天下武功为快不破（一）——VNPY Bybit接口的升级

原创 高庆峰

“竹杖芒鞋轻胜马，一蓑烟雨任平生，有绝对的掌控力才能游刃有余。”

（一）

天下武功为快不破

 CTA策略是一类追涨杀跌，以低胜率换高赔率的策略，下单到成交的低延时很重要。行情像一个漂亮女孩，感觉到位了就要尽快下手，错过便是一生，只能在脑海里回味那惊鸿一瞥多么令人惋惜；但你行动了之后也可能发现彼此并不合适，不是甜甜的恋爱而是无尽的痛苦，怎么办？当然是快速止损！竹杖芒鞋轻胜马，一蓑烟雨任平生，有绝对的掌控力才能游刃有余，哈哈，我指的是交易。所以，提高下单速度，降低交易滑点是精细化优化策略非常重要的一部分。

        此篇先聊聊如何提高下单速度，天下武功为快不破，在CTA策略中如何能做到下单快人一步呢？需要tick级别的bar内开仓！通道类CTA策略用停止单，是个不错的选择，通道上轨作为做多的触发价格，通道下轨作为做空的触发价格，当前一根k线结束下一根k线开始前预埋好停止单，等待被触发，当tick最新成交价向上突破做多触发价格马上追多，向下突破做空触发价格马上追空，而不是等待K线结束才追。

       VNPY默认支持本地停止单，通过订阅web socket tick行情，在本地判断是否达到触发价格，如果达到触发价格迅速向交易所发送高于买1的买单或低于卖1的卖单，迅速成交。本地停止单在数字货币市场的缺点很明显，首先数字货币交易所的服务器一般部署在海外的云服务器上，如果运行策略的服务器和交易所的服务器不在同一区域，网络通信稳定性非常差，网络抖动引起的短时间内web socket频繁异常断开重连问题会导致本地停止单被异常触发，即使我使用了和某交易所同区的服务器，依然会出现这种情况，只是不常出现，但是一旦出现也是致命的；其次当价格超过触发价格订单从策略服务器被发向交易所服务器是有延迟的，这个延迟可能会加大滑点，甚至可能因为行情变化迅速发出的订单不能成交而错过行情。

        问题清楚了，那么上解决方案！很简单，把停止单预埋到交易所，这样上述的两个问题全部都能迎刃而解。VNPY的引擎层是支持服务端停止单的挂撤的，只是接口不支持，那么着手升级接口，以Bybit交易所为例，以下需要升级的代码。

Bybit接口文件目录：vnstudio\Lib\site-packages\vnpy\gateway\bybit

        1. 增加停止单的订单状态。

STATUS\_BYBIT2VT: Dict[str, Status] = { "Created": Status.NOTTRADED, "New": Status.NOTTRADED, "PartiallyFilled": Status.PARTTRADED, "Filled": Status.ALLTRADED, "Cancelled": Status.CANCELLED, "Rejected": Status.REJECTED, "Active": Status.ALLTRADED, "Untriggered": Status.NOTTRADED, "Triggered": Status.PARTTRADED, "Deactivated": Status.CANCELLED,}

        2. 增加停止单订单类型。

ORDER\_TYPE\_VT2BYBIT: Dict[OrderType, str] = { OrderType.LIMIT: "Limit", OrderType.MARKET: "Market", OrderType.STOP: "Stop"}

        3. 增加一个字典self.orders: dict = {}缓存下单参数，后面撤单函数会用到其中的订单类型。

class BybitRestApi(RestClient): """ ByBit REST API """  
 def \_\_init\_\_(self, gateway: BybitGateway): """""" super().\_\_init\_\_()  
 self.gateway: BybitGateway = gateway self.gateway\_name: str = gateway.gateway\_name  
 self.usdt\_base: bool = False self.key: str = "" self.secret: bytes = b""  
 self.order\_count: int = 0 self.contract\_codes: set = set() self.orders: dict = {}

