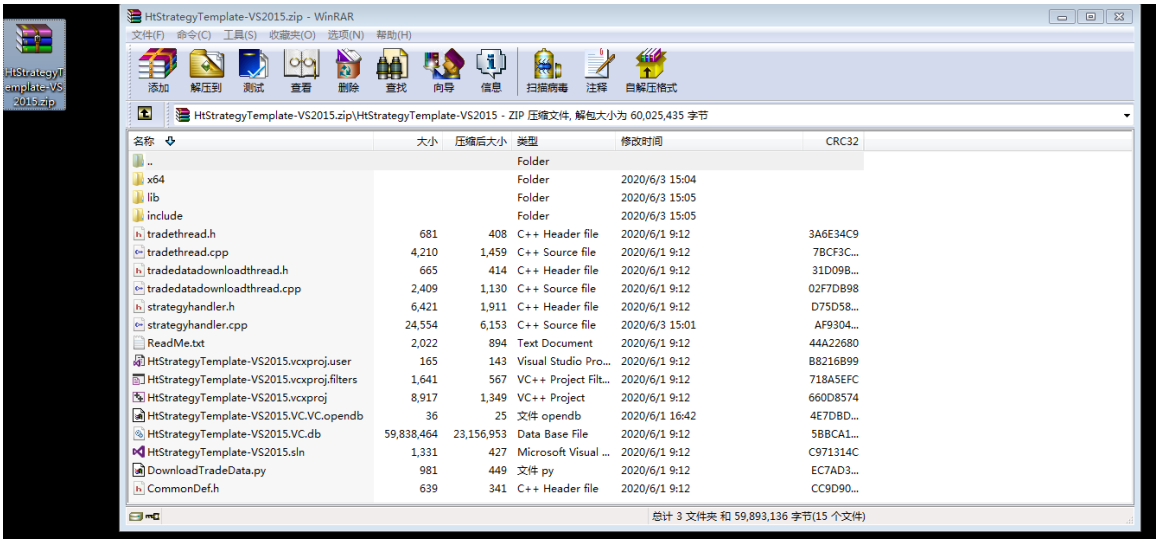
8.1 创建 C++策略

C++版本的操作比较简单，按照如下步骤操作即可：

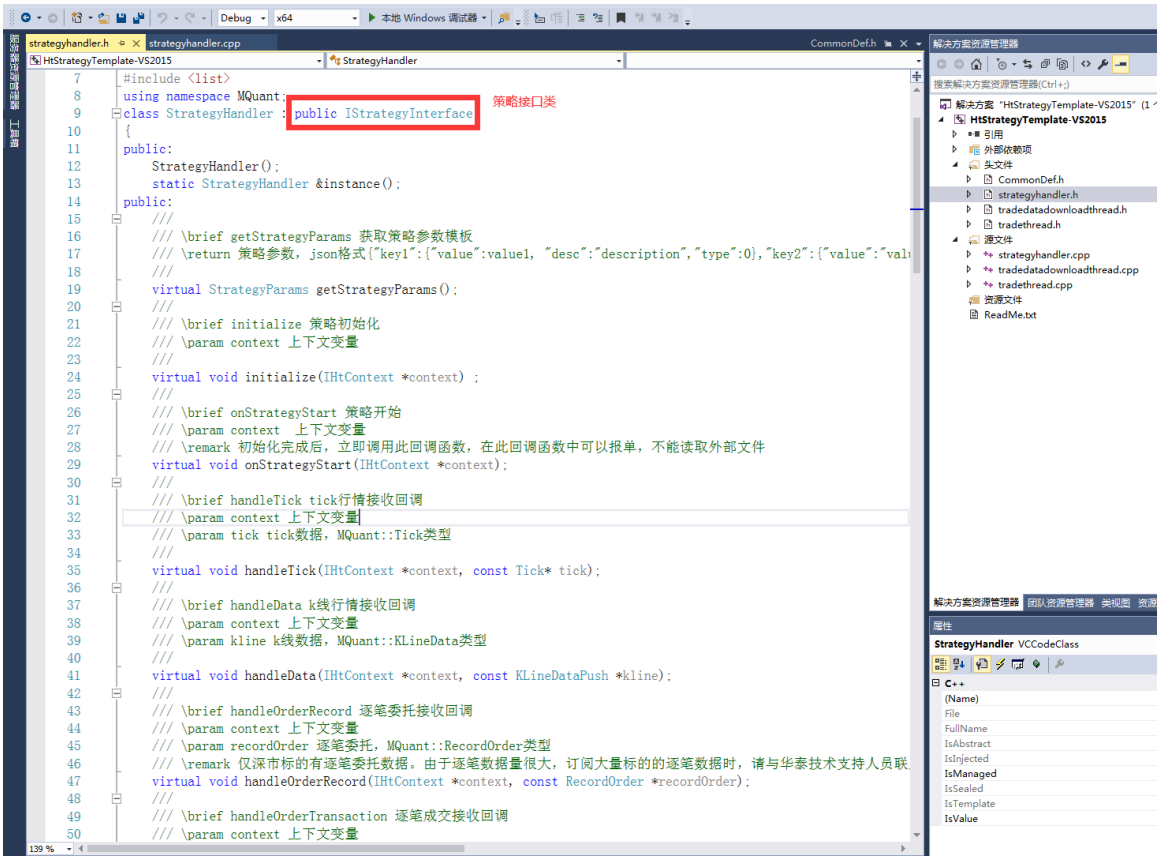
步骤一： 下载 C++工程模板，目前支持 VS 和 QT 两种工程模板。

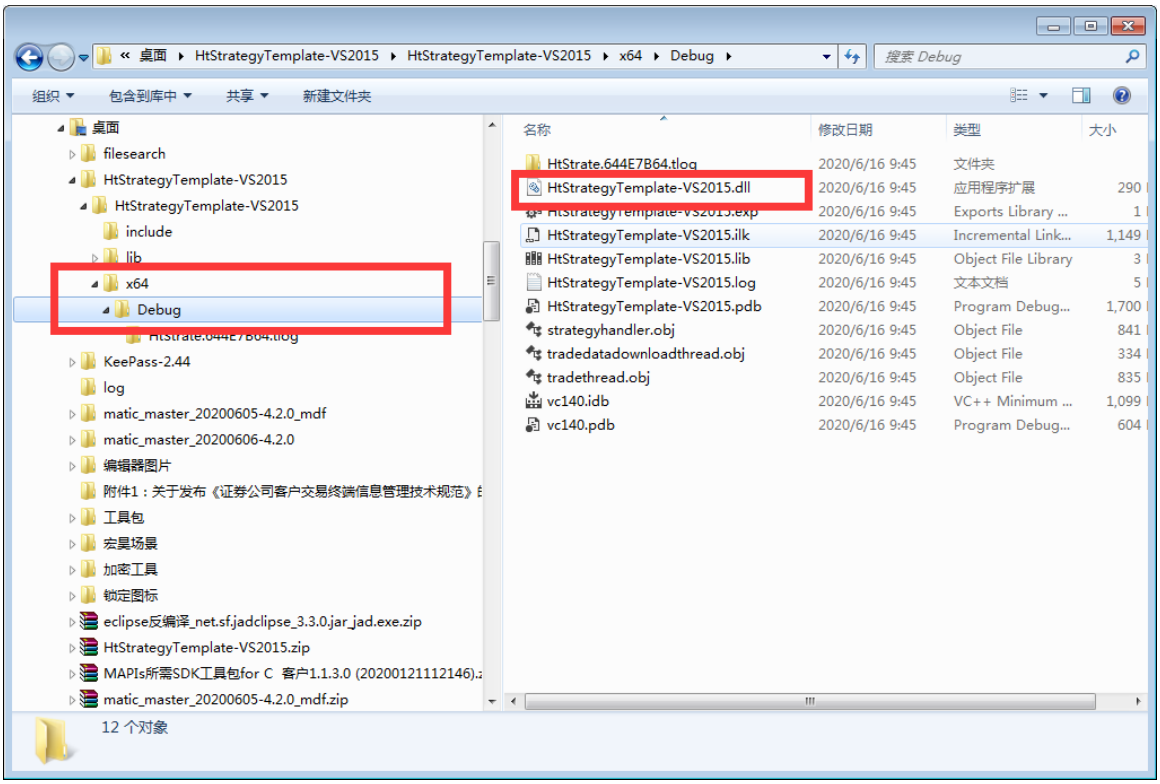


下载的 VS 工程模板如下，是一个配置好的完整的 VS2015 工程，用户也可以选择用其他的 VS 版本打开。

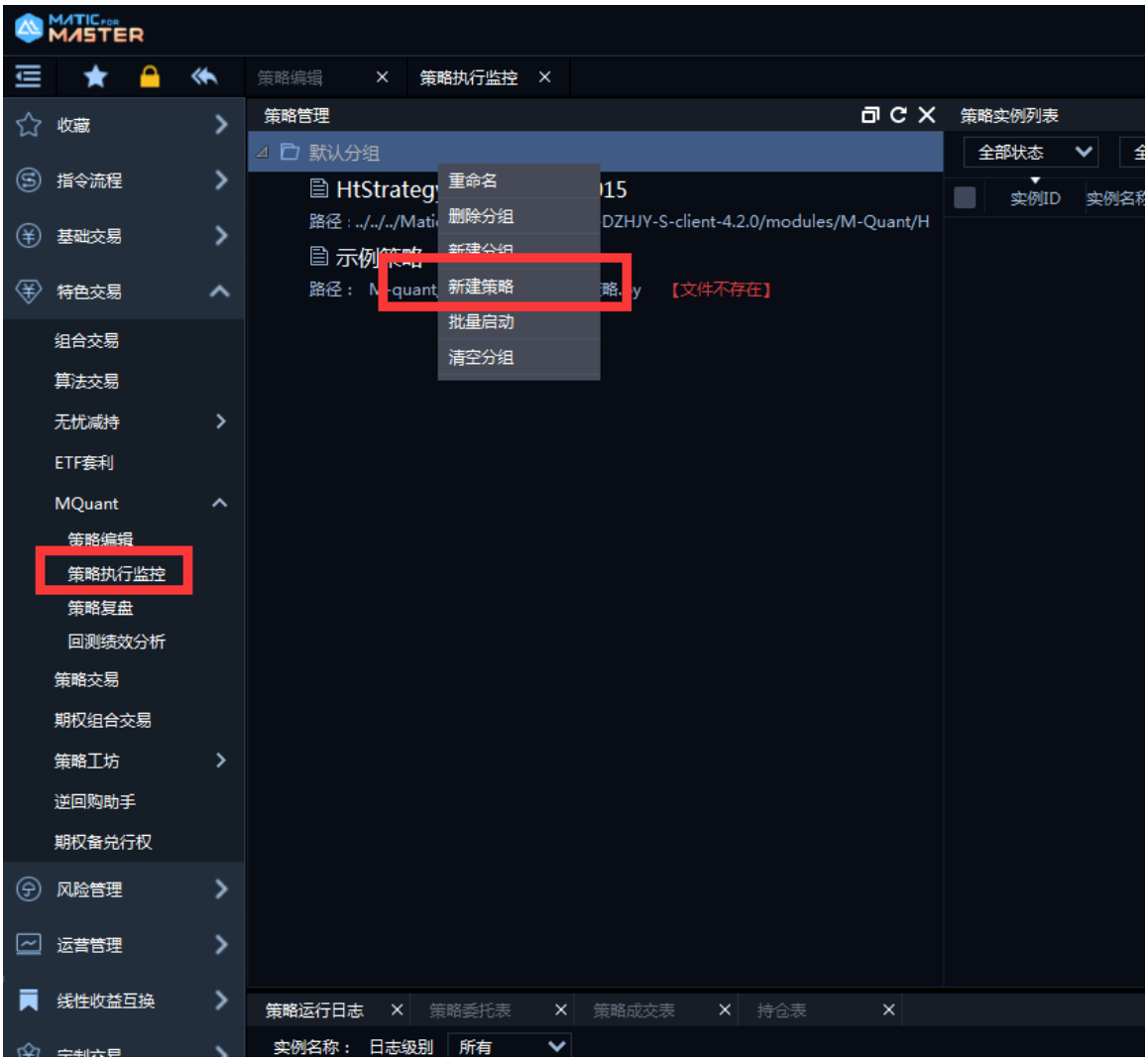


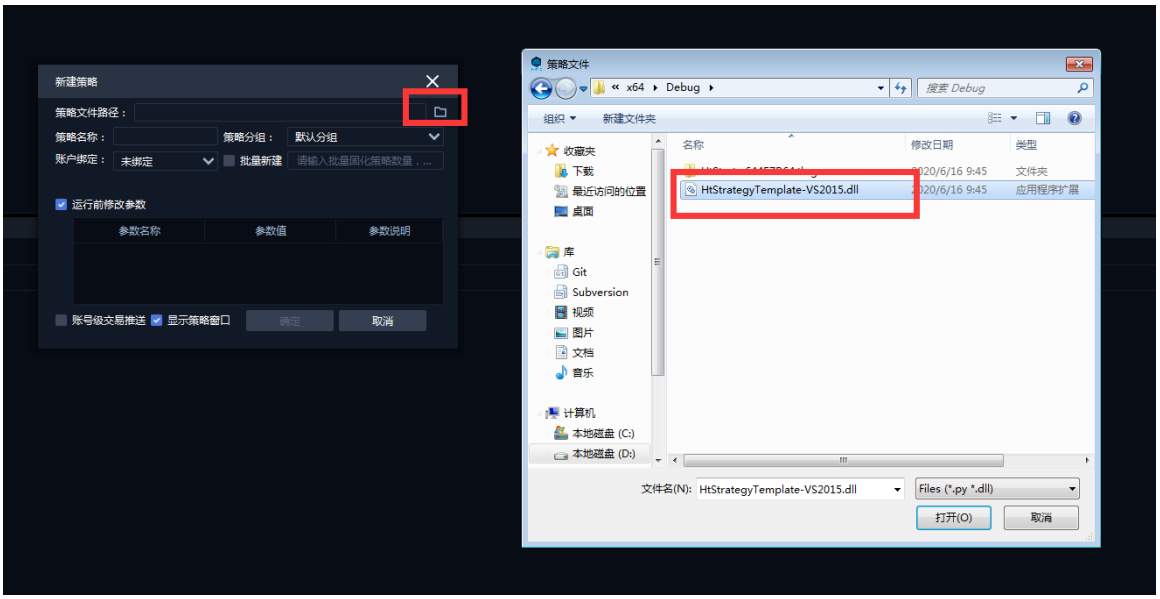
步骤二：编写策略，编译成动态库，支持 Debug 和 Release 版本，仅支持 64 位程序

