**5. 功能代码集合**

**前置说明**

此文档将会提供代码框架在课程学习中，及实际的策略开发时可能用到的功能代码及其使用说明。

|  |
| --- |
| **你可以参考本文档优化代码框架开发的策略代码。** |

此外，为便于展示功能效果，本文档中**案例代码的参数会各有不同，具体数值仅作示范**。

**取消订单**

**功能说明**

|  |
| --- |
| **基本功能：**在行情变化较快时，可以避免行情变化过大，形成错误的买入点 。  **其他功能：**可以与单日资金风控一起使用，释放单日可用资金。 |

在使用量化交易时，我们在某时刻下的订单可能会因为市场撮合等原因导致订单无法成交。而等到要成交时已经不满足原有的策略条件了，那此时再买入就不是我们所期望的。而取消订单功能，则可以用于避免此类情况出现。

此外，你也可以将它与单日资金风控一起使用，避免未成交订单占用金额，释放单日可用资金。

**注意：**这里取消订单功能，仅针对买入订单。对于卖出订单，挂单后不因为超时而取消。除非有特殊情况需要取消订单，这个需要根据个人需求进行设置。

**功能预期**

|  |
| --- |
| 如果委托买入订单的委托时间超过指定的时间，则取消订单。 |

如果功能实现，你将能在终端找到发出了委托，但之后因为超时而被取消的订单。

**功能实现**

实现取消订单功能，我们在代码框架中需要新增的内容如下：



具体的新增代码为：

1. 指标模块-补充设置参数

|  |
| --- |
| Python # 设置订单取消间隔时间为1800秒 context.cancel\_interval = 1800 |

**温馨提示：**这里的订单取消间隔时间你可以根据自身需求自由调整，本代码仅作示范。

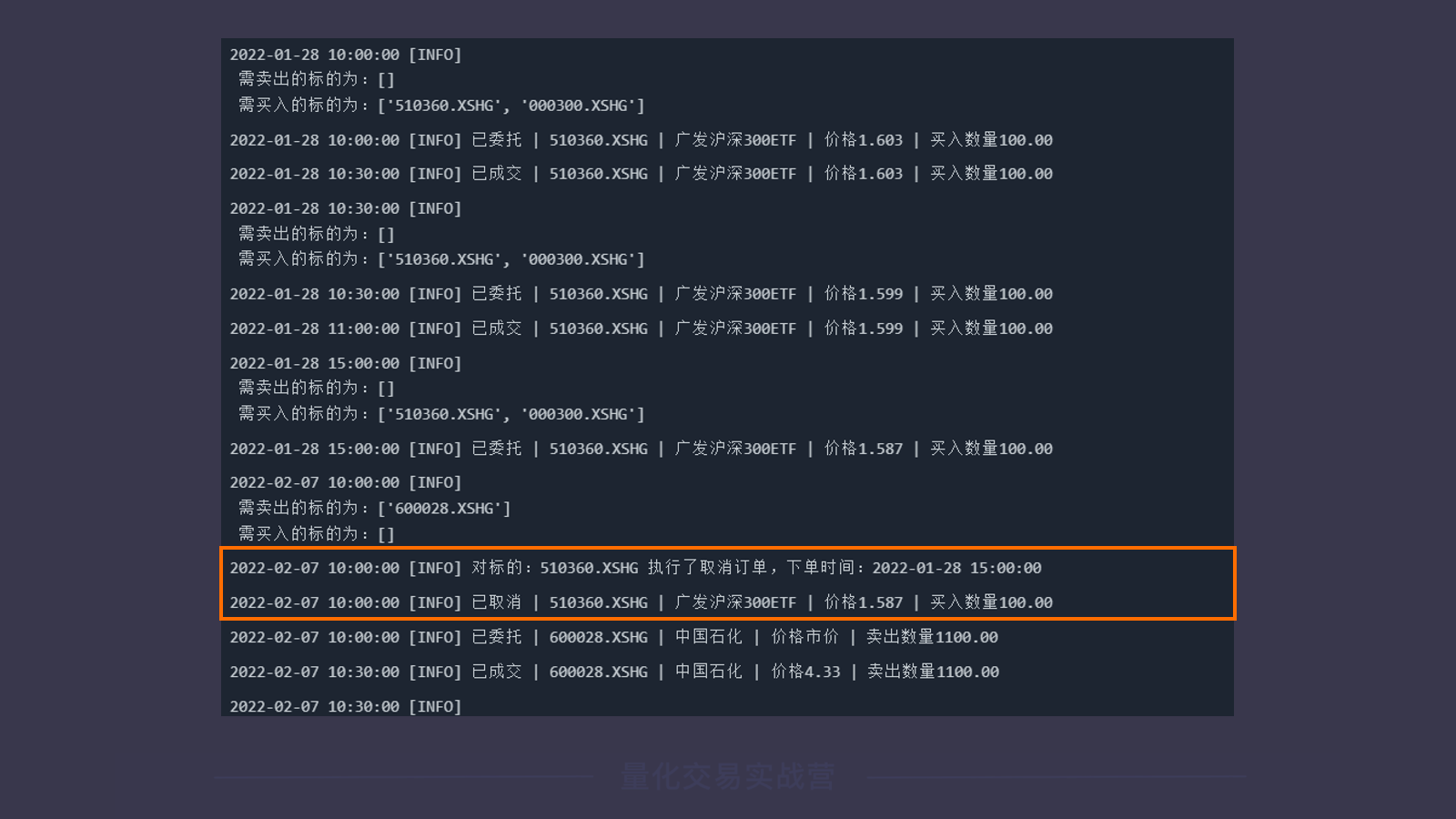
1. 择时模块-取消超时订单

|  |
| --- |
| Python # 获取当前日期时间 current\_datetime = context.datetime.datetime()  # 获取所有已委托的订单 submitted\_orders = context.get\_orders(status='submitted') # 遍历所有已委托的订单 for order in submitted\_orders:  # 获取委托创建时间  created\_time = order.created\_at  # 获取当前距离委托时，间隔的秒数  interval\_now = (current\_datetime - created\_time).seconds   # 如果该委托单为买入并且当前距离委托的时间间隔超过订单取消间隔时间  if order.ordtype == 0 and interval\_now >= context.cancel\_interval:  # 取消订单  context.cancel(order)  context.log('对标的：{} 执行了取消订单，下单时间：{}'.format(order.data.\_name, created\_time)) |

将以上代码添加到以 RSI 交易策略为例的代码框架后，得到的完整代码内容如下：

[**示例代码**](https://prod.pandateacher.com/web/fortrader-web/#/home?share_code=DJvqroQGQzAmx4by)

