**4. 代码框架**

**代码框架有什么用？**

在正式课程中，想必你已经了解了该如何从零到一去编写一个策略了。

但对于量化新手而言，在开发策略时，还是会比较容易遇到问题。

比如：

|  |
| --- |
| 1. 当有了新想法时，却要花几个小时甚至几天的时间，实现从0-1的代码开发。 2. 当想灵活切换单标的或多标的做回测时，却因为代码结构问题，导致修改起来非常的麻烦。 3. 当想把测试好的代码搬上模拟交易或实盘交易时，却总是缺少一些功能要补充，比如符合交易所要求的T+1规则等。 4. ... |

为了避免以上问题，帮助你更高效地开发并应用策略到模拟交易及实盘交易中。我们给你提供了一个逻辑更为清晰、功能更加完善的**代码框架**。

|  |
| --- |
| 可以帮助你规避策略代码开发中的常见问题，把精力更多地聚焦在策略的核心逻辑上。   * 更快速地将**新想法转化为策略代码** * 更灵活、高效地**调试策略**，包括回测标的，策略信号及交易信号 * 可以**实现模拟与实盘交易**（比如时间、资金、价格风控要求、T+1功能、订单取消功能等） |

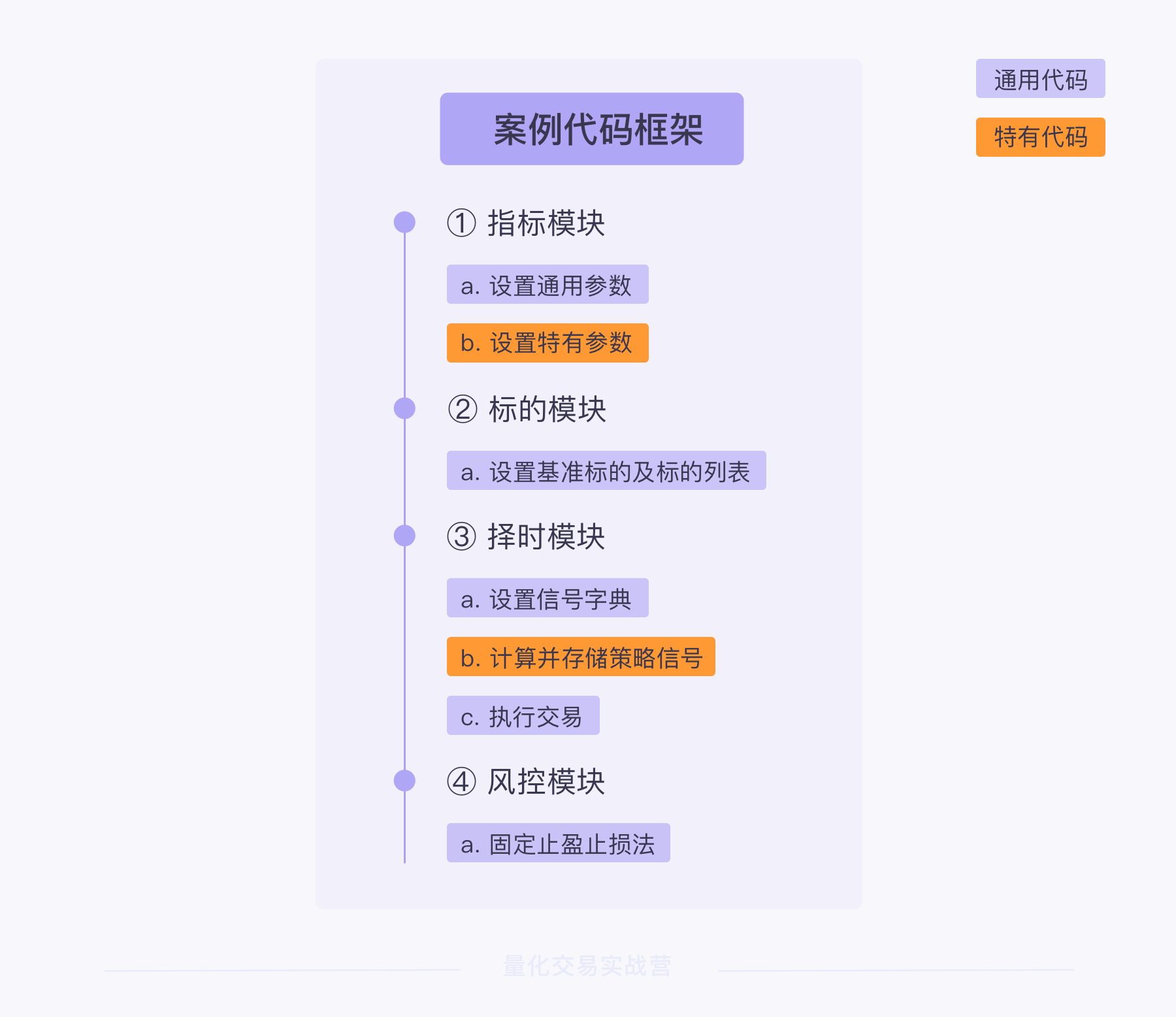
**代码框架是什么？**

|  |
| --- |
| 代码框架是一个提前写好的代码模板 |

**框架结构**

代码框架由两种代码块组成，它们分别是：

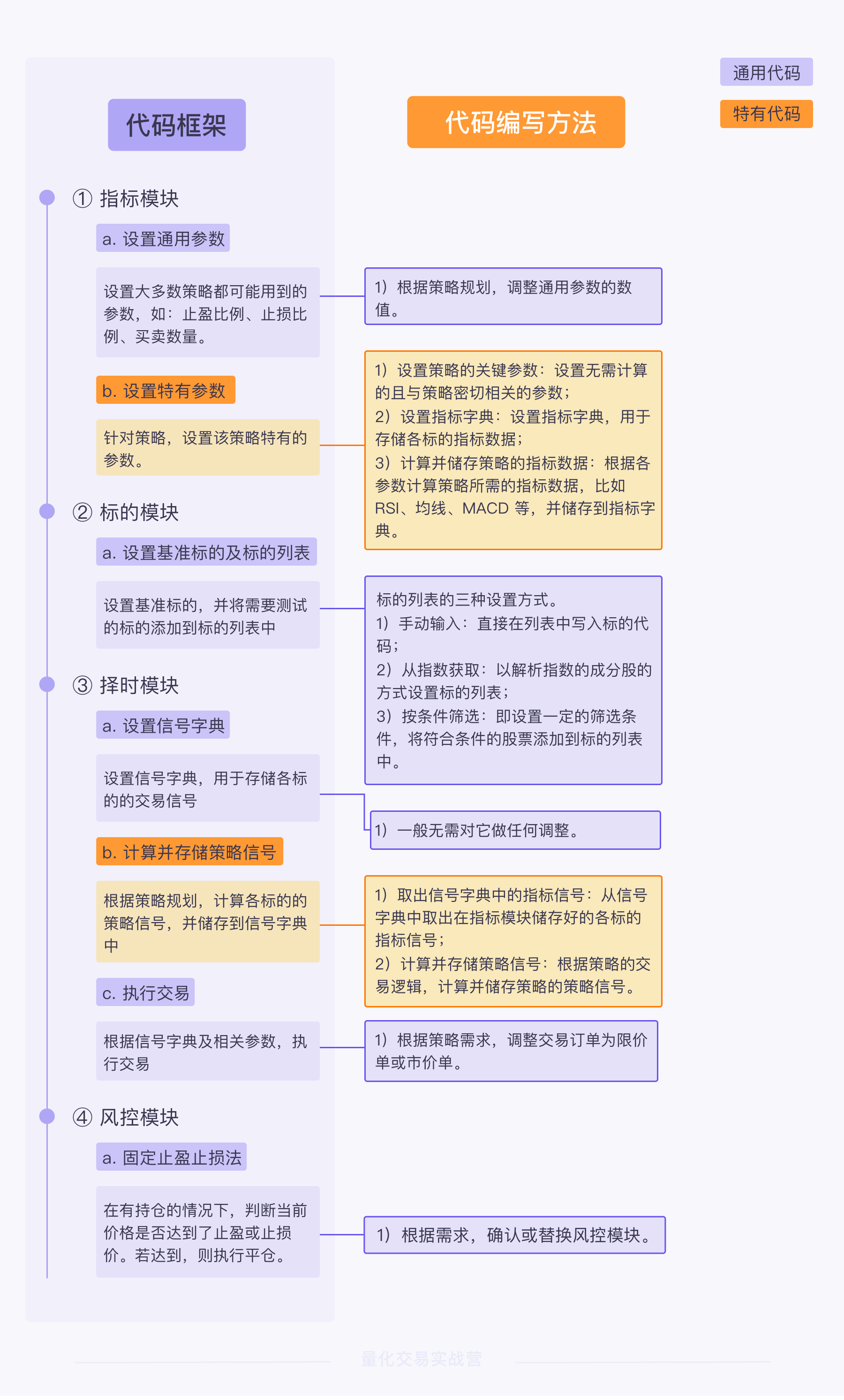
1. **通用代码**：任何交易策略几乎都可以用到的代码内容；
2. **特有代码：**不同交易策略各自特有的代码内容。



我将以 RSI 交易策略作为代示例代码，逐个模块来为你介绍代码框架的具体内容。

|  |
| --- |
| 注意：本文档不会具体讲解代码细节，如果你有疑问，可以复习课程中的讲解内容，或者寻求助教老师的帮助。 |

**编写方法**



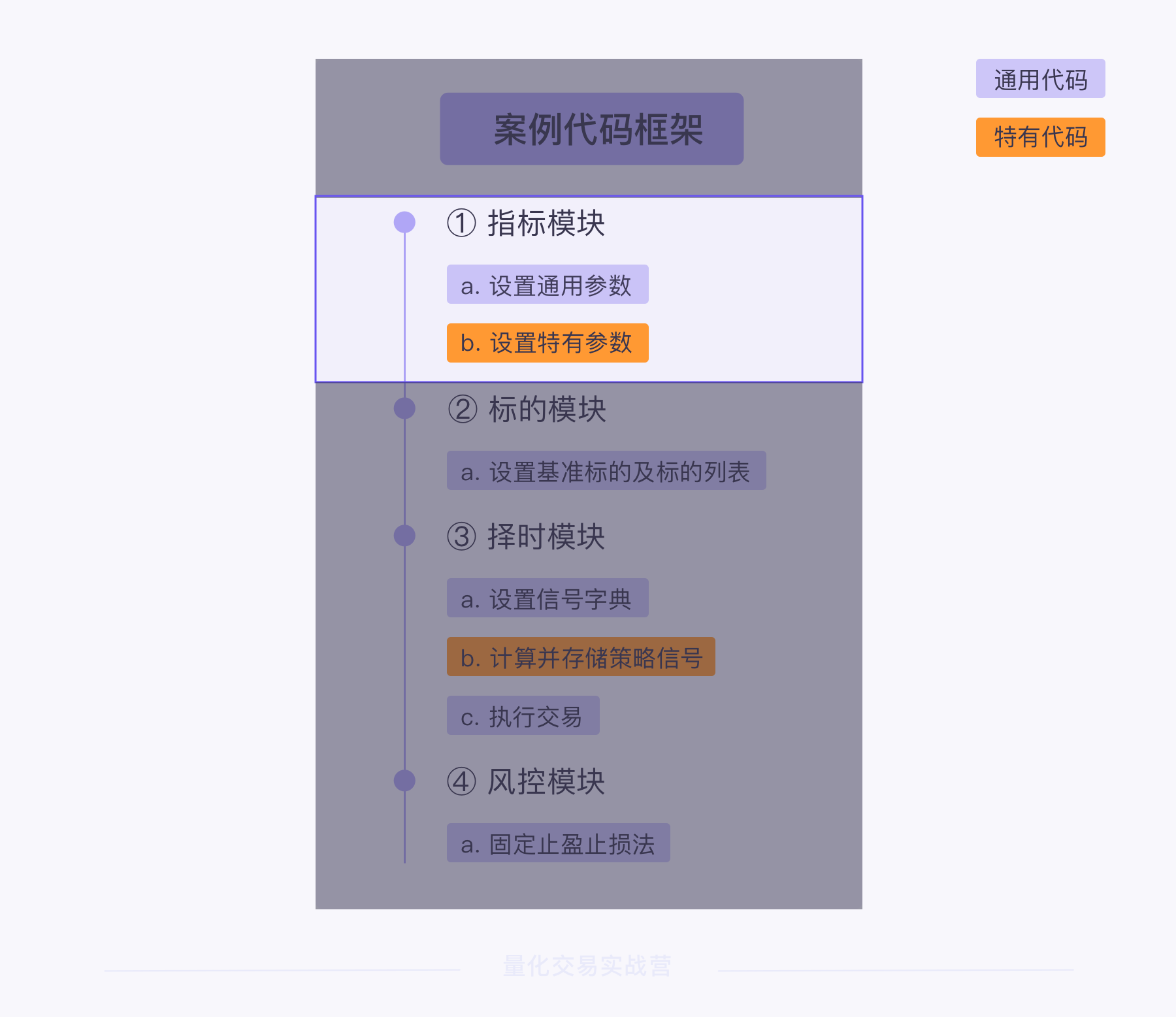
[**框架代码（点击查看完整代码）**](https://prod.pandateacher.com/web/fortrader-web/#/home?share_code=vNanyolVG4AOlgJY)

