关于米筐科技(RiceQuant)

米筐科技专注于为用户提供快速便捷、功能强大的量化交易和分析工具。用户可以使用基于浏览器（网上回测平台）或本地化（RQAlpha等项目）的米筐科技产品，随时、随地开发自己的交易策略，验证自己的投资思路。我们对数据质量、回测系统、模型算法、交互设计、用户界面和用户安全等方面进行了持之以恒的完善，务求使用户获得最佳的产品使用体验。目前，我们已经提供了策略回测和实时模拟交易功能；在将来，我们会进一步提供实盘交易支持，使用户在我们的产品平台上，能够一站式地完成交易策略的开发、测试和实盘执行。

我们是一个创业公司，由一群朝气蓬勃，又有丰富业界经验的年轻人构成。我们渴望和用户一起进步和成长，见证量化投资在中国市场上的普及和发展。如果您有任何产品或服务上的建议，或商业上的合作意向，欢迎随时和我们联系。我们认真对待任何来自用户和合作伙伴的反馈意见，并始终视之为我们进步的动力。

如何使用本文档

在本文档中，我们详细介绍了平台的各项功能和使用方法。由于内容较多，在浏览本文档的时候，我们建议您多使用 "ctrl + f" 的快捷键组合，快速定位到感兴趣的内容上。

千里之行，始于足下。在学习编程的时候，我们都是从打印一句 "Hello World" 开始，踏入到奇妙的程序世界；同样，在本文档的第一部分，我们准备了一个类似 "Hello World" 的简单策略实例，帮助新用户了解量化策略，以及如何使用我们的平台。

在您亲自动手测试上述 "Hello World" 策略，体会到编写和测试交易策略的奇妙和喜悦以后，就可以开始浏览本文档的余下内容：

1. "数据"部分，介绍了平台上可供使用的、丰富多样的数据
2. "回测设置"部分，介绍了回测系统的各项默认设置（例如撮合方式、滑点和期货交易费用等）
3. "进行回测"部分，介绍用户如何使用我们的在线IDE进行策略开发和回测，以及回测过程中可供使用的各个函数和对象；
4. "回测结果分析"部分，我们介绍了策略评估的各类核心量化指标（收益、风险和风险调整后收益），以及进阶分析功能（月度收益、持仓情况等）
5. "策略实例"部分，介绍了更多、更具体的策略实例，帮助你了解常见的量化策略开发思路，和如何灵活使用我们提供的数据和功能
6. "实时模拟交易"部分，介绍如何设定实时模拟交易，以更好地评估策略的实盘表现
7. "外部数据和 Python 模块"部分，介绍了平台支持的外部数据源和 Python模块，以及如何引入自定义的Python模块

希望通过我们的文档和平台，您能够逐步成长为一个优秀的量化策略开发者。如果您对我们文档或平台有任何建议，欢迎随时联系我们；如果希望分享编写策略过程中的疑问或心得体会，欢迎来我们的用户社区一起交流讨论；如果你通过实践，获得了表现出色的量化

Python策略Hello World

以下的策略是最简单的一个买入并持有平安银行（buy and hold）的展示，回测基本流程如下：

1. 在创建策略之后，您需要指定回测的起止日期、初始资金以及回测频率。
2. 在init方法中实现策略初始化逻辑，例如设置合约池、佣金率、保证金率等操作。
3. 可以选择在before_trading进行一些每日开盘之前的操作，比如获取历史行情做一些数据预处理，获取当前账户资金等。
4. 在handle_bar方法中实现策略具体逻辑，包括交易信号的产生、订单的创建等。handle_bar内的逻辑会在每次bar数据更新的时候被触发。
5. 回测完成后，在'回测结果'页面会展示回测的仓位、盈亏、交易、风险等信息。用户可以导出报告方便分析。

您可以点击右上角的clone按钮复制到自己的策略列表中进行修改和测试，非常简单。

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。

# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    context.s1 = "000001.XSHE"
    # order是否被发送出去
    context.fired = False


# 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更新
def handle_bar(context, bar_dict):
    # 开始编写你的主要的算法逻辑

    # bar_dict[order_book_id] 可以拿到某个证券的bar信息
    # context.portfolio 可以拿到现在的投资组合信息

    # 使用order_shares方法进行下单

    # TODO: 开始编写你的算法吧！
    if not context.fired:
        # order_percent并且传入1代表买入该股票并且使其占有投资组合的100%
        order_percent(context.s1, 1)
        context.fired = True
```

创建新策略

创建代码策略

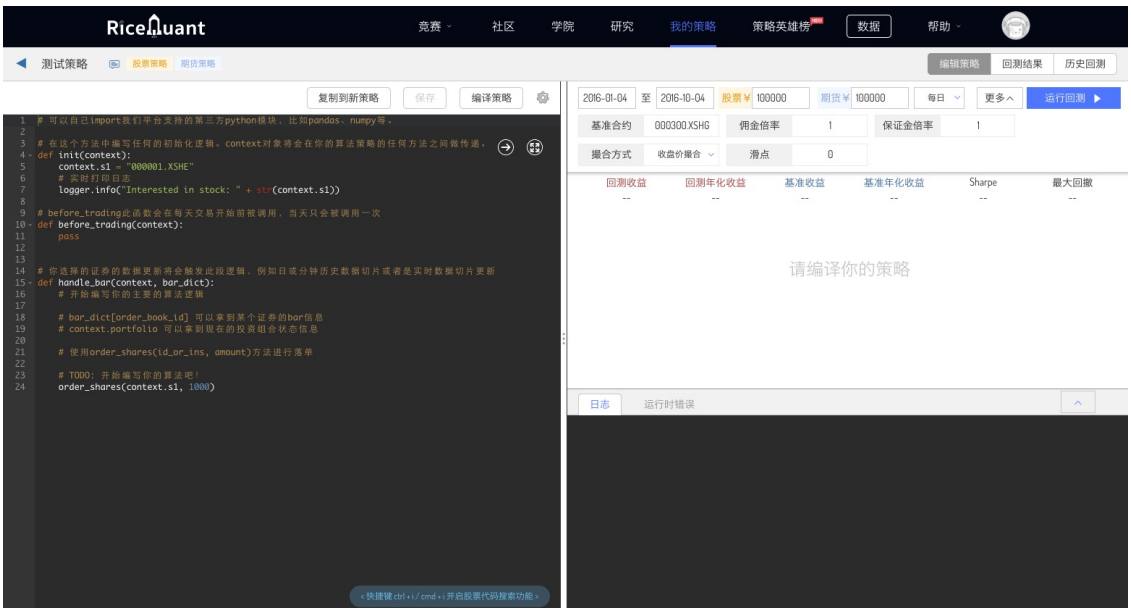
您可以通过在'我的策略'下点击'创建新策略'来创建一个新的回测，如下图：

运行中的策略

需要注意，您在首次创建策略的时候需要指定策略交易的品种（股票，期货，或者两者混合）。需要注意，一旦策略创建完毕，交易品种将不能变更。

运行回测

策略算法编辑页面是运行回测的入口，如下图所示：



您可以在策略编辑页面中进行以下参数的设置： 基础参数：

- 开始日期：回测期间的开始日期
- 结束日期：回测期间的结束日期 - 如果回测结束日期在*今天*之后，将会自动使用最后一天的历史数据
- 起始资金：回测的起始资金 - 您将使用多少钱去投资策略
- 回测频率：可以选择日回测以及分钟回测

高阶设置：

- 基准合约：设置策略表现的对照基准
- 佣金倍率：设置策略佣金在默认佣金基础上的倍数
- 保证金倍率：设置策略保证金在默认保证金率基础上的倍数
- 撮合方式：可以选择'当前bar收盘'或'下一bar开盘'
- 滑点：交易中理想成交价和实际成交价的差异。在进行策略回测时，用户可以设置一个滑点参数，来提高买入价或降低卖出价，模拟实际交易中出现的滑点。该参数取值在 [0, 1) 之间。例如，当股票市价为10元，参数设为0.1，则回测时买入价为10 + 10 * 0.1 = 11元
- 性能分析：勾选该选项之后，策略回测性能分析功能将被开启

- **编译策略** - 点击"编译策略"将会使我们的后台实时编译您的算法策略来查询您的策略是否有编译错误并且可以获得一个策略收益的预览。编译策略会比完整的回测要快，因此您可以试用这个功能在完整回测前先验证下是否有简单的编译或者运行错误。
- **运行回测** - 点击"运行回测"将会带您进入完整的回测结果页面。如果您的策略有编译/运行错误，您将会被停留在改编辑页面以让您先解决问题。

复权机制

RQPro 回测使用的数据采用了 “动态复权” 方式, 以策略回测当前日期为基准进行前复权。

举例来说, 平安银行最近两年的分红派息时间为:

除权除息日	复权因子
2015-04-13	1.210683
2016-06-16	1.217847

假设当前的回测日期为2015-04-14, 那么在此时通过 `history_bars()` 获取4天的历史日线收盘价序列, 返回数据如下表中间列展示。 由于回测当前日期为15年, 2016年的分红拆分当时并未发生, 所以16年的分红拆分事件并未影响回测中获取的价格序列水平。而通过 `get_price()` 获取的前数据则考虑了目标8股票历史上所有的分红派息情况。

回测中这种数据处理方式保证了成交价格全部为市场当时的实际成交价格, 尽量贴近当时市场的实际情况。

日期	原始价格	分段前复权价格 (回测中使用)	前复权价格
2015-04-09	18.00	14.8682	12.2086
2015-04-10	19.80	16.3550	13.4294
2015-04-13	16.54	16.54	13.5813
2015-04- 14(策略当前回测日期)	16.30	16.30	13.3843

基准合约

通过引入基准合约，您可以将策略的表现与基准进行对比，以衡量一定时间内策略的超额表现以及相关风险调整收益指标。

您可以在策略编辑页面"更多"选项下进行基准合约的设置，基准合约可以设置为空。在初次创建策略的时候，包含股票的策略默认基准合约都是**沪深300指数000300.XSHG**；单独期货策略默认没有基准合约。

1。

调整依据基准是交易所规定的最低保证金比例，可以通过instruments这一方法查询到。您可以在策略编辑页面"更多"选项下进行保证金倍率设置。该设置在单一股票类型策略中不会出现。

交易费用

中国A股市场交易费部分主要包含券商手续费和印花税两部分，期货交易主要为佣金费用。

在新版策略框架中，原有通过context.commission设置佣金费率的方式已经被废弃，不再生效。您需要在策略参数高级设置界面中进行佣金倍率的设置，1即代表为默认佣金费率的一倍。该倍率不会影响最低佣金以及印花税的收取标准。

- 中国A股交易费用 A股市场交易费用主要由佣金和印花税两部分组成，佣金目前为双边收费，每笔委托最小收取标准为5元。默认佣金费率是成交额的万分之8，即0.0008。如果设置倍率为10，则实际影响策略的佣金费率就变成了0.008。该倍率并不会影响5元的最小佣金水平。印花税只对卖出方单边征收，对买入方不征税，目前按照成交金额的0.1%收取。**由于是强制征收，印花税已经默认加入到我们的收益计算中。**
- 中国期货交易费用 期货交易费用只包括佣金这一个组成部分。佣金收取方式比较复杂，收取方式有按照名义价值的一定比例收取和按照成交合约张数收取两种方式，同时个别合约区分平今仓费率。初始默认费率表如下请参考：

品种	交易所	佣金类型	回测佣金费率	回测平今费率
铝AL	上期所	按成交量	3	0
锡SN	上期所	按成交量	3	0
橡胶RU	上期所	按成交额	0.00045	0.00045
线材WR	上期所	按成交额	0.00004	0.00004
螺纹钢RB	上期所	按成交额	0.000045	0
燃油FU	上期所	按成交额	0.00002	0.00002
金AU	上期所	按成交量	10	0
铜CU	上期所	按成交额	0.000025	0
银AG	上期所	按成交额	0.00005	0
铅PB	上期所	按成交额	0.00004	0
镍NI	上期所	按成交量	6	0
热轧卷板HC	上期所	按成交额	0.00004	0
锌ZN	上期所	按成交量	3	0

棕榈油P	大商所	按成交量	2.5	0
细木工板BB	大商所	按成交额	0.0001	0.00005
鸡蛋JD	大商所	按成交额	0.00015	0.00015
焦炭J	大商所	按成交额	0.00006	0.00003
聚乙烯L	大商所	按成交量	2	0
聚丙烯PP	大商所	按成交额	0.00005	0.00025
铁矿石I	大商所	按成交额	0.00006	0.00003
豆粕M	大商所	按成交量	1.5	0
玉米C	大商所	按成交量	1.2	0
焦煤JM	大商所	按成交额	0.00006	0.00003
中密度纤维板FB	大商所	按成交额	0.0001	0.00005
玉米淀粉CS	大商所	按成交量	1.5	0
豆一A	大商所	按成交量	2	0
豆二B	大商所	按成交量	2	2
聚氯乙烯V	大商所	按成交量	2	0
豆油Y	大商所	按成交量	2.5	0
锰硅SM	郑商所	按成交量	3	0
白糖SR	郑商所	按成交量	3	0
菜籽粕RM	郑商所	按成交量	1.5	0
油菜籽RS	郑商所	按成交量	2	0
早籼稻RI(ER)	郑商所	按成交量	2.5	2.5
TA	郑商所	按成交量	3	3
动力煤ZC(TC)	郑商所	按成交量	4	0
晚籼稻LR	郑商所	按成交量	3	0
甲醇MA(ME)	郑商所	按成交量	1.4	0
粳稻谷JR	郑商所	按成交量	3	3
硅铁SF	郑商所	按成交量	3	0
菜籽油OI(RO)	郑商所	按成交量	2.5	0
棉花CF	郑商所	按成交量	4.3	0

普麦PM	郑商所	按成交量	5	5
强麦WH(WS)	郑商所	按成交量	2.5	0
沪深300IF	中金所	按成交额	0.000023	0.0023
中证500IC	中金所	按成交额	0.000023	0.0023
上证50IH	中金所	按成交额	0.000023	0.0023
5年期国债TF	中金所	按成交量	3	3
10年期国债T	中金所	按成交量	3	0

撮合机制

在最新的版本中，我们加入了允许用户自定义撮合机制的功能。您可以在策略编辑页面"更多"选项下选择不同的撮合机制。目前提供的撮合方式有以下两种： 1.当前收盘价。即当前bar发单，以当前bar收盘价作为参考价撮合。 2.下一开盘价。即当前bar发单，以下一bar开盘价作为参考价撮合。 限价单（LimitOrder）如果买单价格>=参考价，或卖单价格<=参考价，以参考价加入滑点影响成交（买得更高，卖得更低）。市价单（MarketOrder）直接以以参考价加入滑点影响成交。**成交数量都不超过当前bar成交量的25%。**一旦超过，市价单会在部分成交之后被自动撤单；限价单会一直在订单队列中等待下一个bar数据撮合成交，直到当日收盘。当日收盘后，所有未成交限价单都将被系统自动撤单。

- 分钟回测及实盘模拟：在一个 `handle_bar` 内下单，在该handle_bar结束时统一撮合成交（成交价取决于撮合机制以及滑点设置）。
- 日回测：在一个 `handelbar` 内下单，下单时立刻撮合成交（成交价取决于撮合机制以及滑点设置）。

所以在分钟回测以及实盘模拟中handle_bar内发单之后立刻通过cancel_order对该订单进行撤单操作，是一定会撤单成功的。但在日回测中则很可能撤单失败，因为日回测中下单之后立刻撮合成交。

需要注意，在当前的分钟回测撮合模式下，用户在回测中无法通过在scheduler调用的函数中一次性实现 卖出 -> 资金释放 -> 买入 这种先卖后买的逻辑的。因为在分钟回测中，卖出并不能立刻成交。

举例来说，策略A设置每周一开盘进行调仓操作，先卖后买。那么，以下这种方式在分钟回测中**无法实现卖出资金立刻释放的**（在开启验资的风控情况下，可能导致后面的买入操作因资金不足而拒单）：