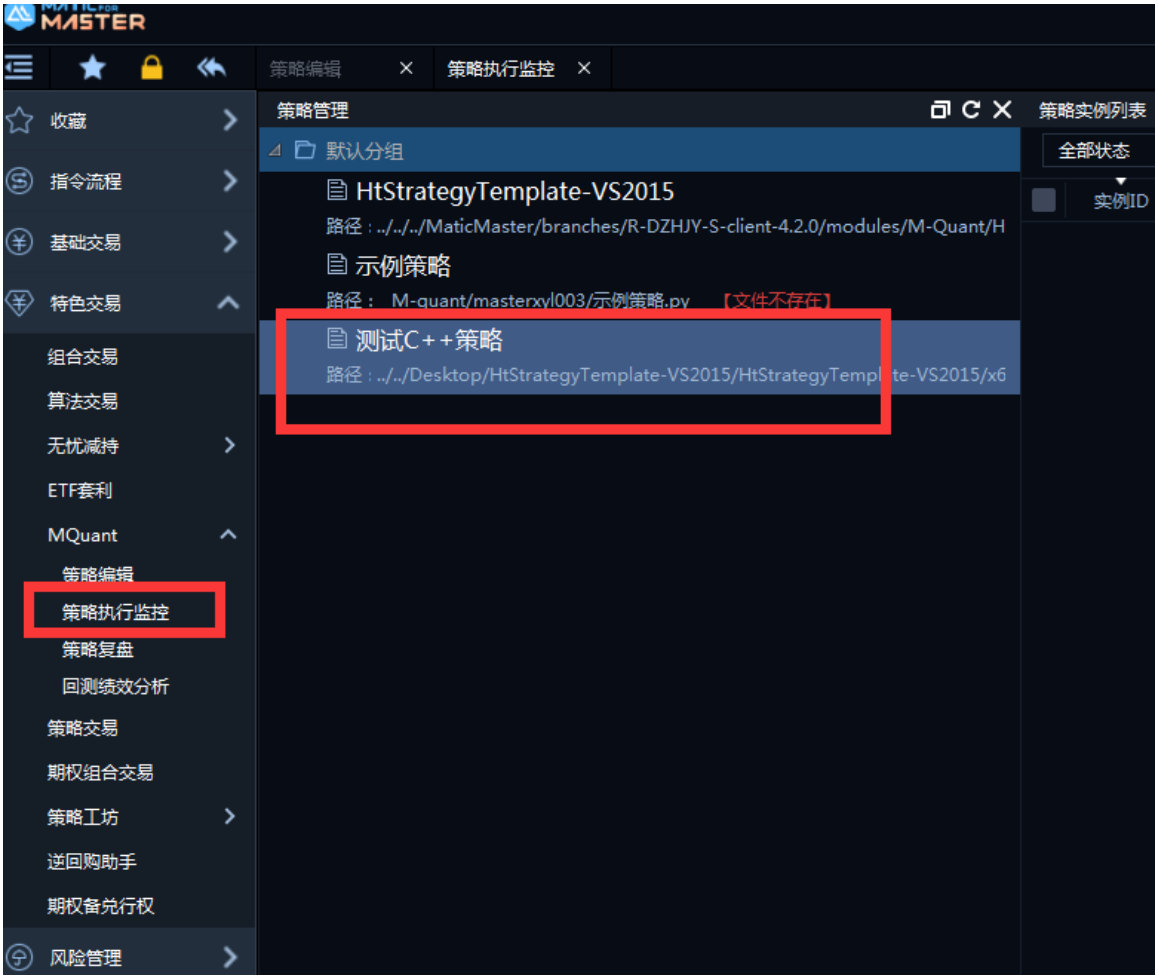




步骤三：在策略执行监控界面创建策略，关联生成的动态库


策略编辑
策略执行监控
策略管理
策略实例列表
默认分组
重命名
删除分组
新建策略
批量启动
清空分组
全部状态
实例ID
收藏
指令流程
基础交易
特色交易
组合交易
算法交易
无忧减持
ETF套利
MQuant
策略编辑
策略执行监控
策略复盘
回测绩效分析
策略交易
期权组合交易
策略工坊
逆回购助手
期权备兑行权
风险管理
运营管理
线性收益互换
策略运行日志
策略委托表
策略成交表
持仓表
实例名称：
日志级别
所有


新建策略
策略文件路径：
策略名称：
策略分组：
默认分组
账户绑定：
未绑定
批量新建
运行前修改参数
参数名称
参数值
参数说明
账号级交易推送
显示策略窗口
确定
取消
策略文件
x64
Debug
搜索 Debug
组织
新建文件夹
名称
修改日期
类型
HtStrategyTemplate-VS2015.dll
2020/6/16 9:45
文件夹
应用程序扩展
收藏夹
下载
最近访问的位置
桌面
库
Git
Subversion
视频
图片
文档
音乐
计算机
本地磁盘 (C:)
本地磁盘 (D:)
文件名(N):
Files (*.py *.dll)
打开(O)
取消