**指标模块**

**框架说明**

在指标模块中，代码框架有两个代码块（标紫的为通用代码、橙色的为特有代码）：

1. 设置通用参数：通用性参数是指无论是你现在的交易策略，还是其他交易策略都可以用到的参数，比如：买卖数量、止盈止损的比例。
2. 设置特有参数：指你当下的策略所特有的参数内容。如 RSI 交易策略中RSI 的买卖阈值。



**代码内容**

|  |
| --- |
| Python def indicators(context):  """指标"""  *# 设置每次买卖的数量*  context.trade\_size = 100  *# 设置止盈比例*  context.take\_profit = 0.15  *# 设置止损比例*  context.stop\_loss = 0.08   *# 以上为每个策略都会用到的常用参数设置部分*  *# 接下来针对具体的策略，可以计算一些针对性的参数*  *# ------------------------------------------------------------------*   *# 初始化RSI指标上限*  context.top = 70  *# 初始化RSI指标下限*  context.bot = 25  *# 设置指标字典*  context.rsi\_dict = {}  *# 设置RSI的计算周期*  rsi\_period = 6  *# 遍历所有标的*  for data in context.datas:  *# 计算周期为6的RSI指标*  rsi = RSI(data.close, period=rsi\_period, safediv=True)  *# 记录标的对应的RSI指标*   context.rsi\_dict[data] = {'rsi': rsi} |

1. 第 3 ~ 8 行标紫色的代码，为通用代码：设置通用参数。
2. 第 14 ~ 27 行标橙色的代码，为特有代码：设置特有参数。

**标的模块**

**框架说明**

在标的模块中，代码框架只有一个通用代码块：

1. 设置基准标的及标的列表：设置基准标的，并将需要测试的标的添加到标的列表中。



**代码内容**

|  |
| --- |
| Python def choose\_stock(context):  """选股"""  # 设置基准标的  context.benchmark = "000300.XSHG"  # 设置标的列表  context.symbol\_list = ["000300.XSHG"] |

1. 第 3 ~ 6 行标紫色的代码，为代码框架中的通用代码：设置基准标的及标的列表。

**择时模块**

**框架说明**

在择时模块中，代码框架一共有三个代码块：

1. 设置信号字典：设置一个信号字典，于存储各标的的策略信号。

|  |
| --- |
| “策略信号”是策略输出的交易判断，内容是：哪些标的该买入，哪些标的该卖出。 |

1. 计算并存储策略信号：根据根据策略的交易逻辑，计算策略中各标的的策略信号，并储存到信号字典中；
2. 执行交易：根据信号字典的内容执行交易。



**代码内容**

|  |
| --- |
| Python def timing(context):  """择时"""  # 设置交易字典  trade\_dict = {'需买入的标的对象': [], '需买入标的的代码': [], '需卖出的标的对象': [], '需卖出的标的代码': []}  # 遍历所有标的  for data in context.datas:  # 获取RSI  rsi = context.rsi\_dict[data]['rsi']   # 当RSI值低于其下限时  if rsi[0] < context.bot:  # 记录需要买入的标的对象  trade\_dict['需买入的标的对象'].append(data)  # 记录需要买入标的对象的代码  trade\_dict['需买入标的的代码'].append(data.\_name)  # 当RSI值高于其上限时  elif rsi[0] > context.top:  # 记录需要卖出的标的对象  trade\_dict['需卖出的标的对象'].append(data)  # 记录需要卖出标的对象的代码  trade\_dict['需卖出的标的代码'].append(data.\_name)   # 如果需买入标的的代码列表或需卖出的标的代码不为空：  if trade\_dict['需买入标的的代码'] or trade\_dict['需卖出的标的代码']:  # 打印列表中的信息  context.log('\n 需卖出的标的为：{} \n 需买入的标的为：{}'.format(trade\_dict['需卖出的标的代码'], trade\_dict['需买入标的的代码']))   # 择时到此，将输出所有标的需要操作的状态，新的策略只需要在这之前根据策略内容输出对应的状态即可  # 接下来执行交易   # 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']:  # 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) |