```
#scheduler调用的函数需要包括context, bar_dict两个参数
def rebalance(context, bar_dict):
    order_shares('000001.XSHE', -100)
    order_shares('601998.XSHG', 100)
```

有如下几种情况无法完成下单：

- portfolio内可用资金不足
- 下单数量不足一手（股票为100股）
- 下单价格超过当日涨跌停板限制
- 当前可卖（可平）仓位不足
- 股票当日停牌
- 合约已经退市（到期）或尚未上市

另外需要注意的是，如果当时市场处于涨停或跌停这种单边市情况，买单（对应涨停），卖单（对应跌停）是无法成交的。尽管bar数据中可能成交量不为0.判断单边市的标准我们采用的是：当前bar数据的收盘价等于涨停价，则当前市场处于涨停状态。跌停也是类似处理。

滑点

为了更好模拟实际交易中订单对市场的冲击，我们引入滑点的设置。您可以在策略编辑页面"更多"选项下进行滑点设置，允许设置的范围是[0, 1)。该设置将在一定程度上使最后的成交价"恶化"，也就是买得更贵，卖得更便宜。我们的滑点方式是按照最后成交价的一定比例进行恶化。例如，设置滑点为0.1，那么如果原本买入交易的成交价为10元，则设置之后成交价将变成11元，即买得更贵。

注意，滑点默认为0，原有的默认0.246%的滑点值以及通过context.slippage设置的方式被废弃。

性能分析

开启该功能之后，策略回测完毕将会在回测详情页面的"性能分析"选项卡下查看策略运行性能报告。需要注意的是，开启该功能将会导致策略运行效率的下降。

图片描述

分红派息

- 拆分（送股、增股）

在回测中您无须担心拆分对股票价格带来的影响因为我们已经在数据的预处理中准确地帮你做了这个工作。

- 股息

在股息事件中有四个关键的日期：

1.**方案实施公告日** ：公司公布股息分配方案的日期。

3.**除权除息日** ：股权登记日的下一个交易日即是除权除息日，该日证券交易所会计算出股票的除权除息价，以作为投资者在除权除息日开盘的参考。

4.**股息到帐日** ：现金股息划拨到投资者资金账户的日期。

当您的投资策略在**股权登记日**时仍持有分股息的股票，那么您的投资策略将有资格参与此次股息分红。在**除权除息日**结束的时候您投资组合中的DividendRecivable会增加对应持有股票的股息分红数目。然后在**股息到帐日**那天DividendRecivable将会被搬入投资组合中的AvailableCash - 您最终拿到了应收股息分红的金额，并且可以用这笔钱进行再投资了。

仓位管理

在交易中国A股市场证券时引入了T+1机制，当日买入的股票需要在下一日才能够卖出。另外，如果持有仓位的股票已经退市，那么系统会自动将仓位清零。此时，投资组合价值将会出现"跳水"；如果持有期货直到到期，那么会按照到期日的结算价进行现金交割。 另外，所有仓位均会在当天收盘之后进行统一清理，被平掉的仓位不会出现在下一交易日初始的持仓中。但在日内，存在持仓为0的仓位记录（它还记录了该仓位的建仓价格、累计盈亏等信息，具体请参考仓位信息）。

日志功能

通常可以通过日志来了解程序的行为，或者用日志类解决程序中暗藏的问题。同样的，您也可以在您的策略中通过日志来协助开发与调试策略。

譬如，您希望在第一个交易日打印一条日志，把这一天发单的情况打印出来。为了实现这一功能，您只需如下图所示在代码中调用日志类即可：

图片描述

需要注意，回测中系统只会保存开始的10000条记录，多出的信息将无法被保存。在实盘模拟中，每天的日志上限是1000条。

股票自动搜索及补全

- **Windows 用户** ： 输入`ctrl` + `i`
- **Mac 用户** ：输入`command` + `i`
- **Linux 用户** ：输入`ctrl` + `i`

当您输入了这个组合键之后，Ricequant在线IDE就会进入股票代码搜索和自动完成模式，接着您可以输入任何一种进行搜索和自动补全：

- **股票数字代码** - 自动补全为股票数字代码，比如**"000024.XSHE"**：
- **股票中文全称** - 自动补全为股票中文全称，比如**"招商地产"**

我们强烈推荐您使用这个功能来避免手动输入股票代码带来的不便，并且容易犯错误，下面还有一个动态图来解释股票自动搜索及补全功能：

图片描述

股票与期货混合策略注意事项

鉴于不同合约交易时间的不同（例如股票没有夜间交易，期货一些品种有夜盘交易），您在编写策略的时候需要注意策略的有效运行时间。比如在2015年12月之前，中金所股指期货的交易时间段是09:15~11:30, 13:00~15:15，比A股市场多出了30分钟。在这个时候进行混合回测的时候就需要通过订阅的方式让策略引擎'知道'handle_bar是要在每天09:16产生第一个bar数据，而不是股票的09:31。

如果您创建的是单一的期货策略，则必须在策略初始化的时候订阅(subscribe)有效期货合约。由于期货有到期日，所以您需要保证在回测期间，始终都有正在交易的合约被订阅。

混合策略的股票、期货子账户信息可以分别通过context.stock_account以及context.future_account获取到。

向导式策略生成器

创建向导式策略

您可以通过在'我的策略'下点击'向导式策略生成器'来通过向导式生成策略，如下图：

图片描述

选股设置

您可以通过选股设置实现投资域的筛选，并利用各类指标设置选股条件，最后对选出的股票进行排序。界面如下图：

图片描述

股票池设置

该部分不仅支持不同股票池的偏好设置，还支持板块、行业、ST股这三个细分投资域的偏好设置。

股票池选择	默认值	选项性质	选项范围
选择股票池	全市场	单选	全市场(全部A股市场)、沪深300、上证50、中证500
板块	全部	多选	全部(所有板块)、主板、中小板、创业板

行业	全部	多选	包括农林牧渔、采掘、化工、钢铁等	
ST股票	包含ST	单选	包含ST、过滤ST、仅包含ST	

选股指标

界面左边是米筐提供的所有指标的展示区域，右边是您所选择指标的设置区域。基于购物车理念，您可以更加方便地对您选择的指标进行参数调整，界面如下图：

图片描述

您可以通过点击选择指标，所选出的指标会右边“我的指标”中展示，并进行相应的设置。选股指标包含五个分类，具体详情如下：

分类	类别详情	
行情	开盘价、收盘价、最高价、最低价、成交量、成交额、换手率、上市天数	
技术指标	MA、MACD、KDJ、CCI、WILLR、ATR、ROC、TRIX、RSI、MFI、阿隆指标、布林线等	
财务指标	估值类、盈利能力类、偿债能力类、营运能力类、成长能力类等	
财务数据	营业收入类、营业支出类、收益利润类、资产类、负债类、权益类、经营现金流、投资现金流、筹资现金流等	
形态指标	两只乌鸦、三只乌鸦、三胞胎乌鸦、红三兵、锤头、倒锤头、乌云盖顶、晨星、黄昏之星等	

其中，财务指标与财务数据的差异为，财务数据是原始数据，财务指标是基于财务数据得到的指标。

排序指标

可用于排序的指标有四类，分别为行情指标、技术指标、财务指标、财务数据，比**选股因子**少了形态指标。

选股条件

您可以利用**选股指标**对股票池粗筛选出的股票进一步进行筛选。您可在**选股条件**部分对选股指标的数值范围进行设置，如下图：

图片描述

您可以在该部分进行如下功能的设置：

分类	支持选项

行情	小于N日均值、介于N1,N2均值间	
技术指标	大于、小于、区间、金叉、死叉、多头、空头、价格多头、价格多头	
财务指标	大于、小于、区间、排名%区间(默认升序)	
财务数据	大于、小于、区间、排名%区间(默认升序)	
形态指标	无选项	

其中，对于技术指标，您可基于自己的偏好调整其对应的参数。

自定义规则

为了支持更多个性化规则，您也可以点击“添加自定义规则”，通过比对两个指标的数值大小，筛选出符合您条件的股票。

排序条件 对于您筛选出的股票，您可以从**排序指标**中选出用于排序的指标，在**排序条件**部分对所选指标进行升序或降序的处理，并赋予不同的权重，从而实现对股票的排序。对于技术指标，您可以对其参数进行偏好设置。如下图：

图片描述

模型选择

您可以在**模型选择**中设置您策略所需要的模型，米筐提供两种模型，分别为**定期轮动**、**条件触发**。

定期轮动 定期轮动为定期调仓交易模型，您可以基于选出的股票，设置调仓周期、最大持仓股票数量。该模型会定期在调仓日卖出不符合选股条件的股票，等权买入符合条件的股票。界面如下图：

图片描述

条件触发 对于条件触发模型，您可以设置买入条件、卖出条件。界面如下：
图片描述

区别于定期轮动模型，条件触发模型将调仓周期分为了三个周期，分别为选股周期、买入周期、卖出周期。

- **选股周期**：该周期决定多少个交易日执行一次选股操作，即筛选出满足**选股设置**里选股条件的股票并按排序条件对其排序。一般用财务指标实现股票池的筛选，由于企业财务指标大多属于季度更新，故该周期可以设置长一些。
- **买入周期**：该周期决定多少个交易日执行一次买入操作，即买入满足**买入条件**的股票。买入条件中一般用于技术指标、形态指标等比较难以捕捉的信号的捕捉，故买入周期可以设置短一些。

买入条件：

您可以在买入条件设置选股周期、买入周期、最大持仓股票数、个股最大持仓比重，同时可以通过对行情、技术指标、财务指标、财务数据、形态指标设置买入条件。这部分提供的指标及指标设置情况可参考**选股指标**、**选股条件**。

卖出条件：

您可以在卖出条件设置卖出周期，同时也可设置行情、技术指标、财务指标、财务数据、形态指标。这部分提供的指标及指标设置情况可参考**选股指标**、**选股条件**。

风险控制

您可以在“风险控制”部分对策略的风险进行设置。风险控制支持对止盈止损、大盘择时(开仓)、大盘择时(平仓)的设置。如下图：

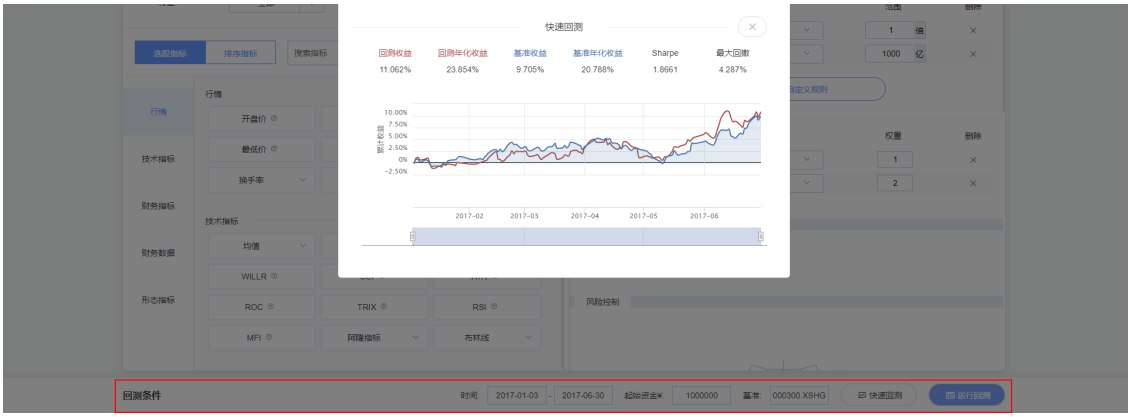
风控名称	作用阶段	可选指标
止盈止损	股票的卖出	个股止损、个股止盈、持仓价值止损、持仓价值止盈、策略最大亏损、策略最大盈利
大盘择时(开仓)	股票的买入	大盘指数的各类指标，包括行情、涨跌幅、技术指标
大盘择时(平仓)	股票的卖出	大盘指数的各类指标，包括行情、涨跌幅、技术指标

风险控制可以实现对用户策略的整个过程风险的全面控制。在开仓前，可以通过大盘择时来判断是否买入；在买入后，可以通过止盈止损和大盘择时(平仓)实现事后风控。

回测设置

您可以在“策略回测”部分对回测的参数进行相应的设置，如下图所示：

运行中的策略



可以设置的参数如下：

参数名	解释
开始日期	回测期间的开始日期
结束日期	回测期间的结束日期 - 如果回测结束日期在今天之后， 将会自动使用最后一天的历史数据
调仓周期	回测期间的调仓周期，可根据偏好进行设置 - 默认收盘后进行相应的调仓
起始资金	回测的起始资金 - 您将使用多少钱去投资策略
基准合约	设置策略表现的对照基准

在设置完回测参数之后，可通过'保存'按钮来保存您的所有设置；也可以点击'快速回测'来对策略进行快速回测，以预览策略收益；或点击'运行回测'直接运行回测。两者的区别如下：

- **快速回测** - 点击"快速回测"将会使我们的后台实时编译您筛选结果生成的策略来获得一个策略收益的预览。快速回测会比完整的回测要快，它能高效地展示策略和基准的收益曲线，因此您可以试用这个功能在完整回测前先验证下策略的收益情况。
- **运行回测** - 点击"运行回测"将会带您进入完整的回测结果页面，您可以查看更加详细的策略回测分析结果，帮助您对策略的表现实现更细致的评估。

分红派息

- 拆分（送股、增股）

在回测中，我们默认对股票拆分的情况进行预处理，提高计算结果的准确性。

- 股息

在股息事件中有四个关键的日期：

1.**方案实施公告日** ：公司公布股息分配方案的日期。

3.**除权除息日** ：股权登记日的下一个交易日即是除权除息日，该日证券交易所会计算出股票的除权除息价，以作为投资者在除权除息日开盘的参考。

4.**股息到帐日** ：现金股息划拨到投资者资金账户的日期。

当您的投资策略在**股权登记日**时仍持有分股息的股票，那么您的投资策略将有资格参与此次股息分红。在**除权除息日**结束的时候您投资组合中的DividendRecivable会增加对应持有股票的股息分红数目。然后在**股息到帐日**那天DividendRecivable将会被搬入投资组合中的AvailableCash - 您最终拿到了应收股息分红的金额，并且可以用这笔钱进行再投资了。

回测结果分析

回测结果

当回测运行没有出错，**回测结果**页面将会载入您的投资组合的各种交易、盈亏和风险信息。

下图是回测结果的介绍：

- 回测设置和状态：在这里您能够看到策略名称、交易的品种（股票还是期货）、回测起止日期、初始资金、回测频率等信息，并能够将回测分享到社区
- 收益概览：展示策略的风险调整收益（Sharpe, InformationRatio等）、回测收益图、每日盈亏图以及成交记录图。您自己通过plot画图结果也会在该页面展示
- 交易详情：展示回测期的历史交易记录，包括成交时间、成交价、费用等信息
- 每日持仓：展示回测期的仓位信息，包括建仓成本、累计盈亏等信息
- 账户信息：展示回测期的账户信息，包括账户总权益、可用资金、当日盈亏、累计盈亏等信息

- 收益表格：您在这里看到回测期间每个月截止往前的1个月、3个月、6个月、12个月的投资策略和基准策略的收益详细情况。
- 风险表格：您在这里看到投资策略回测期间的每个月截止往前的1个月、3个月、6个月、12个月的各种风险指标数值。其中波动值表格中您还可以看到投资策略和基准策略的比较。

回测结果进阶分析

在回测结果页面新加入的"进阶分析"选项卡允许用户在每次**回测运行完毕之后**进行诸如月度收益分布、胜率分布等分析。

净值曲线

反映策略与基准合约在回测期间的累计净值表现，以100%为起点。其中，对数轴累计净值曲线将原有纵轴累计净值在纵轴的刻度以10为底数进行了取对数计算。这种显示方式并未改变累计净值水平，但在收益变动剧烈的情况下（比如暴涨10倍）能够较好地显示出策略净值不同时期水平。波动率调整累计净值意思是将策略的累计净值根据基准的波动率进行调整。举例来说，在回测期间基准合约的波动率为2，而策略的波动率是4.那么尽管策略在回测期间取得了较高的收益，但在波动率调整后，策略整体收益率要变为原有水平的一半。这种方式能够简单直观地对比剔除波动率这一影响因素之后策略与基准的收益水平。

收益详情

月度收益率以等温图的方式显示每月累计收益情况；年度收益率计算每个自然年的策略累计收益情况。

持仓分析

持仓占比为每日股票持仓市值占投资组合总权益的比重；每日换手率计算为每日买、卖成交额绝对值加总除以2再除以投资组合总权益。

1.回测期间发生了以下几笔交易：

- 11-15 09:45 买 平安银行 100股@10元
- 11-15 11:00 买 平安银行 100股@12元
- 11-16 09:45 卖 平安银行 100股@13元
- 11-16 11:00 买 平安银行 100股@14元
- 11-16 14:00 卖 平安银行 100股@15元

2.每日对每只股票的买单、卖单进行汇总，得到一个总和的买单与一个总和的卖单。进行汇总之后：

- 11-15 买 平安银行 200股@11元
- 11-16 买 平安银行 200股@14.5元 卖200股@14元

3.按照先进先出的规则将买单、卖单进行匹配，得到回转交易：

- 回转交易1: 11-15日买入建仓，11-16日卖出平仓
- 回转交易2: 11-16日买入建仓，策略回测到期日的收盘价为成交价自动卖出平仓

情景分析

回测期间如果覆盖以下时间段，分析模块会将基准与策略净值在每段时期的期初全部调整为100%，重点展示特定时期策略表现。

- 2014-12-01 至 2015-03-01 牛市调整期
- 2015-03-03 至 2015-06-03 创业板大牛市
- 2015-06-01 至 2015-09-01 股灾暴跌
- 2015-12-15 至 2016-03-15 熔断暴跌

收益指标

回测收益率: 策略在期限内的收益率。

> $$\text{回测收益} = \frac{\text{期末投资组合总权益} - \text{期初投资组合总权益}}{\text{期初投资组合总权益}}$$

年化收益率: 采用了复利累积以及Actual/365 Fixed的年化方式 计算得到的年化收益。

> $$\text{年化收益率} = (1+R)^{\frac{1}{t}} - 1$$
>
> $$t = \frac{\text{策略运行累计自然日数量}}{365}$$
>
> $$R = \text{累计收益率}$$

基准收益率: 相同条件下，一个简单的买入并持有基准合约策略的收益率（默认基准合约为沪深300指数，这里假设指数可交易，最小交易单位为1）。

每日收益率: 通过投资组合权益计算出的日收益率。

$$\text{每日收益率} = \frac{\text{当前交易日总权益} - \text{前一交易日总权益}}{\text{前一交易日总权益}}$$

阿尔法(alpha, α): CAPM模型表达式中的残余项。表示策略所持有投资组合的收益中和市场整体收益无关的部分，是策略选股能力的度量。当策略所选股票的总体表现优于市场基准组合成分股时，阿尔法取正值；反之取负值。

$$\alpha = E[r_p - [r_f + \beta \cdot (r_b - r_f)]]$$

其中r_p为策略所持有投资组合收益；r_f为无风险组合收益；β为CAPM模型中的贝塔系数；$E[\cdot]$表示随机变量的期望。

风险指标

年化波动率(volatility, σ_t): 策略收益率的标准差，最常用的风险度量。波动率越大，策略承担的风险越高。这里假设一年有244个交易日。

$$\sigma = \sqrt{\frac{244}{n-1}\sum_{i=1}^{n}[r_p(i) - \bar{r_p}]^2}$$

其中，n为回测期内交易日数目；$r_t(i)$表示第 i 个交易日策略所持有投资组合的日收益率；$\bar{r_p}$为回测期内策略日收益率的均值。

年化跟踪误差(tracking error, σ_t): 纯多头主动交易策略（阿尔法策略和基准择时策略）收益和市场基准组合收益之间差异的度量。跟踪误差越大，意味着策略所持有投资组合偏离基准组合的程度越大。需要注意，跟踪误差不适用于多-空结合的对冲策略的风险评估。

$$\sigma_t = \sqrt{\frac{244}{n-1}\sum_{i=1}^{n}[r_{pa}(i) - \bar{r_{pa}}]^2}$$

$$r_{pa}(i) = r_p(i) - r_b(i)$$

其中，n为回测期内交易日数量；$r_{pa}(i)$, $r_p(i)$, $r_b(i)$分别表示第i个交易日策略所持有投资组合的日主动收益、日收益率和基准组合的日收益率。

年化下行波动率(downside risk, σ_d): 相比波动率，下行波动率对收益向下波动和向上波动两种情况做出了区分，并认为只有收益向下波动才意味着风险。在实际计算中，我们统一使用基准组合收益为目标收益，作为向上波动和向下波动的判断标准。

$$\sigma_d = \sqrt{\frac{244}{n-1}\sum_{i=1}^{n}[r_p(i) - r_b(i)]^2 \cdot I(i)}$$

$$I(i) = \begin{cases} 1, & r_p(i) < r_b(i) \\ 0, & r_p(i) \geqslant r_b(i) \end{cases}$$

略所对应投资组合收益减无风险组合收益，则标记为1（高于波动），否则标记为0（高于波动）。

贝塔(beta, β): CAPM模型中市场基准组合项的系数，表示资产收益对市场整体收益波动的敏感程度。

$$\beta = \frac{Cov(r_{p,e}, r_{b,e})}{Var(r_{b,e})}$$

其中$r_{p,e}$为策略超额收益率（策略收益率 - 无风险组合收益率）；$r_{b,e}$为市场基准组合超额收益率（市场基准组合收益率 - 无风险组合收益率）；$Cov(\cdot)$表示协方差；$Var(\cdot)$表示方差

Beta值	解释	举例
β < 0	投资组合和指数基准的走向通常反方向	反向指数ETF或空头头寸
β = 0	投资组合和指数基准的走向没有相关性	固定收益产品，他们的走向通常
0 < β < 1	投资组合和指数基准的走向相同， 但是比指数基准的移动幅度更小	稳定的股票，比如制作肥皂的公 通常和市场的走势相同， 但是受到每日的波动影响更小
β = 1	投资组合和指数基准的走向相同， 并且和指数基准的移动幅度贴近	蓝筹股，指数中占比重大的股票
β > 1	投资组合和指数基准的走向相同， 但是比指数基准的移动幅度更大	受每日市场消息或是受经济情况

最大回撤(max drawdown): 在回测期内，在任一交易日往后推，策略总权益走到最低点时收益率回撤幅度的最大值。最大回撤是评估策略极端风险管理能力的重要指标。其计算方式如下： ：

$$Drawdown_t = \begin{cases} 0 & amp; \text{if } NET_t = min_{j \geq t} NET_j \\ fracNET_t - \min_{j \geq t} NET_j NET_t & amp; else \end{cases}$$

NET为某期净值 $MaxDrawdown = max(Drawdown_t)$

风险调整后收益指标

夏普率(sharpe ratio): 衡量策略相对于无风险组合的表现，是策略所获得风险溢价的度量——即如果策略额外承担一单位的风险，可以获得多少单位的收益作为补偿。

$$Daily\ Sharpe\ Ratio = \frac{\bar{r_e}}{\sigma_e}$$

$$\bar{r_e} = \frac{1}{n}\sum_{i=1}^{n}[r_p(i) - r_f(i)]$$

$$Sharpe\ Ratio = \sqrt{244} \cdot Daily\ Sharpe\ Ratio$$

其中$\overline{r_e}$为回测期内策略日超额收益率均值；n为回测期内交易日数目；$r_p(i), r_f(i)$分别为第i个交易日策略所持有投资组合的日收益率以及无风险组合日收益率；σ_e为策略超额收益率的波动率。

索提诺比率(sortino ratio): 衡量策略相对于目标收益的表现。其使用下行波动率作为风险度量，因此区别于夏普率。在目前的计算中，我们使用基准组合收益作为目标收益，以此作为区分向上波动和向下波动的标准。

$$Daily\ Sortino\ Ratio = \frac{\sqrt{244} \cdot \bar{r_e}}{\sigma_d}$$

$$\bar{r_e} = \frac{1}{n}\sum_{i=1}^{n}[r_p(i) - r_f(i)]$$

$$Sortino\ Ratio = \sqrt{244} \cdot Daily\ Sortino\ Ratio$$

其中$\overline{r_e}$为回测期内策略日超额收益率均值；n为回测期内交易日数目；$r_p(i), r_f(i)$分别为第i个交易日策略所持有投资组合的日收益率以及无风险组合日收益率；σ_d为策略年化下行波动率。

信息比率(information ratio): 衡量策略相对于市场基准组合的表现。一般用于评估纯多头的主动交易策略（包括阿尔法策略和基准择时策略）。需要注意的是，信息率不适用于多-空结合的对冲策略的表现评估。

$$Daily\ Information\ Ratio = \frac{\sqrt{244} \cdot \bar{r_{pa}}}{\sigma_t}$$

$$\bar{r_{pa}} = \frac{1}{n}\sum_{i=1}^{n}[r_p(i) - r_b(i)]$$

$$Information\ Ratio = \sqrt{244} \cdot Daily\ Information\ Ratio$$

其中$\overline{r_{pa}}$为回测期主动日收益率均值；n为回测期内交易日数目；$r_p(i), r_b(i)$分别为第i个交易日策略所持有投资组合的日收益率以及基准组合日收益率；σ_t为策略跟踪误差。

在我们提供的三个风险调整后收益指标中，信息率用于评估投资组合相对于市场基准组合的表现，一般适用于纯多头的主动交易策略（包括阿尔法策略和基准择时策略）；夏普率用于评估投资组合相对于无风险组合的表现，一般适用于多-空结合的交易策略（例如市场中性策略或配对交易策略），或没有公认市场基准组合的投资品种的交易策略（例如期货CTA策略）。

索提诺比率使用下行波动率作为风险度量，因而有别于信息率和夏普率。下行波动率区分了收益向上波动和向下波动两种情况，并认为收益向下波动才代表风险。因此，索提诺比率的优点，在于其使用的风险度量更为切合我们实际投资中面对的风险；而其缺点则是不如信息率和夏普率常用，认知度较低，且其目标收益（区分收益波动是向上还是向下的标

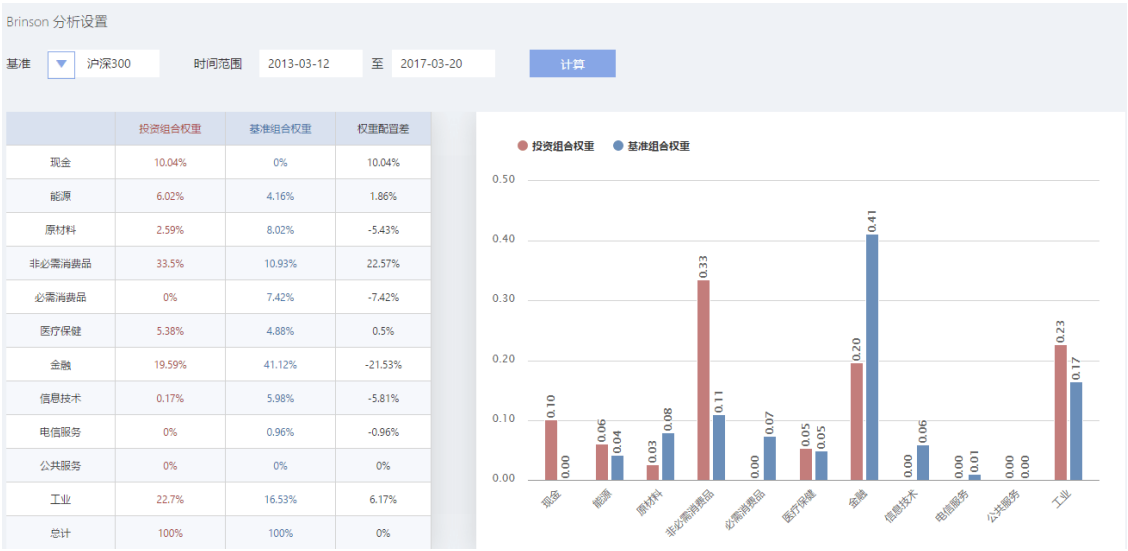
绩效分析

在运行完回测之后, 在回测结果页面点击右上方的 “绩效分析” 按键就开启了对于策略表现的分析。目前该分析主要包括**Brinson分析**,**风格分析**,**净值回归**以及**风险分析**几部分。需要注意的是, 目前绩效分析暂时只支持股票类型投资组合。

Brinson分析

在“Brinson分析”部分，我们使用Brinson Model 对主动投资组合进行业绩归因。 Brinson Model假设用户的投资流程可分为两个步骤：板块配置和板块内选股。 和市场基准组合相比，如果投资者能做到以下任意一项：高配表现好的板块；低配表现差的版块；在某些板块中具备较强的选股能力，都有可能获得比市场基准组合更好的业绩。 为了定量地评估“板块配置”和“板块内选股”这两步对投资组合收益的贡献，用户可以使用 Brinson Model对投资组合的主动收益（投资组合和基准组合的收益差值）进行业绩归因， 以完善其投资流程（例如为选股收益高的板块配置更多的资金等）。

1.投资组合和基准组合的板块权重配置



用户选择基准组合(目前支持沪深300和中证500)，选择板块分类(目前支持MSCI和申万一级行业分类), 设置分析区间的起始和结束时间，点击计算， 该分析图表展示了投资组合和基准组合在每个板块的平均配置权重(time weighted average)，投资组合减去基准组合的板块配置差值即为投资组合在该板块的主动权重。

2.投资组合和基准组合的收益解析归因

Brinson 模型是一个基于板块的业绩归因模型，投资组合相对基准组合的主动收益的来源是板块配置的差异， 以及在板块中收益的差异(选股的差异)，所以下面的表格详细分解了两个组合在各个板块的权重和收益，并且进行归因计算。

能源	0.02%	4.19%	1.86%	-78.94%	-10.34%	-88.39%	-0.41%	-2.83%	-1.06%	-4.52%
原材料	2.59%	8.02%	-5.43%	365.44%	8.08%	357.36%	0.81%	28.66%	-20.65%	8.82%
非必需消费品	33.5%	10.93%	22.57%	634.67%	44.78%	589.89%	-2.68%	64.48%	145.92%	207.71%
必需消费品	0%	7.42%	-7.42%	46.24%	46.24%	0%	-4.27%	0%	0.84%	-3.43%
医疗保健	5.38%	4.88%	0.5%	1197.93%	38%	1159.93%	-1.84%	56.62%	7.87%	62.65%
金融	19.59%	41.12%	-21.53%	953%	42.69%	910.31%	-19.21%	374.29%	-185.99%	169.09%
信息技术	0.17%	5.98%	-5.81%	1433.74%	14.65%	1419.09%	0.3%	84.88%	-83.62%	1.57%
电信服务	0%	0.96%	-0.96%	69.58%	69.58%	0%	-1%	0%	0.33%	-0.67%
公共服务	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
工业	22.7%	16.53%	6.17%	414.84%	33.25%	381.59%	-5.6%	63.07%	31.21%	88.68%
总计	100%	100%	0%	565.1%	34.98%	530.11%	-37.41%	669.14%	-101.62%	530.11%

该表格可以分为三个部分:

- 第一部分为板块权重，详细展示了投资组合和基准组合在不同板块的权重配置情况，并计算了二者的配置差异。
- 第二部分为板块收益，展示了投资组合和基准组合在不同板块的绝对收益情况，两者的差值为板块收益差值。
- 第三部分为归因计算部分，计算了投资组合与基准组合在不同板块上的收益差别的来源，包括板块配置差异，板块内选股差异以及交互作用差异。

投资组合超过基准组合的收益部分为主动收益，Brinson归因分析模型把收益部分分解为三个部分：配置收益，选股收益和交互收益。

- 板块配置收益: 体现了投资组合对板块配置的能力，在图表中的计算公式为 基准组合相对收益 * 板块权重差值
- 板块选股收益: 体现了投资组合在特定板块内的选股能力，在图表中的计算公式为 板块收益差值 * 基准组合权重
- 交互收益: 是板块权重差值和板块收益差值的交互项，在图表中的计算公式为 板块收益差值 * 板块权重差值

板块主动收益为三项收益计算的总和，等于投资组合相对基准组合在该板块的主动收益。

上图表明格雷厄姆数字价值投资法的收益主要来源于精确的选股能力，并且选股带来的收益甚至弥补了部分资产配置不利造成的损失。

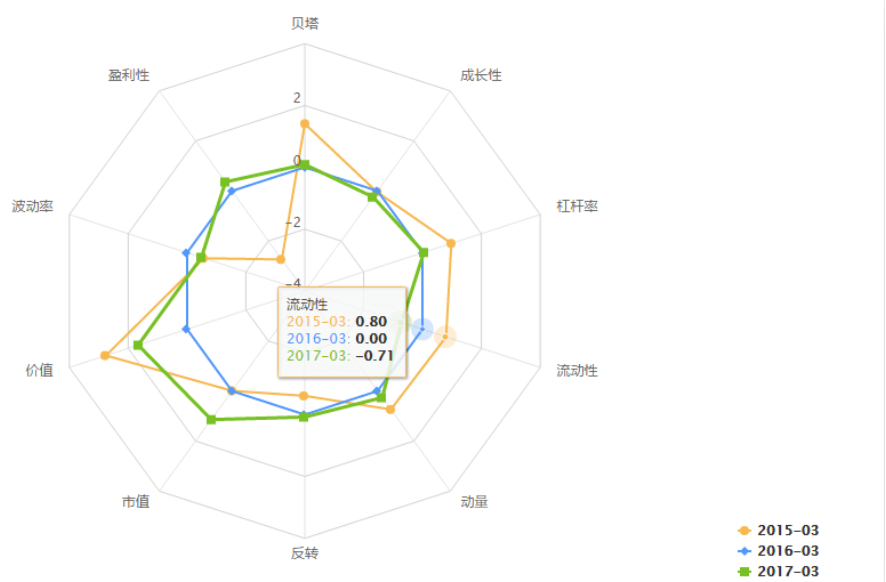
风格分析

风格分析是指对策略或基金的投资风格或偏好进行分析。在RQBeta中，我们通过投资组合中个股的基本面信息（例如规模、价值和盈利能力）和量价信息（例如动量、波动率和流动性）来对投资组合多期的投资风格变化，以及投资组合和基准组合投资风格的差异进行评估。 通过分析基金产品的多期投资风格变化程度（即风格漂移），我们可以判断基金产

对投资组合进行多维度的风格评估。

风格名称	解释
贝塔（Beta）	股票收益对基准组合（沪深300）收益的敏感度
动量（Momentum）	股票价格变化的总体趋势特征
规模（Size）	上市企业的规模
盈利率（Earning Yield）	上市企业的营收能力
波动率（Volatility）	股票价格变化幅度（风险）的度量
成长性（Growth）	上市企业的营收、资产和总体规模的同比增长率
价值（Value）	股票的投资价值（总股本数和市值的比）
杠杆（Leverage）	上市企业的债务相对于其总资产的比例
流动性（Liquidity）	股票的换手率，即其交易的活跃程度
反转（Reversal）	股票价格的均值回归特征

在下图的实例中，我们可以看到在整个投资周期中，投资组合的风格漂移现象并不明显，表现比较稳定。在15年3月份，投资组合更加偏好价值股，在价值因子上风险暴露较多。在16年3月份，整个投资组合在各个因子上的风险暴露度皆为0，说明在该段时期，投资组合持仓为零。 在17年3月份，投资组合更加偏好价值因子和市值因子，表明股票池中的股票持仓逐渐倾向于大市值低市盈率的股票。

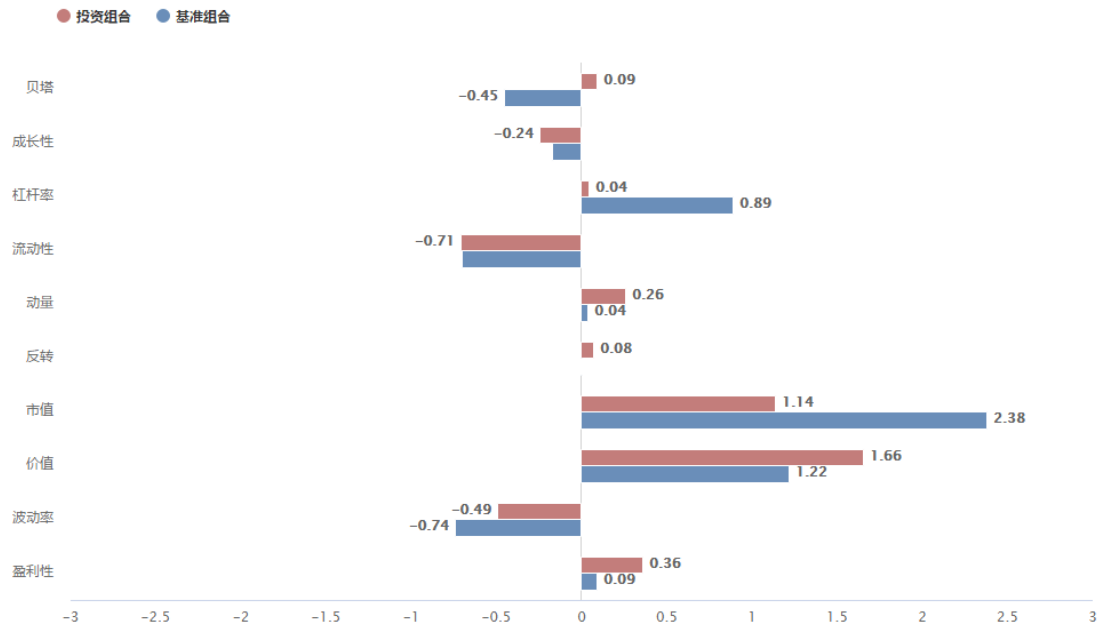


下图显示了最近一次投资组合的风格因子暴露度与基准组合因子暴露度的对比。我们可以看到虽然相比于投资组合初期，市值因子暴露度在不断加大， 但是相比于市场基准组合依然偏小，说明持仓中中小市值股票占比占据主导。投资组合中价值因子暴露度高于市场组合，表明投资组合更加注重低估值股票。 最后，投资组合的杠杠率因子暴露度很低，说明所选公司的偿债能力很强。

在风格因子上相对基准组合而言的相对暴露度。

比如以上范例的投资组合，相对基准组合，它在市值因子和杠杆率因子上的暴露度偏低，在价值因子的暴露度上偏高，说明这个投资组合更偏重小盘股， 同时偏重财务状况较好，偿债能力更强，同时在市场上相对估值便宜的价值股。

投资组合的相对风格暴露度本身只是一种风险的体现，如果用户认为在价值因子和市值因子上的暴露度能获得超额的收益回报，就会接受这样的暴露度风险敞口， 但是如果某些暴露度敞口是用户并不希望在自己的投资组合中出现的，就需要及时调整仓位来避规这些额外的风险。 这张分析图可以帮助用户更好的及时了解自己投资组合面对各个风格因子的相对风险暴露的情况，以做出应对。



净值回归

在这一部分，我们使用投资组合的净值数据进行回归分析，计算投资组合收益对各个RQ风格因子（解释见风格分析部分）收益的敏感度，来对投资组合的收益/风险来源行评估。

通过多期净值归因，我们可以评估投资组合是否有特定的、显著的收益/风险来源。例如，对于某一策略或基金产品，通过多期的净值归因， 发现其一直对规模因子有较大的负敏感度（即当市场上大市值股票总体表现好于小市值股票时，该策略或基金产品的业绩较差）。 这意味着该策略或基金产品可能长期偏好持仓小市值股票，导致风险过于集中。此时，一旦小市值股票总体表现欠佳，该策略或基金产品的业绩可能会出现较大的回撤。

根据上图，在整个持仓期间，投资组合的风格漂移变化并不是十分明显，维持了较强的稳定性，其中变化比较明显的是贝塔因子和成长性因子的回归系数。 以贝塔因子的回归系数为例，图像显示，贝塔值的回归系数随着投资持续期增加而不断降低。一般而言，高贝塔回归系数对应于持仓中包含较多低市值个股， 说明在初期投资组合中持仓个股的市值都偏低。但随着指数上升，投资组合中低市值个股相对减少。

风险分析

在风险分析部分，我们给出了投资组合在测试期内的一系列的收益、风险、风险调整后收益指标。

投资组合详细风险报告

基准 沪深300 时间范围 2013-03-12 至 2017-03-20 计算

详细计算展示了在不同时间区段内，投资组合的风险衡量指标(最大回撤，波动率，市场Beta)以及风险调整后收益(Sharpe, Sortino)。
通过历史VaR的方法计算出95%置信区间的VaR比例，作为投资组合预期风险的参考。

		收益率	alpha	beta	sharpe	sortino	最大回撤	VaR 95% 5天
最近一月	基准组合	10.34%	0%	1.00	0.90	-	-1.8%	0.88%
	投资组合	16.98%	2.89%	1.46	0.94	2.85	-4.26%	1.56%
最近三月	基准组合	16.42%	0%	1.00	1.53	-	-1.8%	0.78%
	投资组合	34%	10.39%	1.54	1.60	4.41	-4.26%	1.68%
最近六月	基准组合	12.49%	0%	1.00	1.01	-	-7.5%	1%
	投资组合	7.49%	-9.73%	1.46	0.36	0.66	-10.67%	1.97%

我们选取了最近一月、最近三月和最近半年期的净值收益率收据，分别计算了投资组合与基准组合的常见风险收益衡量指标， 并给出了 95%的置信区间下在未来五天投资组合收益的最大可能损失幅度。以最近三个月为例， 投资组合收益超越基准组合
17.58%，-4.26%，在 95%的置信区间下，未来五天的最大损失占收益累积净值的比例为 1.68%

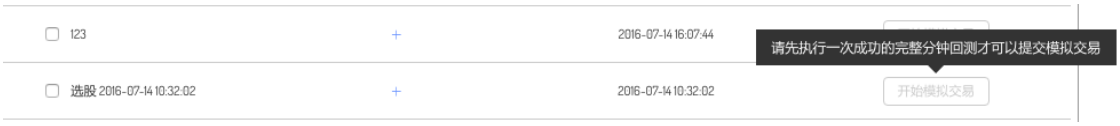
进一步检验策略的有效性。另外您打开微信通知或邮件通知以后，也可以根据收到的实时通知来进行手动落单达到实盘交易的目的。

模拟交易的数据源

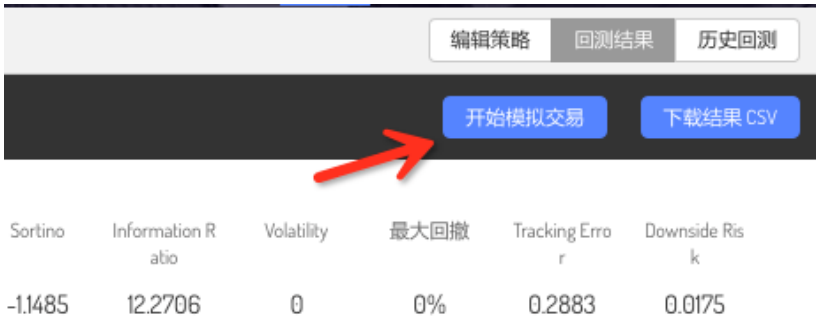
模拟交易使用的是实时更新的Level-1数据，大概会有3-5秒的延迟。

进行模拟交易

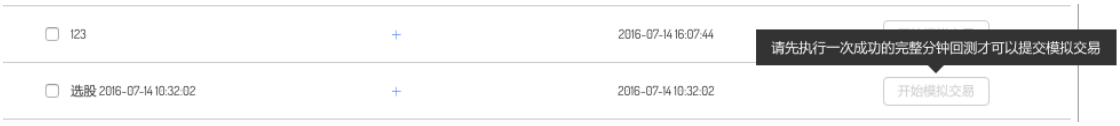
1.首先您需要编写一个策略，然后您会发现提示您需要进行一次完整的分钟回测才可以启动模拟交易：



2.运行完一次完整的分钟回测以后，您可以在回测的结果页面直接启动模拟交易：



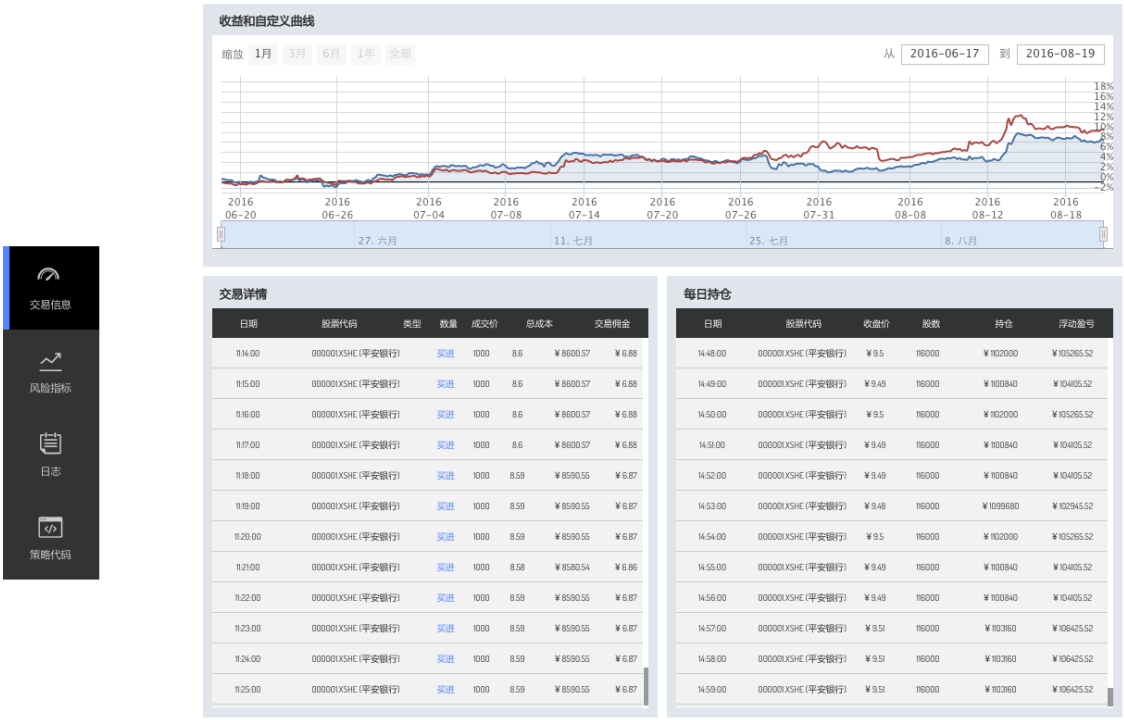
3.或者可以从策略列表页面进行启动：



4.启动之后可以在策略列表看到启动了的模拟交易策略的状态：

5.启动之后的模拟交易策略需要至少等待第一个交易日启动才会有更新：

运行中的策略



模拟交易替换代码

在模拟交易的"设置"选项下可以进行代码更换，您可以选择当前策略的某一次回测代码替换目前正在运行的策略。更换之后，原本的仓位、成交、资金使用情况等信息均会被保存下来。

当前已经暂停的策略在替换代码之后策略仍将处于暂停状态；正在运行的策略在替换代码之后保持运行状态不变；异常结束的策略替换代码之后将自动将策略回复运行。

需要注意，您在替换代码的时候，如果勾选"立即运行init及before_trading，初始化所有参数"，那么原来context中的内容都将会被清除。

图片描述

开启模拟交易的微信通知

1.点击开启微信通知

图片描述

2.之后会弹出微信订阅号的二维码。手机扫码成功后，您的微信将和米筐信号进行捆绑：

图片描述

Clone

3.调仓记录生成后就会被实时推送到您的微信端：

图片描述

4.点击之后跳转到米筐组合调仓界面，首次跳转需要进行登录。

图片描述

您可以选择按照固定频率通知，也可以选择按照具体时间通知：

图片描述

开启模拟交易的邮件通知

非常简单，因为您已经使用了邮箱进行注册，所以只需要点击一下开启邮件通知按钮即可：

图片描述

之后您就可以在产生交易信号的时候收到邮件通知了：

图片描述

本地拿到实盘模拟交易的信号和持仓

有两种方法可以让您本地通过https来拿到您在ricequant上运行的实盘模拟交易策略的**当天的交易列表**和当前**实时的仓位信息**, 您本地拿到信号以后可以自己来做一些展示报告亦或是对接自己的实盘交易通道来做到实盘交易的打通。

1. 通过安装我们发布在github上的RQOpen-Client，详细的说明可见github上的文档说明
2. 通过restful的http请求：

- **登录**: POST url：https://rqopen.ricequant.com/login 请求示例：