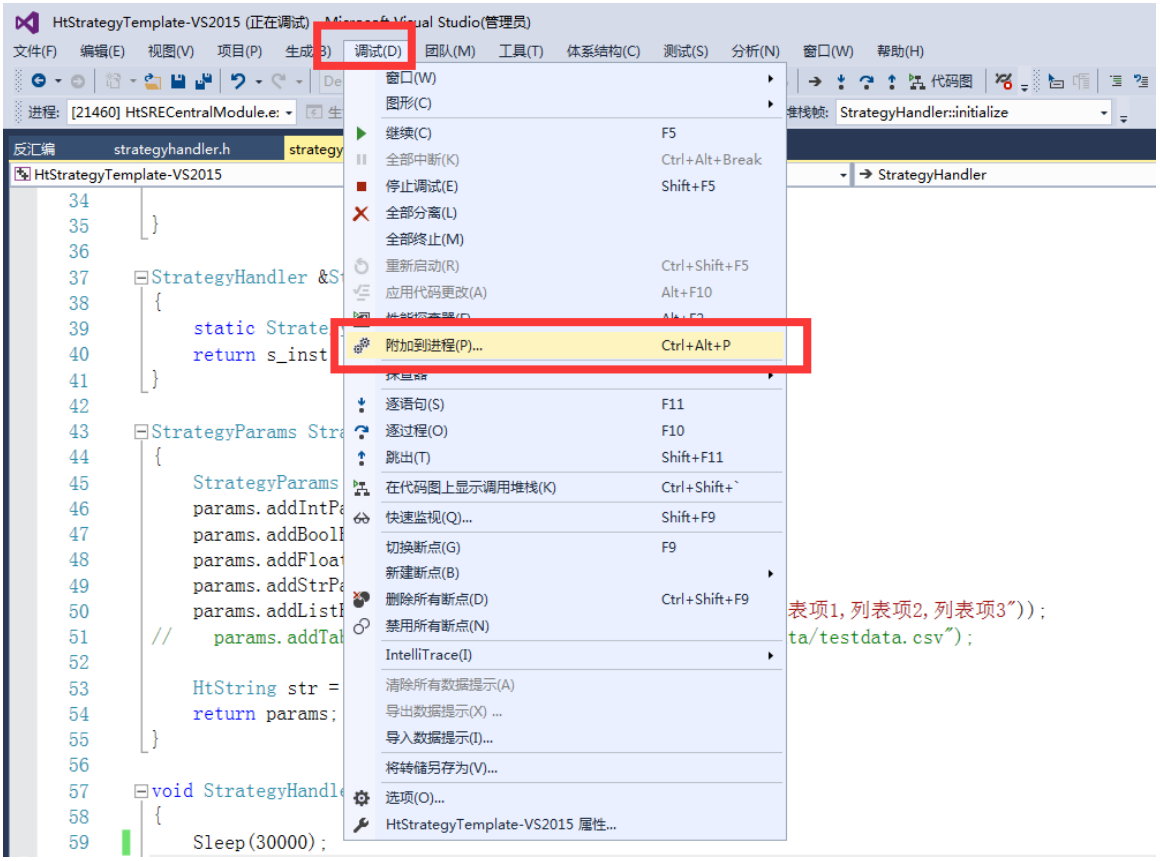


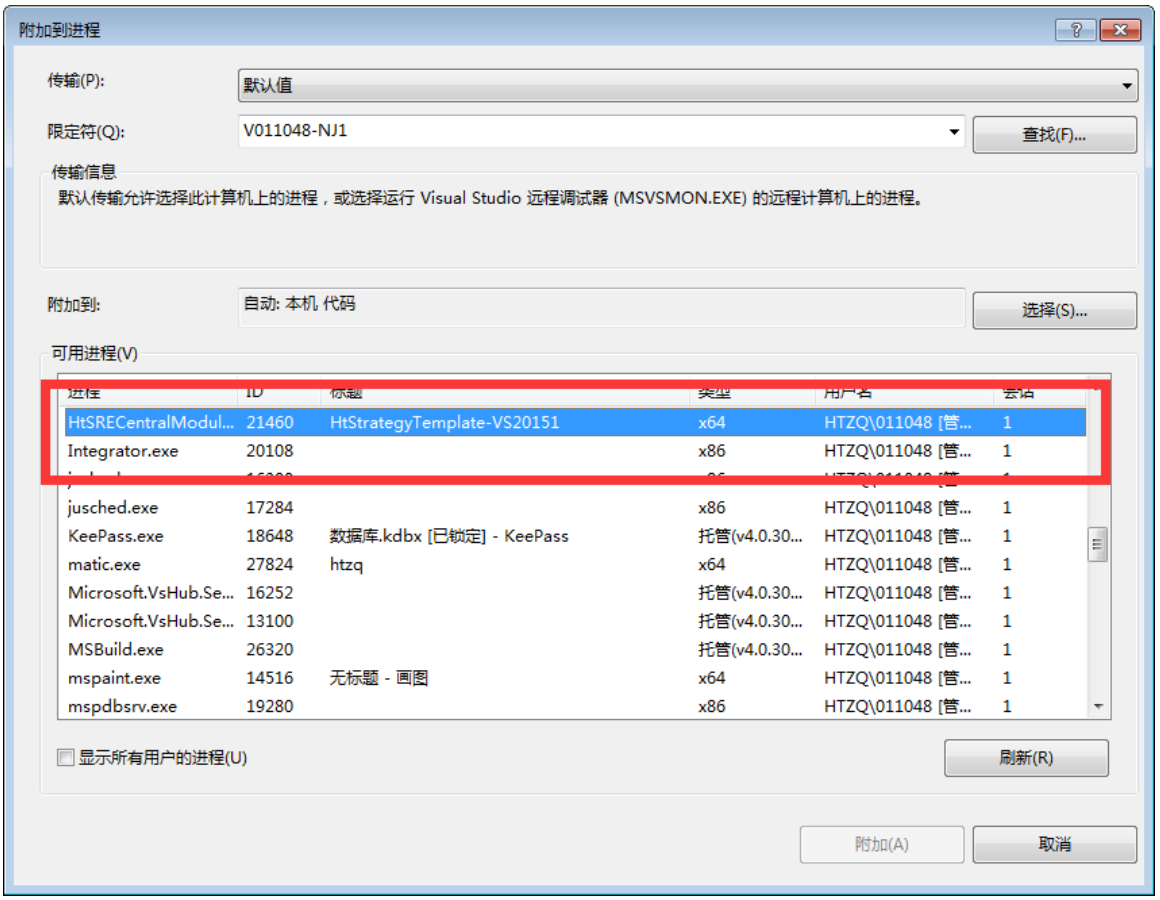
至此，一个完整的 C++策略就创建成功了。

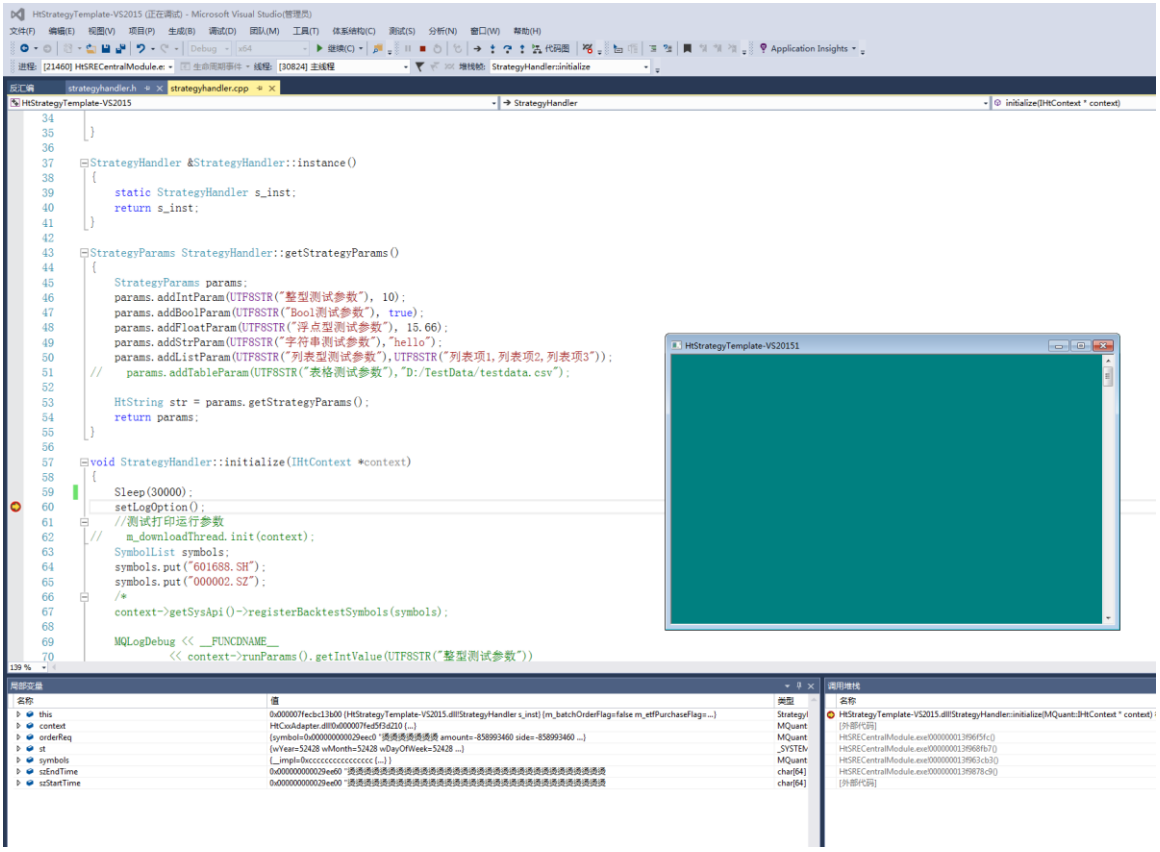
8.2 运行调试 C++策略

C++策略的运行非常简单，直接在策略执行监控的策略关联界面双击相应的策略即可。运行界面和 python 版本完全一致，在此不再赘述。

如果需要进行策略调试，可以编译 Debug 版本，并使用 VS 或者 QT IDE 自带的附加到进程调试的方式来进行，如下图所示：


HtStrategyTemplate-VS2015 (正在调试) - Microsoft Visual Studio(管理员)
文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 团队(M) 工具(T) 体系结构(C) 测试(S) 分析(N) 窗口(W) 帮助(H)
窗口(W)
图形(C)
继续(C) F5
全部中断(K) Ctrl+Alt+Break
停止调试(E) Shift+F5
全部分离(L)
全部终止(M)
重新启动(R) Ctrl+Shift+F5
应用代码更改(A) Alt+F10
附加到进程(P)... Ctrl+Alt+P
逐语句(S) F11
逐过程(O) F10
跳出(T) Shift+F11
在代码图上显示调用堆栈(K) Ctrl+Shift+`
快速监视(Q)... Shift+F9
切换断点(G) F9
新建断点(B)
删除所有断点(D) Ctrl+Shift+F9
禁用所有断点(N)
IntelliTrace(I)
清除所有数据提示(A)
导出数据提示(X) ...
导入数据提示(I)...
将转储另存为(V)...
选项(O)...
HtStrategyTemplate-VS2015 属性...
strategyhandler.h
StrategyHandler
StrategyHandler::initialize


附加到进程
传输(P): 默认值
限定符(Q): V011048-NJ1
查找(F)...
传输信息
默认传输允许选择此计算机上的进程，或选择运行 Visual Studio 远程调试器 (MSVSMON.EXE) 的远程计算机上的进程。
附加到: 自动: 本机 代码
选择(S)...
可用进程(V)
进程 ID 标题 类型 用户名 会话
HtSRECentralModul... 21460 HtStrategyTemplate-VS20151 x64 HTZQ\011048 [管... 1
Integrator.exe 20108 x86 HTZQ\011048 [管... 1
jusched.exe 17284 x86 HTZQ\011048 [管... 1
KeePass.exe 18648 数据库.kdbx [已锁定] - KeePass 托管(v4.0.30... HTZQ\011048 [管... 1
matic.exe 27824 htzq x64 HTZQ\011048 [管... 1
Microsoft.VsHub.Se... 16252 托管(v4.0.30... HTZQ\011048 [管... 1
Microsoft.VsHub.Se... 13100 托管(v4.0.30... HTZQ\011048 [管... 1
MSBuild.exe 26320 托管(v4.0.30... HTZQ\011048 [管... 1
mspaint.exe 14516 无标题 - 画图 x64 HTZQ\011048 [管... 1
mspdbsrv.exe 19280 x86 HTZQ\011048 [管... 1
显示所有用户的进程(U)
刷新(R)
附加(A)
取消



注意，由于 IDE 的附加到进程调试耗时较长，最好在初始化函数开始先 Sleep 一段时间，避免附加成功时程序已经执行过断点所在位置。

8.3 策略库加载失败处理

一般策略库加载失败可能有两种原因：

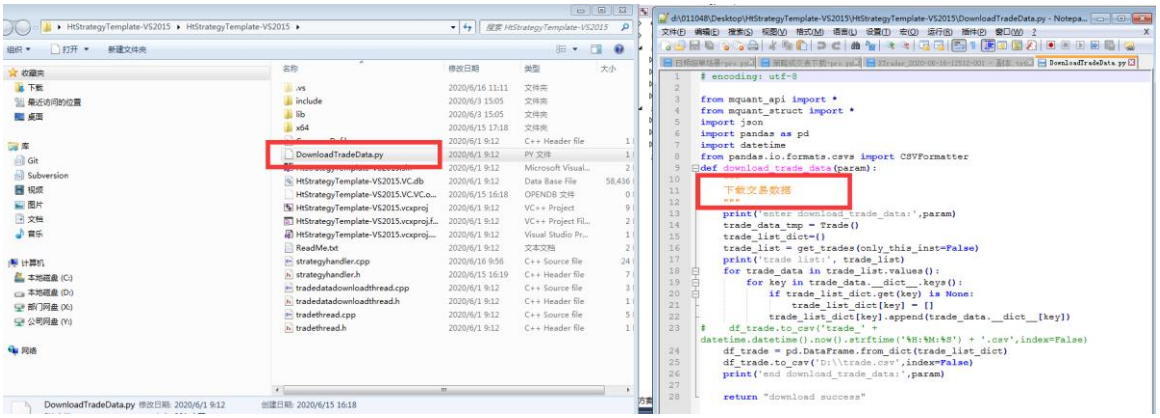
第一种原因是策略库未使用最新的头文件和 lib 重新编译。虽然 MQuant 的 C++接口文件不会频繁变动，这种可能性较小，但是如果出现 matic 升级后策略库加载失败的问题，建议还是重新下载下策略模板，并用最新模板中的接口文件来重新编译。

第二种原因是策略库本身的依赖文件不再当前程序查找路径中。策略执行时，依赖查找路径除掉 Windows 系统路径之外，还有 Matic 根目录和策略文件所在目录（策略文件不需要放在 Matic 根目录下）。对于存在第三方依赖的策略库，需要自行将依赖的文件放在策略所在目录下，直到策略库的依赖完全满足为止。

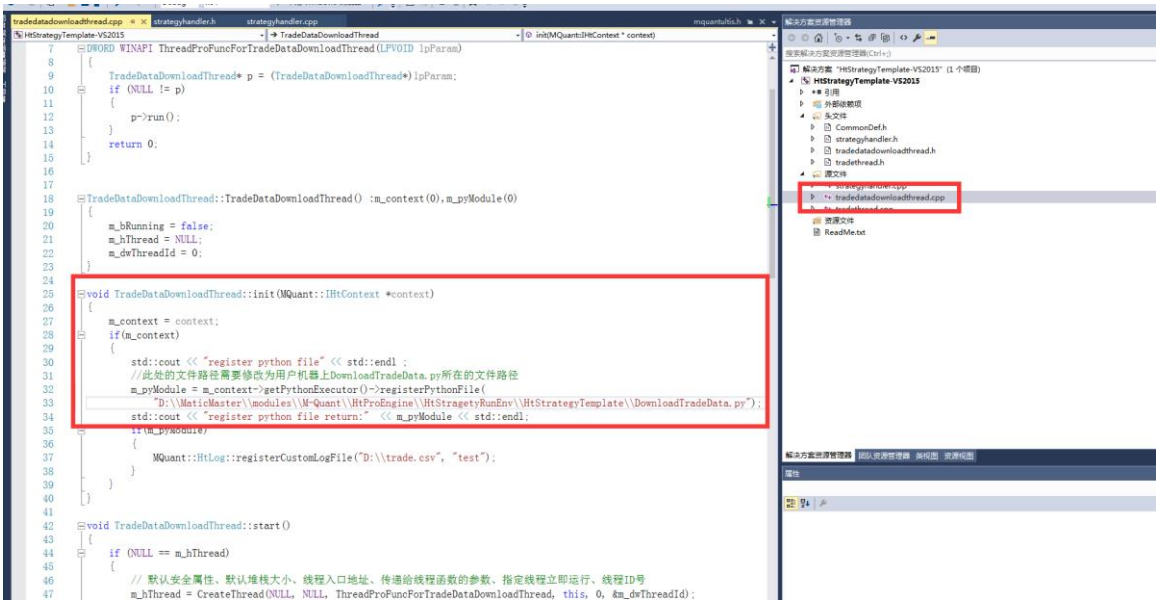
建议客户在出现策略库加载失败的情况时，先查看日志中加载失败的错误码（GetLastError），大致确认错误原因，然后通过 depends.exe 查看策略库依赖，首先将所有依赖文件都拷贝到策略库所在目录，然后在进行下一步操作。策略库依赖的 HtCxxAdapter.dll 库在 Matic 根目录下，不需要拷贝到策略所在目录下。

8.4 C++调用 Python

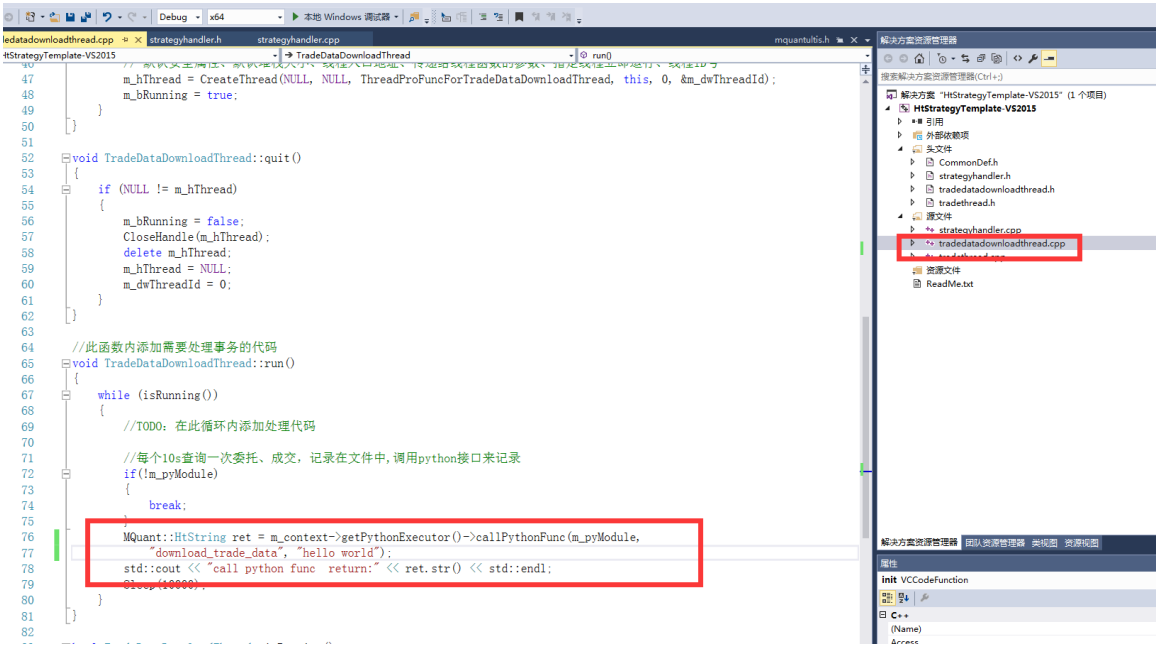
为了方便客户进行策略开发，MQuant C++版本支持调用 python 函数。例如，将交易数据下载功能放在 python 中，C++程序开一个单独的线程来执行 python 脚本。如下所示：



在 C++程序中，需要先调用接口注册 python 模块，如下图所示：



然后通过 MQuant 提供的调用 python 的 API 来进行调用，如下图所示：



目前出入参均只支持字符串类型。

8.5 接口介绍

C++版本接口和 python 接口功能几乎完全一致，同时 C++接口遵循接口即文档的原则，在此不再赘述，有疑问请联系 Matic 技术支持。

8.6 特别说明

C++接口中的容器类数据结构，如 StrategyParams、SymbolList、MarginAssureSecurityList、OrderList 等，是用于屏蔽接口中的容器模板，避免 Debug 版本库无法使用的问题，切记不要直接存储在类的成员变量中，使用方式请遵循即取即用的原则，只用在局部作用域中。

