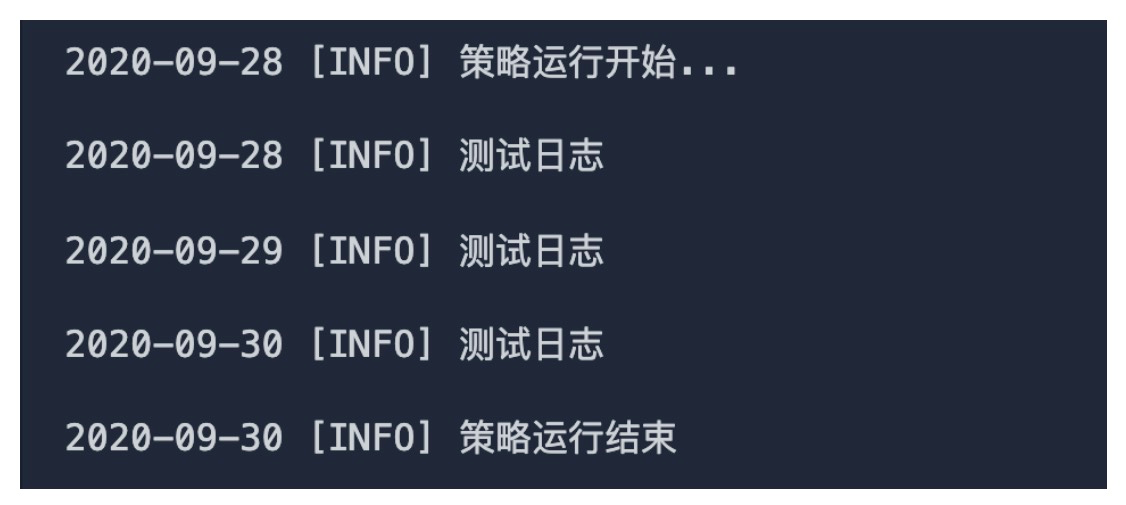
**1. 常用语法**

|  |
| --- |
| 为了更好地编写策略代码，我们需要了解 HiTrader 的语法功能以及其特性。 |

**打印日志**

* **语法**：context.log(想要打印的内容)
* **功能**：在 HiTrader 中，print() 函数是无效的，你必须通过日志功能来打印输出。在编写程序的过程中，少不了调试和 debug，打印日志必然是最常用的功能之一。

|  |
| --- |
| 日志会自动为我们拼接日期数据，如回测执行到具体某一天时，打印日志会在前面自动拼接当天日期，方便我们排查问题。 |



* **日志会拼接日期**这个特性导致普通的打印日志方法**无法**在**指标模块**运行，因为指标模块没有日期概念，如果需要要**指标模块打印日志**，可以加上 level='DEBUG' 参数。

|  |
| --- |
| Python context.log(想要打印的东西, level='DEBUG') |

**经纪商**

**获取账户总资产**

* **语法**：context.broker.getvalue()
* **功能**：返回当前账户中的总资产（总资产 = 账户现金 + 持仓标的总价值）
* **示例**：

|  |
| --- |
| Python def timing(context):  """择时"""  # 获取当前账户总资产  context.broker.getvalue() |

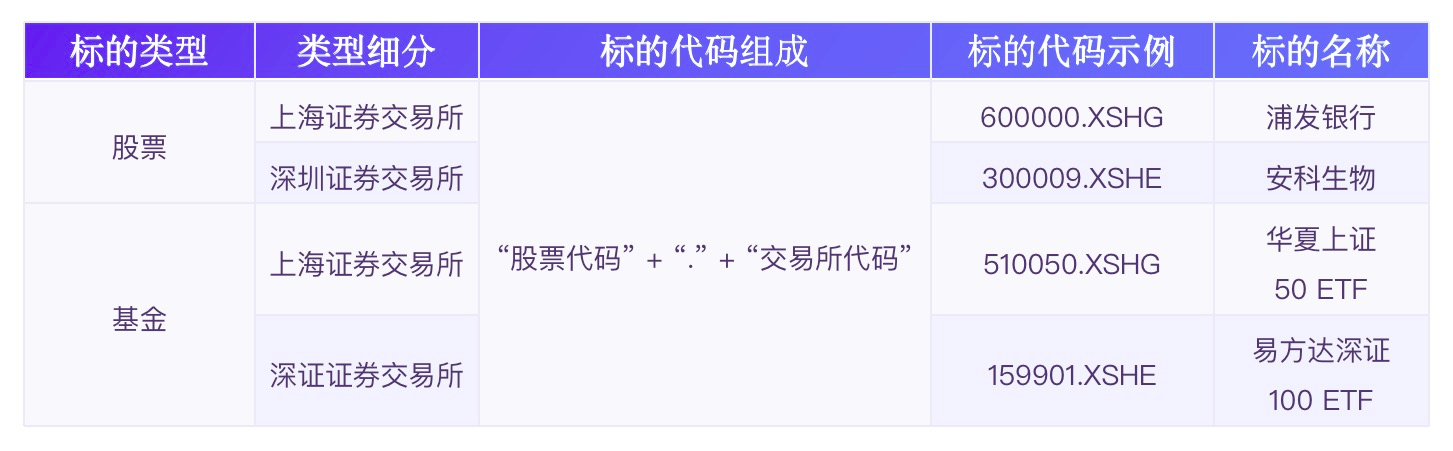
**获取账户现金**

* **语法**：context.broker.cash
* **功能**：返回当前账户中的现金总额。
* **示例**：

|  |
| --- |
| Python def timing(context):  """择时"""  # 获取当前账户中的现金总额  context.broker.cash |

**标的**

**标的类型**



**设置标的池**

* **语法**：context.symbol\_list
* **功能**：每个策略都必须在标的模块设置标的池，标的池是程序初始化各种数据的基础，不在标的池里的标的数据，在程序的运行过程中无法取到。
* **示例**：
* **设置单个标的**

|  |
| --- |
| Python def choose\_stock(context):  # 设置标的池  context.symbol\_list = ['600000.XSHG'] |

* **自定义多个标的**

|  |
| --- |
| Python def choose\_stock(context):  # 设置标的池  context.symbol\_list = ['600000.XSHG', '300131.XSHE', '600321.XSHG'] |

* **设置指数成分股**

当我们希望从某只指数里选股时，我们可以将这只**成分股**放进**标的池里**，然后将context.parse\_index 设置为 True，程序就会自动把指数的成分股进行**解析**。

|  |
| --- |
| Python def choose\_stock(context):  # 设置标的池为一只指数  context.symbol\_list = ['000300.XSHG']  # 打开解析成分股参数  context.parse\_index = True |

|  |
| --- |
| 虽然指数每过一段时间会**更新**成分股，但是程序固定以**回测结束时间**的成分股状态来填充标的池，无论你的回测区间跨越多少个变更周期，都不影响回测过程中的成分股。 |