```
post("https://rqopen.ricequant.com/login", {"username": username, "password": p
```

python范例：

```
import requests
r = requests.Session()
# login
resp = r.post("https://rqopen.ricequant.com/login", {"username": username, "pas
```

返回示例：

```
{'code': 200,
'resp': {'msg': 'Congratulations, login successful! Happy Coding and '
                'Trading!'}}
```

其中，username为米筐账号用户名，password为米筐账号密码

```
get("https://rqopen.ricequant.com/pt/load_day_trades/{}".format(run_id))
```

python范例：

```
import requests
r = requests.Session()
# login
resp = r.post("https://rqopen.ricequant.com/login", {"username": username, "pas
# 拿到当天的trades
resp = r.get("https://rqopen.ricequant.com/pt/load_day_trades/{}".format(run_ic
```

返回示例1（当日无交易）：

```
{'code': 200, 'resp': {'name': '测试信号', 'run_id': 1205691, 'trades': []}}
```

返回示例2（当日有交易）：

```
{'code': 200, 'resp': {'name': 'SVM大法好',
        'trades': [{'order_book_id': '600216.XSHG',
                    'price': 12.77,
                    'quantity': -100.0,
                    'time': '2016-12-23 09:32:00',
                    'trade_id': '2',
                    'transaction_cost': 6.28}]}}
```

run_id为上述模拟交易对应id，可以在这里查看到：
图片描述

- **查询最新持仓** Get url：https://rqopen.ricequant.com/pt/load_current_positions/ 请求示例：

```
get("https://rqopen.ricequant.com/pt/load_current_positions/{}".format(run_id))
```

python范例：

```
import requests
r = requests.Session()
# login
resp = r.post("https://rqopen.ricequant.com/login", {"username": username, "pas
# 已登录，拿到当天的成交信息
resp = r.get("https://rqopen.ricequant.com/pt/load_current_positions/{}".format
```

返回示例：

```
 'quantity': 600.0},
{'order_book_id': '600222.XSHG',
 'price': 8.37,
 'quantity': 900.0},
 ...
```

run_id为上述模拟交易对应id 简单的三个API, 您就可以方便搭建自己的智投系统了！

模拟交易注意事项

模拟交易在每个bar运行后保存状态，当天交易时间结束后结束进程，第二天再恢复。如果您的模拟交易因为非正常原因关闭的话，我们都可以为您恢复到关闭前一个bar的状态。

当天结束进程时会保存这些状态：

- 用户账户，持仓信息
- 使用pickle保存的context对象。
  - 注意：context中不能序列化的变量不会被保存，重启后会不存在，如果你写了如下代码：

    ```
    def init(context):
        context.something = query(valuation)
    ```

    context是不能被保存的，因为query()返回的对象无法被序列化。因此日志中会出现

    ```
    WARN:context.something can not be pickle
    ```

    正确的做法是将query()写在 `before_trading()` 中。

技术分析

米筐平台引入的信号计算方式能够让您更方便地进行技术分析。在技术分析过程中，您既可以使用我们预先提供的常用技术指标，也可以使用自定义指标来产生信号供策略内使用。

```
def MA_SIGNAL():
    # 5周期移动平均上穿10周期移动平均
    return CROSS(MA(CLOSE, 5), MA(CLOSE, 10))

def BOLL_SIGNAL():
    # 收盘价上传布林通道上轨
    MID, TOP, BOTTOM = BOLL(20, 2)
    return CROSS(CLOSE, TOP)

def init(context):
    context.s1 = '000001.XSHE'
    # 在初始化阶段注册指标函数
```

```
    reg_indicator('boll', BOLL_SIGNAL, '1d', win_size=20)

def handle_bar(context, bar_dict):
    # 获取指标结果
    ma_cross = get_indicator(context.s1, 'ma')
    boll_cross = get_indicator(context.s1, 'boll')

    # 设置入场条件
    if ma_cross and boll_cross:
        order_percent(context.s1, 0.1)
```

自定义技术指标

为满足用户对指标要求的多样性，米筐支持用户使用类似通达信公式的方式计算自定义指标，具体方法可以参考如下步骤：

首先，定义指标函数体本身。例如，

```
def KDJ_SIGNAL()):
    # 连续两个周期J值一直在超买区
    K, D, J = KDJ()
    return EVERY(J > 80, 2)
```

其次，在init阶段调用reg_indicator对所需取用的指标进行注册。注册时，需要指定指标名称、参数、适用的周期（比如，5分钟线还是日线）以及初始回溯获取数据的窗口长度。例如，

```
reg_indicator('kdj', KDJ_SIGNAL, '1d', win_size=20)
```

最后，调用get_indicator获取指标计算结果。此时需要指定获取的指标名称以及所需计算的合约代码。例如，

```
get_indicator('000001.XSHE', 'ma')
```

需要注意的是，目前技术指标计算并未包括当前“不完整”分钟线。举例来说，在09:48计算以5分钟线为周期的移动平均时，并不包括09:45~09:48这一“不完整”的5分钟线。

系统预定义指标

- MACD 指数平滑移动平均

```
def MACD(SHORT=12, LONG=26, M=9):
    """
    MACD 指数平滑移动平均线
    """
    DIFF = EMA(CLOSE, SHORT) - EMA(CLOSE, LONG)
    DEA = EMA(DIFF, M)
    MACD = (DIFF - DEA) * 2
```

- KDJ随机指标

```
def KDJ(N=9, M1=3, M2=3):
    """
    KDJ 随机指标
    """
    RSV = (CLOSE - LLV(LOW, N)) / (HHV(HIGH, N) - LLV(LOW, N)) * 100
    K = EMA(RSV, (M1 * 2 - 1))
    D = EMA(K, (M2 * 2 - 1))
    J = K * 3 - D * 2

    return K, D, J
```

- RSI相对指标

```
def RSI(N1=6, N2=12, N3=24):
    """
    RSI 相对强弱指标
    """
    LC = REF(CLOSE, 1)
    RSI1 = SMA(MAX(CLOSE - LC, 0), N1, 1) / SMA(ABS(CLOSE - LC), N1, 1) * 100
    RSI2 = SMA(MAX(CLOSE - LC, 0), N2, 1) / SMA(ABS(CLOSE - LC), N2, 1) * 100
    RSI3 = SMA(MAX(CLOSE - LC, 0), N3, 1) / SMA(ABS(CLOSE - LC), N3, 1) * 100

    return RSI1, RSI2, RSI3
```

- BOLL布林带

```
def BOLL(N=20, P=2):
    """
    BOLL 布林带
    """
    MID = MA(CLOSE, N)
    UPPER = MID + STD(CLOSE, N) * P
    LOWER = MID - STD(CLOSE, N) * P

    return UPPER, MID, LOWER
```

- WR威廉指标

```
def WR(N=10, N1=6):
    """
    W&R 威廉指标
    """
    WR1 = (HHV(HIGH, N) - CLOSE) / (HHV(HIGH, N) - LLV(LOW, N)) * 100
    WR2 = (HHV(HIGH, N1) - CLOSE) / (HHV(HIGH, N1) - LLV(LOW, N1)) * 100

    return WR1, WR2
```

- DMI趋向指标

```
    """
    TR = SUM(MAX(MAX(HIGH - LOW, ABS(HIGH - REF(CLOSE, 1))), ABS(LOW - REF(CLOSE, 1
    HD = HIGH - REF(HIGH, 1)
    LD = REF(LOW, 1) - LOW

    DMP = SUM(IF((HD > 0) & (HD > LD), HD, 0), M1)
    DMM = SUM(IF((LD > 0) & (LD > HD), LD, 0), M1)
    DI1 = DMP * 100 / TR
    DI2 = DMM * 100 / TR
    ADX = MA(ABS(DI2 - DI1) / (DI1 + DI2) * 100, M2)
    ADXR = (ADX + REF(ADX, M2)) / 2

    return DI1, DI2, ADX, ADXR
```

- BIAS乖离率

```
def BIAS(L1=5, L4=3, L5=10):
    """
    BIAS 乖离率
    """
    BIAS = (CLOSE - MA(CLOSE, L1)) / MA(CLOSE, L1) * 100
    BIAS2 = (CLOSE - MA(CLOSE, L4)) / MA(CLOSE, L4) * 100
    BIAS3 = (CLOSE - MA(CLOSE, L5)) / MA(CLOSE, L5) * 100

    return BIAS, BIAS2, BIAS3
```

- ASI震动升降指标

```
def ASI(M1=26, M2=10):
    """
    ASI 震动升降指标
    """
    LC = REF(CLOSE, 1)
    AA = ABS(HIGH - LC)
    BB = ABS(LOW - LC)
    CC = ABS(HIGH - REF(LOW, 1))
    DD = ABS(LC - REF(OPEN, 1))
    R = IF((AA > BB) & (AA > CC), AA + BB / 2 + DD / 4, IF((BB > CC) & (BB > AA), B
    X = (CLOSE - LC + (CLOSE - OPEN) / 2 + LC - REF(OPEN, 1))
    SI = X * 16 / R * MAX(AA, BB)
    ASI = SUM(SI, M1)
    ASIT = MA(ASI, M2)

    return ASI, ASIT
```

- VR容量比率

```
def VR(M1=26):
    """
    VR容量比率
    """
    LC = REF(CLOSE, 1)
```

- ARBR人气意愿指标