MQuant C++版本的 log 支持直接打印 MQuant 内置的数据结构，例如 Order、Position、Execution 等，具体使用方式参照策略模板。

9 性能优化

鉴于平台上部分策略存在瞬时大量报单、在回调函数中进行耗时处理导致所有消息推送延迟严重等性能问题，MQuant 专门进行了针对性的优化。

9.1 高频报单场景优化

对瞬时大量报单的回报性能进行了优化，包括：

（1） 后台绕过消息中间件从交易核心直推网关，需要由 Matic 运维人员在管理端开启，默认不开启，确实有瞬时大量报单场景且对回报性能要求较高的请联系 QQ：952574701

（2） Matic 客户端从网络层、业务层、策略层进行了一次彻底的优化，通过 matic 根目录下 MQuantConfig.ini 配置文件中的 HighSpeedMode 配置项控制，HighSpeedMode=1 表示开启性能优化模式，默认不开启。

（3） 后续可能会默认全部开启优化后模式。

9.2 回调耗时处理优化

针对策略在回调函数中进行耗时处理，导致消息队列严重积压，影响策略运行的场景进行了优化，提供了多线程回调模式。为了避免消息乱序，MQuant 的多线程回调模式不允许同一种类型的消息在多个线程中回调，目前如果开启多线程回调模式，默认**定时信号、行情（tick）、逐笔委托、逐笔成交、订单回报/成交回报**分别在单独的线程中回调，其他消息在同一个线程中回调，如果客户有其他线程规划要求，可以联系 QQ：952574701。

多线程回调模式通过 matic 根目录下的 MQuantConfig.ini 配置文件中的 MsgProcThreadMode 配置项控制，默认不开启，修改 MsgProcThreadMode=1 即可开启多线程模式。

此外，不同消息的丢弃策略也可通过配置实现，如有需要请联系 QQ：952574701。目前多线程模式下默认行情、逐笔委托、逐笔成交如果队列已满就会丢弃，其他消息则不丢弃。单线程模式下所有消息在同一个队列，且队列已满时所有消息都会被丢弃。

特别说明：建议客户尽量不要在回调函数中进行耗时的业务处理，如果有这类处理，可以开启线程执行。切忌在回调函数中调用 sleep 来休眠等待。

10 业务规则说明

10.1 实时资金流向说明

流入单	逐笔成交中字段 TradeBSFlag=1（B）买入为流入成交单
流出单	逐笔成交中字段 TradeBSFlag=2（S）卖出为流出成交单
超大单	单笔成交 20 万股及以上或者 100 万元及以上，或占流通盘比例≥0.3%
大单	单笔成交，6 万股≤股数<20 万股或者 30 万元≤金额<100 万元，或 0.1%≤占流通盘比例<0.3%

中单	单笔成交，1 万股≤股数<6 万股或者 5 万元≤金额<30 万元，或 0.017%≤占流通盘比例<0.1%
小单	单笔成交，股数<1 万股或者金额<5 万元，或占流通盘比例<0.017%
流入	流入成交单的成交金额计入资金流入
流出	流出成交单的成交金额计入资金流出
净流入	资金流入—资金流出
超大户	依据超大单定义
大户	依据大单定义
主力	超大单+大单
散户	中单+小单
资金流向指标线	分时净流入从交易时间开始到结束的累加计算得到的数值随时间变化的曲线
最近五日主力增减仓	最近五个交易日的主力净流入
今日增仓占比	当日主力净流入/（主力流入+散户流入）
3 日增仓占比	3 日累加主力净流入/（3 日主力流入+3 日散户流入）

10.2 分钟 K（沪深股票，与 wind SDK 规则一致，与 wind 界面规则不一致）

	计算说明	说明
数目	全天输出 242 支 K 线，时间戳分别为 092500、093000-112900、130000-150000	
统计标准	时间戳为 HHMM00 的分钟 K 线记录时间周期[HHMM00. HHMM59.999]之间的成交信息 K 线开盘价为该时间周期内第一幅股票快照的最新价，收盘价为其开盘价为该时间周期内最后一幅股票快照的最新价 如果该时间周期内行情快照的最高价发生上升，则以上升之后的最高价为 K 线最高价；否则以该时间周期内的最高的最新价为 K 线最高价 如果该时间周期内行情快照的最低价发生下降，则以下降之后的最低价为 K 线最低价；否则以该时间周期内的最低的最新价为 K 线最低价 K 线的成交笔数/成交量/成交金额为该时间周期内最后一幅快照的成交统计	1、开收盘 092500K 线的开高低收均为当日开盘价，成交量为开盘集合竞价阶段成交总量 150000K 线的开高低收均为当日收盘价，成交量为收盘集合竞价阶段成交总量

	计算说明	说明
	量减去上一时间周期内最后一幅快照的成交统计量的结果	2、休市 记 [113000,130000) 之间第一幅快照时间为 T，则 112900K 线记录时间周期为[112900,T] 之间的成交统计（该 K 线收盘价为 T 时刻最新价，成交统计为 T 时刻快照减去前一周期最后一幅快照）
输出时刻	每个时间周期的 K 线在该时间周期的第一次快照数据到达时开始输出，并随着快照不断刷新；该分钟的最终值在下一时间周期的第一次快照数据到达时输出 在正常刷新时 KLineCategory=0，终值 KLineCategory=11	092500K 线只输出一次，在 092500 时刻之后的第二次行情快照数据到达时输出 150000K 线只输出一次，在 150000 时刻之后的第二次行情快照数据到达时输出
特殊情况	1、开盘时未产生开盘价：如集合竞价阶段未产生开盘价，则在该股票产生开盘价之前的所有 K 线均记开高低收价格为前收盘价、成交量为 0；在产生开盘价的时间周期内，记该时间周期的 K 线开盘价为当日开盘价 2、停牌：股票在停牌阶段正常输出 K 线，并记最新价为当日最后成交价（当日有成交）或前收价（当日无成交），记成交量/笔数/金额为 0 3、暂停上市：暂停上市的股票不输出实时 K 线	
异常处理	1、如计算程序有一段时间未收到实时行情数据，在数据中断的时段内不会输出 K 线。计算程序会在重新收取到实时行情数据之后补充之前缺失的 K 线，记补充的 K 线最新价为当日最后成交价（当日曾经收到过实时行情数据）或前收价（在开盘之前即发生数据中断），记成交量/笔数/金额为 0 计算程序在重新收取到实时行情数据后，不会将缺失的成交计入新的时间周期（即该时间周期的成交为该时间周期最终快照减去第一幅恢复快照的统计，而不是最终快照减去数据中断前最终快照的统计）	

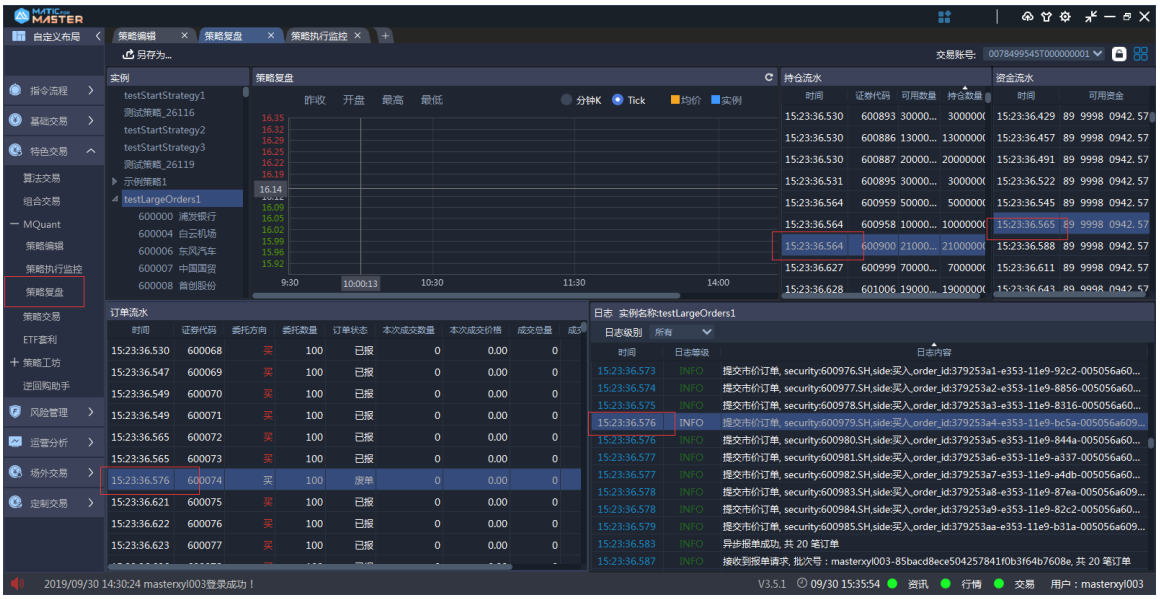
11 策略复盘（已下线）

策略复盘是为了方便用户分析策略，复盘策略运行中的每个细节而设计的，其主要特点有两个：

1) 记录到策略运行过程中的每个订单状态、资金、持仓变动的所有流水信息。

2) 订单、资金、持仓、日志的联动展示，以毫秒时间为基准进行联动。

如下图所示：



点击左侧菜单栏的特色交易→MQuant→策略复盘菜单，可以打开策略复盘界面。策略复盘界面的左上角展示的是已完成的实例列表，每个实例交易的标的会展示为实例的子一级节点。

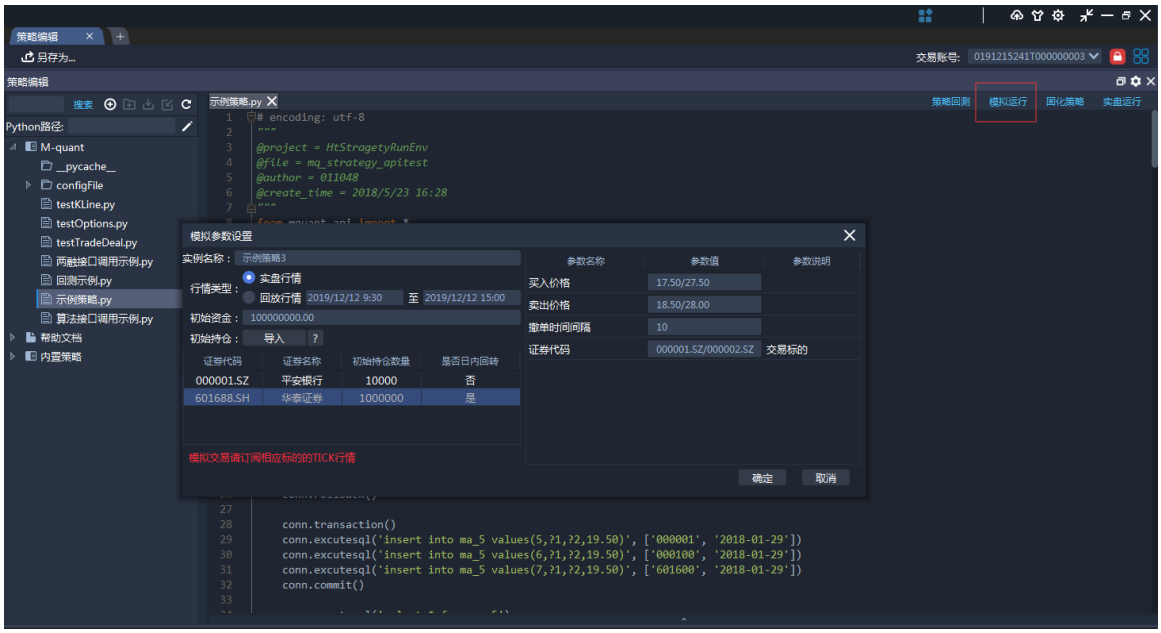
选中实例节点，则图形区域不展示任何内容，订单流水展示该实例交易的所有订单的所有状态记录，持仓和资金流水分别展示实例运行过程中持仓和资金的所有变化过程，日志界面展示实例运行日志。

选中实例下的某一标的，展示该标的的 tick 或者分钟 k 线（测试环境行情数据存在问题，不建议查看行情图形数据），并在线上标注策略报单时间点。同时，订单流水和持仓流水只会展示该标的相关的数据。

选中订单流水、资金流水、持仓流水及运行日志表格中的任何一行数据，其他表格都会联动展示与选中数据时间最相近数据，方便您对策略进行细粒度的复盘。

12 模拟交易（已下线，使用 tick 级回测替代模拟交易）

为了方便用户盘后测试，MQuant 提供了模拟交易功能，支持自定义行情回放，支持设置初始资金、持仓，支持设置标的是否日内回转，方便用户当日测试买卖两个方向。点击策略编辑界面上左上角的模拟交易按钮，弹出模拟交易设置界面如下：



用户可以选择使用实盘行情和回放行情，回放行情可以选择任意时间范围（一次模拟交易仅支持一天之内的数据），系统会在策略调用 subscribe 接口订阅 tick 行情时（目前暂不支持 k 线行情的回放）自动下载标的的 tick 行情并进行回放。

界面支持用户设置初始资金和初始持仓，初始持仓可以通过在表格区域右键 添加标的，也支持通过文件导入。点击导入按钮后面的“？”按钮，弹出持仓模板框，用户可以根据模板定制自己的初始持仓文件，或者下载模板然后再模板中进行修改。

策略模拟交易产生的委托、成交数据在策略监控界面查看，为了方便用户区分实盘和模拟交易数据，在策略委托表和策略成交表右上角增加了“实盘”和“模拟”的选项，如下图所示：