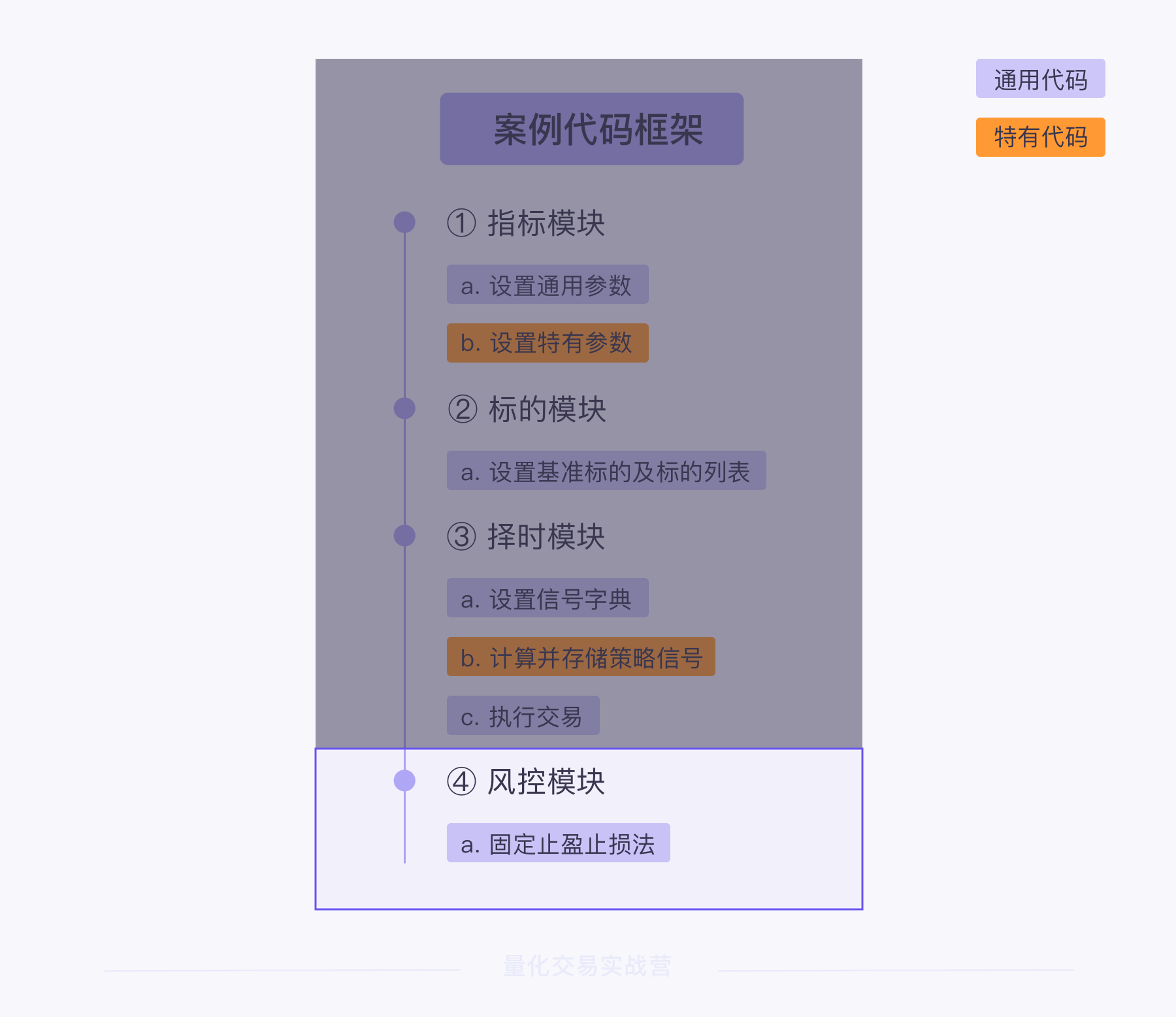
1. 第 3 ~ 4 行标紫色的代码，为通用代码：设置通用参数；
2. 第 5 ~ 21 行标橙色的代码，为特有代码：计算并储存策略信号；
3. 第 23 ~ 47 行代码标紫色的代码，为通用代码：执行交易。

**风控模块**

**框架说明**

在风控模块中，代码框架有一个代码块：

1. 固定止盈止损法：为代码框架在风控模块预设的风控方法固定止盈止损法。



**代码内容**

|  |
| --- |
| Python def control\_risk(context):  """风控"""  # 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") |