**功能效果**



**T+1 交易限制**

**功能说明**

股市交易存在 T+1 规则，即：今天买入的股票是不能卖出的，要等下一个交易日才能卖出。

|  |
| --- |
| T + 1 交易限制功能代码，主要用于避免因 T+1 规则限制导致无法卖出订单的问题。 |

**功能预期**

|  |
| --- |
| 在卖出标的时，只卖出可被交易的订单。 |

如果功能实现，我们在核查卖出仓位时，所卖出的订单均是提前一日持仓的。

**功能实现**

实现T +1 交易限制功能，我们在代码框架中需要调整的内容如下：



需要调整内容逻辑是一样的，但在不同模块的代码中有不同的表现形式，内容如下：

1. 择时模块 - 计算可交易订单数量后执行平仓

|  |
| --- |
| Python # 遍历需要卖出的标的 for sell\_data in trade\_dict['需卖出的标的对象']:  # 获取该标的当天可交易的数量  salable\_size = context.getposition(sell\_data).available  # 如果可交易数量大于0  if salable\_size > 0:  # 卖出所有可交易数量  context.sell(data=sell\_data, size=salable\_size)  # # 执行卖出，订单类型为限价单  # context.sell(data=sell\_data, size=salable\_size, price=sell\_data.close[0], exectype=Order.Limit)   # 遍历需要买入的标的 for buy\_data in trade\_dict['需买入的标的对象']:  # 执行买入，订单类型为市价单  context.buy(data=buy\_data, size=context.trade\_size)  # # 执行买入，订单类型为限价单  # context.buy(data=buy\_data, size=context.trade\_size, price=buy\_data.close[0], exectype=Order.Limit) |

**提示：**标橙部分为调整的内容。

1. 风控模块-计算可交易订单数量后执行平仓

|  |
| --- |
| Python def control\_risk(context):  """风控"""  # 遍历所有标的  for data in context.datas:  # 获取获取该标的当天可交易的数量  salable\_size = context.getposition(data).available  # 如果可交易数量大于0  if salable\_size > 0:  # 获取持仓均价  hold\_price = context.getposition(data).price  # 计算止损价  stop\_price = (1 - context.stop\_loss) \* hold\_price  # 计算止盈价  profit\_price = (1 + context.take\_profit) \* hold\_price   # 如果当前价格达到了止盈或止损价  if data.close[0] < stop\_price or data.close[0] > profit\_price:  # 卖出所有可交易的持仓，订单类型为市价单  context.sell(data=data, size=salable\_size)  context.log("执行了止盈或止损") |