**设置基准标的**

* **语法**：context.benchmark
* **功能**：用于计算“买入持有收益”：
* 标的池中只有一个标的时，基准标的默认是它自身；
* 标的池中大于一个标的时（包括解析成分股的情况），必须指定基准标的。
* **示例**：

|  |
| --- |
| Python def choose\_stock(context):  # 如果标的池中大于一只标的，则必须指定基准标的  context.symbol\_list = ['600000.XSHG', '600321.XSHG', '600863.XSHG']  # 设置基准标的  context.benchmark = '000300.XSHG' |

**标的数据对象**

* **语法**：FtData
* **功能**：标的数据对象在策略编写过程中非常常用，它存储了标的代码、标的历史行情数据等最常用的数据。另一方面，你在订单中指定标的时，也需要用到它。
* **示例**：（以下 data 指代存放“标的数据对象”的变量，实际使用时我们也经常使用 data 作为变量）
* **取标的代码**

|  |
| --- |
| Python # 通过 \_name 属性可以获得标的代码 data.\_name |

* 取标的**历史行情数据**

|  |
| --- |
| Python # 通过 close 属性可以取到收盘价 data.close[0]  # 也可以通过取历史数据的方法取到昨天的收盘价 data.close[-1] |

* **数据表格**：



|  |
| --- |
| 如果一只标的在某个交易日**停牌**了，那这一天它的开盘价、收盘价、最高价、最低价都会填充为前一天的数据，它的成交额、成交量会被填充为 0。 |

**设置默认标的**

* **语法**：context.data
* **功能**：当标的池里只有一个标的时，可以使用 context.data 属性直接取到该标的的数据对象。
* **示例**：

|  |
| --- |
| Python def choose\_stock(context):  # 标的池中只有一个标的  context.symbol\_list = ['600000.XSHG']    # 这个使用使用 context.data 即相当于取到了标的 600000.XSHG 的数据对象  context.data    # 结合标的数据对象的语法，可以取到标的 600000.XSHG 的收盘价等数据  context.data.close[0]  context.data.\_name |

|  |
| --- |
| 当标的池里的标的大于一个时，context.data 会取到池中代码**序号最小**的标的的数据对象。如标的池为 ['600000.XSHG', '000300.XSHG', '600321.XSHG'] ，则 context.data 会取到 '000300.XSHG' 的数据对象。不建议在多标的时使用默认标的属性。 |

**获取多个标的**

* **语法**：context.datas
* **功能**：context.datas 返回一个存着标的池所有标的数据对象的列表，配合 **for 循环**可以遍历所有标的。
* **示例**：

|  |
| --- |
| Python def choose\_stock(context):  # 设置基准标的  context.benchmark = '000300.XSHG'  # 标的池中有多个标的 或 解析了指数成分股  context.symbol\_list = ['600000.XSHG', '600123.XSHG', '600863.XSHG']    # for 循环遍历每一个标的数据对象  for data in context.datas:  # 取标的代码  data.\_name    # 取标的收盘价  data.close[0] |

|  |
| --- |
| context.datas 里只有！只有！只有**标的池里的标的**，不是整个市场的所有标的。 |

**获取指定标的**

* **语法**：context.getdatabyname(标的代码)
* **功能**：context.getdatabyname() 方法能让你通过标的代码直接取到标的数据对象。
* **示例**：

|  |
| --- |
| Python data = context.getdatabyname('600000.XSHG') |

这个方法在你**只有标的代码**，但是要取到标的数据对象时非常管用。

|  |
| --- |
| Python def choose\_stock(context):  # 设置基准标的  context.benchmark = '000300.XSHG'  # 设置标的池  context.symbol\_list = ['600000.XSHG', '600123.XSHG', '600863.XSHG']    # 选股逻辑，并将筛选出的标的代码存起来  context.stock\_list = []  for data in context.datas:  if data.close[0] > data.open[0]:  context.stock\_list.append(data.\_name)     def timing(context):  for name in context.stock\_list:  # 只有标的代码，所以我需要使用这个方法取出指定标的  data = context.getdatabyname(name)    # 正常使用这个标的数据对象  data.close[0] |

示例中的代码只是为了给你展示一下用法，其实你完全可以直接**将 data 存起来**，而不用再取一次。但是在你以后的编写代码过程中，你会遇到一些情况你不得不用标的代码来取数据对象。

|  |
| --- |
| context.getdatabyname() **只能取到标的池中存在的标的**，如果你特意输入一个不在标的池中的代码，程序会报错。 |

**排除指定标的**

* **语法**：
* context.exclude\_symbols=[标的代码]
* **功能**：额外排除指定标的，避免因为停牌导致的运行错误问题。
* **示例：**

|  |
| --- |
| Python def choose\_stock(context):  """标的"""  context.symbol\_list = ["600000.XSHG"]  # 排除指定停牌标的  context.exclude\_symbols=['000540.XSHE','600485.XSHG'] |

**填充停牌标的数据**

* **语法**：
* context.fill\_nan = True // 数据做填充
* context.fill\_nan = False // 数据不做填充
* **功能**：补充停牌期间缺失的数据。填充后，停牌的开盘价、最高价、最低价和收盘价都为最近一个交易日的收盘价，成交量为0

|  |
| --- |
| 系统说明：考虑到获取多只股票的数据时，可能有的股票停牌，有的没有。所以为了保持时间轴的一致，Hitrader 默认处于填充状态，即context.fill\_nan = True 。 |

* **示例：**

|  |
| --- |
| Python def choose\_stock(context):  """标的"""  context.symbol\_list = ["600000.XSHG"]  # 填充停牌日期  context.fill\_nan = True |

**获取日期**

* **语法**：data.datetime.date()
* **功能**：可以获取当前的日期，并且，我们可以通过在括号中填写序号的方式来获取历史日期。
* **示例**：

|  |
| --- |
| Python def choose\_stock(context):  """标的"""  context.symbol\_list = ["600000.XSHG"]   # 获取当前的日期  context.data.datetime.date()  # 获取上一个交易日的日期  context.data.datetime.date(-1)  # 获取上上一个交易日的日期  context.data.datetime.date(-2) |

**持仓**

**持仓数据对象**

* **语法**：position
* **功能**：择时逻辑和风控逻辑经常会将标的的持仓情况作为计算条件。
* **示例**：

|  |
| --- |
| 以下 position 指代存放“持仓数据对象”的**变量**，实际使用时我们也经常使用 position 作为变量 |

* **判断标的是否持仓**

|  |
| --- |
| Python # 如果持仓则做点什么 if position:  ...   # 如果不持仓则做点什么 if not position:  ... |