```
def ARBR(M1=26):
    """
    ARBR人气意愿指标
    """
    AR = SUM(HIGH - OPEN, M1) / SUM(OPEN - LOW, M1) * 100
    BR = SUM(MAX(0, HIGH - REF(CLOSE, 1)), M1) / SUM(MAX(0, REF(CLOSE, 1) - LOW), M

    return AR, BR
```

- DPO

```
def DPO(M1=20, M2=10, M3=6):
    DPO = CLOSE - REF(MA(CLOSE, M1), M2)
    MADPO = MA(DPO, M3)

    return DPO, MADPO
```

- TRIX三重指数平滑均线

```
def TRIX(M1=12, M2=20):
    TR = EMA(EMA(EMA(CLOSE, M1), M1), M1)
    TRIX = (TR - REF(TR, 1)) / REF(TR, 1) * 100
    TRMA = MA(TRIX, M2)

    return TRIX, TRMA
```

工具函数及行情变量

名称	
收盘价	C, CLOSE
开盘价	O, OPEN
最高价	H, HIGH
最低价	L, LOW
成交量	V, VOLUME
10周期前收盘价	REF(CLOSE, 10)
10周期均线	MA(CLOSE, 10)
金叉	CROSS(MA(CLOSE, 5),MA(CLOSE,10))
最大值	MAX(CLOSE, OPEN)

10周期满足条件	EVERY(CLOSE > MA(CLOSE, 5), 10)
10周期收阳线数量	COUNT(CLOSE > OPEN, 10)
10周期收盘价最大值	HHV(CLOSE, 10)
10周期收盘价最小值	LLV(CLOSE, 10)
10周期成交量加总	SUM(VOLUME, 10)

数据

目前我们平台使用的是中国A股市场从2005年到最新交易日的**每分钟以及每日**行情数据。我们后台系统在收到数据提供商提供的原始数据时会自动对其进行预处理。为了使用户摆脱处理数据时的繁琐耗时，我们对数据处理部分进行了大量研究和测试以确保数据的准确性和实时性。另外我们系统采用了面向服务的体系架构以使处理后的数据能安全可靠、极速地传送回用户的策略。

为了使用户能对更多的市场进行算法回测研究，我们自定义了一套统一的金融数据格式。目前我们做了中国A股市场，在不久的将来我们将会支持更多的股市甚至期货期权，如港股，美股或其它您非常感兴趣的市场。美股已经在我们的策略研究平台上支持了。

更多数据内容，请参考数据介绍。

股票数据

我们的原始数据来源于一流的数据提供商，以保障给用户提供最准确的数据源，只有准确的数据才能让策略有保障。目前我们支持多于2700只中国A股市场的股票日线、分钟线行情，其中包括一些已退市的股票，这样可避免所谓的幸存者偏差。另外我们系统也处理了股票的拆分和分红。

每日交易数据纪录了股票在一天内的交易信息，如开盘价或交易量等。目前使用的基准-沪深300（ csi 300 ）很好地反映了中国A股市场的整体走势并且有足够长的时间用于回测，所以我们把它作为平台回测算法的基准参照。

ETF、LOF、分级基金数据

ETF，又称"交易型开放式指数证券投资基金"(Exchange Traded Fund的缩写)，简称"交易型开放式指数基金"，又称"交易所交易基金"。ETF是一种跟踪"标的指数"变化、且在证券交易所上市交易的基金。ETF最大的作用在于投资者可以借助这个金融产品具备的指数期货、商品期货的特性套利操作，有助于提高股市的成交量。

Ricequant上面目前有将近115只ETF的历史数据信息方便您做回测，其价格也如股票历史数据做了拆分的调整，并且也在收益中计算了分红。

可以通过ctrl+i - windows用户，或command+i - mac用户进行代码搜索。

指数数据

我们目前开放了**790多个**每日和每分钟历史指数数据，在代码中的使用方法和股票相同（也可以通过ctrl+i - windows用户，或command+i - mac用户进行代码搜索）。

所支持的指数的详细情况可以查看指数数据字典。

期货数据

Ricequant引入了中国四大期货交易所（大商、郑商、中金、上期）2005年以来的期货每日行情数据以及2010年以来的分钟行情数据。并对合约做了主力连续合约和指数连续合约的合成处理。您可以通过在get_price函数进行获取，并进行相应计算。

需要注意，由于期货合约存续的特殊性，针对每一品种的期货合约，系统中都增加了 **主力连续合约**以及**指数连续合约**两个人工合成的合约来满足使用需求。其中，主力连续合约是由该品种期货不同时期主力合约接续而成，代码以88结尾，例如IF88；指数连续合约是由当前品种全部可交易合约以累计持仓量为权重加权平均得到，代码以99结尾，例如IF99。

主力合约的判断标准：合约首次上市时，以当日收盘同品种持仓量最大者作为从第二个交易日开始的主力合约。当同品种其他合约持仓量在收盘后超过当前主力合约1.1倍时，从第二个交易日开始进行主力合约的切换。日内不会进行主力合约的切换。

财务数据

Ricequant从一流数据提供商处购买并且整理了财务三大表，市场衍生数据和财务指标衍生数据共计400余项，涵盖了A股两市几乎所有股票自上市以来的所有基本面数据。

财务数据/基本面数据可以用来选股，所以主要的应用是在 `init` 或 `before_trading` 函数里的update_universe来加入筛选出来的结果进入股票池或者是使用查询出来的财务指标数据。

在 Ricequant 使用 `get_fundamentals` 函数来查询财务数据库中的所有股票的财务数据。您可以试用类似SQL的语法去查询列、过滤、排序并且限制结果数目等，你可以通过这样来在Ricequant的算法中生成一篮子的证券，基于你选择的公司的财务基本指标。并且你也可以将这些财务指标数据用于你的交易算法或策略研究中。

所有的财务数据指标可以在财务数据字典找到。

财务数据在Ricequant的算法交易中可以有两种使用方法：

- 基于财务数据筛选出股票并加入股票池
- 对你的算法提供数据（在 `handle_bar` 中），来便于做投资决策

有三个函数和财务数据有关： `before_trading` ， `get_fundamentals` ，和 `update_universe` 。

个query。

在下面的例子中，算法策略拿到了每只股票的 pe_ratio 的信息。 get_fundamentals 函数会自动把每个股票的股票代码(order_book_id)包含进来。返回的数据接着会被 filter 按照 pe_ratio 的区间进行过滤。最终，数据会被 order_by 排序并且被 limit 限制最后的查询数据的个数。

以下是一段简单的查询财务数据的代码范例，在 init 里面查询了财务数据，只会在策略运行的时候触发一次，只是用来查询所有股票的收益：

```
# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    context.fundamental_df = get_fundamentals(
        query(
            fundamentals.income_statement.revenue
        )
    )
    logger.info(context.fundamental_df)
```

我们接着看一段进阶的查询范例，查询pe值在25和30之间并且收益排前10的股票：

```
#会在策略开始前触发一次，我们可以从get_fundamentals函数中更新我们的股票池并且保存查询获得
def init(context):
       # 实时打印日志
    context.fundamental_df = get_fundamentals(
        query(
            fundamentals.income_statement.revenue,      fundamentals.eod_derivative
        ).filter(
            fundamentals.eod_derivative_indicator.pe_ratio > 25
        ).filter(
            fundamentals.eod_derivative_indicator.pe_ratio < 30
        ).order_by(
            fundamentals.income_statement.revenue.desc()
        ).limit(
            10
        )
    )
    logger.info(context.fundamental_df)
    update_universe(context.fundamental_df.columns.values)
```

所有的可以查询到的财务数据指标可以在财务数据字典页面找到。我们也提供了一个范例算法来展示如何拿取财务数据结合交易策略。

注意：我们建议您不要使用 get_fundamentals 查询太多的股票，那么会很容易导致程序的运行很慢，因为查询的股票数量过大而10年+的财务数据数据库是十分庞大的。您可以在 query 语句最后加入 limit(limit_number) 来限制查询返回的记录的个数。

调用 get_fundamentals 函数将会返回一个 pandas 的 dataframe ，每一行对应数据库返回的每一行(可能是几个表的联合查询结果的一行), 列索引是你查询的所有字段，可以有几个用法：

```
context.fundamental_df = fundamental_df
```

- 把查询出来的股票池传入 `update_universe` 函数，那么就可以自己手动将这些感兴趣的股票加入整个股票池以获得在 `handle_bar` 中的这些股票的数据更新。下面的代码中的 `context.fundamental_df.columns.values` 即可以拿到查询财务数据返回的 `dataframe` 的列即所有的股票代码的数组：

```
update_universe(context.fundamental_df.columns.values)
```

如果您想打印出来查询完的结果 `logger.info(context.fundamental_df)` ，你将会发现数字的排版非常整洁：

```
                600028.XSHG  600704.XSHG  600019.XSHG  600362.XSHG  \
revenue         1.53684e+12  1.27481e+11  1.22449e+11  1.14881e+11
pe_ratio            26.1758      28.7896      28.3642        28.27

                600755.XSHG  600418.XSHG  601992.XSHG 600600.XSHG  \
revenue         4.31976e+10  3.39457e+10  2.59027e+10  2.4315e+10
pe_ratio            25.3006       26.089      26.1175     26.9428

                601985.XSHG  000921.XSHE
revenue         1.99621e+10  1.92274e+10
pe_ratio            29.6921      27.8182
```

如何只查询某些股票的财务数据？

财务数据时间轴

在ricequant上查询基本面数据时，我们是以所有年报的发布日期（announcement date）为准，因为只有财报发布后才成为市场上公开可以获取的数据。比如某公司第三季度的财报于11月10号发布，那么如果从查询日期为10月5号，也就是早于发布日期，那么返回的只是第二季度的财报数据。这样做最大程度地模拟了真实交易环境，避免了在回测中使用到未来数据。

国债收益率曲线

在 风险计算 中我们使用中国的银行间固定利率国债收益率曲线作为中国市场的零风险利率，来源于中央国债登记结算有限责任公司。收益率曲线如下图所示：

图片描述

收益率曲线数据如下所示：

图片描述

该曲线每日都会变化，我们已经处理和存储了2002年至当日的每日曲线数据从而可以更加准确的计算策略风险值。

在日回测的功能里，Ricequant为您提供了在线编辑器的功能，当您运行策略的过程中遇到问题的时候，您可以在运行时错误tab中您获得错误详情。

譬如，如下图所示，您的代码在运行时抛出异常，那么在运行时错误tab中，您将看到异常发生在策略代码的那一行，以及具体的异常详情。请注意，如果您没有捕获异常的话，策略将会中止。

图片描述

如果你已经遇到了出错，那么在这里您将了解到常见的错误场景，以及应对的办法。如果您遇到无法解决的问题，请联系我们协助您解决。

常见错误如下：

- **所选股票不存在。**
  - 描述： 表示您所选的股票不在我们的日回测历史数据中，因此无法启动该回测。如下图所示：
    图片描述
  - 解决办法 您当前可以使用一些其他的股票资讯网站来查询股票代号，我们非常抱歉还没有推出自动补齐的功能让您方便的查询可交易股票。如果您发现我们有遗漏的股票，我们将非常规感谢您联系我们，帮助我们解决您的问题。

- **策略代码抛出异常。**
  - 描述： 您的策略代码抛出异常且没有被捕获，因此导致策略运行中止。在运行时错误tab中，您可以看到抛出的异常的详情。如下图所示。
    图片描述
  - 解决办法 通过查看抛出异常的详情，您需要可以定位到抛出异常的策略代码中的位置，修复问题，必要时候您可以通过log来打印可能出问题的变量或相关信息。

- **所选股票在所选时间段内无法交易。**
  - 描述： 这是因为所选择的股票在所选的时间范围内没有历史交易数据，因此无法运行您的策略进行回测。
  - 解决办法: 遇到这个错误的原因有多种，后续我们会快速对错误提示进行更细致的调整，改善这个错误的引导提示。如果您遇到这个问题而不知道怎么定位，我们欢迎您联系我们协助。

- **系统繁忙，请稍后重试，如果仍然不行请联系我们。**
  - 描述： ricequant运行您的策略的时候需要使用到硬件资源有限，为了保证服务质量，我们对同时可以运行的策略数量有限制，同时运行的每个用户的回测数上限目前是6个。当系统负荷达到阀值的时候，我们将拒绝新的回测请求。

> - **编译错误。**
>   - 描述： 您的代码中共包含语法错误，在运行时错误的tab中，您可以查询到发生编译错误的策略代码的行号以及错误详情。如下图所示：
>   图片描述
>   - 解决办法： 根据编译错误的提示，您可能需要检查是否按照规范使用Ricequant提供的API，或者您需要检查是否存在python语法错误。如果仍然难以定位问题所在，我们欢迎您联系我们协助。

> - **运行超时。**
>   - 描述： 为了防止恶意的策略常驻在我们的平台上耗费计算资源，我们限制了一个回测最长的生命周期时间。目前限制为8个小时，如果您的策略在8小时内没有能够完成，则会被平台强制停止。而每个事件循环 `handle_bar` 在回测中的限时是 `3分钟` ，实盘模拟中是 `5分钟` 。
>   - 解决办法： 请您不要担心，通常复杂度的策略在平台负载不重的情况下，可以在10-20s内完成。如果您的回测策略确实需要运行比较久的时间，我们欢迎您联系我们协助。

> - **其他错误。**
>   - 描述： 这通常是因为系统内部错误导致。
>   - 解决办法： 欢迎联系我们，我们将第一时间协助您解决这类问题。

隐私及安全

您的策略代码是您宝贵的私有财产。我们**非常重视您的安全和隐私**，会非常严肃地解决安全隐患，Ricequant使用各种措施来保证用户的策略代码及其它私有内容的安全。

除非您自己选择分享您的策略代码，否则它们在通过https加密进入Ricequant平台的一瞬间就会保持加密，其他人不能查看到您的策略，包括Ricequant也不会查阅、分享或以其它方式使用它们。

通常情况下，您会以下列的两种方式授权他人去查看您选择被看到的信息：

- 在Ricequant平台公共处分享您的策略或策略的表现结果 我们不久将推出策略分享的功能，允许您分享自己的策略或策略表现结果，与他人进行研究探讨。
- 您要求Ricequant协助您解决代码中出现的问题 当您的策略出现问题并无法独立解决时，您可以选择让Ricequant协助您去定位问题。如果在提供技术支持或本服务其他维护的过程中，有必要让Ricequant查看您的私有内容，这样的检阅将被限制在非常具体的技术目的，并且我们会向您披露我们的做法。

在Ricequant，我们通过一系列的措施来保护您的策略安全：

- **https通信加密**： 您的浏览器与Ricequant平台之间的通信都经过https加密，这代表着您的策略代码以及代码运行期间产生的所有信息都会被加密，第三者无法窥探。我们

- **加密保存**： 您的策略通过加密网络传输到Ricequant的服务器，我们会对其进行加密，并存放在内网的数据库中。
- **隔离策略运行**： 您的策略在运行的过程中会被放置在一个安全的隔离环境，平台上运行的其他策略无法进入该独立的环境，所以也不能候窥探您的任何信息。在技术上我们通过Java security policy, separete class loader, linux namespace, linux cgroup, linux seccomp, selinux等手段来构建这个隔离环境。如果您对我们使用技术感兴趣的话，可以通过阅读这个安全blog文档获得更多相关内容。
- **云端储存** 我们的所有后台运算和储存都在非常出色的云服务商青云上进行，他们有很专业的安全保护。

如您有对我们的安全保护有任何的疑问，请不要犹豫联系我们。

Python SDK 简介

基本方法

你的算法策略目前必须实现至少两个方法： `init` 和 `handle_bar` ，
而 `before_trading` 和 `after_trading` 是可选择实现的方法。

init (必须实现)

```
init(context)
```

初始化方法 - 在回测和实时模拟交易只会在启动的时候触发一次。你的算法会使用这个方法来设置你需要的各种初始化配置。 `context` 对象将会在你的算法的所有其他的方法之间进行传递以方便你可以拿取到。

参数

参数	类型	注释
context	python简单对象	将会在整个算法中当做一个全局变量来使用。 属性通过点标记（"."）来取到。

返回

无

范例

```
def init(context):
    # cash_limit的属性是根据用户需求自己定义的，你可以定义无限多种自己随后需要的属性，ric
    context.cash_limit = 5000
```

```
handle_bar(context, bar_dict)
```

bar数据的更新会自动触发该方法的调用。策略具体逻辑可在该方法内实现，包括交易信号的产生、订单的创建等。在实时模拟交易中，该函数在交易时间内会每分钟被触发一次。**注意:由于该方法会每分钟被触发，请尽量不要在该函数中放入查询类（如带有query()参数的API）代码以免运行时间过长，该类逻辑可放在 init() 中执行。**

参数

参数	类型	注释
context	python简单对象	储存策略自定义参数、设置、仓位、 投资组合信息的全局变量，属性通过点标记（"."）来取到
bar_dict	dict	key为order_book_id，value为bar数据。 当前合约池内所有合约的bar数据信息都会更新在bar_dict里面

返回

无

范例

```
def handle_bar(context, bar_dict):
    # put all your algorithm main logic here.
    # ...
    order_shares('000001.XSHE', 500)
    # ...
```

before_trading (选择实现)

```
before_trading(context)
```

可选择实现的函数。每天在策略开始交易前会被调用。**不能在这个函数中发送订单。**需要注意，该函数的触发时间取决于用户当前所订阅合约的交易时间。

举例来说，如果用户订阅的合约中存在有夜盘交易的期货合约，则该函数可能会在前一日的20:00触发，而不是早晨08:00.

参数

参数	类型	注释

context	python简单对象	投资组合信息的全局变量，属性通过点标记（"."）来取到。

返回

无

范例

```
def before_trading(context, bar_dict):
    # 拿取财务数据的逻辑，自己构建SQLAlchemy query
    fundamental_df = get_fundamentals(your_own_query)

    # 把查询到的财务数据保存到conext对象中
    context.fundamental_df = fundamental_df

    # 手动更新股票池
    update_universe(context.fundamental_df.columns.values)
```

after_trading (选择实现)

```
after_trading(context)
```

可选择实现的函数。每天在收盘后被调用。**不能在这个函数中发送订单。**您可以在该函数中进行当日收盘后的一些计算。

在实时模拟交易中，该函数会在每天15:30触发。

参数

参数	类型	注释
context	python简单对象	储存策略自定义参数、设置、仓位、 投资组合信息的全局变量，属性通过点标记（"."）来取到。

返回

无

交易相关函数

order_shares - 指定股数交易（股票专用）

```
order_shares(id_or_ins, amount, style=MarketOrder())
```

参数

参数	类型	注释
id_or_ins	str或instrument对象	order_book_id或symbol或instrument对象，用户必须指定
amount	float-required	需要落单的股数。正数代表买入， 负数代表卖出。 将会根据一手xx股来向下调整到一手的倍数，比如中国A股就是调整成100股的倍数。
style	OrderType	订单类型，默认是市价单。 目前支持的订单类型有： • style=MarketOrder() • style=LimitOrder(limit_price)

返回

Order对象

范例

- 购买Buy 2000 股的平安银行股票，并以市价单发送：

```
order_shares('000001.XSHE', 2000)
```

- 卖出2000股的平安银行股票，并以市价单发送：

```
order_shares('000001.XSHE', -2000)
```

- 购买1000股的平安银行股票，并以限价单发送，价格为￥10：

```
order_shares('000001.XSHE', 1000, style=LimitOrder(10))
```

order_lots - 指定手数交易（股票专用）

```
order_lots(id_or_ins, amount, style=OrderType)
```

指定手数发送买/卖单。如有需要落单类型当做一个参量传入，如果忽略掉落单类型，那么默认是市价单（market order）。

参数

参数	类型	注释

id_or_ins	*str*或instrument对象	用户必须指定
amount	*float*	多少手的数目。正数表示买入， 负数表示卖出，用户必须指定
style	*OrderType*	订单类型，默认是市价单。 目前支持的订单类型有： • style=MarketOrder • style=LimitOrder(limit_price)

返回

Order对象

范例

- 买入20手的平安银行股票，并且发送市价单：

```
order_lots('000001.XSHE', 20)
```

- 买入10手平安银行股票，并且发送限价单，价格为￥10：

```
order_lots('000001.XSHE', 10, style=LimitOrder(10))
```

order_value - 指定价值交易（股票专用）

```
order_value(id_or_ins, cash_amount, style=OrderType)
```

使用想要花费的金钱买入/卖出股票，而不是买入/卖出想要的股数，正数代表买入，负数代表卖出。股票的股数总是会被调整成对应的100的倍数（在A中国A股市场1手是100股）。当您提交一个卖单时，该方法代表的意义是您希望通过卖出该股票套现的金额。如果金额超出了您所持有股票的价值，那么您将卖出所有股票。需要注意，如果资金不足，该API将不会创建发送订单。

参数

参数	类型	注释
id_or_ins	*str*或instrument对象	order_book_id或symbol或instrument对象 用户必须指定
cash_amount	*float*	需要花费现金购买/卖出证券的数目。 正数代表买入，负数代表卖出， 用户必须指定

style	*OrderType*	目前支持的订单类型有： • style=MarketOrder() • style=LimitOrder(limit_price)

返回

Order对象

范例

- 买入价值￥10000的平安银行股票，并以市价单发送。如果现在平安银行股票的价格是￥7.5，那么下面的代码会买入1300股的平安银行，因为少于100股的数目将会被自动删除掉：

```
order_value('000001.XSHE', 10000)
```

- 卖出价值￥10000的现在持有的平安银行：

```
order_value('000001.XSHE', -10000)
```

order_percent - 一定比例下单（股票专用）

```
order_percent(id_or_ins, percent, style=OrderType)
```

发送一个等于目前投资组合价值（市场价值和目前现金的总和）一定百分比的买/卖单，正数代表买，负数代表卖。股票的股数总是会被调整成对应的一手的股票数的倍数（1手是100股）。百分比是一个小数，并且小于或等于1（<=100%），0.5表示的是50%.需要注意，如果资金不足，该API将不会创建发送订单。

参数

参数	类型	注释
id_or_ins	*str*或instrument对象	order_book_id或symbol或instrument object，用户必须指定
percent	*float*	占有现有的投资组合价值的百分比。 正数表示买入，负数表示卖出。 用户必须指定
style	*OrderType*	订单类型，默认是市价单。 目前支持的订单类型有： • style=MarketOrder() • style=LimitOrder(limit_price)

范例

- 买入等于现有投资组合50%价值的平安银行股票。如果现在平安银行的股价是￥10/股并且现在的投资组合总价值是￥4000，用来买入的资金为￥2000，那么将会买入200股的平安银行股票。（不包含交易成本和滑点的损失）：

```
order_percent('000001.XSHE', 0.5)
```

order_target_value - 目标价值下单（股票专用）

```
order_target_value(id_or_ins, cash_amount, style=OrderType)
```

买入/卖出并且自动调整该证券的仓位到一个目标价值。如果还没有任何该证券的仓位，那么会买入全部目标价值的证券。如果已经有了该证券的仓位，则会买入/卖出调整该证券的现在仓位和目标仓位的价值差值的数目的证券。需要注意，如果资金不足，该API将不会创建发送订单。

参数

参数	类型	注释
id_or_ins	*str*或instrument对象	order_book_id或symbol或instrument object，用户必须指定
cash_amount	float-required	最终的该证券的仓位目标价值
style	*OrderType*	订单类型，默认是市价单。 目前支持的订单类型有： • style=MarketOrder() • style=LimitOrder(limit_price)

返回

Order对象

范例

- 如果现在的投资组合中持有价值￥3000的平安银行股票的仓位并且设置其目标价值为￥10000，以下代码范例会发送价值￥7000的平安银行的买单到市场。（向下调整到最接近每手股数即100的倍数的股数）：

```
order_target_value('000001.XSHE', 10000)
```

order_target_percent - 目标比例下单（股票专用）

买入/卖出证券以自动调整该证券的仓位到占有一个指定的投资组合的目标百分比。

- 如果投资组合中没有任何该证券的仓位，那么会买入等于现在投资组合总价值的目标百分比的数目的证券。
- 如果投资组合中已经拥有该证券的仓位，那么会买入/卖出目标百分比和现有百分比的差额数目的证券，最终调整该证券的仓位占据投资组合的比例至目标百分比。

其实我们需要计算一个position_to_adjust (即应该调整的仓位)

```
position_to_adjust = target_position - current_position
```

投资组合价值等于所有已有仓位的价值和剩余现金的总和。买/卖单会被下舍入一手股数（A股是100的倍数）的倍数。目标百分比应该是一个小数，并且最大值应该<=1，比如0.5表示50%。

如果 `position_to_adjust` 计算之后是正的，那么会买入该证券，否则会卖出该证券。 需要注意，如果资金不足，该API将不会创建发送订单。

参数

参数	类型	注释
id_or_ins	*str*或instrument对象	order_book_id或symbol或instrument object，用户必须指定
percent	float-required	仓位最终所占投资组合总价值的目标百分比。
style	*OrderType*	订单类型，默认是市价单。 目前支持的订单类型有： • style=MarketOrder() • style=LimitOrder(limit_price)

返回

Order对象

范例

- 如果投资组合中已经有了平安银行股票的仓位，并且占据目前投资组合的10%的价值，那么以下代码会买入平安银行股票最终使其占据投资组合价值的15%：

```
order_target_percent('000001.XSHE', 0.15)
```

buy_open - 买开（期货专用）

```
buy_open(id_or_ins, amount, style=OrderType)
```

参数	类型	注释
id_or_ins	str或instrument对象	order_book_id或symbol或instrument对象
amount	float	下单的手数
style	OrderType	订单类型，默认是市价单。 目前支持的订单类型有： • style=MarketOrder() • style=LimitOrder(limit_price)

返回

Order对象

范例

- 以价格为3500的限价单开仓买入2张上期所AG1607合约：

```
buy_open('AG1607', amount=2, style=LimitOrder(3500))
```

sell_close - 平多仓（期货专用）

```
sell_close(id_or_ins, amount, style=OrderType)
```

平多仓。

参数

参数	类型	注释
id_or_ins	str或instrument对象	order_book_id或symbol或instrument对象
amount	float	下单的手数
style	OrderType	订单类型，默认是市价单。 目前支持的订单类型有： • style=MarketOrder() • style=LimitOrder(limit_price)

返回

Order对象

sell_open - 卖开（期货专用）

卖出开仓。

参数

参数	类型	注释
id_or_ins	str或instrument对象	order_book_id或symbol或instrument对象
amount	float	下单的手数
style	OrderType	订单类型，默认是市价单。 目前支持的订单类型有： • style=MarketOrder() • style=LimitOrder(limit_price)

返回

Order对象

buy_close - 平空仓（期货专用）

```
buy_close(id_or_ins, amount, style=OrderType)
```

平空仓。

参数

参数	类型	注释
id_or_ins	*str或instrument对象*	order_book_id或symbol或instrument对象
amount	*float*	下单的手数
style	*OrderType*	订单类型，默认是市价单。 目前支持的订单类型有： • style=MarketOrder() • style=LimitOrder(limit_price)

返回

Order对象

范例

- 市价单将现有IF1603空仓买入平仓2张：

```
buy_close('IF1603', 2)
```

```
cancel_order(order)
```

参数

参数	类型	注释
order	Order	需要撤销的order对象，用户必须指定

返回

无

get_open_orders - 拿到未成交订单信息

```
get_open_orders()
```

获取一个order对象的list，凡在此list中的order都未被完全成交或取消。

参数

无

返回

list of Orders， 当前所有活跃的订单（未全部成交，未被撤单）。

context属性

now - 当前时间

```
context.now
```

使用以上的方式就可以在 `handle_bar` 中拿到当前的bar的时间，比如day bar的话就是那天的时间，minute bar的话就是这一分钟的时间点。

返回数据类型为datetime.datetime

portfolio - 投资组合信息

```
context.portfolio
```

该投资组合在单一股票或期货策略中分别为股票投资组合和期货投资组合。在股票+期货的混合策略中代表汇总之后的总投资组合。请参考 portfolio

运行中的策略

```
context.stock_account
```

获取股票资金账户信息。请参考 stock_account

future_account - 期货资金账户信息

```
context.future_account
```

获取期货资金账户信息。请参考 future_account

run_info - 策略运行信息

```
context.run_info
```

属性	类型	注释
run_id	str	标识策略每次运行的唯一id
run_type	*RUN_TYPE*	RUN_TYPE.BACKTEST表示当前 RUN_TYPE.PAPER_TRADING表 行实盘模拟
start_date	datetime.date	策略的开始日期
end_date	datetime.date	策略的结束日期
frequency	str	策略频率，'1d'或'1m'
stock_starting_cash	float	股票账户初始资金
future_starting_cash	float	期货账户初始资金
slippage	float	滑点水平
margin_multiplier	float	保证金倍率
commission_multiplier	float	佣金倍率
benchmark	str	基准合约代码
matching_type	*MATCHING_TYPE*	撮合方式， MATCHING_TYPE.NEXT_BAR_C ar开盘价撮合， MATCHING_TYPE.CURRENT_BA 以当前bar收盘价撮合