用户需要勾选“模拟”选项，表格中会自动列举出模拟交易的所有报单。

注意事项：

（1） 为了避免下载 tick 数据占用用户太多网络带宽资源，目前回放行情仅支持盘后使用。

（2） 模拟交易采用的客户端盘口撮合，要求进行交易的标的必须订阅 tick 行情，否则报单会报错

（3） 行情回放一次模拟交易仅支持一天之内的 tick 数据

（4） 目前暂不支持 k 线行情的回放

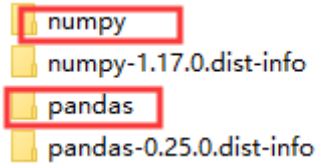
13 FAQ

13.1 客户端下载地址

https://huatech.htsc.com.cn/documents?navId=2&treeId=7

13.2 是否能支持 python 的一些常用库如 pandas 和 numpy 等？

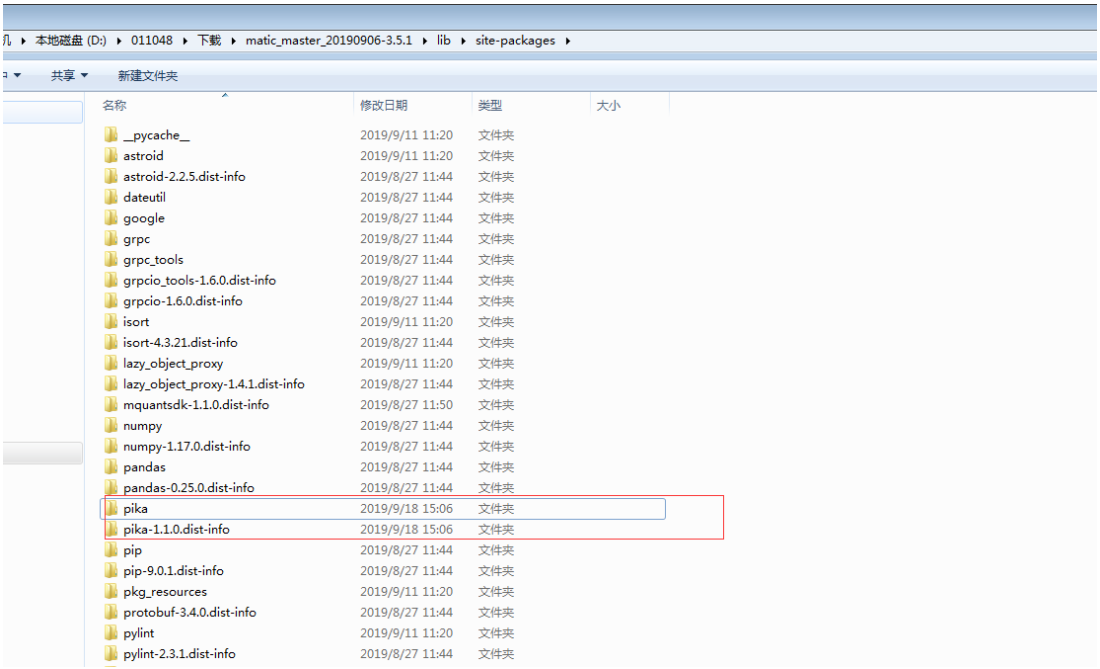
如果用户需要用到第三方库，则需要自定义策略运行环境，参考手册目录 5 。
备注：numpy pandas talib 已经内置了，lib/site-packages 目录下



注意： 我们自带的 python 是 3.6.2，所以自己配环境也必须是 3.6.2 的版本，推荐通过我们自带的绿色 python 解释器导入第三方包

目前 matic 自带的 python 版本中已经集成了 pandas、numpy、talib 这些常用的第三方库，用户可以直接使用，无需安装。

13.2.1 推荐 Mquant 自带的解释器导入第三方包

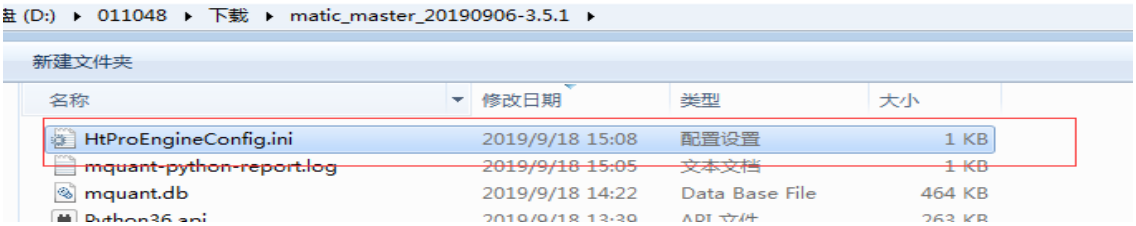


第一种：把安装生成的这两个文件拷贝到我们软件相应的目录下就可以，其他的库也可以这么做：

① 第三方库在 site-packages 里面的文件拷贝到软件目录下的 lib/site-packages 里面

② 去掉自定义的路径。

③ 现在代码检查检查不出来自定义的 python 环境，删掉根目录下面的这个文件，否则还会定位到你自定义的 python 路径下



第二种：直接用你们 scripts 目录下的 pip 安装，会把目录生成到相应的软件目录下
用绝对路径的 python.exe 执行 pip 吧

操作方式：

直接在 matic.exe 程序所在根目录的文件中输入 cmd（重点），回车

输入 python.exe <matic.exe 所在的根目录>\Script\pip.exe install <第三方包的绝对路径>

路径举例：

E:\Work-must\New-matic-master\MATIC-client-test-3.5.1-x64(4)\python.exe E:\Work-must\New-matic-master\MATIC-client-test-3.5.1-x64(4)\Scripts\pip.exe install C:\Users\admin\Downloads\numpy-1.16.5+vanilla-cp36-cp36m-win_amd64.whl

13.3 pandas 中找不到模块

pandas 里面很多函数里面都有用时加载库的代码，加载库就是读文件的操作，例如 io 读写文件，sum 函数使用等，例如报错 No module named "pandas.io.formats.csvs"，
import 区域需要写入：import pandas.io.formats.csvs

5.3.1. No module named "pandas.io.formats.csvs"

import 区域需要写入：import pandas.io.formats.csvs

5.3.2. No module named "pandas.core.computation.expressions"

import 区域需要写入：import pandas.core.computation.expressions as expressions

13.4 可以同时开多个策略么

可以，限制 128 个

13.5 Linux 环境下可以运行么

暂时不支持

13.6 实盘运行、模拟运行和回测在行情和撮合方面的区别？

1.实盘运行

行情：和实际行情时间保持一致，只有交易时间内才有行情推送。其他非交易时间内会有 1 条缓存的数据切面（仅为了程序运行逻辑完整）

行情粒度：tick，逐笔成交，逐笔委托，分钟 K

撮合：集中柜台资金账号，数量 1000 以内全成，超过 1000 成交 1000，剩余可测撤单

（现在提供的测试账号，大部分都是按实际行情盘口撮合的）

快速柜台资金账号，一般默认撮合规则为成交 50%，具体看 tmp 撮合工具设置情况

2.模拟运行（只支持 A 股，不支持两融，不支持期权和期货 ）（该功能已下线）

行情：可选择实盘行情和行情回放

实盘行情即为实盘模式的行情推送，有逐笔行情，非交易时间没有行情推送

行情回放可选择前一个自然月内（包括本日）的任意一天，只可以回放 tick，没有逐笔行情回放。回放 tick 且要在程序中订阅回放的标的代码

（比如：回放 601688.SH，要在代码中 subscribe('601688.SH')），目前最多支持 20 只标的回放

撮合：虚拟的资金账号。可设置初始金额和底仓，底仓可支持本日回转可卖，是按行情撮合成交，报单价格在行情价格范围内才会成交

3.回测（只支持 A 股，不支持两融，不支持期权和期货）：

行情： 目前只支持 tick，不支持逐笔

tick—上一个交易日半年至上一个交易日（以今日为例：20190824-20200221）

分钟 k—20180901 至上一个交易日

日 k—20140101 至上一个交易日

行情支持的品种：沪深 A 股，中小板，创业板，场内基金，指数，期权和期货暂不支持

撮合：虚拟的资金账号。可设置初始金额，底仓需要在回测日期的上一个交易日日期买入建仓（T+1）。按行情撮合，报单价格在行情价格范围内才会成交

13.7 模拟运行和回测支持期权和期货么

模拟运行和回测只支持 A 股，不支持两融，不支持期权和期货，可以用实盘运行测试期权和期货，测试环境的实盘运行不会报到交易所

13.8 回测支持么？

3.6.0 版本已支持（客户端登录按钮下方显示版本信息）

13.8.1 回测数据

目前版本(3.6.2)提供一周的测试数据，回测自动下载历史行情数据，数据粒度包含 tick、分钟 k、日 k。

① 只支持 A 股，不支持两融，不支持期权和期货

② 只支持股票和基金品种的 tick 数据，其他行情数据（逐笔）暂不支持。

③ 回测和撮合要放在一块，否则时序控制不了。

④ 行情数据属于敏感数据，不支持下载到本地或者导入外部数据源。

⑤ 下个版本支持回测时读取本地文件的（待优化）

⑥ mquant 模拟运行和回测的时钟都是按照 tick 产生的行情时钟，不是本地时间也不是 matic 时间。举例：现在本地时间 09:53，写一个单次定时器 run_timely(single_timer_func, -1, '09:20:00')测试模拟运行行情回放，从 09:15 开始回放行情，当 tick 回放到 09:20 时，是会触发定时器的。

其他回测注意事项参考手册 1.4

13.8.2 回测的资金和持仓是自己设置的么

回测是虚拟的资金账号， 可设置初始金额，底仓需要在回测日期的上一个交易日日期买入建仓（T+1）。

13.9 mquant 支持多个账户用同一台电脑运行吗？

可以的，但是要稍微注意下有个问题，如果 matic 进程 1 启动了策略实例，然后再启动 matic 进程 2，这个时候进程 1 的实例会被强制关掉，所以一定要记得先把两个进程启动起来之后再启动策略实例

13.10 mquant 支持一个账户同时登录么？

不支持，Matic 目前是单点登录

13.11 策略和资金账号如何绑定？

选中策略，右键固化策略，固化策略弹框有个“运行前修改参数”的勾选框，如果预先绑定账户，然后预设好运行参数，去掉此框的勾选状态，那么通过策略管理界面运行时不会弹框，会直接启动实例

13.12 mquant 如何同时运行多个资金账户和策略？

可以用策略固化和批量启动，把多个策略固化到一个分组里，可以同时启动

1. 策略执行监控菜单-》策略管理，右键点击空白处新建分组
2. 策略编辑菜单，选中策略，右键固化策略，固化时不同的策略要一个个选择分组
3. 固化策略弹框有个“运行前修改参数”的勾选框，如果预先绑定账户，然后预设好运行参数，去掉此框的勾选状态，那么通过策略管理界面运行时不会弹框，会直接启动实例
4. 策略执行监控菜单-》策略管理，选中分组，右键批量启动策略

13.13 是否支持多线程

目前：始化和结束在另外的线程中，其余回调在一个线程中

允许：可以在初始化阶段创建拉起多个线程，mquant 对多线程的使用不做制约，但是多线程内的操作还是受到 mquant 使用要求的限制，比如禁止读取文件、禁止进程间通信等，行为必须合规。此外，需要对线程安全问题进行处理，做好数据保护。

注意：

我们的系统架构是单线程的，提供的 api 接口不是线程安全的，可以单独开一个线程做一些计算，但是建议最好不要多线程调用系统 api

13.14 数据交互

13.14.1 准则

由于合规要求，mquant 不允许在策略运行过程中进行读数据，仅可以在初始化函数 initialize 和策略结束的回调函数 on_strategy_end 进行读文件，写文件没有限制。

13.14.2 不可使用场景

基于准则，举例一些不能进行数据交互的场景：

1. 回调函数 handle_tick 不允许读文件。
2. 不允许访问网络，所以初始化函数 initialize 中 mongodb/redis 或者本地的 sql 数据库都不能使用 ，只可以访问内存数据库 sqllite，可以将网络的数据先落本地，再在 initialize 函数中读取文件。
3. Sqllite 是实例的内存数据库，不支持实例间共享数据，不支持数据落地。
4. 不允许使用 socket，pipe 跨进程通信

13.15 如何数据交互

13.15.1 支持每隔 10 分钟通过 api 启动新的策略实例

我们提供了策略实例 1 中通过 start_strategy 接口启动策略实例 2 的方式，同时可以传递参数给实例 2，然后就可以在实例 2 的初始化函数中就读取文件，频率最少 10 分钟一次。

```
def start_strategy(strategy_file, run_params='', instance_name=''):
    """
    创建策略实例，允许带参启动，参数会直接传给新启动的实例，不会做任何额外处理
    新启动的实例会继承父实例的资金账号
    :param strategy_file
        策略文件路径 支持相对路径，相对路径为相对用户目录下的文件路径（界面上
    :param run_params 启动参数，最长4096字节,可以为空
    :param instance_name 可选参数，如果未填写默认生成
    :return: 成功返回实例ID，失败返回空字符串
    """
    return start_strategy_impl(strategy_file, run_params, instance_name)
```

举例：

调用脚本.py

被调用脚本.py

13.15.2 手动上传参数

如果只是在运行过程中修改参数的话，可以通过策略实例列表手动上传参数

策略运行过程中参数变更通知函数 on_strategy_params_change(params, path)

```
    pass
def on_strategy_params_change(params, path):
    """
    参数说明:
    :params: 策略参数，可以是任意形式，由您自行传入，并在策略中自行解析
    :path: 如果传入参数文件，path为参数文件路径，否则为空字符串
    调用时间：通过MQuant执行监控界面设置参数时调用
    是否必须：否
    """
    print(params)
    print(path)
    log.debug(params, 'entrust')
```