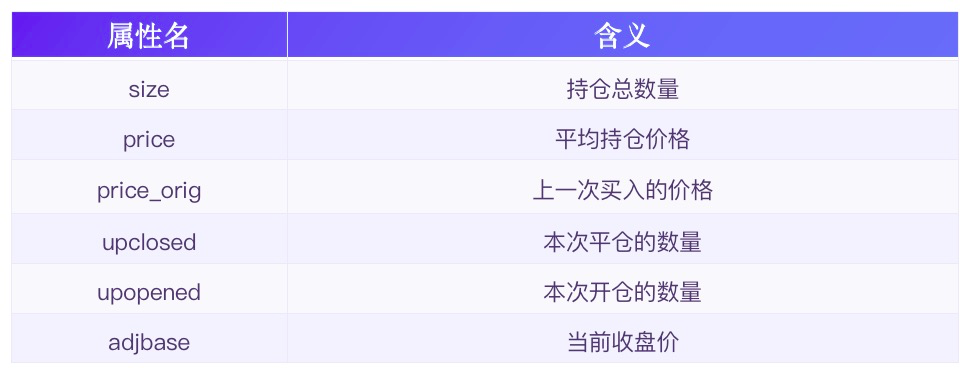
当标的未持仓时，position 的布尔值 为 False，所以我们可以直接使用 position 对象判断标的是否持仓。

* **获取持仓数据**

|  |
| --- |
| Python # 获取持仓数量 position.size  # 获取持仓平均价格 position.price |

当然，你也可以将判断是否持仓的代码写的更直接一点：if position.size == 0:。

* **数据表格**：



**默认标的持仓**

* **语法**：context.position
* **功能**：和 context.data 一样，当标的池中只有一只标的时，你可以使用 context.position 直接取得该标的的持仓数据。存在多标的时，不建议你使用该属性。
* **示例**：

|  |
| --- |
| Python def choose\_stock(context):  # 标的池中只有一个标的  context.symbol\_list = ['600000.XSHG']    # 这个使用 context.position 即相当于取到了标的 600000.XSHG 的持仓数据对象  context.position    # 结合持仓数据对象的语法，可以取到标的 600000.XSHG 的持仓数量、持仓价格等数据  context.position.size  context.position.price |

**获取指定标的持仓数据**

* **语法**：context.getposition(data, side=“long”)
* **功能**：因为持仓数据并没有存放在标的数据内，当存在多个标的时，为了获取每一个标的的持仓数据，就需要用到 context.getposition() 方法来获取指定标的持仓数据。查询参数：
* data: [标的数据对象](https://forchangesz.feishu.cn/docs/doccnLv17GXFHZZlGbfCIzTNLjM#QWeegh)，待查询的标的
* side: 字符串类型，查询的持仓方向，当不设置时，默认是为`long`，即多仓，可选：
* long: 多仓
* short: 空仓
* **示例**：

|  |
| --- |
| Python def timing(context):  # 遍历每个标的数据  for data in context.datas:  # 获取标的持仓数据  position = context.getposition(data)    # 判断是否已持仓  if position:  ... |

**订单**

由于回测的交易频率最小为 1 天，计算时使用的是每日的收盘数据，所以订单会在下一个交易日开始时以开盘价买入/卖出。

**获取当日委托单**

* **语法**：context.get\_orders(status="all")
* status: 字符串类型，订单状态类型，可选如下类型：
* all: 当日全部订单, 当不传递参数时，默认为all
* submitted: 当日全部未结委托单
* completed: 当日部分成交或全部成交订单
* canceled：当日已取消订单
* **功能**：查询当日的全部委托订单列表，列表中每个元素都是订单对象
* **示例**：
* 查询当日全部订单

|  |
| --- |
| Python def choose\_stock(context):     # 设置标的池     context.symbol\_list = ['600000.XSHG']  def timing(context):  # 买入订单  context.buy(size=100)    # 查询当日全部订单列表  orders = context.get\_orders(status="all") |

* 查询当日未结订单，并对订单进行撤单

|  |
| --- |
| Python def choose\_stock(context):     # 设置标的池     context.symbol\_list = ['600000.XSHG']  def timing(context):  # 查询当日全部未结订单列表  orders = context.get\_orders(status="submitted")  for order in orders:  context.cancel(order)    # 买入订单  context.buy(size=100) |