        4. 重中之重来啦，send\_order函数修改的地方比较多，增加停止单的下单链接；创建一个dict对象用以orderid为key来收集下单信息并添加到self.orders容器中缓存。

def send\_order(self, req: OrderRequest) -> str: """""" order\_data: dict = {} orderid = self.new\_orderid() order = req.create\_order\_data(orderid, self.gateway\_name)  
 if req.type == OrderType.STOP: data = { "side": DIRECTION\_VT2BYBIT[req.direction], "symbol": req.symbol, "order\_type": "Market", "qty": int(req.volume), "base\_price": self.gateway.last\_price, "stop\_px": req.price, "time\_in\_force": "GoodTillCancel", "close\_on\_trigger": False, "order\_link\_id": orderid, }  
 if self.usdt\_base: path = "/private/linear/stop-order/create" else: path = "/open-api/stop-order/create" else: data = { "symbol": req.symbol, "side": DIRECTION\_VT2BYBIT[req.direction], "qty": int(req.volume), "order\_link\_id": orderid, "time\_in\_force": "GoodTillCancel", "reduce\_only": False, "close\_on\_trigger": False }  
 data["order\_type"] = ORDER\_TYPE\_VT2BYBIT[req.type] data["price"] = req.price  
 if self.usdt\_base: path = "/private/linear/order/create" else: path = "/v2/private/order/create"  
 self.add\_request( "POST", path, callback=self.on\_send\_order, data=data, extra=order, on\_failed=self.on\_send\_order\_failed, on\_error=self.on\_send\_order\_error, ) data["req\_order\_type"] = req.type order\_data[orderid] = data self.orders.update(order\_data) self.gateway.on\_order(order)        return order.vt\_orderid

        5. 停止单的必传字段有一个为"base\_price"，需要获取tick级的市场最新成交价格，此价格需要在BybitPublicWebsocketApi类下的on\_tick函数获取。

def on\_tick(self, packet: dict) -> None: """""" ...... ...... self.gateway.last\_price = tick.last\_price

        6. 增加撤销订单链接，获取缓存的下单数据判断是否需要撤销的订单是不是停止单。

def cancel\_order(self, req: CancelRequest) -> Request: """""" data = { "symbol": req.symbol, "order\_link\_id": req.orderid } order = self.orders[req.orderid] if order["req\_order\_type"] == OrderType.STOP: if self.usdt\_base: path = "/private/linear/stop-order/cancel" else: path = "/open-api/stop-order/cancel" else: if self.usdt\_base: path = "/private/linear/order/cancel" else: path = "/v2/private/order/cancel" self.gateway.write\_log(path) self.add\_request( "POST", path, data=data, callback=self.on\_cancel\_order )

        7. 在on\_query\_contract函数中的ContractData 数据容器里面增加一个字段stop\_supported=True，以便引擎层判断此交易所是否支持停止单，如果是，则发送交易所停止单。

def on\_query\_contract(self, data: dict, request: Request) -> None: """""" if self.check\_error("查询合约", data): return  
 for d in data["result"]: self.contract\_codes.add(d["name"])  
 contract = ContractData( symbol=d["name"], exchange=Exchange.BYBIT, name=d["name"], product=Product.FUTURES, size=1, pricetick=float(d["price\_filter"]["tick\_size"]), min\_volume=d["lot\_size\_filter"]["min\_trading\_qty"], net\_position=True, history\_data=True, stop\_supported=True, gateway\_name=self.gateway\_name )  
 if self.usdt\_base and "USDT" in contract.symbol: self.gateway.on\_contract(contract) elif not self.usdt\_base and "USDT" not in contract.symbol: self.gateway.on\_contract(contract)

        8. 最后，VN Trader的行情界面的订单状态需要主动请求获取，但是请求频率不能过高，会造成线程阻塞，所以需要加上限速。首先在connect函数下增加self.timer\_count = 0 ，然后在process\_timer\_event 函数下增加计数判断并查询订单。

class BybitGateway(BaseGateway): """ VN Trader Gateway for ByBit connection. """ ....... .......  
 def connect(self, setting: dict) -> None: """""" ...... ...... self.timer\_count = 0 self.event\_engine.register(EVENT\_TIMER, self.process\_timer\_event)  
 ...... ......  
 def process\_timer\_event(self, event): """""" self.query\_position() self.timer\_count += 1 if self.timer\_count < 5: return self.timer\_count = 0 self.rest\_api.query\_order()

        至此，Bybit交易所停止单已经完成，目前已经实盘，未发现问题，如果升级过程中遇到问题，欢迎与我交流，微信号：515119049。

        最后，非常感谢VNPY的支持！！