**提示：**标橙部分为调整的内容。

将以上代码内容调整到以 RSI 交易策略为例的代码框架后，得到的完整代码内容如下：

[**示例代码**](https://prod.pandateacher.com/web/fortrader-web/#/home?share_code=jaOxY8mWPY0JBV3G)

**功能效果**



**总资金风控**

**功能说明**

|  |
| --- |
| 能设定最大使用资金比例，以避免买入资金占比过多的风险。 |

**注：**本功能代码**只能确保总资金使用比例不会因买入标的而超过最大使用资金比例，而无法确保总资金使用比例永远不超过最大使用资金比例。**因为当持有标的价格上涨时，也有可能导致总资金使用比例超出最大使用资金比例，而这部分情况并不在本功能代码的作用范围内。

**功能预期**

|  |
| --- |
| 当买入策略信号中的标的后，总资金使用比例会超过代码中设定的最大使用资金比例时，则不实际买入标的。 |

**功能实现**

实现总资金风控功能，我们在代码框架中需要新增及调整的内容如下：



具体新增及调整的代码为：

1. 指标模块 - 补充设置风控参数

|  |
| --- |
| Python # 设置最大使用资金比例 context.max\_percent = 0.2 |

**注：**此处设置的资金比例仅作示范，你可以根据自身需求自由调整。

1. 择时模块 - 先判断投入的总资金是否超过最大资金使用比例，再执行交易

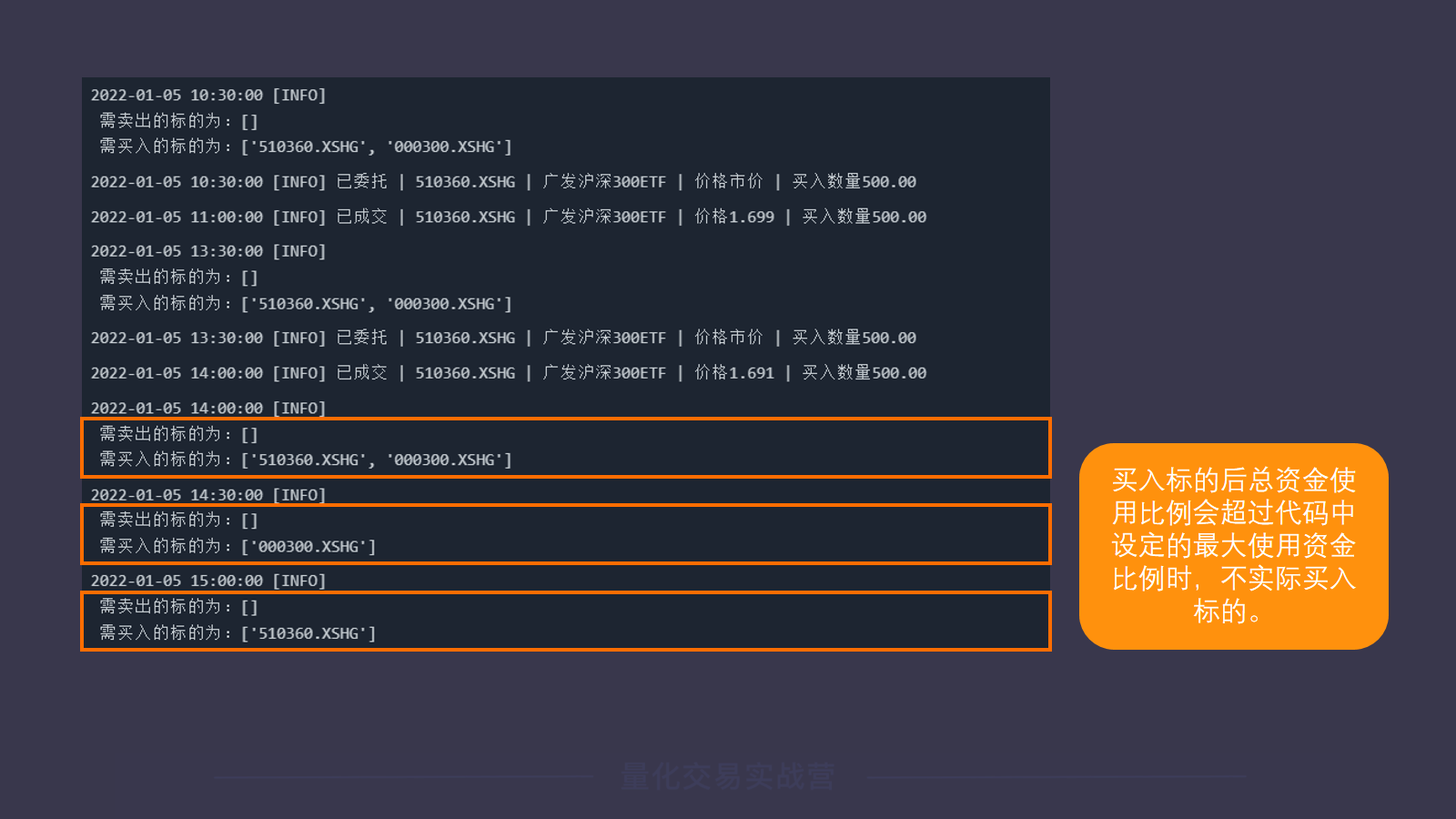
|  |
| --- |
| Python # 遍历需要卖出的标的 for sell\_data in trade\_dict['需卖出的标的对象']:  # 获取持仓数量  hold\_size = context.getposition(sell\_data).size  # 如果持仓数量大于0  if hold\_size > 0:  # 执行平仓，订单类型为市价单  context.close(data=sell\_data)  # # 执行平仓，订单类型为限价单  # context.sell(data=sell\_data, size=hold\_size, price=sell\_data.close[0], exectype=Order.Limit)  # 遍历需要买入的标的 for buy\_data in trade\_dict['需买入的标的对象']:  # 计算当前买入所需的资金  buy\_value = buy\_data.close[0] \* context.trade\_size  # 计算剩余的现金比例  remaining\_cash\_percent = (context.broker.cash - buy\_value) / context.broker.getvalue()   # 如果投入的总资金不超过最大资金使用比例  if remaining\_cash\_percent > (1 - context.max\_percent):  # 执行买入，订单类型为市价单  context.buy(data=buy\_data, size=context.trade\_size)  # # 执行买入，订单类型为限价单  # context.buy(data=buy\_data, size=context.trade\_size, price=buy\_data.close[0], exectype=Order.Limit) |