|  |
| --- |
| * 每天收市后下的订单在第二天依然会生效，在第二天查询到的当日委托单依然会包含这些订单 |

**订单对象**

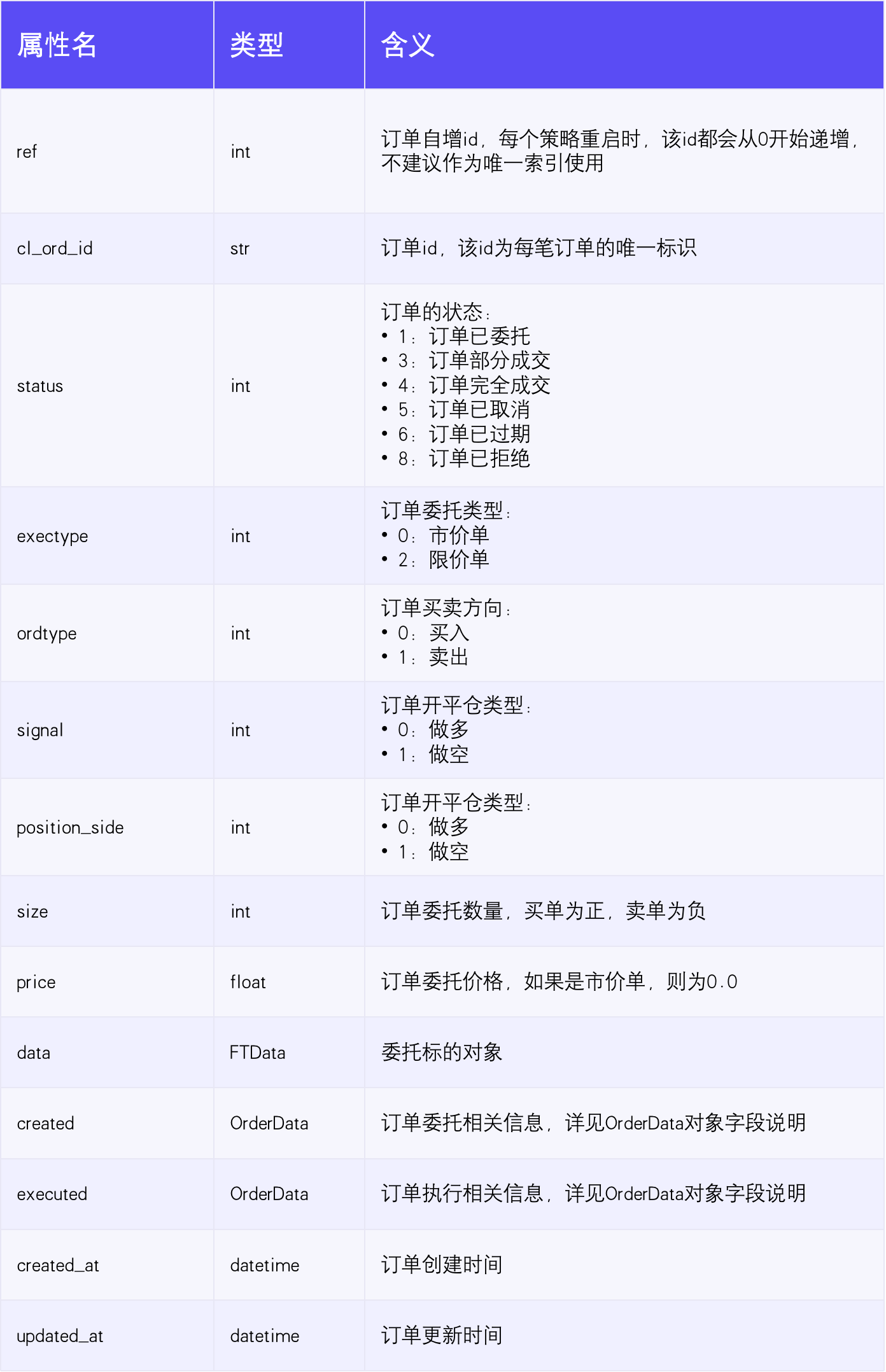
* **语法**：Order
* **功能**：订单对象，它存储了订单的委托、成交等相关信息。
* **示例**：
* 获取订单的状态

|  |
| --- |
| Python order.status |

* 获取订单的标的名称

|  |
| --- |
| Python order.data.\_name |

* **数据表格**：



**订单数据对象**

* **语法**：OrderData
* **功能**：订单数据对象，它存储了订单的委托或者成交等信息。一般是作为Order对象的属性，不会单独使用
* **示例**：
* 获取订单委托的价格

|  |
| --- |
| Python order.created.price |

* 获取订单成交时的佣金

|  |
| --- |
| Python order.executed.comm |

* **数据表格**：



**买入订单**

* **语法**：
* context.buy()
* **功能**：生成一个买入订单，用于择时和风控模块。
* **示例**：
* **不传递任何参数**，默认以【条件设置】内的订单数量买入 context.data。

|  |
| --- |
| Python def choose\_stock(context):  # 设置标的池  context.symbol\_list = ['600000.XSHG']  def timing(context):  # 假设条件设置内订单数量为 300，则生成买入 600000.XSHG 300 股的订单  context.buy() |

* **参数 data 指定买入标的**，需要传入一个标的数据对象。

|  |
| --- |
| Python def choose\_stock(context):  # 设置基准标的  context.benchmark = '000300.XSHG'  # 设置标的池  context.symbol\_list = ['600000.XSHG', '600321.XSHG', '600831.XSHG']  def timing(context):  # 获取指定的标的数据对象  data = context.getdatabyname('600321.XSHG')    # 假设条件设置内订单数量为 300，因为指定了标的 '600321.XSHG'  # 所以生成买入 '600321.XSHG' 300 股的订单  context.buy(data=data) |

* **参数 size 指定订单数量**。

|  |
| --- |
| Python def choose\_stock(context):  # 设置基准标的  context.benchmark = '000300.XSHG'  # 设置标的池  context.symbol\_list = ['600000.XSHG', '600321.XSHG', '600831.XSHG']  def timing(context):  # 获取指定的标的数据对象  data = context.getdatabyname('600321.XSHG')    # 假设条件设置内订单数量为 300，因为指定了标的 600321.XSHG 和订单数量 100 股  # 所以生成买入 600321.XSHG 100 股的订单。  # 保护价设置为 8 个点的涨幅，表示市价交易时，如果当前价格在 8% 涨幅以内，可以撮合成交  # 超过 %8，则不进行市价撮合  context.buy(data=data, size=100, price=data.close[0] \* 1.08) |

* **参数 exectype 指定订单类型。**默认为市价单，可选：
* Order.Market：市价单，根据新规，**沪市** 必须显式指定price，该参数表示为保护价，为你能接受的买入或卖出价格。买入时保护价上限为涨停价，卖出时下限为跌停价。如日线买入交易中，price 可设置为 data.close[0] \* 1.1 表示涨停价（此例为涨幅限制为10%的情形），日内高频交易，请自行处理价格问题
* Order.Close：收盘价单
* Order.Limit：限价单，必须显示指定price
* Order.Stop：止损单，必须显示指定price，price为 止损触发价格，触发订单按市价成交

|  |
| --- |
| Python def choose\_stock(context):  # 设置基准标的  context.benchmark = '000300.XSHG'  # 设置标的池  context.symbol\_list = ['600000.XSHG', '600321.XSHG', '600831.XSHG']  def timing(context):  # 获取指定的标的数据对象  data = context.getdatabyname('600321.XSHG')  # 以 11.27 的价格挂单买入  context.buy( price=11.27, exectype=Order.Limit) |

|  |
| --- |
| **订单创建成功不代表交易成功**，比如说你的钱不够完成订单，那订单会被拒绝掉。所以在涉及到复杂的订单买入时，要注意设计好订单数量的计算逻辑。 |

* **参数signal指定仓位方向。**默认为开仓，可选：
* open: 开仓
* close: 平仓，优先平昨仓，仅支持期货
* close\_today: 平今仓，优先平今仓，仅支持期货（注意：对不支持平今仓的交易所无效）

|  |
| --- |
| Python def choose\_stock(context):  # 设置标的池  context.symbol\_list = ['600000.XSHG']  def timing(context):  # 获取指定的标的数据对象  data = context.getdatabyname('600000.XSHG')    # signal默认为开仓，如果是期货，对做空的仓位进行部分平仓，signal可以设置为“close”或者“close\_today”  context.buy(data=data, size=100, signal="open") |

* **参数 run\_at 指定执行时间。datetime类型，默认为 None，可选**
* **参数 misfire\_grace\_time 指定过期时间间隔。默认 600 秒，可选**
* 详见功能代码集合定时订单