13.16 K 线数据

13.16.1 日 K

（9）可以区分交易日和自然日 ：
交易日 date_type=DateType.TRADE_DATE；
自然日：date_type=DateType.NORMAL_DATE(默认)
（10）是否包含结束时间
不包含：include_init_date = False(默认) ； 包含 include_init_date = True
（11）现在不限制可以获取的时间跨度
（12）要选择实盘交易模式
（13）实盘可以获取当日 K 线状态，当日完整的 K 线数据未准备好，沪深需要等到三点四十五之后

13.16.2 如何获取实时 1 分钟 k

先订阅，有实时推送的分钟 k
subscribe(symbol, MarketDataType.KLINE_1M)， 回调函数 handle_data(context, kline_data)

13.16.3 如何获取 5 分钟 K，5 分钟 K，15 分钟 K

当天没有 5 分钟 K，5 分钟 K，15 分钟 K 是收盘后计算的，盘中实时只能查到 1 分钟 K。
get_kline_data:查询 K 线数据，包括 1 分钟、5 分钟、15 分钟、30 分钟、60 分钟 K 线及日 K 线

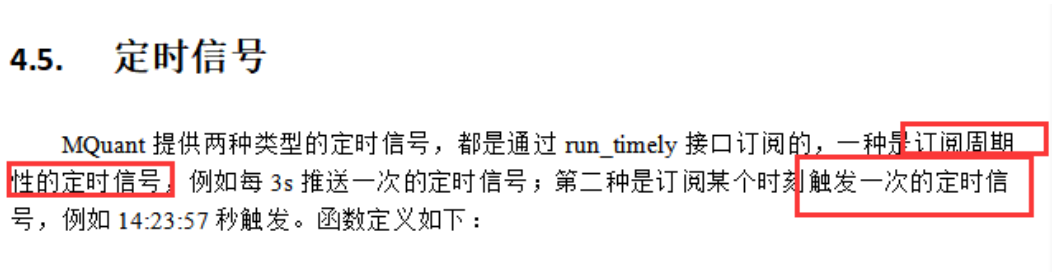
13.17 Mquant 支持的品种

不支持新股申购；
Matic master 不支持可转债转股，可以使用专业版 ii
支持普通 A 股买卖，两融交易，etf 申赎（有内置 ETF 策略模板），可转债买卖 、CDR

期货：一阶段支持股指期货。目前 MATIC 已完成对接的6家期货公司名单：华泰期货、格林大华、永安期货、申万期货、中信期货、银河期货，若有上述任意一家期货的账户，可以直接请客户经理申请添加至 matic。测试环境支持华泰仿真期货、simnow 等，需要客户自行申请，仿真账号下来后提供给技术支持绑定。

期权：深交所即将推出的300ETF 期权是可以接受程序化报备的，需要的话可以联系营业部申请报备。测试环境联系技术支持添加期权资金账号。

13.18 定时



4.5. 定时信号

MQuant 提供两种类型的定时信号，都是通过 run_timely 接口订阅的，一种是订阅周期性的定时信号，例如每 3s 推送一次的定时信号；第二种是订阅某个时刻触发一次的定时信号，例如 14:23:57 秒触发。函数定义如下：

13.18.1 定时函数的执行时间是本地时间么？

run_timely(timer_func, 3, start_time='10:30:30', stop_time=None)
（5） 定时程序和网络没关系
（6） start_time 是本机时间，非交易所时间和 matic 系统显示的时间
本地时间：10:30，交易所时间/matic 时间是 9:57，设置的 start_time 是 10:30，会按本地时间执行

13.18.2 不支持 run_daily 和 run_monthly 函数

run_daily 和 run_monthly 都是不支持的，我们现在只提供了日内的策略。
因为 mquant 是嵌在 matic 软件里面的一个子系统，而 matic 每天会强制重启（大致是在12点到1点之间，另外 matic 登录有效期是8小时，在线时间8小时会强制登出的），所以现在只做了日内的定时函数，日间的

定时函数没有意义，定义接口是为了兼容聚宽的接口。
日间策略：后续我们有计划迁移到云端，但是真正提供出来应该要到明年了

13.18.3 1s 定时信号代码没有执行完会怎么样

信号会积压，等定时函数返回后，新的定时信号会继续推送过来，不会丢掉，所以尽量不要在很短的定时函数中进行大量耗时的处理

13.18.4 外部能否停止定时函数

可通过加判断或者写入时间参数，外部不能停止

13.19 mquant 怎么获取停牌股票

1. 现在行情增加了行情状态，可以查到停牌的股票，self.trading_phase_code = TradingPhaseCode.TemporarilySuspended # 行情状态（TemporarilySuspended = "8" # 临时停牌
2.提供查询停牌股票的接口 get_symbol_detial 返回对象证券详情 SecurityDetials.TradingPhaseCode=8

13.20 行情

13.20.1 如何订阅和处理行情，逐笔委托和成交

（22） 订阅行情
subscribe(["000001.SZ"])#订阅了就会有推送 handle_tick，但是 get_current_tick 之前必选先订阅
subscribe(["600000.SH"])
subscribe('10001979.SH')#期权
subscribe('IF1911.CF')#股指期货

（23） 订阅逐笔委托： subscribe(symbolList,'record_order')#逐笔委托
（24） 订阅逐笔成交：subscribe(["002481.SZ"],'record_transaction')
（25） 处理逐笔委托 ： handle_order_record(context, order_record, msg_type):
（26） 处理逐笔成交 ：handle_record_transaction(context, record_transaction, msg_type):
（27） 处理行情： handle_tick(context, tick, msg_type):

13.20.2 取消订阅 unsubscribe_all 和 unsubscribe

unsubscribe_all：所有信号都取消订阅了,包含订单回报
unsubscribe：取消 tick,逐笔委托和成交的订阅

13.20.3 可以在非初始化函数订阅么

可以，订阅可以单独运行，建议 9 点 15 启动程序，否则后台系统还没初始化好，可能存在问题

13.20.4 集合竞价提供行情以及买卖盘口排序原则

实盘集合竞价 9 点 15 有实时行情，测试环境 9 点 30 之后提供测试

扩展：
集合竞价期间，tick 数据买卖盘口是按照什么原则排序的？
集合竞价期间是虚拟成交，买卖一是能够匹配的买卖量及价格，买卖二是不能匹配的量，因为不能匹配，是没有价格的。
交易状态在开盘集合竞价时段内，其中申买价一和申卖价一为虚拟参考价格，申买量一和申卖量一为行情发布时刻的虚拟匹配量，申买量二为买方虚拟未匹配量，申卖量二为卖方虚拟未匹配量
总结：集合竞价买一卖一为虚拟价格，买二卖二价格为 0

13.20.5 支持指数、期权和期货的行情么

支持指数、期权期货的行情，股指期货还只有一档行情，后续会提供五档
指数行情：只支持了中证指数、国证指数、申万指数，恒生的暂时没支持
期货行情：现在支持中金所股指期货和国债期货的行情和交易，商品期货需要对接期货系统交易，未来会做全品种

13.20.6 支持全市场实时行情订阅吗？

最好不要订阅全市场的，这对你的网络带宽和机器处理能力都有比较高的要求
，另外订阅大量数据处理能力也很重要
方式：先通过 get_symbol_list 获取全市场代码，再传入 subscribe 订阅

13.20.7 支持 level2 行情吗？

支持，tick 本身就是 level2 的， level2 一个是 10 档，还有一些字段是 level2 没有的，比如委托笔数，买卖一档的挂单队列，此外还有逐笔委托和逐笔成交

13.20.8 支持历史 level-1 tick 和多种粒度的历史 K 线下载

13.20.9 Tick 的 datetime 是本地时间还是交易所时间？

tick.datetime 是交易所的时间戳；

注意：建议客户通过 datetime 函数打印本地时间，并将 tick 和逐笔数据都打印一下，通过时间戳比较下本地时间和交易所时间的时间差；

代码示例：

```
print('recv tick msg', tick.code, tick.datetime, datetime.datetime.now(),tick.open)
```

datetime.datetime.now()：操作系统函数，是本地时间

13.20.10 行情的推送速度受什么的影响？

订阅股票的数量，和开启策略的数量无关

13.20.11 行情有延迟么

一般不会太大的延迟，主要和你实际使用的网络环境相关

13.20.12 tick 推送的间隔是多长时间？

活跃的标的正常都是3s 一个，上交所不活跃的标的时间间隔会超过1分钟，观察来看在63秒以内吧。深交所时间间隔比较规整

13.20.13 订阅逐笔和 tick 哪个快？

逐笔是有成交就会推送，l2 是标准的 3s 一个 tick，逐笔的肯定快一些
如果想获取未推送的 tick，可以订阅逐笔委托和成交

13.20.14 get_current_tick

13.20.14.1 get_current_tick()获取失败返回 tick 对象

get_current_tick()如果获取失败，是抛出异常还是返回 None？
会打印错误日志，同时返回的数据是一个 Tick 对象，里面的值是空的
参考手册 4.1（3）

```
def get_current_tick(security):
    接口功能：获取最新的 tick 数据
    参数说明：
        security:标的代码， 支持股票、商品期货和股指期货。 不可以使用主力合约和指数合约代码
    返回值：
        security 对应的最新 Tick 对象 需要判断 tick 里面的代码是否=security，如果不相等，说明获取
    失败
```

13.20.14.2 get_current_tick(security)每次只能获取一只股票的 tick 数据吗？

是的，单次只支持一支股票

13.20.14.3 get_current_tick()提示组装行情失败

get_current_tick 是获取的是客户端缓存的最新的 tick，客户端收到推送后会将最新 tick 缓存下来

此函数不可单独使用，必须订阅 subscribe 才能取到的，这个是从缓存里面取的，注意要订阅需要获取

行情的股票

13.21 回调函数

13.21.1 回报函数 handle_order_report 需要注意的点

① 当订单状态或者订单成交数量改变（分笔成交）的时候，就会回调 handle_order_report

② 非 Mquant 报单也能收到回报，需要勾选账号级交易推送（matic 系统外委托同步最大延迟为 5s，一般在 2s 以内）

③ 撤单没有回报，能收到撤单导致的原单回报

④ 获取分笔成交的成交价格，可通过 handle_execution_report 成交回报查每笔成交情况

⑤ 被风控拦截的订单没有报到柜台，只会有废单推送，不会有实际的订单，不会在委托和成交记录查询到，废单原因 cancel_info 不带信息， 界面日志和系统日志里面可以看到废单原因

⑥ 通过 get_orders 查询订单查到当前资金账号所有满足条件的订单进行撤单

13.21.2 订单新增报单失败和撤单失败状态

MQuant 新增了报单失败和撤单失败的订单回报推送，报单失败推送 reject 状态，撤单失败推送 cancel_failed 状态，属于新增推送，之前的推送不受影响。

注意：MQuant 订单状态不能保证一定按序，存在极小的可能订单状态顺序会乱，需要策略自行处理

13.21.3 一个订单提交之后第一个 open 的状态正常多久能推送回来？

正常情况下是很快的，毫秒级

13.21.4 回调函数是否存在并发执行的可能？

不存在，初始化函数和其他回调函数不在一个线程执行，除初始化函数之外，其他的都是顺序执行的，比如你在 callback1 里面修改了 context 里面的一个变量，然后进到 callback2，那么 callback2 里面取到的一定是修改过的，不可能是未修改的或者修改了一半的

13.22 两融

13.22.1 查询资金和持仓

s=MarginTradeHandler.get_margin_assert()#查询信用资产

信用资产是定时查询，30s 更新一次

注意：添加主动刷新信用资产的接口意义不大，即使提供也不会突破柜台同步查询的限制。

13.22.2 两融报单的速度怎么样

我们系统内部处理很快，主要是融资买入涉及柜台头寸的计算，这个我们没法控制

13.22.3 在融资融券账户会被限制报单数量么

你可以试试 matic 的组合交易，我们系统内部没有做限制，我刚问过柜台，也没做限制，只不过融资买入会慢一些

13.22.4 担保品买卖函数中单标的怎么取

取 list 第一条

13.22.5 融券卖出的仓位要用什么接口查看呢？

可以通过 get_margin_contract 获取。

13.22.6 融券卖出未成交订单怎么查看？

未成订单是通过 get_open_orders 接口，里面有个 account_type 的字段，指定信用账户可以查两融未成订单

13.22.7 买券卖券、卖券还款函数无返回值（待新版本）

目前的 api 接口返回的是 None

13.22.8 怎么查剩余可融券额度？

可融券数量：get_margin_security_info 查询融券标的信息，返回对象字段中 loan_available_aty 为融券可用额度

13.22.9 实时计算的可交易的额度

get_avaliable_qty_for_trade 是计算可交易的数量，返回指定条件交易的最大可交易数量
计算融券卖出的额度：融券卖出只支持限价和专项头寸的

```
print(MarginTradeHandler.get_avaliable_qty_for_trade('
000001.SZ',EntrustType.creditMargin, side=OrderSide.SELL, style=LimitOrderStyle(17.73), position_prop=PositionT
ype.vip))
```

13.23 报单/撤单

13.23.1 统一报单接口：支持 A 股、两融、期货、期权报单

order_normal(order_request,account_type=AccountType.normal,batch_no=-1,last_batch_flag=0):
接口功能：
提供统一报单接口，支持 A 股、两融、期货、期权报单，不包含 ETF 申赎、直接还款、备兑锁定解锁等

13.23.2 cancel_order 和 cancel_orders

13.23.2.1 返回对象

cancel_order：成功返回 Order，失败返回 None，
cancel_orders：成功接口返回 list，列表元素是 Order 对象，失败返回的是空 list