```
context.universe
```

在运行update_universe，subscribe或者unsubscribe的时候，合约池会被更新。

需要注意，合约池内合约的交易时间（包含股票的策略默认会在股票交易时段触发）是handle_bar被触发的依据。

scheduler定时器

scheduler.run_daily - 每天运行

```
scheduler.run_daily(function)
```

每日运行一次指定的函数，**只能在init内使用。**

注意，schedule一定在其对应时间点的handle_bar之前执行，如果定时运行函数运行时间较长，则中间的handle_bar事件将会被略过。

参数

参数	类型	注释
function	function	使传入的 `function` 每日运行。**注意**，function函数一定要包含（并且只能包含）context, bar_dict两个输入参数

返回

无

范例

以下的范例代码片段是一个非常简单的例子，在每天交易开始时查询现在 `portfolio` 中剩下的cash的情况：

```
#scheduler调用的函数需要包括context, bar_dict两个输入参数
def log_cash(context, bar_dict):
    logger.info("Remaning cash: %r" % context.portfolio.cash)

def init(context):
    #...
    # 每天运行一次
    scheduler.run_daily(log_cash)
```

scheduler.run_weekly - 每周运行

每周运行一次指定的函数，**只能在init内使用。**

注意：

- `tradingday` 中的负数表示倒数。
- `tradingday` 表示交易日。如某周只有四个交易日，则此周的 `tradingday=4` 与 `tradingday=-1` 表示同一天。
- `weekday` 和 `tradingday` 不能同时使用。

参数

参数	类型	注释
function	*function*	使传入的 `function` 每日交易开始前运行。**注意**，function函数一定要包含（并且只能包含）context, bar_dict两个输入参数
weekday	*int*	1~5 分别代表周一至周五，用户必须指定
tradingday	*int*	范围为[-5,1],[1,5] 例如， 1代表每周第一个交易日，-1代表每周倒数第一个交易日，用户可以不填写

返回

无

范例

以下的代码片段非常简单，在每周二固定运行打印一下现在的 `portfolio` 剩余的资金：

```
#scheduler调用的函数需要包括context, bar_dict两个参数
def log_cash(context, bar_dict):
    logger.info("Remaning cash: %r" % context.portfolio.cash)

def init(context):
    #...
    # 每周二打印一下剩余资金:
    scheduler.run_weekly(log_cash, weekday=2)
    # 每周第二个交易日打印剩余资金:
    #scheduler.run_weekly(log_cash, tradingday=2)
```

scheduler.run_monthly - 每月运行

```
scheduler.run_monthly(function,tradingday=t)
```

每月运行一次指定的函数，**只能在init内使用。**

- `tradingday` 表示交易日，如某月只有三个交易日，则此月的tradingday=3与tradingday=-1表示同一。

参数

参数	类型	注释
function	*function*	使传入的 `function` 每日交易开始前运行。**注意**，function函数一定要包含（并且只能包含）context, bar_dict两个输入参数
tradingday	*int*	范围为[-23,1], [1,23] ，例如， 1代表每月第一个交易日，-1代表每月倒数第一个交易日，用户必须指定

返回

无

范例

以下的代码片段非常简单的展示了每个月第一个交易日的时候我们进行一次财务数据查询，这样子会非常有用在一些根据财务数据来自动调节仓位股票组合的算法来说：

```
#scheduler调用的函数需要包括context, bar_dict两个参数
def query_fundamental(context, bar_dict):
        # 查询revenue前十名的公司的股票并且他们的pe_ratio在25和30之间。打fundamentals的
    fundamental_df = get_fundamentals(
        query(
            fundamentals.income_statement.revenue, fundamentals.eod_derivative_indi
        ).filter(
            fundamentals.eod_derivative_indicator.pe_ratio > 25
        ).filter(
            fundamentals.eod_derivative_indicator.pe_ratio < 30
        ).order_by(
            fundamentals.income_statement.revenue.desc()
        ).limit(
            10
        )
    )

    # 将查询结果dataframe的fundamental_df存放在context里面以备后面只需：
    context.fundamental_df = fundamental_df

    # 实时打印日志看下查询结果，会有我们精心处理的数据表格显示：
    logger.info(context.fundamental_df)
    update_universe(context.fundamental_df.columns.values)

 # 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    # 每月的第一个交易日查询以下财务数据，以确保可以拿到最新更新的财务数据信息用来调整仓位
    scheduler.run_monthly(query_fundamental, tradingday=1)
```

`scheduler` 还可以用来做定时间运行，比如在每天开盘后的一小时后或一分钟后定时运行，这里有很多种组合可以让您达到各种自己想要达到的定时运行的目的。

使用的方法是和上面
的 `scheduler.run_daily`，`scheduler.run_weekly` 和 `scheduler.run_monthly` 进行组合加入 `time_rule` 来一起使用。

注意:

- `market_open` 与 `market_close` 都跟随中国A股交易时间进行设置，即09:31~15:00。
- 使用 `time_rule` 定时运行只会在分钟级别回测和实时模拟交易中有定义的效果，在日回测中只会默认依然在该天运行，并不能在固定的时间运行。
- 在分钟回测中如未指定 `time_rule` ,则默认在开盘后一分钟运行,即09:31分。
- 如果两个 `schedule` ，分别使用 `market_open` 与 `market_close` 规则，但规则触发时间在同一时刻，则 `market_open` 的 `handle` 一定在 `market_close` 的 `handle` 前执行。
- 目前暂不支持开盘交易(即 09:30分交易) ,所以 `time_rule(minute=0)` 和 `time_rule(hour=0)` 将不会触发任何事件。
- market_open(minute=120)将在11:30执行， market_open(minute=121)在13:01执行，中午休市的区间会被忽略。
- time_rule='before_trading'表示在开市交易前运行scheduler函数。该函数运行时间将在before_trading函数运行完毕之后handle_bar运行之前。

参数

参数	类型	注释
time_rule	market_open, market_close, str	定时具体几点几分运行某个函数。 time_rule='before_trading' 表示开始交易前运行； market_open(hour=x, minute=y) 表示A股市场开市后x小时y分钟运行， market_close(hour=x, minute=y) 表示A股市场收市前x小时y分钟运行。 如果不设置 `time_rule` 默认的值是中国A股市场开市后 钟运行。

market_open, market_close参数如下：

参数	类型	注释
hour	int - option [1,4]	具体在market_open/market_close后/前第多少小时执行, 股票的交易时间为[9:31 - 11:30],[13:01 - 15:00]共240分钟， 所以hour的范围为 [1,4]
minute	int - option [1,240]	具体在market_open/market_close的后/前第多少分钟执行, 同上，股票每天交易时间240分钟，所以minute的范围为 [1,240],中午休市的时间区间会被忽略。

范例

- **每天**的开市后10分钟运行：

```
scheduler.run_daily(function, time_rule=market_open(minute=10))
```

- **每周**的第t个交易日闭市前1小时运行：

```
scheduler.run_weekly(function, tradingday=t, time_rule=market_close(hour=1))
```

- **每月**的第t个交易日开市后1小时运行：

```
scheduler.run_monthly(function, tradingday=t, time_rule=market_open(hour=1))
```

- **每天开始交易前**运行：

```
scheduler.run_daily(function, time_rule='before_trading')
```

数据查询相关函数

get_fundamentals - 查询财务数据

```
get_fundamentals(query, entry_date=None, interval='1d', report_quarter=False)
```

获取历史财务数据表格。目前支持中国市场超过400个指标，具体请参考 财务数据文档 。目前仅支持中国市场。需要注意，一次查询过多股票的财务数据会导致系统运行缓慢。

参数

参数	类型	说明
query	*SQLAlchemyQueryObject*	SQLAlchmey的Query对象。 其中可在'query'内填写需要查询的指标，'filter'内填写数据过滤条件。具体可参考 sqlalchemy's query documentation 学习使用更多的方便的查询语句。 从数据科学家的观点来看，sqlalchemy的使用比sql更加简单和强大

entry_date	*datetime.datetime*, *pandasTimestamp*	应早于策略当前日期。 默认为策略当前日期前一天。
interval	*str*	查询财务数据的间隔， 默认为'1d'。例如，填写'5y'， 则代表从entry_date开始 （包括entry_date）回溯5年， 返回数据时间以**年**为间隔。'd' - 天，'m' - 月， 'q' - 季，'y' - 年
report_quarter	*bool*	是否显示报告期，默认为False， 不显示。'Q1' - 一季报，'Q2' - 半年报，'Q3' - 三季报，'Q4' - 年报

返回

pandas DataPanel 如果查询结果为空，返回空*pandas DataFrame* 如果给定间隔为1d, 1m, 1q, 1y，返回*pandas DataFrame*

范例

- 获取财务数据中的pe_ration和revenue指标：

```
# 并且通过filter过滤掉得到符合一定范围的pe_ration的结果
# 最后只拿到按照降序排序之后的前10个
fundamental_df = get_fundamentals(
     query(
                fundamentals.income_statement.revenue, fundamentals.eod_derivative_
            ).filter(
                fundamentals.eod_derivative_indicator.pe_ratio > 25
            ).filter(
                fundamentals.eod_derivative_indicator.pe_ratio < 30
            ).order_by(
                fundamentals.income_statement.revenue.desc()
            ).limit(
                10
            )
    )
context.stocks = fundamental_df.columns.values
update_universe(context.stocks)
```

- 获取某些指定股票的历史财务数据：

```
def init(context):
    context.stocks = industry('A01')
    logger.info("industry stocks: " + str(context.stocks))

    #每个表都有一个stockcode在用来方便通过股票代码来过滤掉查询的数据，比如次数是只查询'A0
    #最后加入 entry_date 参数获取2015年12月31日的数据
    context.fundamental_df = get_fundamentals(
```

```
            fundamentals.eod_derivative_indicator.pe_ratio > 5
        ).filter(
            fundamentals.eod_derivative_indicator.pe_ratio < 300
        ).filter(
            fundamentals.income_statement.stockcode.in_(context.stocks)
            ), entry_date='20151231'
    )
    logger.info(context.fundamental_df)
    update_universe(context.fundamental_df.columns.values)
```

all_instruments - 所有合约基础信息

```
all_instruments(type=None)
```

获取某个国家市场的所有合约信息。使用者可以通过这一方法很快地对合约信息有一个快速了解，目前仅支持中国市场。

参数

参数	类型	说明
type	*str*	需要查询合约类型，例如：type='CS'代表股票。默认是所有类型

其中type参数传入的合约类型和对应的解释如下：

合约类型	说明
CS	Common Stock, 即股票
ETF	Exchange Traded Fund, 即交易所交易基金
LOF	Listed Open-Ended Fund，即上市型开放式基金
FenjiMu	Fenji Mu Fund, 即分级母基金
FenjiA	Fenji A Fund, 即分级A类基金
FenjiB	Fenji B Funds, 即分级B类基金
INDX	Index, 即指数
Future	Futures，即期货，包含股指、国债和商品期货

返回

pandas DataFrame - 所有合约的基本信息。

范例

- 获取中国市场所有分级基金的基础信息：

```
   abbrev_symbol    order_book_id    product    sector_code    symbol
0    CYGA    150303.XSHE    null    null    华安创业板50A
1    JY500A    150088.XSHE    null    null    金鹰500A
2    TD500A    150053.XSHE    null    null    泰达稳健
3    HS500A    150110.XSHE    null    null    华商500A
4    QSAJ    150235.XSHE    null    null    鹏华证券A
...
```

instruments - 合约详细信息

```
instruments(id_or_symbols)
```

获取某个国家市场内一个或多个合约的详细信息。目前仅支持中国市场。

参数

参数	类型	说明
id_or_symbols	*str* OR *str* *list*	合约代码或合约代码列表，可传入order_book_id, order_book_id list。 中国市场的order_book_id通常类似'000001.XSHE'。 需要注意，国内股票、ETF、 指数合约代码分别应当以'.XSHG'或'.XSHE'结尾， 前者代表上证，后者代表深证。期货则无此要求

返回

一个Instrument对象，或一个instrument list。

目前系统**并不支持**跨国家市场的同时调用。传入 order_book_id list必须属于同一国家市场，不能混合着中美两个国家市场的order_book_id。

范例

- 获取单一股票合约的详细信息：

```
[In]instruments('000001.XSHE')
[Out]
Instrument(order_book_id=000001.XSHE, symbol=平安银行, abbrev_symbol=PAYH, listed_da
```

- 获取多个股票合约的详细信息：

```
[In]instruments(['000001.XSHE', '000024.XSHE'])
[Out]
[Instrument(order_book_id=000001.XSHE, symbol=平安银行, abbrev_symbol=PAYH, listed_d
```

- 获取合约已上市天数：

- 获取合约距离到期天数：

```
instruments('IF1701').days_to_expire()
```

history - 合约历史数据（已废弃）

```
history(bar_count, frequency, field)
```

该API已废弃，建议使用history_bars的方式进行数据获取。

`history` 函数返回所有已关注证券（当前universe中包括的所有证券）的历史行情，同时支持**日以及分钟**历史数据。获取不在股票池（universe）内的合约历史行情将会导致效率低下。 **注意**该API只支持股票历史行情的获取，不支持期货历史行情。如有需要，请使用 `history_bars` 进行获取。

日回测获取分钟历史数据：不支持

日回测获取日历史数据history(1, '1d', 'close')

调用时间	返回数据
T日before_trading	T-1日day bar
T日handle_bar	T日day bar

分钟回测获取日历史数据history(1, '1d', 'close')

调用时间	返回数据
T日before_trading	T-1日day bar
T日handle_bar	T-1日day bar

分钟回测获取分钟历史数据history(1, '1m', 'close')

调用时间	返回数据
T日before_trading	T-1日最后一个minute bar
T日handle_bar	T日当前minute bar

参数

参数	类型	注释
bar_count	*int*	表示回溯的bar的数量，用户必须填写

frequency	*str*	例如"1d"或"1m"分别表示每日和每分钟，用户必须填写
field	*str*	制定返回的DataFrame中以哪个指标作为数据值， 用户必须填写。见下方列表

field	字段名
open	开盘价
high	最高价
low	最低价
close	收盘价
volume	成交量
total_turnover	成交额
datetime	时间

返回

pandas DataFrame - 当前策略股票池中所有股票对应数据的回溯结果。 **注意**，history函数可以通过后面使用[]的方式传入证券的order_book_id来返回某一证券的回溯结果。这一证券不一定存在于当前的universe中。用户进行如此调用将返回Series类型数据，并且这一证券并不会自动添加到当前universe中。

范例

- 获取当前universe的历史数据（假设当前universe中包含601998.XSHG 000002.XSHE两只股票，当前策略日期为2016-01-04 ）：

```
print(history(4, '1d', 'close')
```

```
            601998.XSHG  000002.XSHE
2015-12-29         7.31        24.43
2015-12-30         7.32        24.43
2015-12-31         7.22        24.43
2016-01-04         6.76        24.43
```

history_bars - 某一合约历史数据

```
history_bars(order_book_id, bar_count, frequency, fields=None, skip_suspended=True,
```

获取指定合约的历史行情，同时支持**日以及分钟**历史数据。不能在init中调用。

参数

bar_count	*int*	获取的历史数据数量，必填项
frequency	*str*	获取数据什么样的频率进行。'1d'或'1m'分别表示每日和 必填项。您可以指定不同的分钟频率，例如'5m'代表5分
fields	*str* OR *str* *list*	返回数据字段。必填项。见下方列表
skip_suspended	*bool*	是否跳过停牌，默认True，跳过停牌
include_now	*bool*	是否包括不完整的bar数据。默认为False，不包括。举 在09:39的时候获取上一个5分钟线， 默认将获取到09:31~09:35合成的5分钟线。如果设置为 则将获取到09:36~09:39之间合成的"不完整"5分钟线

fields	字段名
datetime	时间戳
open	开盘价
high	最高价
low	最低价
close	收盘价
volume	成交量
total_turnover	成交额
datetime	int类型时间戳
open_interest	持仓量（期货专用）
basis_spread	期现差（股指期货专用）
settlement	结算价（期货日线专用）
prev_settlement	结算价（期货日线专用）

返回

ndarray ，方便直接与talib等计算库对接，效率较history返回的DataFrame更高。

范例

- 获取最近5天的日线收盘价序列（策略当前日期为20160706）

```
[ 8.69  8.7   8.71  8.81  8.81]
```

- 获取最近2个5分钟bar数据时间戳以及成交量（策略当前日期为2015-02-10）

```
[In]
    logger.info('INCLUDE NOW')
    logger.info(history_bars(context.s1, 2, '5m', ['datetime','volume'],include_now
    logger.info('NO INCLUDE NOW')
    logger.info(history_bars(context.s1, 2, '5m', ['datetime','volume'],include_now


[Out]
2016-07-01 09:31:00.00  INFO   INCLUDE NOW
2016-07-01 09:31:00.00  INFO   [(20160630150000, 1420219) (20160701093100, 665317)]
2016-07-01 09:31:00.00  INFO   NO INCLUDE NOW
2016-07-01 09:31:00.00  INFO   [(20160630145500, 654006) (20160630150000, 1420219)]
```

注：

- 因为撮合逻辑是当前bar收盘或者下一个bar开盘，所以history_bars()可以获取到**包含当前bar**及之前所有的bar数据
- bar数据为切片数据，例如当前时间为12：00，使用 `history_bars(context.s1,3,'60m','close',include_now=True)` ，获取到的是(9:30,10:30],(10:30,11:30],(11:30,12:00]三个时间段的数据（切片是从开盘计算）。

get_price - 合约历史数据

```
get_price(order_book_ids, start_date, end_date=None, frequency='1d', fields=None, a
```

获取指定合约或合约列表的历史行情（包含起止日期，日线或分钟线），不能在'handle_bar'函数中进行调用。

注意，这一函数主要是为满足在研究平台编写策略习惯而引入。在编写策略中，使用history_bars进行数据获取会更方便。

参数

参数	类型	说明
order_book_ids	*str* OR *str list*	合约代码，合约代码，可传入order_book_id, order_book_id list
start_date	*str*, *datetime.date*, *datetime.datetime*, *pandasTimestamp*	开始日期，用户必须指定

end_date	*datetime.datetime*, *pandasTimestamp*	结束日期，默认为策略当前日期前一天
frequency	*str*	历史数据的频率。 现在支持**日**/**分钟级别**的历史数据，默认为'1d'。使用者可自由选取不同频率，例如'5m'代表5分钟线
fields	*str* OR *str list*	返回字段名称
adjust_type	*str*	权息修复方案。前复权 - `pre` ，后复权 - `post` ，不复权 - `none` 。
skip_suspended	*bool*	是否跳过停牌数据。默认为False，不跳过，用停牌前数据进行补齐。True则为跳过停牌期。**注意**，当设置为True时，函数order_book_id只支持单个合约传入

返回

pandas Panel/DataFrame/Series

- 传入一个order_book_id，多个fields，函数会返回一个*pandas DataFrame*
- 传入一个order_book_id，一个field，函数会返回*pandas Series*
- 传入多个order_book_id，一个field，函数会返回一个*pandas DataFrame*
- 传入多个order_book_id，函数会返回一个*pandas Panel*

参数	类型	说明
~~OpeningPx~~ open	*float*	开盘价
~~ClosingPx~~ close	*float*	收盘价
~~HighPx~~ high	*float*	最高价
~~LowPx~~ low	*float*	最低价
~~LimitUp~~ limit_up	*float*	涨停价
~~LimitDown~~ limit_down	*float*	跌停价
~~TotalTurnover~~ total_turnover	*float*	总成交额
~~TotalVolumeTraded~~ volume	*float*	总成交量

acc_net_value	*float*	累计净值（仅限基金日线数据）
~~UnitNetValue~~ unit_net_value	*float*	单位净值（仅限基金日线数据）
~~DiscountRate~~ discount_rate	*float*	折价率（仅限基金日线数据）
~~SettlPx~~ settlement	*float*	结算价（仅限期货日线数据）
~~PrevSettlPx~~ prev_settlement	*float*	昨日结算价（仅限期货日线数据）
~~OpenInterest~~ open_interest	*float*	累计持仓量（期货专用）
~~BasisSpread~~ basis_spread	*float*	基差点数（股指期货专用， 股指期货收盘价-标的指数收盘价）
~~TradingDate~~ trading_date	*pandasTimeStamp*	交易日期 （仅限期货分钟线数据）， 对应期货夜盘的情况

范例

- 获取单一股票历史日线行情（返回*pandas DataFrame*）：

```
[In]get_price('000001.XSHE', start_date='2015-04-01', end_date='2015-04-12')
[Out]
open    close    high    low    total_turnover    volume    limit_up    limit_down
2015-04-01    10.7300    10.8249    10.9470    10.5469    2.608977e+09    236637563
2015-04-02    10.9131    10.7164    10.9470    10.5943    2.222671e+09    202440588
2015-04-03    10.6486    10.7503    10.8114    10.5876    2.262844e+09    206631550
2015-04-07    10.9538    11.4015    11.5032    10.9538    4.898119e+09    426308008
2015-04-08    11.4829    12.1543    12.2628    11.2929    5.784459e+09    485517069
2015-04-09    12.1747    12.2086    12.9208    12.0255    5.794632e+09    456921108
2015-04-10    12.2086    13.4294    13.4294    12.1069    6.339649e+09    480990210
...
```

current_snapshot - 当前快照数据

```
current_snapshot(id_or_symbol)
```

获得当前市场快照数据。只能在日内交易阶段调用，获取当日调用时点的市场快照数据。市场快照数据记录了每日从开盘到当前的数据信息，可以理解为一个动态的day bar数据。在目前分钟回测中，快照数据为当日所有分钟线累积而成，一般情况下，最后一个分钟线获取到的快照数据应当与当日的日线行情保持一致。**需要注意**，在实盘模拟中，该函数返回的是调用当时的市场快照情况，所以在同一个handle_bar中不同时点调用可能返回的数

参数

参数	类型	注释
id_or_symbol	*str*	合约代码或简称

返回

调用时点的市场快照snapshot

范例

- 在handle_bar中调用该函数，假设策略当前时间是20160104 09:33

```
[In]
logger.info(current_snapshot('000001.XSHE'))
[Out]
2016-01-04 09:33:00.00  INFO
Snapshot(order_book_id: '000001.XSHE', datetime: datetime.datetime(2016, 1, 4, 9, 3
```

get_dominant_future - 期货主力合约

```
get_dominant_future(underlying_symbol)
```

获取某一期货品种策略当前日期的主力合约代码。 合约首次上市时，以当日收盘同品种持仓量最大者作为从第二个交易日开始的主力合约。当同品种其他合约持仓量在收盘后超过当前主力合约1.1倍时，从第二个交易日开始进行主力合约的切换。日内不会进行主力合约的切换。合约品种详情请见交易费用。

参数

参数	类型	说明
underlying_symbol	*str*	期货合约品种，例如沪深300股指期货为'IF'

返回

str - 主力合约order_book_id

范例

- 获取某一天的主力合约代码（策略当前日期是20160801）

```
[In]
get_dominant_future('IF')
[Out]
'IF1608'
```

```
get_future_contracts(underlying_symbol)
```

获取某一期货品种在策略当前日期的可交易合约order_book_id列表。按照到期月份，下标从小到大排列，返回列表中第一个合约对应的就是该品种的近月合约。

参数

参数	类型	说明
underlying_symbol	*str*	期货合约品种，例如沪深300股指期货为'IF'

返回

str list - 当日可交易的order_book_id list

范例

```
[In]
logger.info(get_future_contracts('IF'))
[Out]
['IF1612', 'IF1701', 'IF1703', 'IF1706']
```

get_securities_margin - 融资融券信息

```
get_securities_margin(id_or_symbols, count=1, fields=None)
```

获取融资融券信息。包括深证融资融券数据以及上证融资融券数据情况。既包括个股数据，也包括市场整体数据。需要注意，T日的数据将在T+1日上午09:00左右更新，所以可能无法在before_trading阶段获取到上一交易日的最新数据。融资融券的开始日期为2010年3月31日。

参数

参数	类型	说明
id_or_symbols	*str* or *str list*	可输入order_book_id, order_book_id list。另外，输入'XSHG'或'sh'代表整个上证整体情况；'XSHE'或'sz'代表深证整体情况
count	*int*	回溯获取的数据个数。 默认为当前能够获取到的最近的数据
fields	*str* OR *str list*	默认为所有字段。见下方列表

buy_on_margin_value	融资买入额
margin_repayment	融资偿还额
short_balance	融券余额
short_balance_quantity	融券余量
short_sell_value	融券卖出额
short_sell_quantity	融券卖出量
short_repayment_quantity	融券偿还量
total_balance	融资融券余额

返回

- 多个order_book_id，单个field的时候返回*pandas DataFrame*，index为date，column为order_book_id
- 单个order_book_id，多个fields的时候返回*pandas DataFrame*，index为date，column为fields
- 单个order_book_id，单个field返回*pandas Series*
- 多个order_book_id，多个fields的时候返回*pandas Panel* Items axis为fields Major_axis axis为时间戳 Minor_axis axis为order_book_id

范例

- 获取沪深两个市场一段时间内的融资余额