1. 第 3 ~ 20 行标紫色的代码，为通用代码：固定止盈止损法；

**代码框架怎么用？**

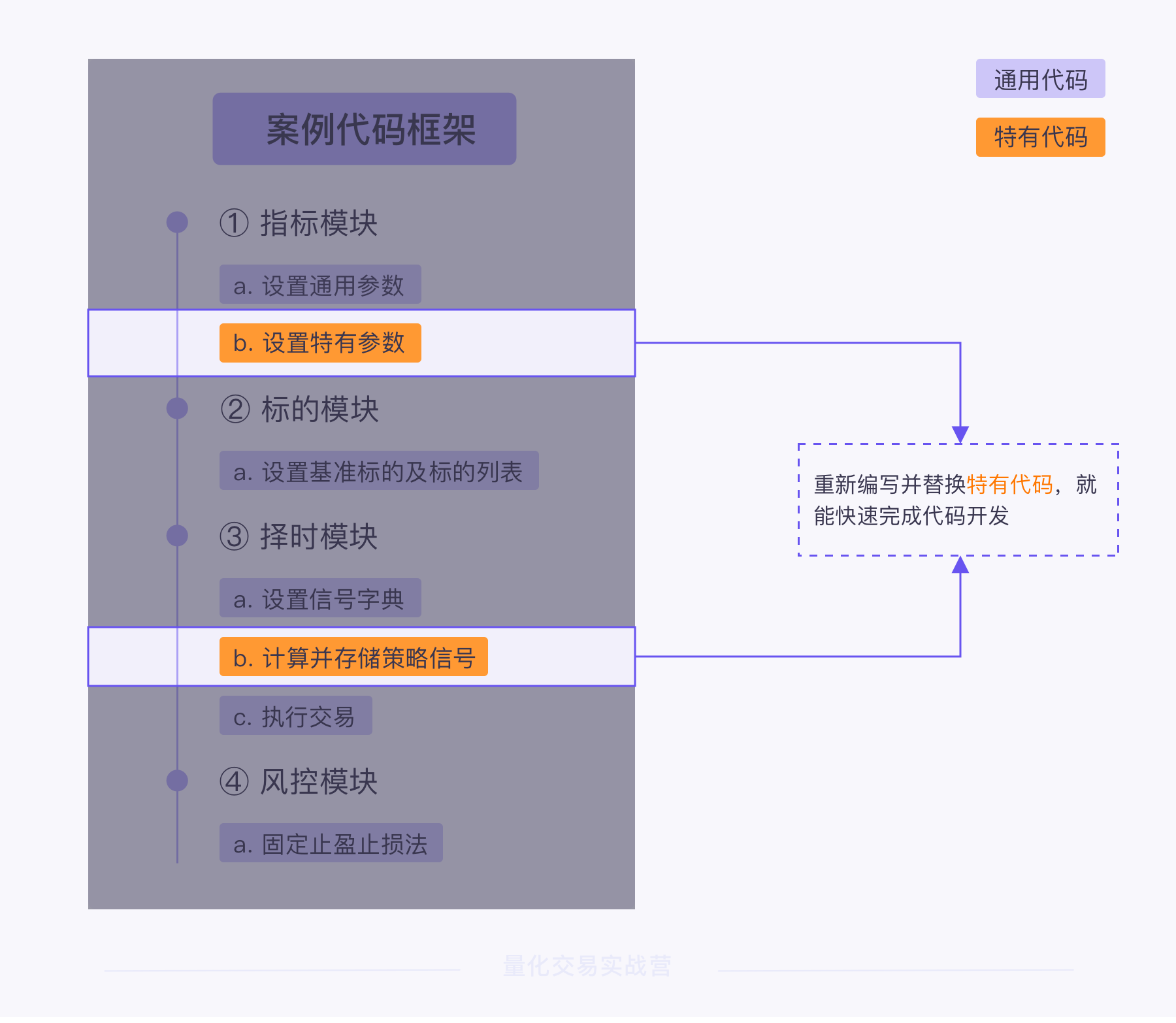
在具体应用代码框架时，有以下**三个角度**：

1. 代码开发
2. 策略调试
3. 策略优化/增加功能

**代码开发**

**使用说明**

|  |
| --- |
| 在用代码框架开发时，你只需要**新增或微调通用代码**，再根据代码框架的编写步骤**重新编写并替换特有代码**，以实现策略的快速落地。 |



**使用示例**

例如，你要开发单均线策略，那么你只需修改对于原代码框架（RSI 交易策略）中的**指标模块、**及**择时模块**的内容，修改后的结果如下（**标橙部分为改动的内容**）：

1. 指标模块

|  |
| --- |
| Python def indicators(context):  """指标"""  # 设置每次买卖的数量  context.trade\_size = 100  # 设置止盈比例  context.take\_profit = 0.15  # 设置止损比例  context.stop\_loss = 0.08   # 以上为每个策略都会用到的常用参数设置部分  # 接下来针对具体的策略，可以计算一些针对性的参数  # ------------------------------------------------------------------    # 设置指标字典  context.sma\_dict = {}  # 设置均线的计算周期  sma\_period = 15  # 遍历所有标的  for data in context.datas:  # 计算均线  sma = SMA(data.close, period=sma\_period)  # 记录标的对应均线  context.sma\_dict[data] = {'SMA': sma} |

1. 择时模块

|  |
| --- |
| Python def timing(context):  """择时"""  # 设置信号字典  trade\_dict = {'需买入的标的对象': [], '需买入标的的代码': [], '需卖出的标的对象': [], '需卖出的标的代码': []}  # 遍历所有标的  for data in context.datas:  # 获取均线  sma = context.sma\_dict[data]['SMA']   # 当标的收盘价上穿15日均价时  if data.close[-1] < sma[-1] and data.close[0] > sma[0]:  # 记录需要买入的标的对象  trade\_dict['需买入的标的对象'].append(data)  # 记录需要买入标的对象的代码  trade\_dict['需买入标的的代码'].append(data.\_name)  # 当标的收盘价下穿15日均价时  elif data.close[-1] > sma[-1] and data.close[0] < sma[0]:  # 记录需要卖出的标的对象  trade\_dict['需卖出的标的对象'].append(data)  # 记录需要卖出标的对象的代码  trade\_dict['需卖出的标的代码'].append(data.\_name)   # 如果需买入标的的代码列表或需卖出的标的代码不为空：  if trade\_dict['需买入标的的代码'] or trade\_dict['需卖出的标的代码']:  # 打印列表中的信息  context.log('\n 需卖出的标的为：{} \n 需买入的标的为：{}'.format(trade\_dict['需卖出的标的代码'], trade\_dict['需买入标的的代码']))   # 择时到此，将输出所有标的需要操作的状态，新的策略只需要在这之前根据策略内容输出对应的状态即可  # 接下来执行交易   # 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']:  # 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) |

**策略调试**

**使用说明**

在策略调试中，代码框架可以更快捷地完成**标的切换**及**策略排查**。

1. **标的切换**

|  |
| --- |
| 在代码框架中切换标的时，你只需**调整/重新设置标的列表**即可。 |

标的列表的设置方式包括：

1. 手动输入：直接在列表中写入标的代码；
2. 从指数获取：以解析指数的成分股的方式设置标的列表；
3. 按条件筛选：即设置一定的筛选条件，将符合条件的股票添加到标的列表中。
4. **策略排查**