13.23.2.2 cancel_orders() 和 orders()并列使用

cancel_orders 和 orders 都是异步执行的，并不能保证函数调用结束就撤单成功；如果需要严格等待撤单成功再报单，需要等待原单状态变成已撤或部撤再报单；为了避免订单回报消息推丢，保险的做法还应该订阅一个定时信号，定时查询一次原单的状态

13.23.3 策略未初始化成功，禁止报单

不可以在初始化阶段报单；
初始化、停止是单独的线程，其他回调都在一个线程。假如在 hand_tick 中收到行情的时候报单，因为订阅是在初始化阶段订阅的，这时候会没有初始化完，所以不可以报单。
参考手册4.4（1）初始化阶段不允许报单和撤单

（1） 初始化函数
函数定义：def initialize(context)
调用时间：策略初始化阶段调用
是否必须：是
特别说明：策略在初始化函数中，一般进行一些参数存储、信号订阅等初始化操作，允许读取文件，不允许报单、撤单等交易行为和访问网络
示例：

13.23.4 报单买卖方向

买卖方向是由数量的正负来决定的，正为买，负为卖

13.23.5 单笔报单和批量报单的区别？

批量会快一些，不过也要考虑到组装订单的耗时，具体可以测试一下，另外批量报单为了提升性能，我们会拆成最多20笔一组分多组报出去。

13.23.6 报单是异步还是同步报单？

异步报单，不用等第一个成交第二个再成交

13.23.7 报单之后立即撤单可以么？

报单和撤单都是异步执行的，并不能保证函数调用结束就撤单成功

13.23.8 报单和撤单返回 id 不同

对系统来说报单和撤单都是委托，生成不同的 id

13.23.9 Mquant 订单未完成，如何补单

在订单回报函数对未完成的订单进行撤单（已报和部成可以撤单）
订单撤单成功之后，部撤的订单需要注意下：订单 cancelled 状态并不能判断为最终状态，还要看看已成数量+已撤数量是不是等于委托数量，如果不等，可能是撤单成交先回来了，但是最后一笔报单成交还没回来，此时 filled 字段是不准确的，但是撤单数量一定是准确的，所以这种情况可以直接按照撤单数量补单

13.23.10 Mquant 怎么判断订单终结

终结状态：已成、已撤、废单，部撤。
部撤的状态要注意下：因为报单成交和撤单成交柜台是多线程推送，所以不能保证谁先到，如果先后到撤单成交，状态会改成部撤，撤单数量是对的，但是成交数量可能随着报单成交收到逐步更新成正确的值，这时部撤可能会有多条记录；
我们判断部撤的时候，状态是部撤，并且委托数量=撤单数量+成交数量，判断订单终结
例如：mquant 的 cancelled（已撤/部撤）状态并不能判断为最终状态，此时 filled 字段是不准确的，但是撤单数量始终是准确的，还要看看已成数量+已撤数量是不是等于委托数量

13.23.11 日志的订单状态说明

Matic 系统定义的状态，这张图，结合日志判断，但是 mquant 代码里面的判断还是根据手册：

```
namespace OrderStatus {
const QString pendingNew = "1"; ///待报
const QString newOrder = "2"; ///已报交易所
const QString pendingCancel = "3"; ///已报待撤
const QString partiallyFilledForCancel = "4"; ///部分成交待撤
const QString partiallyCancel = "5"; ///部分撤销
const QString canceled = "6"; ///已撤
const QString partiallyFilled = "7"; ///部分成交
const QString filled = "8"; ///全部成交
const QString reject = "9"; ///废单
const QString cancelFailed = "E"; ///撤废

const QString calucated = "B"; ///表示成交回报
const QString confirmed = "V"; ///确认
}
```

策略里面的状态定义：
# 订单新创建未委托，用于盘前/隔夜单，订单在开盘时变为 open 状态开始撮合(待报) new = 8
# 订单未完成, 无任何成交（已报）
open = 0
# 订单未完成, 部分成交（部成）
filled = 1

# 订单完成，已撤销，可能有成交，需要看 Order.filled 字段（已撤/部撤）
canceled = 2
# 订单完成，交易所已拒绝（废单）
rejected = 3
# 订单完成，全部成交, Order.filled 等于 Order.amount（已成）
held = 4
# 订单取消中，只有实盘会出现，回测/模拟不会出现这个状态（已报待撤/部成待撤
pending_cancel = 9

13.23.12 order 对象字段

13.23.12.1 订单的买卖方向

is_buy 暂不提供，可以是 order.Side 判断
输出 order.side=long ,代表买入
Short：代表卖出

13.23.12.2 order_id

是系统内部订单号，不是柜台委托编号
参考手册 Order 数据结构

self.order_id = '' # 订单 ID，为 M-Quant 系统内部生成的订单 id，非委托编号，下单/撤单立即返回

13.23.12.3 orig_order_id

这个字段现在没用，撤单委托本身我们没有开放

self.orig_order_id = -1 # 原始订单号 撤单订单有效，目前保留字段

13.23.12.4 add_time

self.add_time = None # 订单创建时间, datetime.datetime 对象，为后台时间

self.create_time = datetime.datetime.now()

create_time 不要使用,用 add_time
订单的本地创建时间如果需要的话要自行维护

13.23.12.5 订单的成交时间

后台成交时间：

如果要后台时间只能通过查询成交

成交到客户端的时间：需要自己创建

13.23.13 委托时间和本地时间和后台时间

后台系统时间获取不到，可以 拿行情时间戳和本地时间对比下，行情时间和交易时间差别不大，找找规律，延迟应该在 1s 以内

监控界面显示的是后台时间，和你本地机器可能存在时间差
监控界面上除日志之外显示的都是后台时间
报单你如果打印的本地时间就是本地时间
策略日志的委托时间是您本机时间，订单列表中的时间是后台订单系统时间
matic 的委托时间是和交易所对过时的，可能和您本地有一定的误差（交易所时间和北京时间也有误差）

13.23.14 报单 order 没有响应？

order 报单没响应一般是因为诸如资金不足、持仓不足，参数填写错误等原因导致报单不成功，现在在策略运行界面有订单的日志打印出来可以查看详细委托信息

13.23.15 废单

13.23.15.1 资金不足批量报单，会全部废单么

不同的市场会分开控：比如上海和深圳的放在一个批次里面，可能上海的会成功，深圳的会都失败；还会分版块，例如主板和创业板也是分开控的。

注意：

注意：

1.资金不是很充裕的情况下建议尽量选择限价单报单，市价单会按照涨停价冻结的
2.Matic 不检查股票是否可用以及资金是否够用的，在后台交易系统才会检查
3.可用资金有的，context.portfolio.available_cash，但是和实际情况可能有延迟，正常情况延迟2s 以内
4.最多可用买多少股票是自己计算的，建议留一点余量

13.23.15.2 卖出没有的持仓股票，订单查询不到？

1. 持仓不足或者资金不足，柜台拒单之后会废单，废单之后，订单 id（这个订单 id 是 mquant 生成的）是 order 这个函数返回值里面就有，但是 order 返回时并不知道这笔订单废单了
2. 废单这种订单状态接收要在 handle_order_report 里面接收，接收的订单 Order 对象不为空，可以查到废单的委托信息，例如 symbol，amount 和 entrust_price 等委托的一些信息。
3. 如果是柜台或者交易所的废单通过 get_orders 查询也能查到

13.23.15.3 mquant 中废单怎么查询

有废单的时候麻烦看看界面的委托列表中是否有这笔废单，如果有，通过 get_orders 就能查到，如果没有，那么只能通过 handle_order_report 来接收（查不到：被 matic 拒单了，例如自成交)

13.23.16 报单报价格校验不通过

期权精确到后四位，股票到后两位，基金是三位
上海债券是 2 位，深圳债券是 3 位

13.23.17 Mquant 报单什么时候返回 None?

现在返回 None 只有没有勾选 A 股资金账号会出现，废单之后 Order 也不会返回 None

13.23.18 提供自动拆单

13.24 查询订单 get_orders

get_orders(order_id='', security='', status=None, page_no=1, page_size=1000, only_this_inst=True, msg_type=-1 , account_type = AccountType.normal, inst_id=’’):

1. 按状态查询：不应该传递的是数字例如 status=4，应该传递 status=OrderStatus.filled
2. get_orders：only_this_inst=true，只能取当前实例的订单
   only_this_inst=False：可以取所有的订单
   only_this_inst=False 并且指定实例 id（inst_id）：可以取指定实例的订单
3. 如果原单是中间状态部成，变成终结状态已成或者部撤，只会查到部撤或者已成的记录，查询时只会查询到当前订单最新状态的记录。

13.25 如何在界面实时监控订单状态？

现在在策略运行界面有订单的日志打印出来可以查看详细委托信息

13.26 全局变量 g 怎么用

g 就是一个全局变量，没有任何成员的，主要用于给你们存储数据，里面的内容你自己定义就可以，这个变量的成员是用户自己添加的，没有任何结构，可以参考示例策略中 g 的用法

13.27 日志

13.27.1 日志可以设置为只写入文件？

可以用自定义日志打印一些非必要的 debug 信息，这些信息会写到文件里面，不会展示在界面上，界面上展示重要的日志信息就行（推荐）；或者调用接口 def enable_console_output(enable, label='')
接口功能：设置是否允许 log 模块打印的日志在控制台输出,默认系统日志允许输出到控制台，用户自定义日志不输出到控制台

13.27.2 注意事项

1,支持格式化字符
log.info ('%s.2f,%s.2f'%('bb','aa'))
log.info ('{} {} '.format('bb','aa'))
2.不支持打印对象，例如 exception 对象，获取到异常信息字符串的话可以打印出来,
3.不支持打印 list 对象，可以把 list 的 orderId 打印出来
4.日志时间是本地时间
5.log/20190606/HTSRE 打头文件
6.XTader 是 matic 的，HTPEM 是 Matic 进程 MQuant 专用的，HTSRE 是每个策略进程的
7.目前只保留自然日 3 天日志，可在 res 目录下 config.ini 修改 log_maxkeeptime

13.28 控制台窗口不运行了？

先排除下是不是误点了控制台窗口，win10有这个问题，停了按 ctrl+q 也能恢复运行
或者想避免误点，可以
在配置里面隐藏掉控制台窗口：
文件系统根目录下 HtProEngineConfig.ini，ShowSubProcess 设置为 0，重启系统
实盘一般不会调试策略，我们会隐藏窗口，窗口也有容量限制，不能一直 print

备注：误点之后日志的输出和控制台输出都暂停了，策略还在运行，恢复之后，之前的暂停的输出会推送

过来

13.29 查询持仓和资金

13.29.1 查询持仓的两种方式

（3）context.portfolio.positions 是动态获取的，需要指定获取单只股票的持仓才会有数据；获取方式：context.portfolio.positions[symbol] ，参考手册 4..3.2
context.portfolio..long_positions 用法参考 context.portfolio.positions

注意：context.portfolio.positions[security] 如果没有该股票持仓的话，这句代码会返回 None，从而导致策略失败，需要判断下，例如：if not context.portfolio.positions['601688.SH'] is None:

（4）get_positions()：可以获取所有持仓，参考手册 4.3.2 章节
（5）get_positions()： 如果一只标的当日初始仓位不为 0，在交易过程中仓位被卖掉变成 0 了，可以查询到

13.29.2 查询可用资金

资金通过 context.portfolio 结构下的相关成员变量获取
示例：
获取账户可用资金：context.portfolio.available_cash，参考手册 4.3.3

13.29.3 Mquant 资金和持仓是实时更新的么

可用资金和持仓这类数据后台是根据需要实时查询更新的；
但是目前实盘资金和持仓的更新是有延迟的，一般在 1.5s 以内，有时会到 2s，所以强烈建议各位的策略不要强依赖资金和持仓，如果强依赖，建议自己在策略中维护一个虚拟的资金和持仓。此外，强依赖资金和持仓的策略在回测的时候也可能出现相同参数回测结果不一致的问题，原理是一样的

13.29.4 Mquant 获取持仓的接口

1.通过 get_positions_ex 获取单只标的或者所有的持仓，支持所有 Mquant 品种（包括期权和期货），成功返回 list<Position>,失败返回 None

2.通过 context.portfolio.positions[symbol]来获取指定标的的持仓

```
def get_positions(account_type=AccountType.normal, symbol=''):
    """
    查询持仓，不支持期货、期权
    :param account_type:账户类型，默认为A股账户
    :param symbol:标的，如果标的为空，返回dict<symbol,Position>对象，否则，返回Position或者None
    :return:失败返回None,成功返回dict<symbol, Position>, 传入symbol时，成功返回Position对象，失败返回None
    """
    return get_positions_impl(account_type, symbol)


def get_positions_ex(account_type=AccountType.normal, symbol=''):
    """
    获取所有持仓，支持所有MQuant品种
    :param account_type:
    :param symbol:
    :return:失败返回None,成功返回list<Position>
    """
    return get_positions_ex_impl(account_type, symbol)
```

13.30 创建算法失败

第一步，beta 环境后台 ATS 是否正常
第二步，账号算法是否授权
第三步，是否是在实盘模式，而不是模拟模式，模拟交易不支持算法交易

13.31 策略执行监控

13.31.1 策略运行日志要选中实例才会有数据

13.32 暂停策略

暂停策略，不会重新运行初始化函数，策略进程还存在，但是所有信号都不会推送；
策略停止，进程停止

13.33 期货

13.33.1 怎么申请 matic 期货测试

matic 里面的期货账户是期货公司开的，可以在 matic 下单

13.33.2 华泰期货开户

https://www.htfc.com/main/wsyyt/wykh/jrfzjgkh/index.shtml?id=10531
这是华泰期货仿真开户链接，一般 1 到 3 天能申请下来；
初始密码（一般是身份证后 6 位）会发到客户手机号上，要下载一个手机 app 修改初始密码，期货账号**首次登陆必须修改资金密码**，可以通过华泰期货的快期客户端或者华泰期货的 **期赢天下 app** 修改期货的资金