**卖出订单**

* **语法**：context.sell()
* **功能**：生成一个卖出订单，用于择时和风控模块。
* **示例**：
* **不传递任何参数**，默认以【条件设置】内的订单数量卖出 context.data。

|  |
| --- |
| Python def choose\_stock(context):  # 设置标的池  context.symbol\_list = ['600000.XSHG']  def timing(context):  # 假设条件设置内订单数量为 300，则生成卖出 600000.XSHG 300 股的订单  context.sell() |

* **参数 data 指定卖出标的**，需要传入一个标的数据对象。

|  |
| --- |
| Python def choose\_stock(context):  # 设置基准标的  context.benchmark = '000300.XSHG'  # 设置标的池  context.symbol\_list = ['600000.XSHG', '600321.XSHG', '600831.XSHG']  def timing(context):  # 获取指定的标的数据对象  data = context.getdatabyname('600321.XSHG')    # 假设条件设置内订单数量为 300，因为指定了标的 600321.XSHG  # 所以生成卖出 600321.XSHG 300 股的订单  context.sell(data=data) |

* **参数 size 指定订单数量**

|  |
| --- |
| Python def choose\_stock(context):  # 设置基准标的  context.benchmark = '000300.XSHG'  # 设置标的池  context.symbol\_list = ['600000.XSHG', '600321.XSHG', '600831.XSHG']  def timing(context):  # 获取指定的标的数据对象  data = context.getdatabyname('600321.XSHG')    # 假设条件设置内订单数量为 300，因为指定了标的 600321.XSHG 和订单数量 100 股  # 所以生成卖出 600321.XSHG 100 股的订单  context.sell(data=data, size=100) |

|  |
| --- |
| HiTrader 支持做空操作（A 股除外），所以即使你的订单数量不足，你仍然可以卖出更多的订单数量。比如说，你只有 200 股，但是你可以卖出 400 股。这个特性同样要求你设计好订单卖出数量的计算逻辑，避免出现意料之外的情况。 |

* **参数signal指定仓位方向。**默认为平仓close，可选：
* open: 开仓, 仅支持期货
* close: 平仓，优先平昨仓
* close\_today: 平今仓，优先平今仓（注意：对不支持平今仓的交易所无效）

|  |
| --- |
| Python def choose\_stock(context):  # 设置标的池  context.symbol\_list = ['600000.XSHG']  def timing(context):  # 获取指定的标的数据对象  data = context.getdatabyname('600000.XSHG')    # signal默认为开仓，如果是期货，对做空的仓位进行部分平仓，signal可以设置为“close”或者“close\_today”  context.sell(data=data, size=100, signal="close") |

**参数 run\_at 指定执行时间。datetime类型，默认为 None，可选**

* **参数 misfire\_grace\_time 指定过期时间间隔。默认 600 秒，可选**
* 详见功能代码集合定时订单

**平仓订单**

* **语法**：context.close()
* **功能**：生成一个平仓订单，用于择时和风控模块。平仓订单与卖出订单的功能几乎完全一致，除了不需要你指定订单数量。
* **示例**：

|  |
| --- |
| Python  def choose\_stock(context):  # 设置标的池  context.symbol\_list = ['600000.XSHG']  def timing(context):  # 生成 600000.XSHG 的平仓订单  context.close() |

* **参数 data 指定平仓标的**，需要传入一个标的数据对象。

|  |
| --- |
| Python def choose\_stock(context):  # 设置基准标的  context.benchmark = '000300.XSHG'  # 设置标的池  context.symbol\_list = ['600000.XSHG', '600321.XSHG', '600831.XSHG']  def timing(context):  # 获取指定的标的数据对象  data = context.getdatabyname('600321.XSHG')    # 因为指定了标的 600321.XSHG，所以生成了 600321.XSHG 的平仓订单  context.close(data=data) |

* **参数side指定平仓方向**，默认为多仓long，可选：
* long: 多仓
* short: 空仓，仅支持期货

|  |
| --- |
| Python def choose\_stock(context):  # 设置基准标的  context.benchmark = '000300.XSHG'  # 设置标的池  context.symbol\_list = ['600000.XSHG', '600321.XSHG', '600831.XSHG']  def timing(context):  # 获取指定的标的数据对象  data = context.getdatabyname('600321.XSHG')    # 因为指定了标的 600321.XSHG，所以生成了 600321.XSHG 的平仓订单  context.close(data=data, side="long") |

**调整仓位**

* **语法**：context.order\_target\_percent(data, target)
* data：指定标的，需要传入一个标的数据对象 context.data
* target：用于设置目标仓位比例，如 0.9 即是将仓位调整至 90%
* **功能**：按照一定的持仓比例调整标的仓位。
* **示例**：
* 将标的 159931.XSHE 的**仓位调整到 90%**

|  |
| --- |
| Python def choose\_stock(context):     # 设置标的池     context.symbol\_list = ['159931.XSHE']  def timing(context):  # 调整 159931.XSHE 仓位  context.order\_target\_percent(data=context.data, target=0.9) |

* 当标的**未持仓**时，代码将计算持仓 90% 需要买入的数量，并生成对应的买入订单。
* 当标的**已持仓**
* 如果当前仓位小于 90%，代码将计算出距离持仓 90% **差多少数量**，并生成新的买入订单。
* 如果当前仓位大于 90%，代码将计算出比 90% **多了多少**数量，并将多余的订单卖出。

|  |
| --- |
| * 调仓操作使用的是**账户总资产**来计算调仓数量，并**不会考虑你的余额是否足够**。所以在多个标的、复杂的策略逻辑中使用调仓操作时，我们需要谨慎设计计算逻辑，否则容易出现仓位不足的情况。 * 对于股票，如果调整后需要买卖的数量不是100的整数，系统会自动按100向下取整，比如当买入订单数量为252时，则按照200买入 |

**滑点**

**设置百分比滑点**

* **语法**：context.slip\_perc
* **功能**：设置百分比滑点，在回测时使用，如果设置了百分比滑点，则在下一根k线时，会按照开盘价的百分比进行滑点成交，比如：
* 买入订单时：成交的价格为下一根k线的开盘价 \* (1 + context.slip\_perc)
* 卖出订单时：成交的价格为下一根k线的开盘价 \* (1 - context.slip\_perc)
* **示例**：

|  |
| --- |
| Python def indicators(context):  """指标"""  # 计算15日的均价，赋值给变量context.sma  context.sma = SMA(period=15)  context.slip\_perc = 0.005 # 设置百分比滑点  def choose\_stock(context):  """标的"""  context.symbol\_list = ["600000.XSHG"]  def timing(context):  """择时"""  # 判断是否持仓，如果不持仓，则判断是否出现买入信号  if not context.position:  # 当股票收盘价上升并且交叉穿过15日均价时，出现买入信号  if context.data.close[-1] < context.sma[-1] and context.data.close[0] > context.sma[0]:  # 买入信号出现时，发送买入指令，系统自动执行买入交易  context.order = context.buy()   # 如果持仓，则判断是否出现卖出信号  else:  # 当股票收盘价小于15日均价时，出现卖出信号  if context.data.close[-1] > context.sma[-1] and context.data.close[0] < context.sma[0]:  # 卖出信号出现时，发送卖出指令，系统自动执行卖出交易  context.order = context.sell() |