```
[In]
logger.info(get_securities_margin('510050.XSHG', count=5))
[Out]
margin_balance    buy_on_margin_value    short_sell_quantity    margin_repayment
2016-08-01    7.811396e+09    50012306.0    3597600.0    41652042.0    15020600.0
2016-08-02    7.826381e+09    34518238.0    2375700.0    19532586.0    14154000.0
2016-08-03    7.733306e+09    17967333.0    4719700.0    111043009.0    16235600.0
2016-08-04    7.741497e+09    30259359.0    6488600.0    22068637.0    17499000.0
2016-08-05    7.726343e+09    25270756.0    2865863.0    40423859.0    14252363.0
```

- 获取沪深两个市场一段时间内的融资余额

```
[In]
logger.info(get_securities_margin(['XSHE', 'XSHG'], count=5, fields='margin_balance
[Out]
        XSHE        XSHG
2016-08-01    3.837627e+11    4.763557e+11
2016-08-02    3.828923e+11    4.763931e+11
2016-08-03    3.823545e+11    4.769321e+11
2016-08-04    3.833260e+11    4.776380e+11
2016-08-05    3.812751e+11    4.766928e+11
```

```
[In]
logger.info(get_securities_margin(['XSHG', '601988.XSHG', '510050.XSHG'], count=5))
[Out]
<class 'pandas.core.panel.Panel'>
Dimensions: 8 (items) x 5 (major_axis) x 3 (minor_axis)
Items axis: margin_balance to total_balance
Major_axis axis: 2016-08-01 00:00:00 to 2016-08-05 00:00:00
Minor_axis axis: XSHG to 510050.XSHG
```

- 获取50ETF融资偿还额情况

```
[In]
logger.info(get_securities_margin('510050.XSHG', count=5, fields='margin_repayment'
[Out]
2016-08-01     41652042.0
2016-08-02     19532586.0
2016-08-03    111043009.0
2016-08-04     22068637.0
2016-08-05     40423859.0
Name: margin_repayment, dtype: float64
```

get_shares - 流通股信息

```
get_shares(id_or_symbols, count=1, fields=None)
```

获取某只股票在一段时间内的流通情况。

参数

参数	类型	说明
id_or_symbols	*str*	可输入order_book_id, order_book_id list
count	*int*	回溯获取的数据个数。 默认为当前能够获取到的最近的数据
fields	*str* oR *str list*	默认为所有字段。见下方列表

fields	字段名
total	总股本
circulation_a	流通A股
management_circulation	已流通高管持股
non_circulation_a	非流通A股合计

返回

- 多个order_book_id，多个fields的时候返回*pandas Panel*
- 单个order_book_id，多个fields的时候返回*pandas DataFrame*
- 单个order_book_id，单个field返回*pandas Series*

范例

- 获取平安银行总股本数据

```
[In]
logger.info(get_shares('000001.XSHE', count=5, fields='total'))
[Out]
2016-08-01    1.717041e+10
2016-08-02    1.717041e+10
2016-08-03    1.717041e+10
2016-08-04    1.717041e+10
2016-08-05    1.717041e+10
Name: total, dtype: float64
```

get_turnover_rate - 历史换手率

```
get_turnover_rate(id_or_symbols, count=1, fields=None)
```

参数

参数	类型	说明
id_or_symbols	*str* or *str list*	可输入order_book_id, order_book_id list
count	*int*	回溯获取的数据个数。 默认为当前能够获取到的最近的数据
fields	*str* OR *str list*	默认为所有字段。见下方列表

fields	字段名
today	当天换手率
week	过去一周平均换手率
month	过去一个月平均换手率
three_month	过去三个月平均换手率
six_month	过去六个月平均换手率

current_year	当年平均换手率
total	上市以来平均换手率

返回

- 如果只传入一个order_book_id，多个fields，返回*pandas DataFrame*
- 如果传入order_book_id list，并指定单个field，函数会返回一个*pandas DataFrame*
- 如果传入order_book_id list，并指定多个fields，函数会返回一个*pandas Panel*

范例

- 获取平安银行历史换手率情况

```
[In]
logger.info(get_turnover_rate('000001.XSHE', count=5))
[Out]
            today    week   month  three_month  six_month    year  \
2016-08-01  0.5190  0.4478  0.3213       0.2877     0.3442  0.5027
2016-08-02  0.3070  0.4134  0.3112       0.2843     0.3427  0.5019
2016-08-03  0.2902  0.3460  0.3102       0.2823     0.3432  0.4982
2016-08-04  0.9189  0.4938  0.3331       0.2914     0.3482  0.4992
2016-08-05  0.4962  0.5031  0.3426       0.2960     0.3504  0.4994

            current_year   total
2016-08-01        0.3585  1.1341
2016-08-02        0.3570  1.1341
2016-08-03        0.3565  1.1339
2016-08-04        0.3604  1.1339
2016-08-05        0.3613  1.1338
```

industry - 行业股票列表

```
industry(code)
```

获得属于某一行业的所有股票列表。

参数

参数	类型	注释
code	*str* OR *industry_code item*	行业名称或行业代码。例如，农业可填写industry_code.A01 或 'A01'

返回

获得属于某一行业的所有股票的order_book_id list。

```
def init(context):
    stock_list = industry('A01')
    logger.info("农业股票列表：" + str(stock_list))
```

得到的结果是：

```
INITINFO 农业股票列表：['600354.XSHG', '601118.XSHG', '002772.XSHE', '600371.XSHG',
```

我们目前使用的行业分类来自于中国国家统计局的国民经济行业分类，可以使用这里的任何一个行业代码来调用行业的股票列表：

行业代码	行业名称
A01	农业
A02	林业
A03	畜牧业
A04	渔业
A05	农、林、牧、渔服务业
B06	煤炭开采和洗选业
B07	石油和天然气开采业
B08	黑色金属矿采选业
B09	有色金属矿采选业
B10	非金属矿采选业
B11	开采辅助活动
B12	其他采矿业
C13	农副食品加工业
C14	食品制造业
C15	酒、饮料和精制茶制造业
C16	烟草制品业
C17	纺织业
C18	纺织服装、服饰业
C19	皮革、毛皮、羽毛及其制品和制鞋业
C20	木材加工及木、竹、藤、棕、草制品业
C21	家具制造业

C23	印刷和记录媒介复制业
C24	文教、工美、体育和娱乐用品制造业
C25	石油加工、炼焦及核燃料加工业
C26	化学原料及化学制品制造业
C27	医药制造业
C28	化学纤维制造业
C29	橡胶和塑料制品业
C30	非金属矿物制品业
C31	黑色金属冶炼及压延加工业
C32	有色金属冶炼和压延加工业
C33	金属制品业
C34	通用设备制造业
C35	专用设备制造业
C36	汽车制造业
C37	铁路、船舶、航空航天和其它运输设备制造业
C38	电气机械及器材制造业
C39	计算机、通信和其他电子设备制造业
C40	仪器仪表制造业
C41	其他制造业
C42	废弃资源综合利用业
C43	金属制品、机械和设备修理业
D44	电力、热力生产和供应业
D45	燃气生产和供应业
D46	水的生产和供应业
E47	房屋建筑业
E48	土木工程建筑业
E49	建筑安装业
E50	建筑装饰和其他建筑业
F51	批发业

G53	铁路运输业
G54	道路运输业
G55	水上运输业
G56	航空运输业
G57	管道运输业
G58	装卸搬运和运输代理业
G59	仓储业
G60	邮政业
H61	住宿业
H62	餐饮业
I63	电信、广播电视和卫星传输服务
I64	互联网和相关服务
I65	软件和信息技术服务业
J66	货币金融服务
J67	资本市场服务
J68	保险业
J69	其他金融业
K70	房地产业
L71	租赁业
L72	商务服务业
M73	研究和试验发展
M74	专业技术服务业
M75	科技推广和应用服务业
N76	水利管理业
N77	生态保护和环境治理业
N78	公共设施管理业
O79	居民服务业
O80	机动车、电子产品和日用产品修理业
O81	其他服务业

Q83	卫生
Q84	社会工作
R85	新闻和出版业
R86	广播、电视、电影和影视录音制作业
R87	文化艺术业
R88	体育
R89	娱乐业
S90	综合

sector - 板块股票列表

```
sector(code)
```

获得属于某一板块的所有股票列表。

参数

参数	类型	注释
code	*str* OR *sector_code items*	板块名称或板块代码。例如，能源板块可填写'Energy'、'能源'或sector_code.Energy

返回

属于该板块的股票order_book_id或order_book_id list.

范例

```
def init(context):
    ids1 = sector("consumer discretionary")
    ids2 = sector("非必需消费品")
    ids3 = sector("ConsumerDiscretionary")
    assert ids1 == ids2 and ids1 == ids3
    logger.info(ids1)
```

得到的结果是：

```
INIT INFO
['002045.XSHE', '603099.XSHG', '002486.XSHE', '002536.XSHE', '300100.XSHE', '600633
```

Energy	能源	energy
Materials	原材料	materials
ConsumerDiscretionary	非必需消费品	consumer discretionary
ConsumerStaples	必需消费品	consumer staples
HealthCare	医疗保健	health care
Financials	金融	financials
InformationTechnology	信息技术	information technology
TelecommunicationServices	电信服务	telecommunication services
Utilities	公共服务	utilities
Industrials	工业	industrials

concept - 概念股票列表

```
concept(concept_name1, concept_name2, ...)
```

获取属于某个或某几个概念的股票列表。

参数

参数	类型	说明
concept_names	*str* or 多个*str*	概念名称。 可以从概念列表中选择一个或多个概念填写

返回

属于该概念的股票order_book_id或order_book_id list.

范例

- 得到一个概念的股票列表：

```
[In]concept('民营医院')
[Out]
['600105.XSHG',
'002550.XSHE',
'002004.XSHE',
'002424.XSHE',
...]
```

- 得到某几个概念的股票列表：

```
'600748.XSHG',
'600630.XSHG',
...]
```

概念列表

3D打印 3D玻璃 4G概念 5G概念 360私有化 AB股 AH股 BDI指数 IPV6 ITO导电玻璃
LED MLCC MSCI概念 O2O概念 OLED P2P概念 PET薄膜 PM2.5 PPP模式 QFII重仓
ST概念 S股 VA VB1 VC VD3 VE 一带一路 万达私有化 三七概念
三沙概念 三网融合 三聚氰胺 上海国资改革 上海房价上涨 上海自贸区 不锈钢概念 丙烯类
丝绸之路 两桶油改革 中厚板 中央政务区 中字头 中山房价上涨 中成药 中药饮片 中超概念
临沂房价上涨 丹参概念 举牌概念 乙二醇概念 乳业 二胎概念 云计算 互联网+ 互联网金融
人免疫球蛋白概念 人凝血因子概念 人参概念 人工智能 人工牛黄概念 人民币贬值概念 人脑工
低碳经济 体检与健康管理 体育产业 何首乌概念 佛山房价上涨 供应链金融 供热 保健酒 债
充电桩 光伏概念 光伏电池 光学膜 光学镜头 免疫治疗 党参概念 兜底增持概念 全息技术
六氟磷酸锂 兰州房价上涨 共享单车 养老概念 养老金持股 军工 军民融合 农业现代化 农机
冬虫夏草概念 冷轧 冷链物流 切片 创投 制冷剂 券商 前海概念 动力煤 化学制剂
化学原料药 北京冬奥会 北京房价上涨 北斗导航 区块链 医用医疗器械 医用耗材 医疗器械
医药工业配套服务 医药电商 医药设备和实验室工程 医院 单抗概念 单晶硅 南京房价上涨 南
印刷电路板（PCB） 厦门房价上涨 参股券商 参股新股 可燃冰 合同能源 合肥房价上涨 吉林
呼和浩特房价上涨 咖啡因概念 哈尔滨房价上涨 唐山房价上涨 徐州房价上涨 微信小程序 快递
啤酒 固废处理 国产软件 国企改革 土地流转 在线教育 地下管网 地热能 地黄概念 型材
型材（钢材） 基因工程药物 基因测序 塑料钞票 增强现实 壳资源 复合肥 大数据 大理房价
大连房价上涨 大金融 大飞机 天津房价上涨 家用电器 宽带中国 尼龙66切片 尼龙薄膜 尾气
天津自贸区 天然气 天然气供应 太阳能 头孢 宁波房价上涨 安防 宜兴房价上涨 实体药店
川贝概念 工业4.0 工业导爆索 工业雷管 己二酸概念 布洛芬概念 常州房价上涨 广州房价上
彩票概念 成都房价上涨 房地产开发 房屋租赁 手游概念 振兴东北 摘帽 新零售 无人机 无
数字电视 文化传媒 新三板 新型城镇化 新材料 新疆基建 新疆振兴 新股与次新股 新能源
无人驾驶 无汞电解二氧化锰 无烟煤 棒材（钢材） 武汉房价上涨 民营医院 民营银行 氟化工
无线充电 无锡房价上涨 昆山房价上涨 普通电解二氧化锰 景点旅游 智慧停车 智慧城市 智能
智能电网 智能电视 智能穿戴 智能音箱 有机硅类 期货概念 机器人概念 杀菌剂 杀虫剂 材
杭州房价上涨 杭州湾大湾区 板材 板材（钢材） 板蓝根概念 染料类 柴油 柴胡概念 核电
氨纶 水利建设 水泥 水电 汕头房价上涨 江苏国资改革 污水处理 汽油 汽车电子概念 沈阳
沥青类 沪港通 油品升级 油改概念 油气设备服务 泉州房价上涨 济南房价上涨 海参 海口房
海工装备 海洋经济 海绵城市 涤纶类 液晶面板 液氨 液碱 深圳国资改革 深圳房价上涨 涪
湖州房价上涨 滨海新区 火电 炭黑概念 炸药 烟台房价上涨 烧碱 热轧 焦炭概念 焦煤
煤化工 煤改气 燃料乙醇 燃料电池 牛黄概念 物流电商平台 物联网 特斯拉 特色小镇 特钢
特高压 独家药品 猪 王者荣耀概念 环戊烷 环氧丙烷 玻璃基板 玻璃概念 玻璃纤维 玻纤类
珠海房价上涨 生态农业 生物医药 生物疫苗 生物质能 甲醇概念 电力改革 电商概念 电子信
电子竞技 疫苗 病毒防治 白炭黑 白酒 白银 白马股 盖板玻璃 石墨烯 石油
磁性材料 磷矿石 磷酸 磷酸盐 磷酸铁 磷铵 票交所 福州房价上涨 福建自贸区 禽流感
移动互联网 移动支付 稀土 稀土永磁 稀缺资源 管材 管材（钢材） 粘胶短纤 粘胶长丝 粤
精对苯二甲酸（PTA） 糖 红参概念 红花概念 纯碱概念 线材（钢材） 绍兴房价上涨 维生素
网约车 网络安全 网络游戏 美丽中国 耐火材料 职业教育 联碱 联通混改 聚丙烯 聚氨酯
聚氨酯胶 聚氯乙烯 聚氯乙烯树脂 聚氯乙烯糊树脂 聚酯切片 聚酯薄膜 聚醚类 肉制品 股权
胶印油墨 能源互联网 腾讯概念 自来水 自由贸易港 航母概念 航空煤油 节能照明 节能环保
芯片概念 苏州房价上涨 苹果概念 茯苓概念 草甘膦 药品分销 葡萄酒 蓝宝石 虚拟现实 蝮
蚌埠房价上涨 蛋氨酸 融资融券 螺纹钢 血塞（栓）通 血液制品 装配式建筑 西安房价上涨
覆铜板 触摸屏概念 证金持股 诊断试剂 语音技术 调味品 贵阳房价上涨 赛马概念 超导概念
超级电容 超细纤维类 足球概念 跨境电商 车联网 转融券标的 轮胎 连翘概念 迪士尼 送转
通用航空 郑州房价上涨 醋酸丁酯 重庆房价上涨 量子通信 金华房价上涨 金融IC 金融改革
钒电池 钛白粉 钢坯 钨 钴 钼 铁精粉 铁路基建 铅 铜
铝 铝电解电容器 银川房价上涨 银杏叶概念 锂 锂电池概念 锂离子电解液 锂锰电解二氧化锰
锦纶丝类 锦纶切片 镇江房价上涨 镍 长春房价上涨 长沙房价上涨 阻燃树脂 阿胶概念 阿里
雄安新区 集成电路 青岛房价上涨 青霉素类概念 靶材 页岩气 顺酐类 风电 风能 食品安全

index_components - 指数成分股

```
index_components(order_book_id, date=None)
```

获取某一指数的股票构成列表，也支持指数的历史构成查询。

参数	类型	说明
order_book_id	*str*	指数代码，可传入order_book_id
date	*str*, *date*, *datetime*, *pandas Timestamp*	查询日期，默认为策略当前日期。 如指定， 则应保证该日期不晚于策略当前日期

返回

构成该指数股票的order_book_id list

范例

- 得到上证指数在策略当前日期的构成股票的列表：

```
[In]index_components('000001.XSHG')
[Out]['600000.XSHG', '600004.XSHG', ...]
```

get_dividend - 分红数据

```
get_dividend(order_book_id,  start_date)
```

获取某只股票到策略当前日期前一天的分红情况（包含起止日期）。

参数

参数	类型	说明
order_book_id	*str*	可输入order_book_id或symbol
start_date	*str*, *date*, *datetime*, *pandasTimestamp*	开始日期，用户必须指定， 需要早于策略当前日期

返回

pandas DataFrame - 查询时间段内某个股票的分红数据

- declaration_announcement_date: 分红宣布日，上市公司一般会提前一段时间公布未来的分红派息事件

- payable_date: 分红到帐日，这一天最终分红的现金会到账
- round_lot: 分红最小单位，例如：10代表每10股派发dividend_cash_before_tax单位的税前现金

范例

- 获取平安银行2013-01-04 到策略当前日期前一天的分红数据：

```
[In]
get_dividend('000001.XSHE', start_date='20130104')

[Out]
                            book_closure_date  dividend_cash_before_tax  \
declaration_announcement_date
2013-06-14                         2013-06-19                    0.9838

                            ex_dividend_date payable_date  round_lot
declaration_announcement_date
2013-06-14                        2013-06-20   2013-06-20       10.0
```

get_split - 拆分数据

```
get_split(order_book_id,  start_date)
```

获取某只股票到策略当前日期前一天的拆分情况（包含起止日期）。

参数

参数	类型	说明
order_book_id	*str*	证券代码，证券的独特的标识符，例如：'000001.XSHE'
start_date	*str*, *date*, *datetime*, *pandasTimestamp*	开始日期，用户必须指定，需要早于策略当前日期

返回

pandas DataFrame - 查询时间段内的某个股票的拆分数据

- ex_dividend_date: 除权除息日，该天股票的价格会因为拆分而进行调整
- book_closure_date: 股权登记日
- split_coefficient_from: 拆分因子（拆分前）
- split_coefficient_to: 拆分因子（拆分后）

例如：每10股转增2股，则split_coefficient_from = 10, split_coefficient_to = 12.

范例

```
[Out]
                book_closure_date payable_date  split_coefficient_from  \
ex_dividend_date
2013-06-20              2013-06-19   2013-06-20                      10

                 split_coefficient_to
ex_dividend_date
2013-06-20                       16.0
```

get_trading_dates - 交易日列表

```
get_trading_dates(start_date, end_date)
```

获取某个国家市场的交易日列表（起止日期加入判断）。目前仅支持中国市场。

参数

参数	类型	说明
start_date	*str*, *date*, *datetime*, *pandasTimestamp*	开始日期，用户必须指定
end_date	*str*, *date*, *datetime*, *pandasTimestamp*	结束日期，用户必须指定

返回

datetime.date list

范例

```
[In]get_trading_dates(start_date='2016-05-05', end_date='20160505')
[Out]
[datetime.date(2016, 5, 5)]
```

get_previous_trading_date - 上一交易日

```
get_previous_trading_date(date, n=1)
```

获取指定日期的上一（或n）交易日。

参数

参数	类型	说明
date	*str*, *date*, *datetime*, *pandasTimestamp*	指定日期
n	*int*	第 n 个交易日，默认为1

范例

```
[In]get_previous_trading_date(date='2016-05-02')
[Out]
[datetime.date(2016, 4, 29)]
```

get_next_trading_date - 下一交易日

```
get_next_trading_date(date, n=1)
```

获取指定日期的下一交易日

参数

参数	类型	说明
date	*str*, *date*, *datetime*, *pandasTimestamp*	指定日期
n	*int*	第 n 个交易日，默认为1

返回

datetime.date

范例

```
[In]get_next_trading_date(date='2016-05-01')
[Out]
[datetime.date(2016, 5, 3)]
```

get_price_change_rate - 获取历史涨跌幅

```
get_price_change_rate(id_or_symbols, count=1)
```

获取股票或指数的历史涨跌幅。**注意**目前只支持股票与指数两类合约，基金、期货等目前并不支持。

参数

参数	类型	说明
id_or_symbols	*str* or *str list*	可输入order_book_id, order_book_id list

count	*int*	默认为当前能够获取到的最近的数据

返回

- 传入多个order_book_id，函数会返回*pandas DataFrame*
- 传入一个order_book_id，函数会返回*pandas Series*

范例

- 获取平安银行以及沪深300指数一段时间的涨跌幅情况。

```
[In]
get_price_change_rate(['000001.XSHE', '510050.XSHG'], 1)
[Out]
2016-06-01 15:30:00.00  INFO   order_book_id  000001.XSHE  510050.XSHG
                               date
                               2016-05-31        0.026265     0.033964
2016-06-02 15:30:00.00  INFO   order_book_id  000001.XSHE  510050.XSHG
                               date
                               2016-06-01       -0.006635    -0.008308
```

get_yield_curve - 收益率曲线

```
get_yield_curve(date=None, tenor=None)
```

获取某个国家市场指定日期的收益率曲线水平。

数据为2002年至今的**中债国债收益率曲线**，来源于中央国债登记结算有限责任公司。

参数

参数	类型	说明
date	*str*, *date*, *datetime*, *pandasTimestamp*	查询日期，默认为策略当前日期前一天
tenor	*str*	标准期限，'0S' - 隔夜，'1M' - 1个月，'1Y' - 1年，默认为全部期限

返回

pandas DataFrame - 查询时间段内无风险收益率曲线

范例

```
[In]
get_yield_curve('20130104')

[Out]
                0S      1M      2M      3M      6M      9M      1Y      2Y  \
```

```
2013-01-04  0.0314  0.0318  ...  0.0342  0.0350  0.0353  0.0357  0.0361
...
```

is_suspended - 全天停牌判断

```
is_suspended(order_book_id, count=1)
```

判断某只股票是否全天停牌。

参数

参数	类型	说明
order_book_id	*str*	某只股票的代码，可传入单只股票的order_book_id, symbol
count	*int*	回溯获取的数据个数。 默认为当前能够获取到的最近的数据

返回

count为1时 *bool*

count>1时 *list*

is_st_stock - ST股判断

```
is_st_stock(order_book_id, count=1)
```

判断一只或多只股票在一段时间内是否为ST股（包括ST与*ST）。

ST股是有退市风险因此风险比较大的股票，很多时候您也会希望判断自己使用的股票是否是'ST'股来避开这些风险大的股票。另外，我们目前的策略比赛也禁止了使用'ST'股。

参数

参数	类型	注释
order_book_id	*str*	股票的代码，可传入order_book_id, symbol
count	*int*	回溯获取的数据个数。 默认为当前能够获取到的最近的数据

返回

count为1时 *bool*

count>1时 *list*

```
fenji.get_a_by_yield(current_yield, listing=True)
```

通过传入当前的本期利率拿到对应的分级A的order_book_id list

参数

参数	类型	注释
current_yield	*float*	本期利率，用户必须指定
listing	*bool*	默认为True，该分级基金是否在交易所可交易

返回

符合当前利率水平的分级A基金的order_book_id list；如果无符合内容，则返回空列表。

范例

- 拿到当前收益率为4的A基的代码列表：

```
[In] fenji.get_a_by_yield(4)
[Out]
['150039.XSHE']
```

```
fenji.get_a_by_interest_rule(interest_rule)
```

通过传入当前的利率规则拿到对应的分级A的order_book_id list

参数

参数	类型	注释
interest_rule	*str*	利率规则，例如："+3.5%", "+4%", "=7%", "*1.4+0.55%", " etc. 您也可以在研究平台使用fenji.get_all来进行查询所有的组合 用户必须填写
listing	*bool*	该分级基金是否在交易所可交易，默认为True

返回

符合当前利率规则的分级A基金的order_book_id list

范例

- 拿到符合利率规则"+3%"的A基的代码列表：

```
[ 502011.XSHG ,  150219.XSHE ,  150181.XSHE ,  150209.XSHE ,  150179.XSHE ,  150217
```

```
fenji.get_all(field_list)
```

获取所有分级基金信息

参数

参数	类型	注释
field_list	*str list*	希望输出的数据字段名（见下表），默认为所有字段

返回

pandas DataFrame - 分级基金各项数据

字段名	注释
a_b_propotion	分级A：分级B的比例
conversion_date	下次定折日
creation_date	创立日期
current_yield	本期利率
expire_date	到期日，可能为NaN - 即不存在
fenji_a_order_book_id	A基代码
fenji_a_symbol	A基名称
fenji_b_order_book_id	B基代码
fenji_b_symbol	B基名称
fenji_mu_orderbook_id	母基代码
fenji_mu_symbol	母基名称
interest_rule	利率规则
next_yield	下期利率
track_index_symbol	跟踪指数

范例

- 拿到所有的分级基金的信息：

```
[In] fenji.get_all()
[Out]
```

2 1.1 2018-12-13 2015-07-01 5.5 NaN 150335.XSHE 半上成A

- 拿到只有2个字段的所有分级基金的信息：