**提示：**标橙部分为调整的内容。

将以上代码内容调整到以 RSI 交易策略为例的代码框架后，得到的完整代码内容如下：

[**示例代码**](https://prod.pandateacher.com/web/fortrader-web/#/home?share_code=jaOxY8mWPY0JBV3G)

**功能效果**



**单日资金风控**

**功能说明**

|  |
| --- |
| 能设定每日最大的买入资金比例，以避免单日买入资金过大的风险。 |

在使用量化交易做日内交易时，我们每天可能多次买入而导致仓位超出预期。用单日资金风控功能代码则可以避免这个风险。

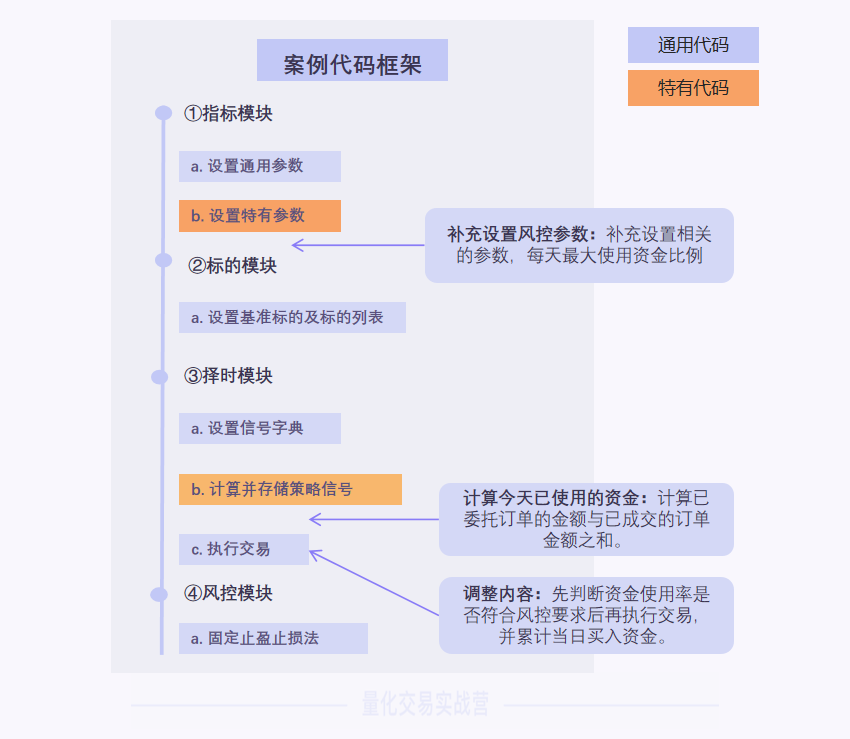
**功能预期**

|  |
| --- |
| 每日买入的资金比例不超过代码中设定的最大资金比例。 |

如果功能实现，策略交易结果中每日交易的资金比例会均小于你在代码中设定的值。

**功能实现**

实现单日资金风控功能，我们在代码框架中需要新增及调整的内容如下：



具体新增及调整的代码为：

1. 指标模块 - 补充设置相关的参数，每天最大使用资金比例

|  |
| --- |
| Python # 设置每天最大使用资金比例 context.day\_max\_percent = 0.1 |

**注：**此处设置的资金比例仅作示范，你可以根据自身需求自由调整。

1. 择时模块-计算今天已使用的资金

|  |
| --- |
| Python # 设置记录当天买入资金为0 day\_buy\_value = 0 # 获取所有已委托的订单 submitted\_orders = context.get\_orders(status='submitted') # 遍历所有已委托的订单 for order in submitted\_orders:  *# 如果该委托单为买入*  if order.ordtype == 0:  *# 计算订单金额*  order\_value = order.created.price \* order.created.size  *# 将订单金额累加到当天买入资金变量中*  day\_buy\_value += order\_value |

1. 择时模块-先判断资金使用率是否符合风控要求后再执行交易，并累计当日买入资金。

|  |
| --- |
| Python # 遍历需要卖出的标的 for sell\_data in trade\_dict['需卖出的标的对象']:  # 获取持仓数量  hold\_size = context.getposition(sell\_data).size  # 如果持仓数量大于0  if hold\_size > 0:  # 执行平仓，订单类型为市价单  context.close(data=sell\_data)  # # 执行平仓，订单类型为限价单  # context.sell(data=sell\_data, size=hold\_size, price=sell\_data.close[0], exectype=Order.Limit)  # 遍历需要买入的标的 for buy\_data in trade\_dict['需买入的标的对象']:  # 计算当前买入所需的资金  buy\_value = buy\_data.close[0] \* context.trade\_size  # 计算当天用于买入的资金比例  day\_buy\_percent = (day\_buy\_value + buy\_value) / context.broker.getvalue()   # 如果当天用于买入的资金比例小于每天最大买入资金比例  if day\_buy\_percent < context.day\_max\_percent:  # 执行买入，订单类型为市价单  order = context.buy(data=buy\_data, size=context.trade\_size)  # # 执行买入，订单类型为限价单  # order = context.buy(data=buy\_data, size=context.trade\_size, price=buy\_data.close[0], exectype=Order.Limit)   # 如果order不为None  if order:  # 将本次买入金额累加到当天用于买入的资金中  day\_buy\_value += buy\_value |

**提示：**标橙部分为调整的内容。

将以上代码内容调整到以 RSI 交易策略为例的代码框架后，得到的完整代码内容如下：

[**示例代码**](https://prod.pandateacher.com/web/fortrader-web/#/home?share_code=e4vXV8Xy55oYLD9p)

**功能效果**



**其他课程涉及功能代码**

**打印可交易订单数量**

**功能说明**

在代码框架中添加了 T+1 交易限制功能代码后，打印各标的可交易的订单数量，方便我们在终端中做检查。

**功能代码**

在择时模块，取消订单的代码后添加一段代码。

|  |
| --- |
| Python # 遍历需要卖出的标的 for sell\_data in trade\_dict['需卖出的标的对象']:  # 获取该标的当天可交易的数量  salable\_size = context.getposition(sell\_data).available  # 打印可交易数量  context.log('当前标的：{}当天可交易的订单数量为：{}'.format(sell\_data.\_name, salable\_size))    # 如果可交易数量大于0  if salable\_size > 0:  # 卖出所有可交易数量  context.sell(data=sell\_data, size=salable\_size)  # # 执行卖出，订单类型为限价单  # context.sell(data=sell\_data, size=salable\_size, price=sell\_data.close[0], exectype=Order.Limit) |