**设置绝对值滑点**

* **语法**：context.slip\_fixed
* **功能**：设置绝对值滑点，在回测时使用，如果设置了绝对值滑点，则在下一根k线时，会按照绝对值进行滑点成交，比如：
* 买入订单时：成交的价格为下一根k线的开盘价 + context.slip\_fixed
* 卖出订单时：成交的价格为下一根k线的开盘价 - context.slip\_fixed
* **示例**：

|  |
| --- |
| Python def indicators(context):  """指标"""  # 计算15日的均价，赋值给变量context.sma  context.sma = SMA(period=15)  context.slip\_fixed = 0.01 # 设置绝对值滑点  def choose\_stock(context):  """标的"""  context.symbol\_list = ["600000.XSHG"]  def timing(context):  """择时"""  # 判断是否持仓，如果不持仓，则判断是否出现买入信号  if not context.position:  # 当股票收盘价上升并且交叉穿过15日均价时，出现买入信号  if context.data.close[-1] < context.sma[-1] and context.data.close[0] > context.sma[0]:  # 买入信号出现时，发送买入指令，系统自动执行买入交易  context.order = context.buy()   # 如果持仓，则判断是否出现卖出信号  else:  # 当股票收盘价小于15日均价时，出现卖出信号  if context.data.close[-1] > context.sma[-1] and context.data.close[0] < context.sma[0]:  # 卖出信号出现时，发送卖出指令，系统自动执行卖出交易  context.order = context.sell() |

|  |
| --- |
| * 如果滑点后的价格超过了k线的最高价和最低价，则在会自动取最高价或最低价进行成交 * 一般只需设置其中一种滑点即可，如果两个同时设置，则默认只配置context.slip\_perc |

**数据处理**

**And 逻辑与**

* 函数名：And
* 中文名称：逻辑与
* 简介：
* 该运算符表示并且的意思，同时为真才返回真，否则返回假。
* 计算公式：——
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.signal = And(context.data.open(0) > context.data.open(-1),  context.data.close(0) > context.data.close(-1)) |

* 参数：
* 无参数
* 返回
* signal：1.0代表真；0.0代表假。

**Or 逻辑或**

* 函数名：Or
* 中文名称：逻辑或
* 简介：
* 该运算符表示或者的意思，同时为假才返回假，否则返回真。
* 计算公式：——
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.signal = Or(context.data.open(0) > context.data.open(-1),  context.data.close(0) > context.data.close(-1)) |

* 参数：
* 无参数
* 返回
* signal：1.0代表真；0.0代表假。

**Max 最大值**

* 函数名：Max
* 中文名称：最大值
* 简介：
* 该函数返回历史数据中的最大值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.max\_close = Max(context.data.close) |

* 参数：
* 无参数
* 返回
* max\_close：最大值

**Min 最小值**

* 函数名：Min
* 中文名称：最小值
* 简介：
* 该函数返回历史数据中的最小值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.min\_close = Min(context.data.close) |

* 参数：
* 无参数
* 返回
* min\_close：最小值

**Any 任意**

* 函数名：Any
* 中文名称：任意
* 简介：
* 该函数判断给定的数据线是否全部为假，则返回 0.0，若任意一个为真，则返回 1.0。
* 计算公式：——
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.signal = Any(context.data.close) |

* 参数：
* 无参数
* 返回
* signal：1.0代表真；0.0代表假。

**All 全部**

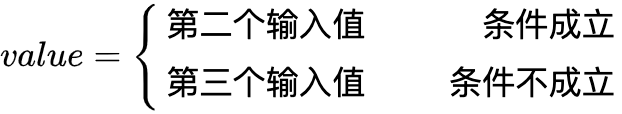
* 函数名：All
* 中文名称：全部
* 简介：
* 该函数判断给定的数据线是否全部为真，则返回 1.0，如果有一个为假，则返回 0.0。
* 计算公式：——
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.signal = All(context.data.close) |

* 参数：
* 无参数
* 返回
* signal：1.0代表真；0.0代表假。

**If 条件判断**

* 函数名：If
* 中文名称：条件判断
* 简介：
* 该函数判断给定的条件是否成立；
* 该函数需传入三个值/数据线；
* 第一个输入为判断条件，若为真，返回第二个输入；否则，返回第三个输入。
* 计算公式：



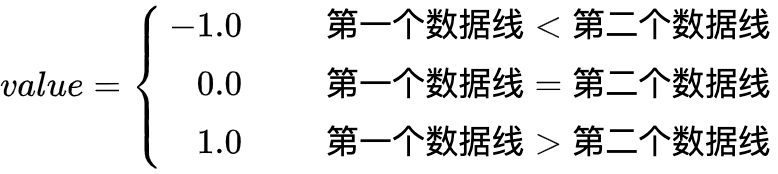
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = If(context.data.close(0) > context.data.close(-1), 1.0 ,0.0) |

* 参数：
* 无参数
* 返回
* value：输入的返回值

**Cmp 比较**

* 函数名：Cmp
* 中文名称：比较
* 简介：
* 该函数可用于比较两个值/数据线的大小；
* 该函数需传入两个值/数据线；
* 计算公式：



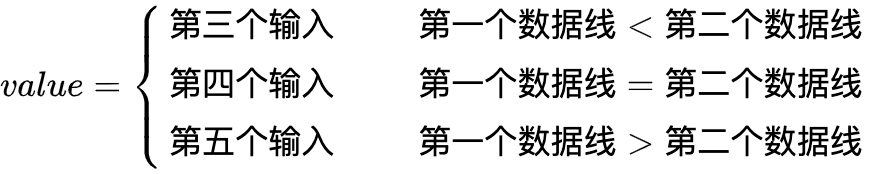
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = Cmp(context.data.close, context.data.close(-1)) |

* 参数：
* 无参数
* 返回
* value：比较结果

**CmpEx 比较**

* 函数名：CmpEx
* 中文名称：比较
* 简介：
* 该函数可用于比较两个值/数据线的大小；
* 该函数需传入五个值/数据线；
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = CmpEx(context.data.close,   context.data.close(-1),  Max(1),   Max(0),   Max(-1)) |