代码框架对指标数据、策略信号及交易信号（实际执行的交易行为）做了区分。

其中：

1. **指标数据**及**策略信号**分别被保存在**指标字典**及**信号字典**中，**本身不参与交易的执行。**
2. **执行交易**的部分则**由一块独立代码块负责**。

|  |
| --- |
| 在用代码框架进行策略排查时，你可以**从终端的打印结果来区分是策略信号还是交易信号出了问题，**然后再对目标代码块进行修改。 |

**使用示例**

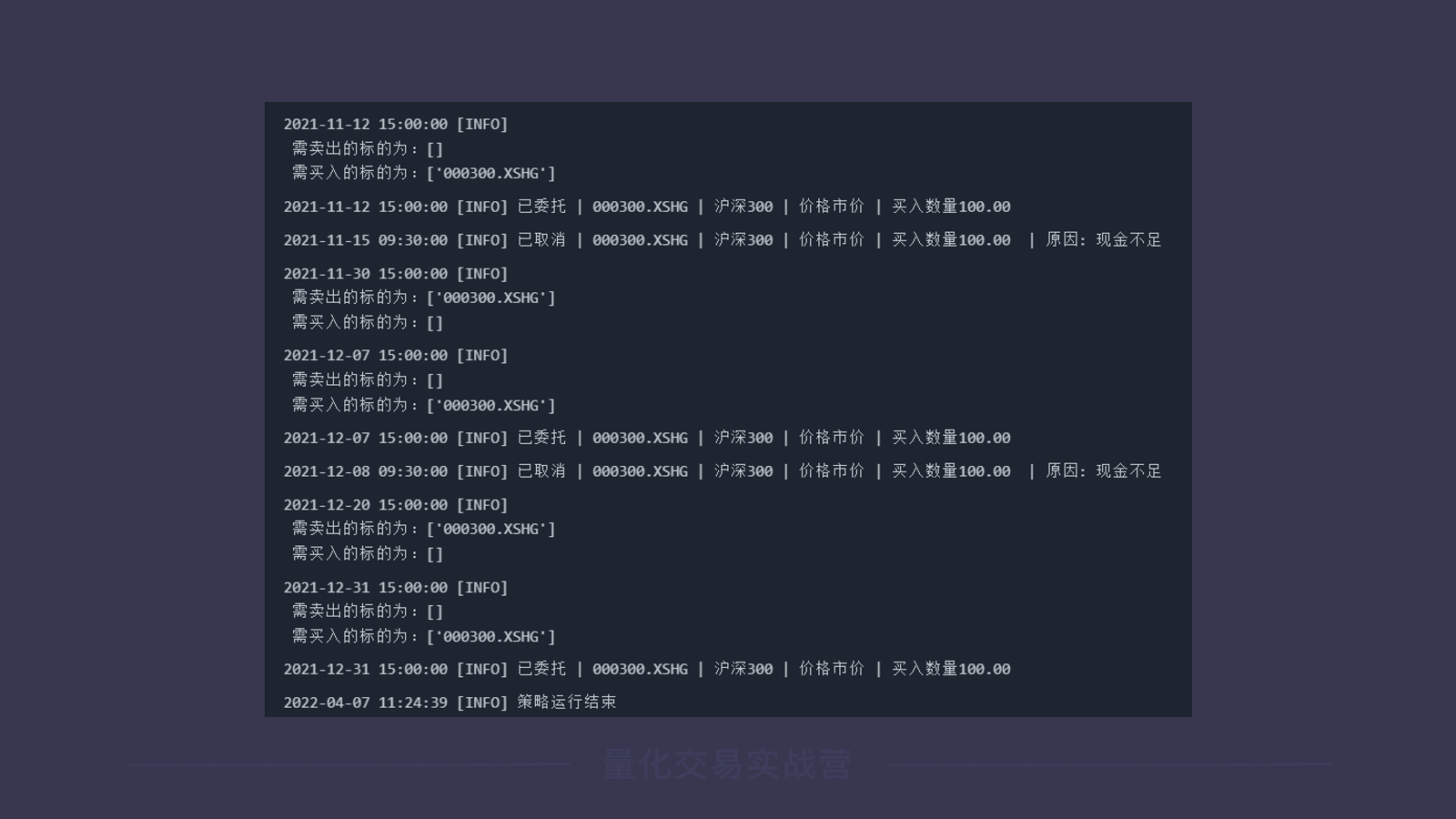
1. **标的切换**

在具体操作上，你可以在代码中手动输入设置基准标的及标的列表。

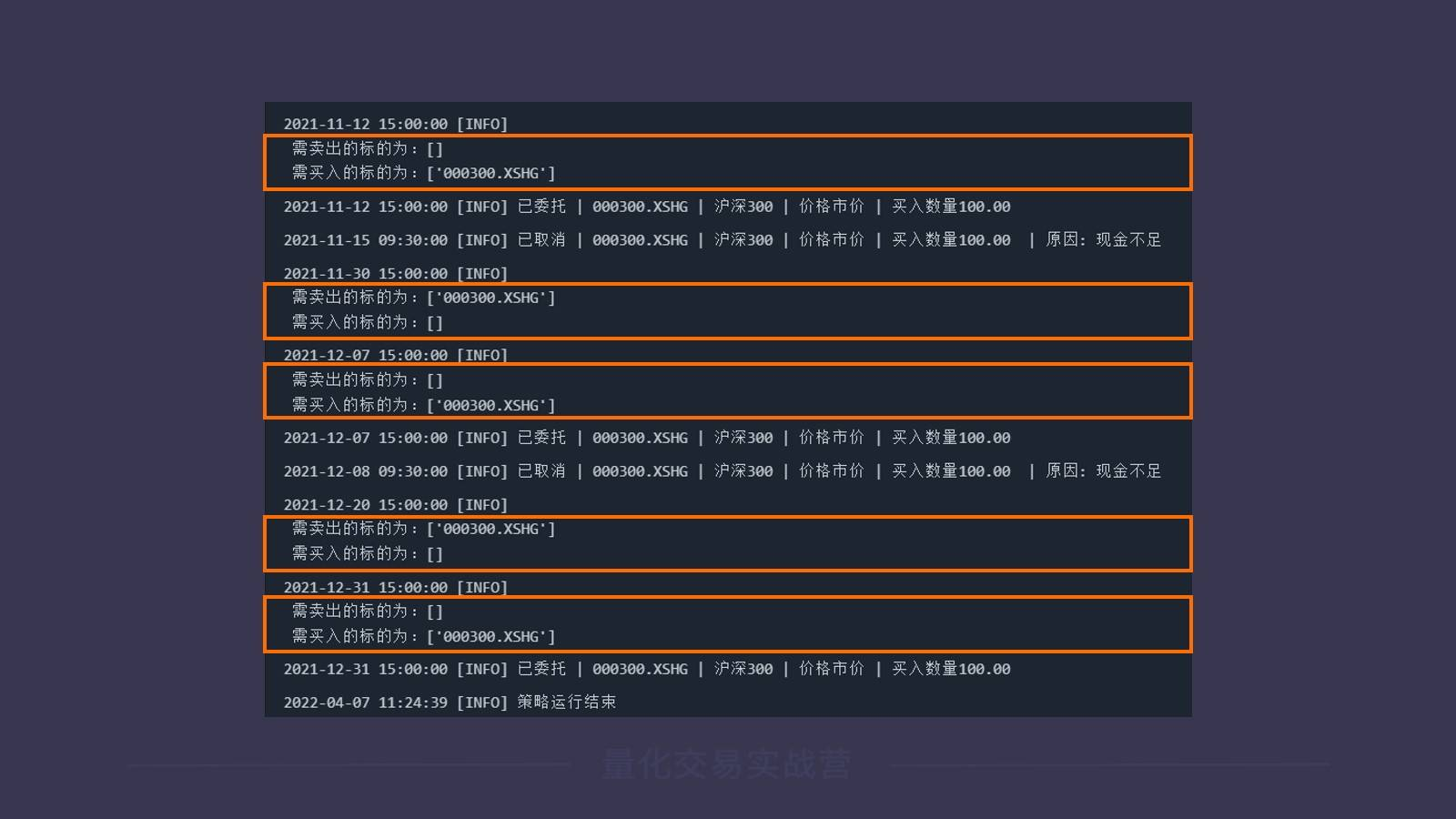
|  |
| --- |
| Python def choose\_stock(context):  """选股"""  # 设置基准标的  context.benchmark = "000300.XSHG"  # 设置标的列表  context.symbol\_list = ["000300.XSHG"] |