```
[In] fenji.get_all(field_list = ['fenji_a_order_book_id', 'current_yield'])
[Out]
current_yield    fenji_a_order_book_id
0    2.5    161828
1    5    150213.XSHE
2    5.5    150335.XSHE
```

雪球舆论数据

```
xueqiu.top_stocks(field, date=None, frequency='1d', count=10)
```

获取每日、每周或每月的某个指标的雪球数据的股票排名情况以及它的对应的统计数值.

参数

参数	类型	说明
field	*str*	目前支持的雪球数据统计指标有: 昨日新增评论 - `new_comments` ，总评论 - `total_comments` ，昨日新增关注者 - `new_followers` ，总关注者数目 - `total_followers` ，卖出行为 - `sell_actions` ，买入行为 - `buy_actions`
date	*str*, *datetime.date*, *datetime.datetime*, *pandasTimestamp*	查询日期。默认为策略当前日期前一天。 如指定，则该日期应早于策略当前日期。 注意： 我们最早支持的雪球数据只到2015年4月23日，之后的数据我们都会保持更新
frequency	*str*	默认是 `1d` ，即每日的数据统计。也支持每周 - `1w` 和每月 - `1M` 的统计
count	*integer*	指定返回多少个结果，默认是10个

返回

pandas DataFrame - 各项舆情数据

范例

- 获取前一天的新增留言最多的10支股票：

```
logger.info(u)
```

其他方法

update_universe - 更新股票池

```
update_universe(order_book_id)
```

该方法用于更新现在关注的证券的集合（e.g.：股票池）。PS：会在下一个bar事件触发时候产生（新的关注的股票池更新）效果。并且update_universe会是覆盖（overwrite）的操作而不是在已有的股票池的基础上进行增量添加。比如已有的股票池为 `['000001.XSHE', '000024.XSHE']` 然后调用了 `update_universe(['000030.XSHE'])` 之后，股票池就会变成 `000030.XSHE` 一个股票了，随后的数据更新也只会跟踪 `000030.XSHE` 这一个股票了。

参数

参数	类型	注释
order_book_id	*str* OR *str list*	合约代码，可传入order_book_id, order_book_id list

范例

下面的代码是将股票池变更为只有2个股票 `000001.XSHE` 和 `000024.XSHE`：

```
update_universe(['000001.XSHE', '000024.XSHE'])
```

当然，您也可以使用合约简称：

```
update_universe(['平安银行', '招商地产'])
```

subscribe - 订阅行情

```
subscribe(order_book_id)
```

订阅合约行情。该操作会导致合约池内合约的增加，从而影响 `handle_bar` 中处理bar数据的数量。

需要注意，用户在初次编写策略时候需要首先订阅合约行情，否则 `handle_bar` 不会被触发。

参数

返回

无

unsubscribe - 取消订阅行情

```
unsubscribe(order_book_id)
```

取消订阅合约行情。取消订阅会导致合约池内合约的减少，如果当前合约池中没有任何合约，则策略直接退出。

参数

参数	类型	说明
order_book_id	*str*, *str list*	合约代码，或代码列表

返回

无

reg_indicator - 注册指标

```
reg_indicator(name, func_obj, freq='1d', win_size=10)
```

参数

参数	类型	说明
name	*str*	定义的指标名称
func_obj	*function*	函数对象
freq	*str*	指标计算的周期。 支持日级别与分钟级别，'1d'代表每日，'5m'代表5分钟
win_size	*int*	获取数据回溯窗口。 该指标用于在注册指标时让系统获取回溯获取数据的最大窗口，便于数据的加载与预计算

返回

无

get_indicator - 获取指标

参数

参数	类型	说明
order_book_id	*str*	合约代码
name	*str*	定义的指标名称

name为您注册的指标中的name参数，使用案例请见自定义技术指标

返回

定义指标返回值

get_file - 读取文件

```
get_file(file_path)
```

读取您的私有文件（您的私有文件可以在研究模块中看到）

参数	类型	注释
file_path	*str*	相对路径， 相对于您的私有的在研究模块空间的根目录的路径

返回

返回文件的原始内容，不做任何decode。

范例

以下代码可以在回测中使用，读取自己的私有文件：day_px.csv 之后，然后通过pandas转换成dataframe类型进行方便使用：

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。
import pandas as pd
from six import BytesIO

# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    # 由于保存的是原数据文件，因此需要用BytesIO进行转换
    body = get_file('day_px.csv')
    data=pd.read_csv(BytesIO(body))
    logger.info(data)
```

put_file - 存储文件

```
put_file(file_path, data, append=False)
```

参数

参数	类型	注释
file_path	*str*	相对路径， 相对于您的私有的在研究模块空间的根目录的路径
data	*str*, *bytes*	保存的信息。如果为str类型，则会默认以utf-8方式编码存储
append	*bool*	是否续写文件。默认为False， 即每次调用时都会清除原文件内容

返回

无

范例

运行结束之后，您就能够在研究平台中看到test_file.csv文件了。

```
put_file('test_file.csv', 'hello world')
```

get_csv - 读取csv数据

```
get_csv(csv_file_path)
```

参数

get_csv函数支持 pandas.read_csv 的全部参数。

参数	类型	注释
'csv_file_path'	str - required	ipython策略研究部分上传的csv文件的路径。

返回

`pandas Dataframe` - 里面保存着csv文件中的数据内容

范例

如果仅仅是想使用csv文件格式的话可以使用get_csv接口，我们在ipython策略研究部分提供了上传自己数据的功能，您从这里进入ipython策略研究：

图片描述

接着可以点击右上角的上传文件：

图片描述

我们看下这个csv文件中的数据内容：

```
2015-01-10,2937842929.6,113463565.69,1733702964.32
2014-01-10,2926316060.58,116181575.59,883935497.71
2013-01-10,2616532214.37,90146425.57,948898049.5
2012-01-10,2681016310.35,,620593405.65
2011-01-10,2034147254.71,,499812019.44
2010-01-10,,,508985888.73
```

随后我们可以在编写策略的时候使用 `get_csv` 来调用：

```
def init(context):
    context.csv_df = get_csv("revenue.csv")
    logger.info(context.csv_df)
```

我们把renuve.csv中的数据读取出来以后是一个 `dataframe` 然后存放到了 `context.csv_df` 里面以供之后的策略使用，可以运行以后看下打印出来的结果：

```
INITINFO    2015-01-10  2937842929.6  113463565.69  1733702964.32
         0  2014-01-10  2.926316e+09  1.161816e+08   8.839355e+08
         1  2013-01-10  2.616532e+09  9.014643e+07   9.488980e+08
         2  2012-01-10  2.681016e+09           NaN   6.205934e+08
         3  2011-01-10  2.034147e+09           NaN   4.998120e+08
         4  2010-01-10           NaN           NaN   5.089859e+08
```

plot - 画图

```
plot(series_name, value)
```

`plot` 函数可以将时间序列的数据传给页面进行绘图，结果是以时间为横轴，value为纵轴的曲线。

参数

参数	类型	注释
series_name	*str*	绘制曲线的名称，用户必须填写
value	*float*	当前日期的曲线的点的值，用户必须填写

范例

```
 # 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更新
def handle_bar(context, bar_dict):
    # TODO: 开始编写你的算法吧！
```

```
plot( 'open' , bar_dict[ '000001.XSHE' ].open)
```

以上代码画图的结果截图：

图片描述

重要对象

Bar对象

属性	类型	注释
order_book_id	*str*	合约代码
symbol	*str*	合约简称
datetime	*datetime.datetime*	时间戳
open	*float*	开盘价
close	*float*	收盘价
high	*float*	最高价
low	*float*	最低价
volume	*float*	成交量
total_turnover	*float*	成交额
prev_close	*float*	昨日收盘价
limit_up	*float*	涨停价
limit_down	*float*	跌停价
isnan	*bool*	当前bar数据是否有行情。例如， 获取已经到期的合约数据，isnan此时为True
suspended	*bool*	是否全天停牌
prev_settlement	*float*	昨结算（期货日线数据专用）
settlement	*float*	结算（期货日线数据专用）

注意,在股票策略中bar对象可以拿到所有股票合约的bar信息。如下

```
def handle_bar(context,bar_dict):
    volume = bar_dict['000001.XSHE'].volume
    #拿到'000001.XSHE'当前bar的成交量
```

Snapshot对象

属性	类型	注释
order_book_id	*str*	股票代码
datetime	*datetime.datetime*	当前快照数据的时间戳
open	*float*	当日开盘价
last	*float*	当前最新价
high	*float*	截止到当前的最高价
low	*float*	截止到当前的最低价
prev_close	*float*	昨日收盘价
volume	*float*	截止到当前的成交量
total_turnover	*float*	截止到当前的成交额
open_interest	*float*	截止到当前的持仓量（期货专用）
prev_settlement	*float*	昨日结算价（期货专用）

需要注意，在回测中通过snapshot获取prev_close以及prev_settlement较为低效，推荐通过bar数据进行获取。

Order对象

属性	类型	注释
order_id	*int*	唯一标识订单的id
order_book_id	*str*	合约代码
datetime	*datetime.datetime*	订单创建时间
side	*SIDE*	订单方向
price	*float*	订单价格， 只有在订单类型为'限价单'的时候才有意义
quantity	*int*	订单数量
filled_quantity	*int*	订单已成交数量
unfilled_quantity	*int*	订单未成交数量
type	*ORDER_TYPE*	订单类型
transaction_cost	*float*	费用

status	*ORDER_STATUS*	订单状态
message	*str*	信息。 比如拒单时候此处会提示拒单原因
trading_datetime	*datetime.datetime*	订单的交易日期（对应期货夜盘）
position_effect	*POSITION_EFFECT*	订单开平（期货专用）

Clone

Portfolio对象

- portfolio对象

属性	类型	注释
cash	*float*	可用资金，为子账户可用资金的加总
frozen_cash	*float*	冻结资金，为子账户冻结资金加总
total_returns	*float*	投资组合至今的累积收益率
daily_returns	*float*	投资组合每日收益率
daily_pnl	*float*	当日盈亏，子账户当日盈亏的加总
market_value	*float*	投资组合当前的市场价值， 为子账户市场价值的加总
total_value	*float*	总权益，为子账户总权益加总
units	*float*	份额。在没有出入金的情况下， 策略的初始资金
unit_net_value	*float*	单位净值
static_unit_net_value	*float*	静态单位权益
transaction_cost	*float*	当日费用
pnl	*float*	当前投资组合的累计盈亏
start_date	*datetime.datetime*	策略投资组合的回测/ 实时模拟交易的开始日期
annualized_returns	*float*	投资组合的年化收益率
positions	*dict*	一个包含所有仓位的字典， 以order_book_id作为键， `position` 对象作为值， 关于position的更多的信息可以在下面的部分找到。

属性	类型	注释
cash	*float*	可用资金
frozen_cash	*float*	冻结资金
market_value	*float*	投资组合当前所有证券仓位的市值的加总
total_value	*float*	总权益
transaction_cost	*float*	当日费用
positions	*dict*	一个包含股票子组合仓位的字典， 以order_book_id作为键， `position` 对象作为值， 关于position的更多的信息可以在下面的部分找到。
dividend_receivable	*float*	投资组合在分红现金收到账面之前的应收分红部分。 具体细节在分红部分

- 期货账户future_account对象

属性	类型	注释
cash	*float*	可用资金
frozen_cash	*float*	冻结资金
market_value	*float*	投资组合当前所有期货仓位的**名义市值**的加总
daily_pnl	*float*	当日盈亏，当日浮动盈亏 + 当日平仓盈亏 - 当日费用
holding_pnl	*float*	当日浮动盈亏
realized_pnl	*float*	当日平仓盈亏
total_value	*float*	总权益
transaction_cost	*float*	当日费用
positions	*dict*	一个包含期货子组合仓位的字典， 以order_book_id作为键， `position` 对象作为值
margin	*float*	已占用保证金
buy_margin	*float*	多头保证金
sell_margin	*float*	空头保证金

Position对象

- 股票position对象

quantity	*int*	当前持仓股数
pnl	*float*	持仓累计盈亏
sellable	*int*	该仓位可卖出股数。T + 1的市场中sellable = 所有持仓-今日买入的仓位
market_value	*float*	获得该持仓的实时市场价值
value_percent	*float*	获得该持仓的实时市场价值在总投资组合价值中所占比例，取值范围[0, 1]
avg_price	*float*	平均建仓成本

- 期货position对象

属性	类型	注释
order_book_id	*str*	合约代码
pnl	*float*	累计盈亏
daily_pnl	*float*	当日盈亏，当日浮动盈亏+当日平仓盈亏
holding_pnl	*float*	当日持仓盈亏
realized_pnl	*float*	当日平仓盈亏
transaction_cost	*float*	仓位交易费用
margin	*float*	仓位总保证金
market_value	*float*	当前仓位的名义价值。 如果当前净持仓为空方向持仓，则名义价值为负
buy_daily_pnl	*float*	多头仓位当日盈亏
buy_pnl	*float*	多头仓位累计盈亏
buy_transaction_cost	*float*	多头费用
closable_buy_quantity	*float*	可平多头持仓
buy_margin	*float*	多头持仓占用保证金
buy_today_quantity	*int*	多头今仓
buy_quantity	*int*	多头持仓
buy_avg_open_price	*float*	多头开仓均价
buy_avg_holding_price	*float*	多头持仓均价
sell_daily_pnl	*float*	空头仓位当日盈亏

sell_transaction_cost	*float*	空头费用
closable_sell_quantity	*int*	可平空头持仓
sell_margin	*float*	空头持仓占用保证金
sell_today_quantity	*int*	空头今仓
sell_quantity	*int*	空头持仓
sell_avg_open_price	*float*	空头开仓均价
sell_avg_holding_price	*float*	空头持仓均价

Instrument对象

- **股票，ETF，指数Instrument对象**

参数	类型	说明
order_book_id	*str*	证券代码，证券的独特的标识符。 应以'.XSHG'或'.XSHE'结尾，前者代表上证， 后者代表深证
symbol	*str*	证券的简称，例如'平安银行'
abbrev_symbol	*str*	证券的名称缩写，在中国A股就是股票的拼音缩写。 例如：'PAYH'就是平安银行股票的证券名缩写
round_lot	*int*	一手对应多少股，中国A股一手是100股
sector_code	*str*	板块缩写代码，全球通用标准定义
sector_code_name	*str*	以当地语言为标准的板块代码名
industry_code	*str*	国民经济行业分类代码， 具体可参考下方"Industry列表"
industry_name	*str*	国民经济行业分类名称
listed_date	*str*	该证券上市日期
de_listed_date	*str*	退市日期
type	*str*	合约类型，目前支持的类型有: 'CS', 'INDX', 'LOF', 'ETF', 'FenjiMu', 'FenjiA', 'FenjiB', 'Future'
concept_names	*str*	概念股分类，例如：'铁路基建'，'基金重仓'等
exchange	*str*	交易所，'XSHE' - 深交所, 'XSHG' - 上交所
board_type	*str*	板块类别，'MainBoard' - 主板,'GEM' - 创业板,'SME' - 中小板

status	*str*	'TemporarySuspended' - 暂停上市, 'PreIPO' - 发行配售期间, 'FailIPO' - 发行失败
special_type	*str*	特别处理状态。'Normal' - 正常上市, 'ST' - ST处理, 'StarST' - *ST代表该股票正在接受退市警告, 'PT' - 代表该股票连续3年收入为负，将被暂停交易, 'Other' - 其他

- **期货Instrument对象**

参数	类型	说明
order_book_id	*str*	期货代码，期货的独特的标识符（郑商所期货合约数字部分进行了补齐。例如原有代码'ZC609'补齐之后变为'ZC1609'）。主力连续合约UnderlyingSymbol+88，例如'IF88'；指数连续合约命名规则为UnderlyingSymbol+99
symbol	*str*	期货的简称，例如'沪深1005'
margin_rate	*float*	期货合约最低保证金率
abbrev_symbol	*str*	期货的名称缩写，例如'HS1005'。主力连续合约与指数连续合约都为'null'
round_lot	*float*	期货全部为1.0
listed_date	*str*	期货的上市日期。主力连续合约与指数连续合约都为'0000-00-00'
type	*str*	合约类型，'Future'
contract_multiplier	*float*	合约乘数，例如沪深300股指期货的乘数为300.0
underlying_order_book_id	*str*	合约标的代码，目前除股指期货(IH, IF, IC)之外的期货合约，这一字段全部为'null'
underlying_symbol	*str*	合约标的名称，例如IF1005的合约标的名称为'IF'
maturity_date	*str*	期货到期日。主力连续合约与指数连续合约都为'0000-00-00'
settlement_method	*str*	交割方式，'CashSettlementRequired' - 现金交割, 'PhysicalSettlementRequired' - 实物交割
product	*str*	产品类型，'Index' - 股指期货, 'Commodity' - 商品期货, 'Government' - 国债期货

exchange	*str*	上海期货交易所，'CFFEX' - 中国金融期货交易所, 'CZCE'- 郑州商品交易所

Instrument对象也支持如下方法：

- 合约已上市天数。

```
instruments(order_book_id).days_from_listed()
```

如果合约首次上市交易，天数为0；如果合约尚未上市或已经退市，则天数值为-1

- 合约距离到期天数。

```
instruments(order_book_id).days_to_expire()
```

如果策略已经退市，则天数值为-1

- 获取合约最小价格变动单位。

```
tick_size()
```

例如，instruments('IF1608').tick_size()获取的就是股指期货的最小价格变动单位，为0.2，即“一跳”的水平。

枚举常量

ORDER_STATUS - 订单状态

枚举值	说明
PENDING_NEW	待报
ACTIVE	可撤
FILLED	全成
CANCELLED	已撤
REJECTED	拒单

SIDE - 买卖方向

枚举值	说明
BUY	买

POSITION_EFFECT - 开平

枚举值	说明
OPEN	开仓
CLOSE	平仓

ORDER_TYPE - 订单类型

枚举值	说明
MARKET	市价单
LIMIT	限价单

RUN_TYPE - 策略运行类型

枚举值	说明
BACKTEST	回测
PAPER_TRADING	实盘模拟

MATCHING_TYPE - 撮合方式

枚举值	说明
CURRENT_BAR_CLOSE	以当前bar收盘价撮合
NEXT_BAR_OPEN	以下一bar数据开盘价撮合

外部数据和Python模块

引入自己的Python模块

我们已提供了不少丰富的第三方Python库可以用，如果你还有引入自己的Python模块的需求的话可以通过IPython研究平台新建一个python文件，写好自己的python库，然后在回测以及实盘模拟交易这边import来实现：

步骤很简单，只需3步：

1. 在研究平台新建一个文本：

   图片描述

2. 点击名字，修改为xxx.py, 比如本次例子修改为frank.py (一定要以.py结尾哦！)

然后保存：

图片描述

4. 在策略中调用你的自定义库。首先"import xxx"，这里面就不需要打.py了，只需要库的名字就好，然后就这么完成了！

图片描述

支持的其他外部数据源

Tushare

Tushare是一个国内很流行的爬虫数据源，支持的数据种类挺丰富的，API设计也是围绕着pandas 来做的，因此也非常适合Ricequant的用户上手:

只需要简单的 `import tushare as ts` 就可以在ricequant的研究环境中使用了：

```
[In]
import tushare as ts
ts.get_hist_data('600848') #一次性获取全部日k线数据

[Out]
            open    high    close    low    volume    price_change    p_change    m
            v_ma20  turnover
date                                                                              Clone
2016-03-15    15.24    15.51    15.42    15.12    24615.96    0.18    1.18    15.14
2016-03-14    15.02    15.55    15.24    15.02    25581.68    0.50    3.39    15.20
2016-03-11    14.97    14.97    14.75    14.49    25404.35    -0.26    -1.73    15.
2016-03-10    15.25    15.41    15.01    15.01    23066.78    -0.26    -1.70    15.
```

重要的是目前使用Tushare可以支持一些ricequant暂时没有的数据，具体的使用方法和文档可以去tushare上面寻找，比如：

- 投资参考数据
    - 分配预案
    - 业绩预告
    - 限售股解禁
    - 基金持股
    - 新股数据
    - 融资融券（沪市）
    - 融资融券（深市）
- 宏观经济数据
    - 存款利率
    - 贷款利率
    - 存款准备金率
    - 货币供应量

- 三大需求对GDP贡献
- 三大产业对GDP拉动
- 三大产业贡献率
- 居民消费价格指数
- 工业品出厂价格指数

- 新闻事件数据
  - 即时新闻
  - 信息地雷
  - 新浪股吧
- 龙虎榜数据
  - 每日龙虎榜列表
  - 个股上榜统计
  - 营业部上榜统计
  - 机构席位追踪
  - 机构成交明细
- 银行间同业拆放利率
  - Shibor拆放利率
  - 银行报价数据
  - Shibor均值数据
  - 贷款基础利率（LPR）
  - LPR均值数据

目前支持的Python模块

我们现在支持如下表格所列的多种强大的Python模块，您**需要手动自己引入**，比如可以打入以下代码来支持引入pandas模块：

```
import pandas as pd
df = pd.DataFrame(xxxx)
```

您可以引入我们目前支持的Python模块来做各种神奇的数据处理。

如果您有自己擅长和特别喜欢的Python模块希望我们支持， 请让我们知道

下面的列表是现在Ricequant已经支持的Python模块：

模块名	简介	文档链接

talib	程序员常用的金融数据技术分析库。 包含了超过150+的技术指标比如ADX,MACD,RSI,Stochastic,Bollinger Bands等	TA-Lib官网
pandas	最流行的Python数据分析库	pandas文档
numpy	numpy是一个Python的科学计算基础库。	numpy文档
scipy	SciPy是一个Python的数学、 科学和工程计算的生态系统库。	scipy文档
statsmodels	Statsmodels是一个Python的模块可以让您研究数据， 构架统计模型和进行统计测试。功能包括： 线性回归模型（Linear regression models）等	statsmodels文
bisect	Python的排序模块	bisect文档
cmath	提供可以对复数计算的数学模块	cmath文档
collections	提供除了Python内嵌的容器之外的容器种类选择 - dict, list, set 和 tuple	collections文
sklearn	Python的机器学习模块（machine learning）	sklearn文档
hmmlearn	Python的隐马尔可夫模型（Hidden Markov Models）模块，类似scikit-learn的API	hmmlearn文
hsmmlearn	Python的无监督学习隐马尔可夫模型（Hidden Markov Models）模块，类似scikit-learn的API	hsmmlearn文
pykalman	超级简单的卡尔曼滤波（Kalman Filter），Kalman Smoother和EM模块	pykalman文档
cvxopt	cvxopt提供了凸优化（convex optimization） 的解的python库。	Clone
arch	arch提供了Univariate volatility模型， Bootstrapping和Multiple comparison procedures	arch文档
dateutil	dateutil模块提供了对标准的datetime模块的强大的拓展	dateutil文档
Edward	一个用于概率建模、推理和评估的 Python 库, 融合了以下三个领域：贝叶斯统计学和机器学习、 深度学习、概率编程	Edward文档
Funcat	将同花顺、通达信、文华财经等的公式移植到了 Python 中	Funcat文档
datetime		datetime文档
functools		functools文档
heapq		heapq文档

tensorflow	Tensor flow is an open source software library for machine intelligence.	tensorflow文档
tushare	国内流行的开源数据库，燥起来吧，各种数据。	tushare网站
pybrain	pybrain是一个流行的机器学习库。PyBrain is a modular Machine Learning Library for Python.	pybrain文档
nltk	一个流行的人类语言分析库。	nltk文档
keras	Theano和Tensorflow的深度学习库。	keras文档
requests	易用的HTTP库	requests文档
bs4	beautifulsoup是网页爬取数据的利器！	beautifulsoup
lxml	处理XML和HTML的最好用的python库	lxml中文文档
urllib	python自带的url处理库	urllib文档
xgboost	速度快效果好的boosting模型	xgboost文档
plotly	强大优美的图表库，支持三种不同类型的图表，包括地图，箱形图和密度图，以及更常见的产品如，条状和线形图	plotly文档
fbprophet	简单强大的数据预测工具包	Prophet使用指
pytorch	流行的神经网络工具包	Pytorch文档
sonnet	基于tensorflow快速构建神经网络的工具	sonnet文档
itertools		itertools文档
math		math文档
pytz		pytz文档
queue		queue文档
random		random文档
re		re文档
time		time文档
array		array文档
copy		copy文档
json		json文档
operator		operator文档
xml		xml文档

在下面部分我们列举了一些常用的算法范例，您也可以通过右上角的clone按钮很方便的把范例算法复制到自己的算法列表中进行调试、更改甚至模拟、真实交易。

第一个策略-买入&持有

万事开头难，这是一个最简单的策略：在回测开始的第一天买入资金量的100%的平安银行并且一直持有。可以通过右上角的clone按钮很方便的把范例算法复制到自己的算法列表中， 您可以更改一下股票代码以及回测时间、金钱等就可以很快的知道某个时期您测试某个股票的表现了，各种风险数值也可以很快地计算出来。

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。


# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    context.s1 = "000001.XSHE"
    # 是否已发送了order
    context.fired = False