**提示：**标橙部分为调整的内容。

将以上代码内容调整到添加了 T+1 功能的 RSI 交易策略代码框架后，得到的完整代码内容如下：

[示例代码](https://hitrader.yueniusz.com/#/home?share_code=W4JkV0dPMk03bvga)

**功能效果**



**区分标的类型**

**功能说明**

区分标的类型，用于资金风控中花费总资金的计算。

**功能代码**

|  |
| --- |
| Python # 获取标的类型 data\_type = context.get\_symbol\_information(buy\_data.\_name)['type'] # 如果标的是基金 if data\_type == 'fund':  # 买入价格等于收盘价乘以0.999，并保留3位小数  buy\_price = round(buy\_data.close[0] \* 0.999, 3) # 如果是股票 elif data\_type == 'stock':  # 买入价格等于收盘价乘以0.999，并保留2位小数  buy\_price = round(buy\_data.close[0] \* 0.999, 2) # 其他情况 else:  # 买入价格等于收盘价乘以0.999，并取整  buy\_price = round(buy\_data.close[0] \* 0.999)  # 计算当前买入所需的资金 buy\_value = buy\_price \* context.trade\_size # 计算当天用于买入的资金比例 day\_buy\_percent = (day\_buy\_value + buy\_value) / context.broker.getvalue() # 计算剩余的现金比例 remaining\_cash\_percent = (context.broker.cash - buy\_value) / context.broker.getvalue()  # 如果当天用于买入的资金比例小于每天最大买入资金比例，并且投入的总资金不超过最大资金使用比例 if day\_buy\_percent < context.day\_max\_percent and remaining\_cash\_percent > (1 - context.max\_percent): |

**提示：**标橙部分为调整的内容。

**时间风控（交易日）**

**功能说明**

|  |
| --- |
| 为了规避策略的时限性风险，如果策略超过 N 个交易日还没有卖出信号，就强制平仓 |

该功能可用于 1min级、30min 级、日级

**功能预期**

|  |
| --- |
| 检查数据图表区的 K 线和买卖点，预期是持仓周期（如果有多次买入，则为最近一次买入到卖出的时间）均小于预设的 N 个交易日；  同时，如果指标线不满足卖出条件，但是出现了卖出点，那么在终端对应的时间处能看到触发时间风控的记录“已达到最长持仓天数，执行平仓”。 |

**功能实现**

实现时间风控功能，我们在代码框架中需要新增及调整的内容如下：



具体新增及调整的代码为：

1. 指标模块 - 补充设置参数及字典

|  |
| --- |
| Python # 设置最近一次买入时间记录字典 context.recently\_buy\_dict = {}  # 设置最长持仓时间为1天 context.hold\_time = 1 |

**提示：**标橙部分为调整的内容。

1. 风控模块 - 更新买入时间

|  |
| --- |
| Python  # ------------------时间风控--更新买入时间  # 获取所有已成交的订单  completed\_orders = context.get\_orders(status='completed')  # 遍历所有已成交的订单  for order in completed\_orders:  # 获取已成交的时间戳  completed\_timestamp = order.updated\_at.timestamp()  # 如果该成交单为买入，并且记录的成交的时间比当前订单成交的时间晚  if order.ordtype == 0 and context.recently\_buy\_dict[order.data.\_name] < completed\_timestamp:  # 更新当前标的最近一次买入时间  context.recently\_buy\_dict[order.data.\_name] = completed\_timestamp |

**提示：**标橙部分为调整的内容。

1. 风控模块 - 计算当前距离最近一次买入相隔的交易日

|  |
| --- |
| Python  # 遍历所有标的  for data in context.datas:  # 获取标的当前持仓数量  hold\_size = context.getposition(data).size  # 如果有持仓  if hold\_size > 0:  # 获取当前的时间戳  now\_timestamp = context.datetime.datetime().timestamp()  # 获取当前标的最近一次买入的时间戳  recently\_timestamp = context.recently\_buy\_dict[data.\_name]  # 计算当前距离最近一次买入相隔的交易日  days = context.get\_symbol\_timedelta(symbol\_exchange=data.\_name, start\_time=recently\_timestamp, end\_time=now\_timestamp)   # 获取持仓均价  hold\_price = context.getposition(data).price  # 计算止损价  stop\_price = (1 - context.stop\_loss) \* hold\_price  # 计算止盈价  profit\_price = (1 + context.take\_profit) \* hold\_price   # 如果当前价格达到了止盈或止损价  if data.close[0] < stop\_price or data.close[0] > profit\_price:  # 执行平仓  context.close(data=data)  context.log("执行了止盈或止损")   # 如果当前距离最近一次买入的交易日数 大于等于最长持仓天数  elif days >= context.hold\_time:  # 执行平仓  context.close(data=data)  context.log("已达到最长持仓天数，执行平仓") |