1. **策略排查**

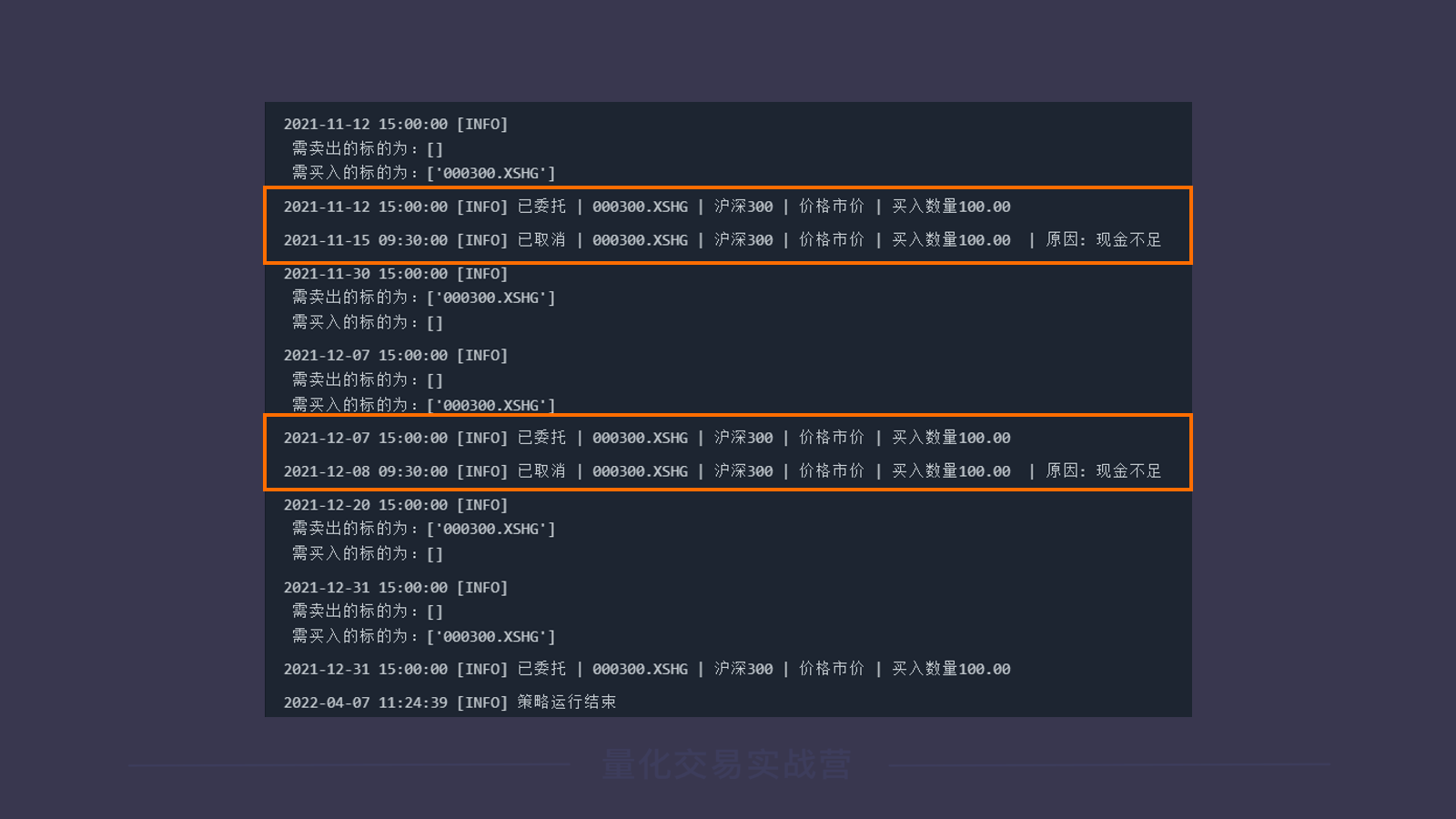
假如，你的策略出现成交失败的情况时，在终端看到的打印结果是这样的：



那么，你就可以先检查策略信号：



再检查交易信号：



来找到问题的根源。

在这个示例中，你可以清楚看到是交易信号因为现金不足出了问题。

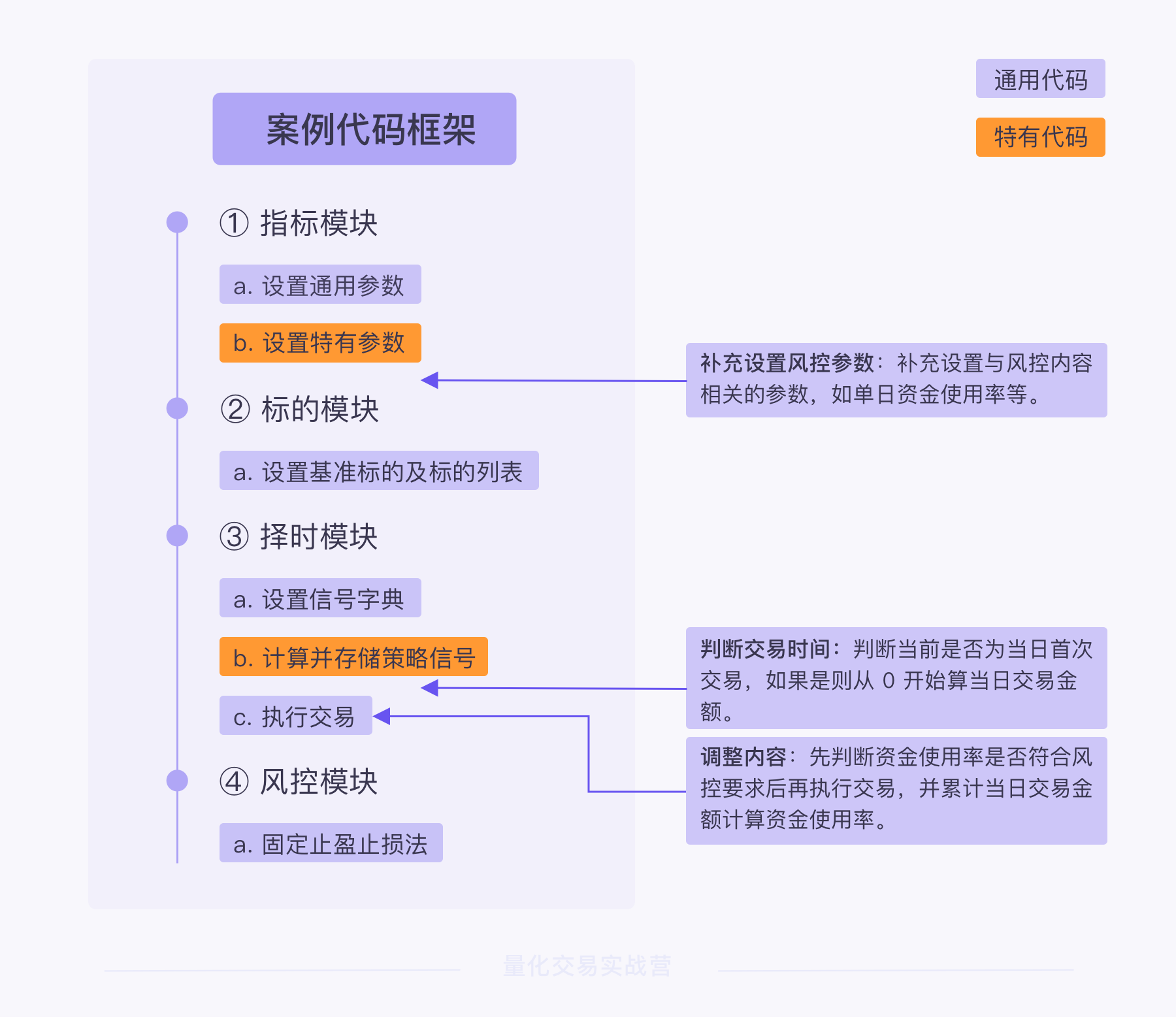
**策略优化/功能增加**

**使用说明**

|  |
| --- |
| **在原代码基础上新增并调整新的通用代码块**，你就可以实现代码框架的策略优化及功能增加。 |

**使用示例**

例如，我们想要在原代码基础上新增一个时间风控功能：**如果最近一次买入的时候与当前时间相比，超过了 1 天，则执行平仓。**



各部分具体新增及调整的代码为：

1. 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 |

1. 补充记录当前买入时的时间戳

|  |
| --- |
| Python # 记录当前买入时的时间戳 context.recently\_buy\_dict[buy\_data.\_name] = context.datetime.datetime().timestamp() |

1. 补充计算持仓天数，并设置风控逻辑

|  |
| --- |
| Python # 获取当前的时间戳 now\_timestamp = context.datetime.datetime().timestamp() # 过去最近一次买入的时间戳 recently\_timestamp = context.recently\_buy\_dict[data.\_name]  # 计算距离最近一次买入相隔的天数(一天86400秒) days = (now\_timestamp - recently\_timestamp) / 86400  # 如果当前持仓天数大于等于最长持仓天数 if days >= context.hold\_time:  # 执行平仓  context.close(data=data)  context.log("超过最长持仓天数，执行平仓") |

|  |
| --- |
| 其他代码功能的添加方法，你可以在本文档中的[功能代码集合](https://forchangesz.feishu.cn/wiki/wikcnOaW9UK2okdnvPpDR2a5jQf?sheet=5a930cf)的部分查看。 |