# 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更新
def handle_bar(context, bar_dict):
    # 开始编写你的主要的算法逻辑

    # bar_dict[order_book_id] 可以拿到某个证券的bar信息
    # context.portfolio 可以拿到现在的投资组合状态信息

    # 使用order_shares(id_or_ins, amount)方法进行落单

    # TODO: 开始编写你的算法吧！
    if not context.fired:
        # order_percent并且传入1代表买入该股票并且使其占有投资组合的100%
        order_percent(context.s1, 1)
        context.fired = True
```

Golden Cross算法示例

以下是一个我们使用TALib在我们的平台上编写的golden cross算法的示例，使用了simple moving average方法：

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。
import talib
# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    context.s1 = "000001.XSHE"

    # 设置这个策略当中会用到的参数，在策略中可以随时调用，这个策略使用长短均线，我们在这里i
    context.SHORTPERIOD = 20
    context.LONGPERIOD = 120
```

```
def handle_bar(context, bar_dict):
    # 开始编写你的主要的算法逻辑

    # bar_dict[order_book_id] 可以拿到某个证券的bar信息
    # context.portfolio 可以拿到现在的投资组合状态信息

    # 使用order_shares(id_or_ins, amount)方法进行落单

    # TODO: 开始编写你的算法吧！

    # 因为策略需要用到均线，所以需要读取历史数据
    prices = history_bars(context.s1,context.LONGPERIOD+1, '1d', 'close')

    # 使用talib计算长短两根均线，均线以array的格式表达
    short_avg = talib.SMA(prices, context.SHORTPERIOD)
    long_avg = talib.SMA(prices, context.LONGPERIOD)

    plot("short avg", short_avg[-1])
    plot("long avg", long_avg[-1])

    # 计算现在portfolio中股票的仓位
    cur_position = context.portfolio.positions[context.s1].quantity
    # 计算现在portfolio中的现金可以购买多少股票
    shares = context.portfolio.cash/bar_dict[context.s1].close

    # 如果短均线从上往下跌破长均线，也就是在目前的bar短线平均值低于长线平均值，而上一个bar
    if short_avg[-1] - long_avg[-1] < 0 and short_avg[-2] - long_avg[-2] > 0 and cu
        # 进行清仓
        order_target_value(context.s1, 0)

    # 如果短均线从下往上突破长均线，为入场信号
    if short_avg[-1] - long_avg[-1] > 0 and short_avg[-2] - long_avg[-2] < 0:
        # 满仓入股
        order_shares(context.s1, shares)
```

单股票MACD算法示例

以下是一个我们使用TALib在我们的平台上编写的单股票MACD算法示例，使用了TALib的MACD方法：

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。
import talib


# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    context.s1 = "000001.XSHE"

    # 使用MACD需要设置长短均线和macd平均线的参数
    context.SHORTPERIOD = 12
    context.LONGPERIOD = 26
    context.SMOOTHPERIOD = 9
    context.OBSERVATION = 100
```

```
# 开始编写你的主要的算法逻辑

# bar_dict[order_book_id] 可以拿到某个证券的bar信息
# context.portfolio 可以拿到现在的投资组合状态信息

# 使用order_shares(id_or_ins, amount)方法进行落单

# TODO: 开始编写你的算法吧！

# 读取历史数据，使用sma方式计算均线准确度和数据长度无关，但是在使用ema方式计算均线时建
prices = history_bars(context.s1,context.OBSERVATION,'1d','close')

# 用Talib计算MACD取值，得到三个时间序列数组，分别为macd, signal 和 hist
macd, signal, hist = talib.MACD(prices, context.SHORTPERIOD,
                                context.LONGPERIOD, context.SMOOTHPERIOD)

plot("macd", macd[-1])
plot("macd signal", signal[-1])

# macd 是长短均线的差值，signal是macd的均线，使用macd策略有几种不同的方法，我们这里采

# 如果macd从上往下跌破macd_signal

if macd[-1] - signal[-1] < 0 and macd[-2] - signal[-2] > 0:
    # 计算现在portfolio中股票的仓位
    curPosition = context.portfolio.positions[context.s1].quantity
    #进行清仓
    if curPosition > 0:
        order_target_value(context.s1, 0)

# 如果短均线从下往上突破长均线，为入场信号
if macd[-1] - signal[-1] > 0 and macd[-2] - signal[-2] < 0:
    # 满仓入股
    order_target_percent(context.s1, 1)
```

多股票RSI算法示例

以下是一个我们使用TALib在我们的平台上编写的多股票RSI算法示例，使用了TALib的RSI方法：

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。
import talib


# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):

    # 选择我们感兴趣的股票
    context.s1 = "000001.XSHE"
    context.s2 = "601988.XSHG"
    context.s3 = "000068.XSHE"
    context.stocks = [context.s1, context.s2, context.s3]

    context.TIME_PERIOD = 14
```

```
# 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更新
def handle_bar(context, bar_dict):
    # 开始编写你的主要的算法逻辑

    # bar_dict[order_book_id] 可以拿到某个证券的bar信息
    # context.portfolio 可以拿到现在的投资组合状态信息

    # 使用order_shares(id_or_ins, amount)方法进行落单

    # TODO: 开始编写你的算法吧！

    # 对我们选中的股票集合进行loop，运算每一只股票的RSI数值
    for stock in context.stocks:
        # 读取历史数据
        prices = history_bars(stock,context.TIME_PERIOD+1, '1d', 'close')

        # 用Talib计算RSI值
        rsi_data = talib.RSI(prices, timeperiod=context.TIME_PERIOD)[-1]

        cur_position = context.portfolio.positions[stock].quantity
        # 用剩余现金的30%来购买新的股票
        target_available_cash = context.portfolio.cash * context.ORDER_PERCENT

        # 当RSI大于设置的上限阀值，清仓该股票
        if rsi_data > context.HIGH_RSI and cur_position > 0:
            order_target_value(stock, 0)

        # 当RSI小于设置的下限阀值，用剩余cash的一定比例补仓该股
        if rsi_data < context.LOW_RSI:
            logger.info("target available cash caled: " + str(target_available_cash
            # 如果剩余的现金不够一手 - 100shares，那么会被ricequant 的order management
            order_value(stock, target_available_cash)
```

财务数据策略

在回测开始前，通过查询回测开始当天的财务数据，获得市盈率大于55且小于60，营业总收入前10的股票，然后将所有资金平摊到这10个股票Buy & Hold的策略。您可以clone之后自行修改，通过查询回测开始当天的其它财务数据指标来筛选股票。但是注意需要调整资金量到￥300,000以上才会有比较好的落单效果（否则资金量不足以满足买入如此多股票组成的一篮子投资组合）

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。


# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    # 查询revenue前十名的公司的股票并且他们的pe_ratio在55和60之间。打fundamentals的时候
    fundamental_df = get_fundamentals(
        query(
            fundamentals.income_statement.revenue, fundamentals.eod_derivative_indi
        ).filter(
```

```
        ).order_by(
            fundamentals.income_statement.revenue.desc()
        ).limit(
            10
        )
    )

    # 将查询结果dataframe的fundamental_df存放在context里面以备后面只需：
    context.fundamental_df = fundamental_df

    # 实时打印日志看下查询结果，会有我们精心处理的数据表格显示：
    logger.info(context.fundamental_df)
    update_universe(context.fundamental_df.columns.values)

    # 对于每一个股票按照平均现金买入：
    context.stocks = context.fundamental_df.columns.values
    stocks_number = len(context.stocks)
    context.average_percent = 0.99 / stocks_number
    logger.info("Calculated average percent for each stock is: %f" % context.averag
    context.fired = False


# 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更新
def handle_bar(context, bar_dict):
    # 开始编写你的主要的算法逻辑

    # bar_dict[order_book_id] 可以拿到某个证券的bar信息
    # context.portfolio 可以拿到现在的投资组合状态信息

    # 使用order_shares(id_or_ins, amount)方法进行落单

    # TODO: 开始编写你的算法吧！

    # 对于选择出来的股票按照平均比例买入：
    if not context.fired:
        for stock in context.stocks:
            order_target_percent(stock, context.average_percent)
            logger.info("Bought: " + str(context.average_percent) + " % for stock:
        context.fired = True
```

根据收益和市盈率定期调仓策略

这是一个利用财务数据定期选股并且调仓的策略：选取市盈率在53到67之间，盈利前10的股票，然后按照等比例分配资金买入一个10个股票的投资组合：

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。
def query_fundamental(context, bar_dict):
    # 查询revenue前十名的公司的股票并且他们的pe_ratio在53和67之间。打fundamentals的时候
    fundamental_df = get_fundamentals(
        query(
            fundamentals.income_statement.revenue, fundamentals.eod_derivative_indi
        ).filter(
            fundamentals.eod_derivative_indicator.pe_ratio > 53
        ).filter(
```

```
        ).limit(
            10
        )
    )

    # 将查询结果dataframe的fundamental_df存放在context里面以备后面只需：
    context.fundamental_df = fundamental_df

    # 实时打印日志看下查询结果，会有我们精心处理的数据表格显示：
    logger.info(context.fundamental_df)

    # 对于每一个股票按照平均现金买入：
    context.stocks = context.fundamental_df.columns.values
    update_universe(context.stocks)

    stocksNumber = len(context.stocks)
    context.average_percent = 0.99 / stocksNumber
    logger.info("Calculated average percent for each stock is: %f" % context.averag

    # 先查一下选出来的股票是否在已有的portfolio里面：
    # 这样做并不是最好的，只是代码比较简单
    # 先清仓然后再买入这一个月新的符合条件的股票
    logger.info("Clearing all the current positions.")
    for holding_stock in context.portfolio.positions.keys():
        if context.portfolio.positions[holding_stock].quantity != 0:
            order_target_percent(holding_stock, 0)

    logger.info("Building new positions for portfolio.")
    for stock in context.stocks:
        order_target_percent(stock, context.average_percent)
        logger.info("Buying: " + str(context.average_percent) + " % for stock: " +


# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    scheduler.run_monthly(query_fundamental, monthday=1)


# 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更新
def handle_bar(context, bar_dict):
    # 开始编写你的主要的算法逻辑

    # bar_dict[order_book_id] 可以拿到某个证券的bar信息
    # context.portfolio 可以拿到现在的投资组合状态信息

    # 使用order_shares(id_or_ins, amount)方法进行落单

    # TODO: 开始编写你的算法吧！
    pass
```

海龟交易系统

海龟交易系统也是非常经典的一种策略，我们也放出了范例代码如下，而关于海龟交易系统的介绍也可以参照这篇帖子。

```
import talib
import math

def get_extreme(array_high_price_result, array_low_price_result):
    #抛开最新价格的价格序列
    np_array_high_price_result = np.array(array_high_price_result[:-1])
    np_array_low_price_result = np.array(array_low_price_result[:-1])
    #序列最大值
    max_result = np_array_high_price_result.max()
    #最小值
    min_result = np_array_low_price_result.min()
    #返回一个两个元素的list
    return [max_result, min_result]

#拿到真实震幅与头寸
def get_atr_and_unit( atr_array_result,  atr_length_result, total_value_result):
    #atr为真实平均振幅的最新值
    atr =  atr_array_result[ atr_length_result-1]
    #头寸
    unit = math.floor(total_value_result * .01 / atr)
    return [atr, unit]

#得到止损价格
def get_stop_price(first_open_price_result, units_hold_result, atr_result):
    stop_price = first_open_price_result - 2 * atr_result \
                 + (units_hold_result - 1) * 0.5 * atr_result
    return stop_price


def init(context):
    context.trade_day_num = 0
    context.unit = 0
    context.atr = 0
    context.trading_signal = 'start'
    context.pre_trading_signal = ''
    context.units_hold_max = 4
    context.units_hold = 0
    context.quantity = 0
    context.max_add = 0
    context.first_open_price = 0
    context.s = '000300.XSHG'
    context.open_observe_time = 55
    context.close_observe_time = 20
    context.atr_time = 20


def handle_bar(context, bar_dict):
    #当前合约的价值
    total_value = context.portfolio.total_value
    #context.open_observe_time+1个bar的每日最高价
    high_price = history_bars(context.s,context.open_observe_time+1, '1d', 'high')
    low_price_for_atr = history_bars(context.s,context.open_observe_time+1, '1d', '
    low_price_for_extreme = history_bars(context.s,context.close_observe_time+1, '1
    close_price = history_bars(context.s,context.open_observe_time+2, '1d', 'close'
    close_price_for_atr = close_price[:-1]
    #talib.ATR平均真实振幅
    atr_array = talib.ATR(high_price, low_price_for_atr, close_price_for_atr, timep
    #得到max_result
    maxx = get_extreme(high_price, low_price_for_extreme)[0]
```

```
atr = atr_array[-2]

if context.trading_signal != 'start':
    if context.units_hold != 0:
        context.max_add += 0.5 * get_atr_and_unit(atr_array, atr_array.size, to
else:
    context.max_add = bar_dict[context.s].last

#当前context.s持仓股数
cur_position = context.portfolio.positions[context.s].quantity
#当前仓位的现金
available_cash = context.portfolio.cash
#当前仓位的市场价值
market_value = context.portfolio.market_value

#当前仓位大于0并且当前价格小于止损价，产生信号"stop"
if (cur_position > 0 and
        bar_dict[context.s].last < get_stop_price(context.first_open_price, con
    context.trading_signal = 'stop'
else:
    #仓位大于0，并且当前价格小于观察时期的low的最小值
    if cur_position > 0 and bar_dict[context.s].last < minn:
        context.trading_signal = 'exit'
    else:
        #仓位大于0但小于策略设计仓位，并且当前价格大于观察时期的high的最大值，产生追加
        if (bar_dict[context.s].last > context.max_add and context.units_hold !
                context.units_hold < context.units_hold_max and
                available_cash > bar_dict[context.s].last*context.unit):
            context.trading_signal = 'entry_add'
        else:
            #价格大于maxx并且持仓为0，产生信号"entry"
            if bar_dict[context.s].last > maxx and context.units_hold == 0:
                context.max_add = bar_dict[context.s].last
                context.trading_signal = 'entry'

atr = get_atr_and_unit(atr_array, atr_array.size, total_value)[0]
if context.trade_day_num % 5 == 0:
    #每5个bar计算一次context.unit寸头
    context.unit = get_atr_and_unit(atr_array, atr_array.size, total_value)[1]
context.trade_day_num += 1
context.quantity = context.unit

#不同信号下的操作
if (context.trading_signal != context.pre_trading_signal or
        (context.units_hold < context.units_hold_max and context.units_hold > 1
        context.trading_signal == 'stop'):
    if context.trading_signal == 'entry':
        context.quantity = context.unit
        if available_cash > bar_dict[context.s].last*context.quantity:
            order_shares(context.s, context.quantity)
            context.first_open_price = bar_dict[context.s].last
            context.units_hold = 1

    if context.trading_signal == 'entry_add':
        context.quantity = context.unit
        order_shares(context.s, context.quantity)
        context.units_hold += 1

    if context.trading_signal == 'stop':
```

```
        if context.trading_signal == 'exit':
            if cur_position > 0:
                order_shares(context.s, -cur_position)
                context.units_hold = 0

    context.pre_trading_signal = context.trading_signal
```

股指期货MACD日回测

以下是一个使用TALib进行股指期货主力合约日级别回测MACD算法示例：

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等
import talib


# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递
def init(context):
    # context内引入全局变量s1，存储目标合约信息
    context.s1 = 'IF1606'

    # 使用MACD需要设置长短均线和macd平均线的参数
    context.SHORTPERIOD = 12
    context.LONGPERIOD = 26
    context.SMOOTHPERIOD = 9
    context.OBSERVATION = 50

    #初始化时订阅合约行情。订阅之后的合约行情会在handle_bar中进行更新
    subscribe(context.s1)


# 你选择的期货数据更新将会触发此段逻辑，例如日线或分钟线更新
def handle_bar(context, bar_dict):
    # 开始编写你的主要的算法逻辑
    # 获取历史收盘价序列，history_bars函数直接返回ndarray，方便之后的有关指标计算
    prices = history_bars(context.s1, context.OBSERVATION, '1d', 'close')

    # 用Talib计算MACD取值，得到三个时间序列数组，分别为macd,signal 和 hist
    macd, signal, hist = talib.MACD(prices, context.SHORTPERIOD,
                                    context.LONGPERIOD, context.SMOOTHPERIOD)

    # macd 是长短均线的差值，signal是macd的均线，如果短均线从下往上突破长均线，为入场信号
    if macd[-1] - signal[-1] > 0 and macd[-2] - signal[-2] < 0:
        sell_qty = context.portfolio.positions[context.s1].sell_quantity
        # 先判断当前卖方仓位，如果有，则进行平仓操作
        if sell_qty > 0:
            buy_close(context.s1, 1)
        # 买入开仓
        buy_open(context.s1, 1)

    if macd[-1] - signal[-1] < 0 and macd[-2] - signal[-2] > 0:
        buy_qty = context.portfolio.positions[context.s1].buy_quantity
        # 先判断当前买方仓位，如果有，则进行平仓操作
        if buy_qty > 0:
            sell_close(context.s1, 1)
```

商品期货跨品种配对交易

该策略为分钟级别回测。运用了简单的移动平均以及布林带（Bollinger Bands）作为交易信号产生源。有关对冲比率（HedgeRatio）的确定，您可以在我们的**研究平台**上面通过 `import statsmodels.api as sm` 引入statsmodels中的OLS方法进行线性回归估计。具体估计窗口，您可以根据自己策略需要自行选择。

策略中的移动窗口选择为60分钟，即在每天开盘60分钟内不做任何交易，积累数据计算移动平均值。当然，这一移动窗口也可以根据自身需要进行灵活选择。下面例子中使用了黄金与白银两种商品期货进行配对交易。简单起见，例子中期货的价格并未做对数差处理。

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。
import numpy as np


# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    context.s1 = 'AG1612'
    context.s2 = 'AU1612'

    # 设置全局计数器
    context.counter = 0

    # 设置滚动窗口
    context.window = 60

    # 设置对冲手数,通过研究历史数据进行价格序列回归得到该值
    context.ratio = 15

    context.up_cross_up_limit = False
    context.down_cross_down_limit = False

    # 设置入场临界值
    context.entry_score = 2

    # 初始化时订阅合约行情。订阅之后的合约行情会在handle_bar中进行更新
    subscribe([context.s1, context.s2])


# before_trading此函数会在每天交易开始前被调用，当天只会被调用一次
def before_trading(context):
    # 样例商品期货在回测区间内有夜盘交易,所以在每日开盘前将计数器清零
    context.counter = 0


# 你选择的期货数据更新将会触发此段逻辑，例如日线或分钟线更新
def handle_bar(context, bar_dict):

    # 获取当前一对合约的仓位情况。如尚未有仓位,则对应持仓量都为0
    position_a = context.portfolio.positions[context.s1]
    position_b = context.portfolio.positions[context.s2]

    context.counter += 1
```

```
# 获取当天历史分钟线价格队列
price_array_a = history_bars(context.s1, context.window, '1m', 'close')
price_array_b = history_bars(context.s2, context.window, '1m', 'close')

# 计算价差序列、其标准差、均值、上限、下限
spread_array = price_array_a - context.ratio * price_array_b
std = np.std(spread_array)
mean = np.mean(spread_array)
up_limit = mean + context.entry_score * std
down_limit = mean - context.entry_score * std

# 获取当前bar对应合约的收盘价格并计算价差
price_a = bar_dict[context.s1].close
price_b = bar_dict[context.s2].close
spread = price_a - context.ratio * price_b

# 如果价差低于预先计算得到的下限,则为建仓信号,'买入'价差合约
if spread <= down_limit and not context.down_cross_down_limit:
    # 可以通过logger打印日志
    logger.info('spread: {}, mean: {}, down_limit: {}'.format(spread, mean,
    logger.info('创建买入价差中...')

    # 获取当前剩余的应建仓的数量
    qty_a = 1 - position_a.buy_quantity
    qty_b = context.ratio - position_b.sell_quantity

    # 由于存在成交不超过下一bar成交量25%的限制,所以可能要通过多次发单成交才能够成
    if qty_a > 0:
        buy_open(context.s1, qty_a)
    if qty_b > 0:
        sell_open(context.s2, qty_b)
    if qty_a == 0 and qty_b == 0:
        # 已成功建立价差的'多仓'
        context.down_cross_down_limit = True
        logger.info('买入价差仓位创建成功!')

# 如果价差向上回归移动平均线,则为平仓信号
if spread >= mean and context.down_cross_down_limit:
    logger.info('spread: {}, mean: {}, down_limit: {}'.format(spread, mean,
    logger.info('对买入价差仓位进行平仓操作中...')

    # 由于存在成交不超过下一bar成交量25%的限制,所以可能要通过多次发单成交才能够成
    qty_a = position_a.buy_quantity
    qty_b = position_b.sell_quantity
    if qty_a > 0:
        sell_close(context.s1, qty_a)
    if qty_b > 0:
        buy_close(context.s2, qty_b)
    if qty_a == 0 and qty_b == 0:
        context.down_cross_down_limit = False
        logger.info('买入价差仓位平仓成功!')

# 如果价差高于预先计算得到的上限,则为建仓信号,'卖出'价差合约
if spread >= up_limit and not context.up_cross_up_limit:
    logger.info('spread: {}, mean: {}, up_limit: {}'.format(spread, mean, u
    logger.info('创建卖出价差中...')
    qty_a = 1 - position_a.sell_quantity
    qty_b = context.ratio - position_b.buy_quantity
    if qty_a > 0:
```

```
        if qty_a == 0 and qty_b == 0:
            context.up_cross_up_limit = True
            logger.info('卖出价差仓位创建成功')

    # 如果价差向下回归移动平均线,则为平仓信号
    if spread < mean and context.up_cross_up_limit:
        logger.info('spread: {}, mean: {}, up_limit: {}'.format(spread, mean, u
        logger.info('对卖出价差仓位进行平仓操作中...')
        qty_a = position_a.sell_quantity
        qty_b = position_b.buy_quantity
        if qty_a > 0:
            buy_close(context.s1, qty_a)
        if qty_b > 0:
            sell_close(context.s2, qty_b)
        if qty_a == 0 and qty_b == 0:
            context.up_cross_up_limit = False
            logger.info('卖出价差仓位平仓成功!')
```

Alpha策略----多因子对冲

多因子模型是应用最广泛的一种选股模型，基本原理是采用一系列的因子作为选股标准，满足这些因子的股票则被买入，不满足的则卖出。各种多因子模型核心的区别第一是在因子的选取上，第二是在如何用多因子综合得到一个最终的判断。股票端我们按照多因子的模型买入，期货端则一手空单对冲风险。

```
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。
import numpy as np
import math
# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
#选取板块
    context.stks=[]
    context.stks.append(sector("consumer discretionary"))
    context.stks.append(sector("consumer staples"))
    context.stks.append(sector("health care"))
    context.stks.append(sector("telecommunication services"))
    context.stks.append(sector("utilities"))
    context.stks.append(sector("materials"))

    context.flag=True
    # 确定运行频率
    scheduler.run_daily(rebalance)

    # 手动添加银行板块
    context.stocks = ['000001.XSHE','002142.XSHE','600000.XSHG','600015.XSHG','6000
    # 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更
def get_stocks(context, bar_dict):
    stocks=set([])

    for i in range(0,6):

        # 在这个循环里，首先获取每个板块的财务数据
        fundamental_df = get_fundamentals(

            query(fundamentals.financial_indicator.adjusted_return_on_equity_weight
```

```
                fundamentals.eod_derivative_indicator.pe_ratio<999
            ).order_by(
                fundamentals.eod_derivative_indicator.pb_ratio
            )
        )

        # 使用pandas对财务数据进行排名并打分
        df=fundamental_df.T
        df=df.sort(columns='pb_ratio')
        df['pb_score']=list(range(1,len(df)+1))
        df=df.sort(columns='pe_ratio')
        df['pe_score']=list(range(1,len(df)+1))
        scores=[]
        for stock in df.T.columns.values:

            scores.append(df.loc[stock,'pe_score']+df.loc[stock,'pb_score'])
        df['scores']=list(scores)
        df=df.sort(columns='scores')

        #取得分最低的三个股票
        df=df.head(3)
        #logger.info(df)

        stocks = stocks | set(df.T.columns.values)
        #logger.info(i)

    # 银行板块单独按照市净率取市净率最低的两个
    fundamental_df = get_fundamentals(
        query(fundamentals.financial_indicator.adjusted_return_on_equity_weighted_a
        ).filter(
            fundamentals.income_statement.stockcode.in_(context.stocks)
        ).order_by(
            fundamentals.eod_derivative_indicator.pb_ratio
        ).limit(
            2
        )
    )


    # 买入的股票，进行调仓操作
    stocks =stocks | set(fundamental_df.columns.values)
    return stocks
def rebalance(context, bar_dict):
    stocks =  get_stocks(context, bar_dict)
    holdings = set(get_holdings(context))

    to_buy = stocks - holdings
    to_sell = holdings - stocks
    to_buy2= stocks - holdings

    for stock in to_sell:
        if bar_dict[stock].suspended == False:
            order_target_percent(stock , 0)

    if len(to_buy) == 0:
        return

    to_buy = get_trading_stocks(to_buy, context, bar_dict)
    cash = context.portfolio.cash
```

```
        if average_value > total_value/len(to_buy):
            average_value = total_value/len(to_buy)

    for stock in to_buy:
        if (bar_dict[stock].suspended == False)and(context.portfolio.cash>average_v
            order_target_value(stock, average_value)
    if context.flag==True :
        sell_open('IF88', 1)
        context.flag=False

# 得到交易的股票
def get_trading_stocks(to_buy, context, bar_dict):
    trading_stocks = []
    for stock in to_buy:
        if bar_dict[stock].suspended == False:
            trading_stocks.append(stock)

    return trading_stocks

# 持仓的股票
def get_holdings(context):
    positions = context.portfolio.stock_account.positions

    holdings = []
    for position in positions:
        if positions[position].quantity > 0:
            holdings.append(position)

    return holdings
def handle_bar(context, bar_dict):
    # TODO: 开始编写你的算法吧！
    pass
```