* 参数：
* 无参数
* 返回
* value：比较结果

**ADD 加法**

* 函数名：ADD
* 中文名称：加法
* 简介：
* 该函数接受两个输入值/数据线，返回两个输入的加总值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.ADD(context.data.close(0), context.data.close(-1)) |

* 参数：
* 无参数
* 返回
* value：和

**SUB 减法**

* 函数名：SUB
* 中文名称：减
* 简介：
* 该函数接受两个输入值/数据线，返回两个输入的差值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.SUB(context.data.close(0), context.data.close(-1)) |

* 参数：
* 无参数
* 返回
* value：差

**MULT 乘法**

* 函数名：MULT
* 中文名称：乘积
* 简介：
* 该函数接受两个输入值/数据线，返回两个输入的乘积
* 计算公式：



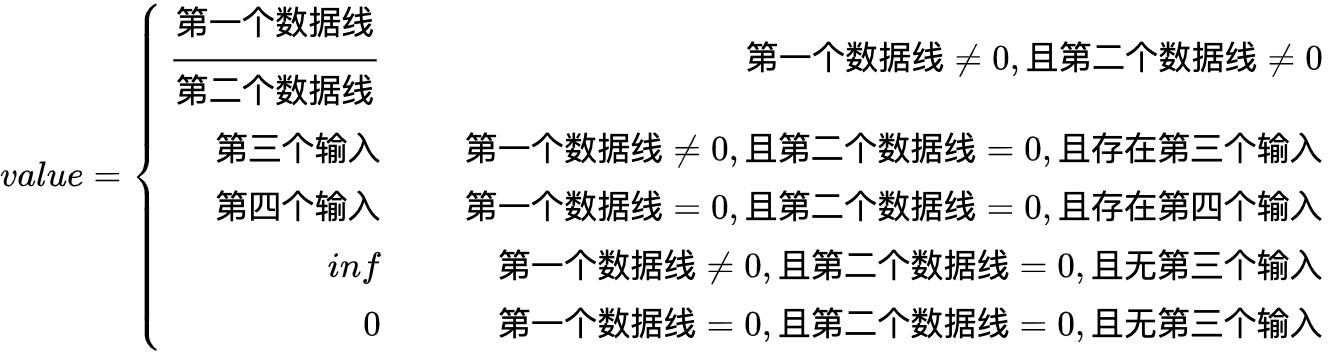
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.MULT(context.data.close(0), context.data.close(-1)) |

* 参数：
* 无参数
* 返回
* value：积

**DivZeroByZero 除法**

* 函数名：DivZeroByZero
* 中文名称：除法
* 简介：
* 该函数考虑除法时分母为0的情况
* 该函数至少需要输入两个值/数据线，最多可以输入四个值/数据线
* 计算公式：



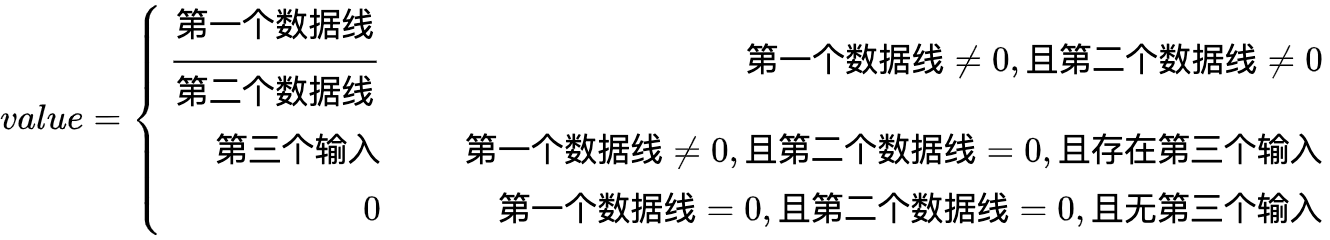
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = DivZeroByZero(context.data.close(0), context.data.close(-1), 1, 0) |

* 参数：
* 无参数
* 返回
* value：商

**DivByZero 除法**

* 函数名：DivByZero
* 中文名称：除法
* 简介：
* 该函数考虑除法时分母为0的情况
* 该函数至少需要输入两个值/数据线，最多可以输入三个值/数据线
* 计算公式：



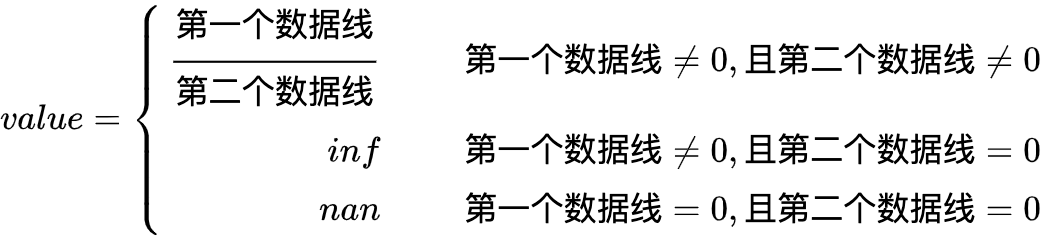
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = DivByZero(context.data.close(0), context.data.close(-1), 0) |

* 参数
* 无参数
* 返回
* value：商

**DIV 除法**

* 函数名：DIV
* 中文名称：除法
* 简介：
* 该函数接受两个输入值/数据线，返回两个输入的商
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.DIV(context.data.close(0), context.data.close(-1)) |

* 参数：
* 无参数
* 返回
* value：商

**AllN 全部**

* 函数名：AllN
* 中文名称：全部
* 简介：
* 该函数用于判断给定的周期数据的所有元素是否都为真，如果是返回1.0，否则返回0.0；
* 计算公式：——
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.alln = AllN(context.data.close, period=1) |

* 参数：

无法复制加载中的内容

* 返回：
* alln：1.0表示True；0.0 表示False

**AnyN 任意**

* 函数名：AnyN
* 中文名称：全部
* 简介：
* 该函数用于判断给定的周期数据的所有元素是否都为假，如果是返回0.0；若任意一个为真，返回1.0；
* 计算公式：——
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.anyn = AnyN(context.data.close, period=1) |

* 参数：



**点击图片可查看完整电子表格**

* 返回：
* anyn ：1.0 表示True；0.0 表示False

**ApplyN**

* 函数名：ApplyN
* 中文名称：——
* 简介：
* 该函数可获取周期为period 的数据，并调用指标func计算。
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.applyn = ApplyN(context.data.close, func=min, period=1) |