密码，然后 matic 客户端才能校验资金密码

13.33.3 simnow 开户

在 simnow 注册一个期货仿真户，注册地址：http://www.simnow.com.cn/product.action，注册好了之后期货账号**首次登陆必须修改资金密码，**同时需要在 simnow 修改一下初始密码。然后联系技术支持人员把账号提供一下，技术支持会给您绑定 Matic 账号以及开权限。

13.33.4 如果 matic 里面的期货账户是期货公司开的，能再 matic 里下单不？

可以的，菜单需要授权（基础交易=》期货交易）；测试环境可以联系技术支持，实盘联系营业部客户经理。

13.33.5 matic 里是不是华泰期货 还是其他期货公司也可以

生产对接期货柜台：已完成华泰期货、格林大华、永安期货、申万期货、中信期货、银河期货 CTP 柜台对接

Beta 期货仿真交易：华泰期货仿真交易和 Simnow 仿真平台

13.33.6 Matic 期货支持的品种是什么

目前 MATIC 已支持的期货品种是中金所的股指期货和国债期货，其他交易所不支持

13.33.7 假如产品里绑定华泰以外其他期货商的，怎么清算？

都是柜台清算。比如：华泰证券户+格林大华期货，股票部分华泰证券柜台清算，期货部分格林大华期货柜台清算

13.33.8 手工期货交易

基础交易=》期货交易，测试环境可以联系技术支持，实盘联系营业部客户经理。

13.33.9 测试环境期货支持哪些投保类型

目前测试环境手工期货和 mquant 都只支持投机模式，套利和保值模式现不支持。

13.33.10 程序化期货交易

mquant 提供期货单笔下单接口 order_future 和统一报单 order_normal

13.33.11 Matic 实盘中绑定期货以后，没有办法显示股票资产和期货资产合计的总资产是吗？

暂时还看不到期货+证券的总市值

13.33.12 Mquant 支持指数、期权和期货的行情么

支持指数、期权期货的行情，股指期货还只有一档行情，期权目前已降至 level-1。
指数行情：只支持了中证指数、国证指数、申万指数，恒生的暂时没支持
期货行情：现在支持中金所股指期货和国债期货的行情和交易，商品期货需要对接期货系统交易，未来会做全品种

13.34 期权

13.34.1 怎么申请 matic 期权测试

1.交易所要求程序化报备，现在没用开报备通道，报备开始的时候可以通过营业部咨询下网络金融部-周全老师，可以先测试
2.测试账号可以联系技术支持分配 88888 开头的期权测试账号
3.技术支持添加期权账号的界面注意要绑定 A 股账号（客户不用管）

注意：目前系统的期权功能，T 日开户，要 T+1 日才可以交易，所以明天才可以交易
测试环境新加的账号，T+1 日才可以交易 。

13.34.2 手工期权支持么

实盘支持手工期权交易，可以联系营业部申请。
基础交易=》期权交易；特色交易=》期权组合交易

13.34.3 期权程序化支持么

目前深交所开准备放开 300etf 期权的报备
Mapis:有 etf 期权请求接口，需要满足私募产品 5 个亿的要求
Mquant（推荐）:深交所推出的 300ETF 期权测试环境支持，其他期权的暂不支持，期权下单接口 order_option
期货公司有期权，期权也可以对接 ctp

13.34.4 Mquant 支持期权行情么

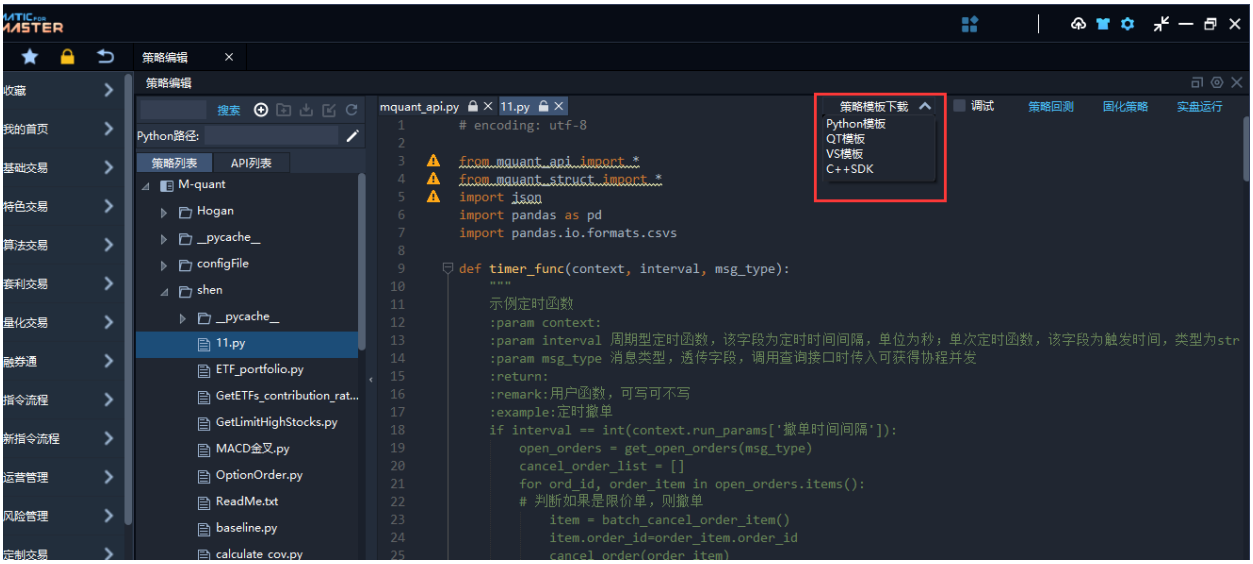
支持

13.34.5 get_symbol_list 获取所有合约的代码

个股期权不支持，支持 ETF 期权
1. get_symbol_list(ExchangeType.SZ, SecurityType.OptionType, '')
   get_symbol_list 的第三个参数默认 A 股，所以取不到，第三个参数填空字符串就可以取到了
2.symbolList=get_symbol_list('SZ',7,'07002')
3.get_symbol_list(ExchangeType.SZ, SecurityType.OptionType, SecuritySubType.ETFoption)

13.35 C++版本

13.35.1 C++的模板工程文件在哪里下载



13.35.2 怎么运行 C++版本

在策略执行监控中，鼠标右键策略管理空白处，新建策略，加载本地编译好的 C++策略

13.35.3 用 vs2019 编译 c++策略会编不过，需要在配置里面加上这个，或者 vs 配置里面配一下

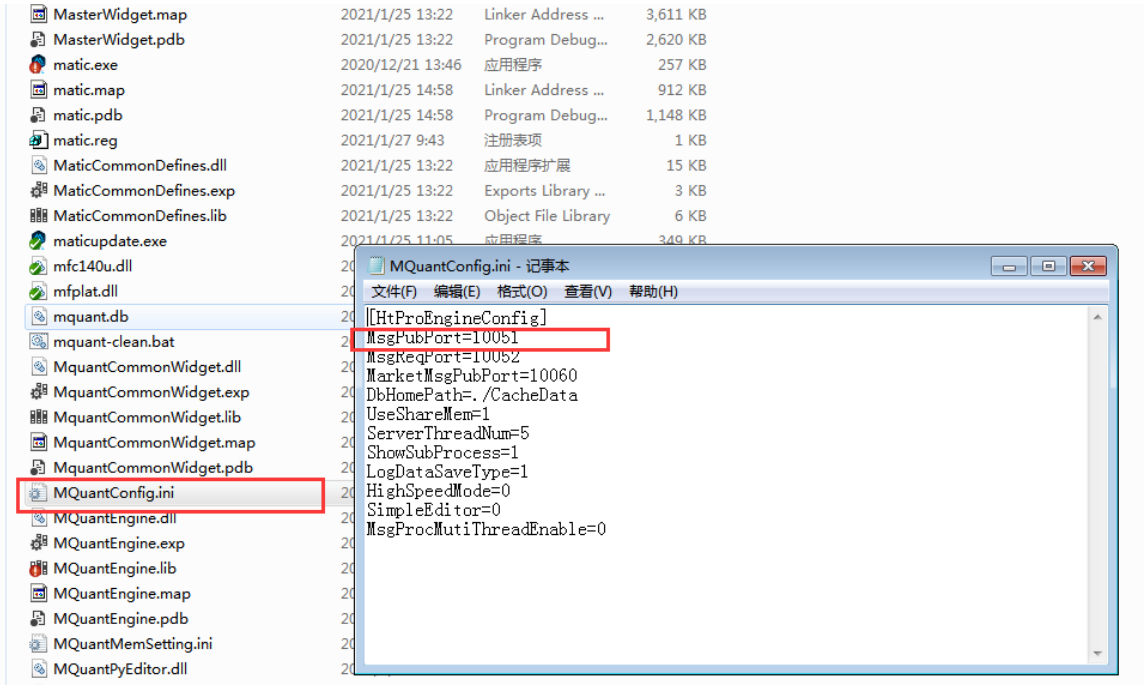
```
unix {
    target.path = /usr/lib
    INSTALLS += target
}
QMAKE_CXXFLAGS += -permissive

win32: LIBS += -L$$PWD/lib/x64/ -lHtCxxAdapter
#win32: LIBS += $$PWD/lib/x64/HtCxxAdapter.lib

INCLUDEPATH += $$PWD//lib/x64
DEPENDPATH += $$PWD//lib/x64
```

13.36 软件崩溃自助排查

目前遇到两类情况可能导致打开 MQuant 菜单就会崩溃：

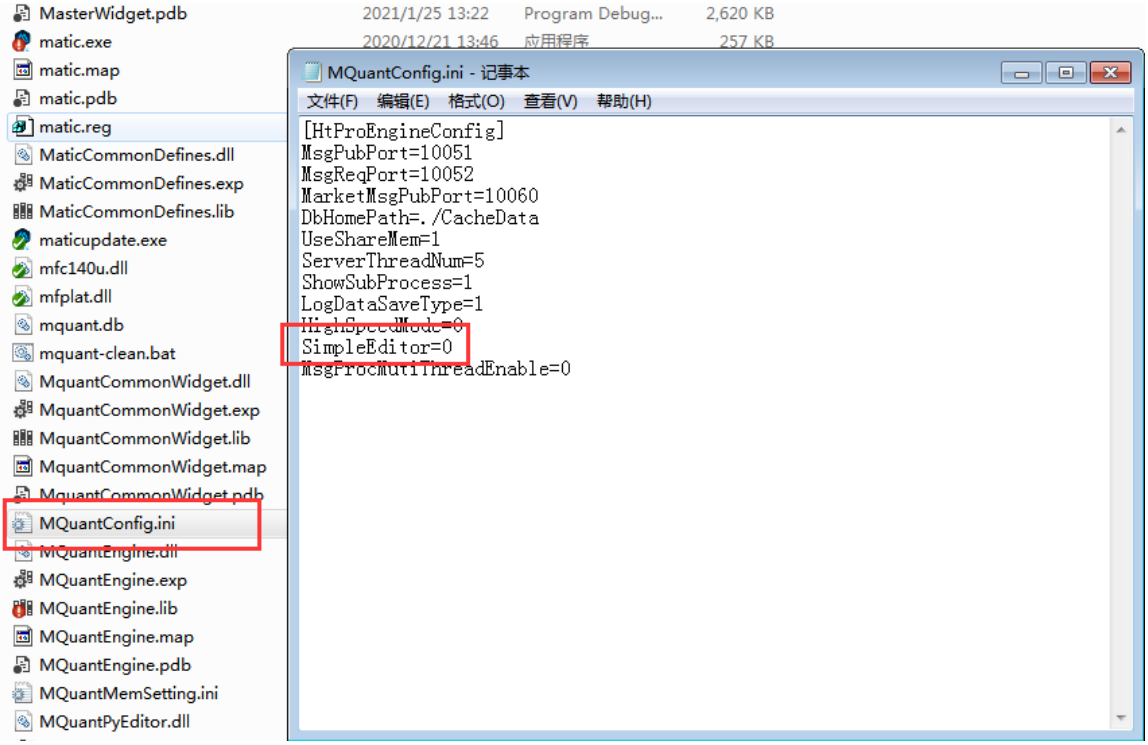
（1） 情况一：内部通信端口被占用

解决方案：打开 matic 软件根目录下的 MQuantConfig.ini 配置文件，修改 MsgPubPort 配置项为 1024-65535 之间的其他端口，建议修改为较小端口号，如 3841。



（2） 情况二：本机环境无法初始化 python 解释器

解决方案：这种问题目前还未明确原因，目前的解决方案是打开菜单时先禁用 python 解释器，后续使用 C++语言开发策略。配置方式为：打开 matic 软件根目录下的 MQuantConfig.ini 配置文件，修改 SimpleEditor 配置项为 1,如下图所示。



14 温馨提示

测试环境主要的定位是功能性的验证，数据方面不比生产环境有保障。在测试过程中如若遇到查 K 线数据

不全，行情延迟比实盘大很多，报单提示标的错误，回测提示 no market data 等，均是测试环境数据不全的原因。如果测试对数据要求很高，麻烦联系营业部申请生产账号上生产测试，敬请理解，谢谢。