**提示：**标橙部分为调整的内容。

1. 风控模块 - 针对当前距离最近一次买入的交易日数超过最长持仓天数的标的，执行平仓

|  |
| --- |
| Python  # 遍历所有标的  for data in context.datas:  # 获取标的当前持仓数量  hold\_size = context.getposition(data).size  # 如果有持仓  if hold\_size > 0:  # 获取当前的时间戳  now\_timestamp = context.datetime.datetime().timestamp()  # 获取当前标的最近一次买入的时间戳  recently\_timestamp = context.recently\_buy\_dict[data.\_name]  # 计算当前距离最近一次买入相隔的交易日  days = context.get\_symbol\_timedelta(symbol\_exchange=data.\_name, start\_time=recently\_timestamp, end\_time=now\_timestamp)   # 获取持仓均价  hold\_price = context.getposition(data).price  # 计算止损价  stop\_price = (1 - context.stop\_loss) \* hold\_price  # 计算止盈价  profit\_price = (1 + context.take\_profit) \* hold\_price   # 如果当前价格达到了止盈或止损价  if data.close[0] < stop\_price or data.close[0] > profit\_price:  # 执行平仓  context.close(data=data)  context.log("执行了止盈或止损")   # 如果当前距离最近一次买入的交易日数 大于等于最长持仓天数  elif days >= context.hold\_time:  # 执行平仓  context.close(data=data)  context.log("已达到最长持仓天数，执行平仓") |

**提示：**标橙部分为调整的内容。

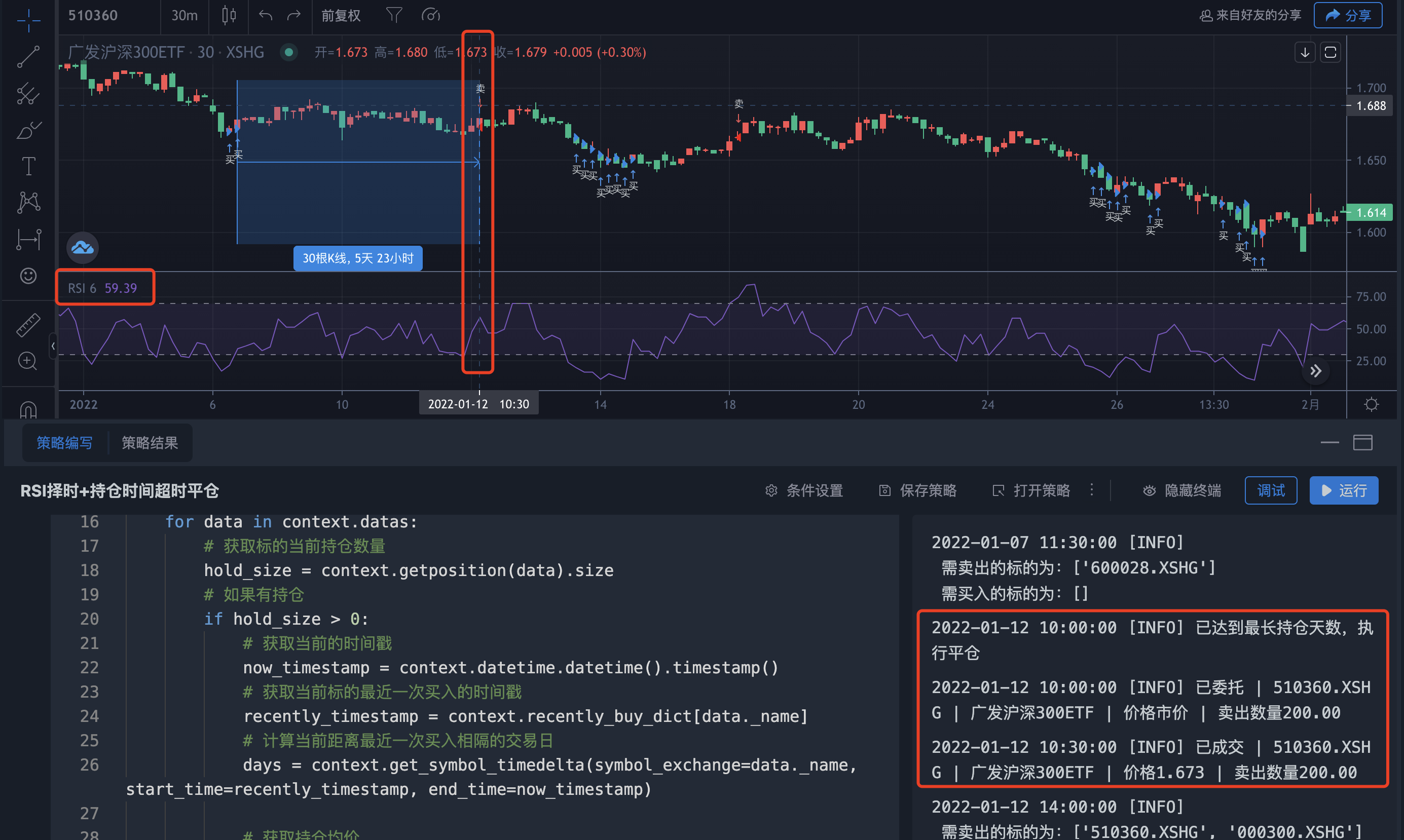
[**示例代码**](https://prod.pandateacher.com/web/fortrader-web/#/home?share_code=lyvkn8ykn4016bWj)