* 参数：

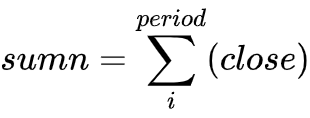


**点击图片可查看完整电子表格**

* 返回：
* applyn：所调用指标的结果值

**SumN 求和**

* 函数名：SumN
* 中文名称：求和
* 简介：
* 该函数可计算周期内的数据的加总值。
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.sumn = SumN(context.data.close, period=1) |

* 参数：

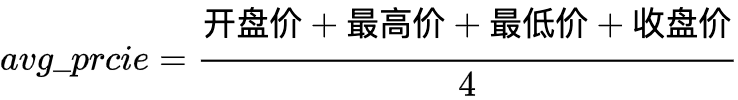


**点击图片可查看完整电子表格**

* 返回：
* sumn：加总值

**AVGPRICE 平均价格**

* 函数名：AVGPRICE
* 中文名称：平均价格
* 简介：
* 该函数返回当前 bar 的开盘价、最高价、最低价、收盘价的均值
* 计算公式：



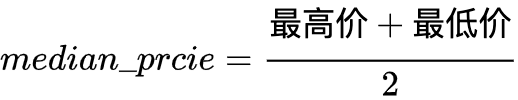
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.avg\_price = talib.AVGPRICE(context.data.open,  context.data.high,  context.data.low,  context.data.close) |

* 参数：
* 无参数
* 返回：
* avg\_price：平均价格

**MEDPRICE 平均价格**

* 函数名：MEDPRICE
* 中文名称：平均价格
* 简介：
* 该函数返回当前 bar 的最高价和最低价的均值
* 计算公式：



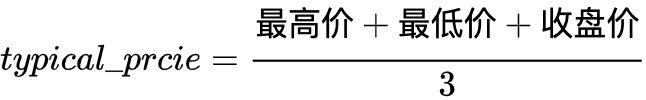
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.median\_price = talib.MEDPRICE(context.data.high, context.data.low) |

* 参数：
* 无参数
* 返回：
* median\_price：平均价格

**TYPPRICE 代表性价格**

* 函数名：TYPPRICE
* 中文名称：代表性价格
* 简介：
* 该函数返回当前 bar 的最高价、最低价、收盘价的均值
* 计算公式：



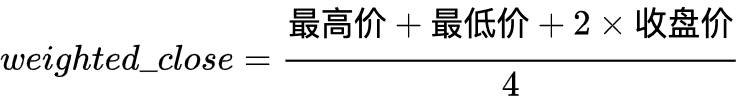
* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.typical\_price = talib.TYPPRICE(context.data.high,   context.data.low,   context.data.close) |

* 参数：
* 无参数
* 返回：
* typical\_price：代表性价格

**WCLPRICE 加权收盘价**

* 函数名：WCLPRICE
* 中文名称：代表性价格
* 简介：
* 该函数返回当前 bar 的最高价、最低价、收盘价的均值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.typical\_price = talib.WCLPRICE(context.data.high,   context.data.low,   context.data.close) |

* 参数：
* 无参数
* 返回：
* weighted\_close：加权收盘价

**EXP 指数**

* 函数名：EXP
* 中文名称：指数
* 简介：
* 该函数返回输入数据的指数
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.EXP(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：指数值

**LN 自然对数**

* 函数名：LN
* 中文名称：自然对数
* 简介：
* 该函数返回输入数据的自然对数
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.LN(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：自然对数值

**LOG10 对数函数log**

* 函数名：LOG10
* 中文名称：对数函数log
* 简介：
* 该函数返回输入数据以 10 为底的对数.
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.LOG10(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：对数函数log值

**SQRT 平方根**

* 函数名：SQRT
* 中文名称：平方根
* 简介：
* 该函数返回非负实数的平方根，若输入数据为负值，则返回nan。
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.SQRT(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：平方根

**CEIL 向上取整数**

* 函数名：CEIL
* 中文名称：向上取整数
* 简介：
* 该函数返回大于等于输入数据的最小整数值。
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.CEIL(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：向上整数值

**FLOOR 向下取整数**

* 函数名：FLOOR
* 中文名称：向下取整数
* 简介：
* 该函数返回输入数据的的下舍整数，小于或等于输入数据。
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.FLOOR(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：下舍整数值

**ACOS 反余弦函数**

* 函数名：ACOS
* 中文名称：反余弦函数
* 简介：
* 该函数返回输入数据的反余弦函数值；
* 该函数以 -1 到 1 的一个数为参数，返回一个 0 到 pi 弧度）的数值。如果传入的参数值超出了限定的范围，将返回 nan。
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.ACOS(context.data.close(0)/context.data.close(-1)) |

* 参数：
* 无参数
* 返回：
* value：反余弦函数值

**ASIN 反正弦函数**

* 函数名：ASIN
* 中文名称：反正弦函数
* 简介：
* 该函数返回输入数据的反正弦函数值；
* 该函数接受 -1 到 1 之间的数值作为参数，如果接受的参数值超出范围，则返回 nan。
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.ASIN(context.data.close(0)/context.data.close(-1)) |

* 参数：
* 无参数
* 返回：
* value：反正弦函数值

**ATAN 反正切函数**

* 函数名：ATAN
* 中文名称：反正切函数
* 简介：
* 该函数返回输入数据的反正切函数值。
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.ATAN(context.data.close(0)/context.data.close(-1)) |

* 参数：
* 无参数
* 返回：
* value：反正切函数值

**COS 余弦函数**

* 函数名：COS
* 中文名称：余弦函数
* 简介：
* 该函数返回输入数据的余弦函数值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.COS(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：余弦函数值

**COSH 双曲余弦函数**

* 函数名：COSH
* 中文名称：双曲余弦函数
* 简介：
* 该函数返回输入数据的双曲余弦函数值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.COSH(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：双曲余弦函数值

**SIN 正弦函数**

* 函数名：SIN
* 中文名称：正弦函数
* 简介：
* 该函数返回输入数据的正弦函数值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.SIN(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：正弦函数值

**SINH 双曲正弦函数**

* 函数名：SINH
* 中文名称：双曲正弦函数
* 简介：
* 该函数返回输入数据的双曲正弦函数值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.SINH(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：双曲正弦函数值

**TAN 正切函数**

* 函数名：TAN
* 中文名称：正切函数
* 简介：
* 该函数返回输入数据的正切函数值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.TAN(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：正切函数值

**TANH 双曲正切函数**

* 函数名：TANH
* 中文名称：双曲正弦函数
* 简介：
* 该函数返回输入数据的双曲正切函数值
* 计算公式：



* 调用方式：

|  |
| --- |
| Plain Text def indicators(context):  """指标"""  context.value = talib.TANH(context.data.close) |

* 参数：
* 无参数
* 返回：
* value：双曲正切函数值