**功能效果**

在数据图表区，持仓时间均小于 4 个交易日（数据频次为 30min 时，4 个交易日有32 根 K 线）。



定位到某一次卖出，可以看到，在2022-01-12 10:30:00，RSI<70，但是产生了卖出点，找到终端，发现“已达到最长持仓天数，执行平仓”。



**定时委托**

**功能说明**

|  |
| --- |
| 策略的选股、择时以及风控函数的执行，是由 K线的推送信号驱动的，即必须有一根新的 K线推送过来时，才会驱动策略去计算，这样就会产生一个问题，日内的最后一根 K线推送过来时，已经休市了，此时如果策略计算出了买卖信号，委托下单是会被柜台拒绝的。这种场景下，下午 3 点的最后一根 K线产生的信号，应该要在第二天的开盘时进行下单，但由于上文提到的，策略的运算时机是由 K线驱动的，开盘时交易所是没有新的 K线推送的（日线不影响，日线是另一种机制），每天的第一根 K线一定是第二个频次开始的时候，比如小时频次，第一根 K线到来肯定是 10:30。因此，为了解决这种场景，框架实现了定时任务，可指定时间点执行买卖操作。 |

**功能实现**

要实现定时订单，主要是靠context.buy 和 context.sell中的两个参数去控制的。

* run\_at：datetime 类型，表示要执行的时间点，默认为 None，表示立即执行，即不开启定时
* misfire\_grace\_time：int类型，单位秒。表示允许错过执行的最长时间间隔，默认 600秒，这个参数主要是控制定时任务过期是否丢弃，比如设定 9:30 下单，但是我 9:31 才启动策略，那么此时我的定时订单已经过期了，但如果设置 misfire\_grace\_time=600，表示允许过期 10 分钟，那么在 9:40 之前启动策略，9:30 这个时间的订单依然会执行。

|  |
| --- |
| Python def timing(context):  """择时"""  import datetime  now = datetime.datetime.now()  run\_at = None  if now.hour >= 15: # 判断是否是下午三点了  # 获取下一个交易日  next\_day = context.get\_next\_trading\_date(datetime.date.today())  # 下一个交易日的 9:30  run\_at = datetime.datetime.combine(next\_day, datetime.time(hour=9, minute=30))    if not context.position:  if run\_at:   context.buy(data=context.data.\_name,price=context.data.close[0]\*1.1,exectype=Order.Market,run\_at=run\_at,misfire\_grace\_time=600)  else:  context.buy(data=context.data.\_name,price=context.data.close[0]\*1.1,exectype=Order.Market)   # 如果持仓，则判断是否出现卖出信号  else:  if run\_at:  context.sell(data=context.data.\_name,price=context.data.close[0]\*0.9,run\_at=run\_at,misfire\_grace\_time=600)  else:  context.sell(data=context.data.\_name,price=context.data.close[0]\*0.9) |

**注意！！**：定时订单是框架的一个定时任务，只是到了**指定的时间**去向交易所发起委托，因此，在执行context.buy或context.sell时，并不会预先冻结资金，也不会冻结仓位，即在定时订单真正执行之前，调用 get\_cash 或者 get\_value 函数时，结果是不变的，你需要自行处理计算仓位和现金的问题。