背景

前面，通过图文 如何利用 C# 爬取 ONE 的交易数据？ 向大家介绍了如何爬取在 BigOne 上线的数字资产的交易数据。

其次，通过图文 如何利用BigOne的API制作自动化交易系统 -- 身份验证
向大家介绍了利用 BigOne API 函数之前的身份验证问题。

然后，通过图文 如何利用BigOne的API制作自动化交易系统 -- 获取账户资产
向大家介绍了利用 BigOne API 函数获取自己的账户资产数据。

接着，通过图文 如何利用BigOne的API制作自动化交易系统 -- 订单系统
向大家介绍了 利用 BigOne API 函数进行挂单、撤单、查询订单状态等交易操作。

最后，通过图文 如何进行代码的重构？以封装 BigOne API 为例 向大家介绍了
利用 Layers 软件体系结构风格重构对 BigOne API 的封装过程。

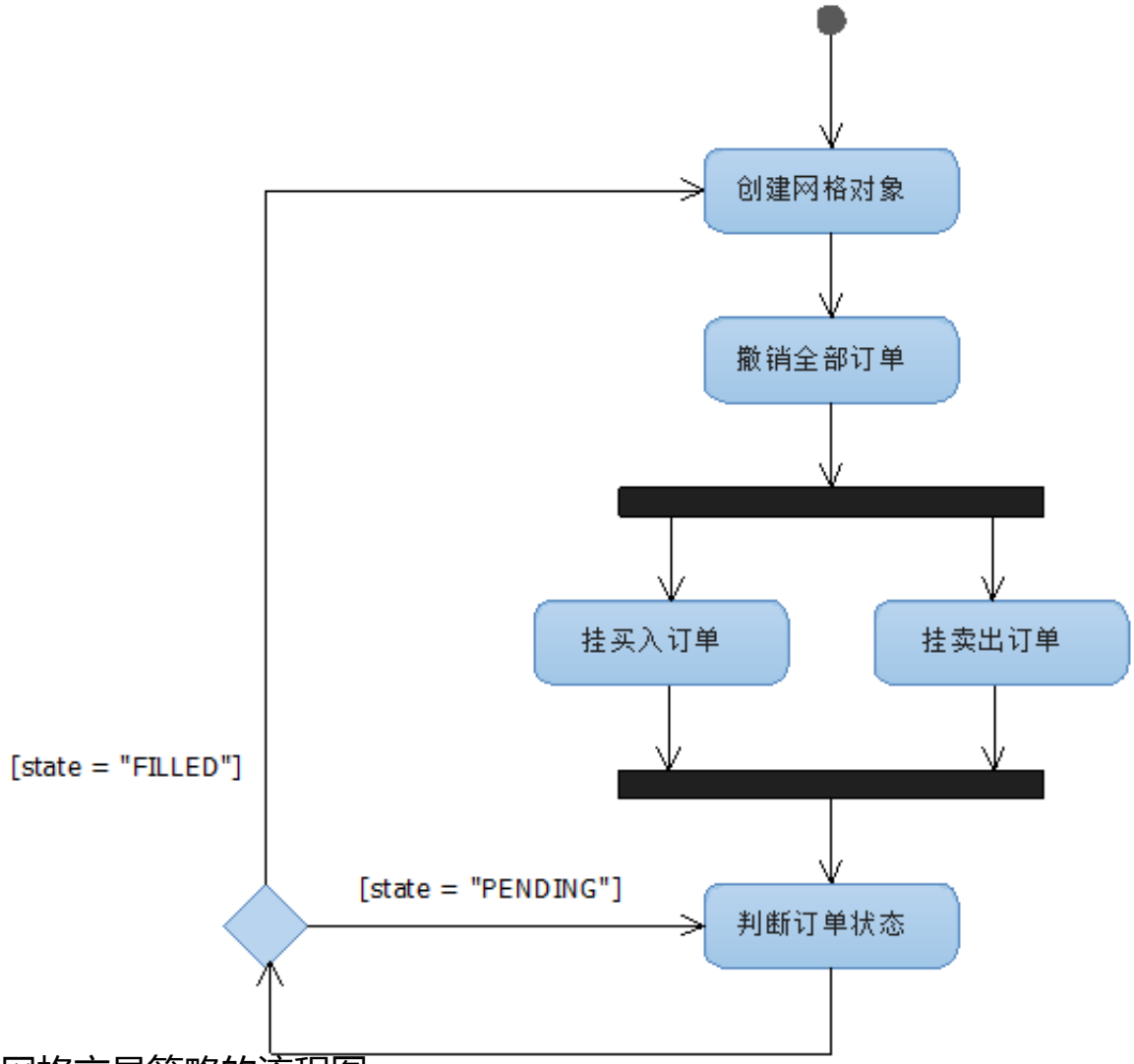
有了以上的基础，我们就可以制定交易策略，通过自动化的方式进行数字资产的交易了。从此告别手动挂单、撤单、查看是否成交的烦恼！

技术分析

道氏理论指出，金融市场的价格变动可以分为三种情况，分别是上涨、下跌和牛皮。而在上涨或下跌的过程中也会在较短周期上出现连续的波动，价格最终或者以无趋势波动呈现出来，或者以短期的波动和长期的趋势呈现出来的。在外汇市场上有一种被称为渔网交易法的交易理念，这种理念认为，在一定的时间周期上，价格基本处于往复波动的状态，投资者可以通过较高的频率交易，利用限价单的交易方式来获得价格波动的收益。

什么是网格交易法呢？

> 网格交易法，就是跌买涨卖。具体做法是把资金分成 n 份，每次投入固定金额，先初始建仓，再设定一个百分比，比如 5%，股价跌 5% 就买入一份，涨 5% 就卖出一份，如此反复买卖，不断的低吸高抛，不断产生盈利，从而积少成多。



网格交易策略的流程图

- 创建网格对象：根据历史成交数据确定向上、向下的网格密度（宽度）以及确定每个网格卖出、买入的价格和数量。
- 撤销全部订单：撤销当前未完成的订单。
- 挂买入订单：形成买入的网格。
- 挂卖出订单：形成卖出的网格。
- 判断订单状态：监测数字资产价格是否触及网格，如果触及并成交则重新创建网格对象，如此往复执行。

代码实现

Step1 定义网格交易用到的数据结构。

表示订单的结构 Order

```
public class Order{ // ?? public double Price { get; set; }
// ?? public double Amount { get; set; } public Order(double
 price, double amount) { Price = price; Amount = amount; }}
```

表示数字货币资产的接口 IDigitalCurrency。

```
public interface IDigitalCurrency{ // ??ID string AssetId {
get; } // ?????????? List<AssetData> GetHistoryData(string p
eriod, string time, string limit); // ?????????? List<AssetD
ata> GetHistoryData();}
```

利用 BTC 来实现数字资产接口举例。

代码原理参见：如何利用 C# 爬取 ONE 的交易数据？

```
public class Btc : IDigitalCurrency{ public string AssetId {
 get; } = "BTC"; public List<AssetData> GetHistoryData(strin
g period, string time, string limit) { string url = "https:/
/b1.run/api/xn/v1/asset_pairs/550b34db-696e-4434-a126-196f82
7d9172/candles?" + "period=" + period + "&time=" + time + "&
limit=" + limit; ServicePointManager.SecurityProtocol = Secu
rityProtocolType.Tls12; IHtmlDocument document = new JumonyP
arser().LoadDocument(url); List<IHtmlNode> nos = document.No
des().ToList(); string str = nos[0].ToString(); StringReader
 sr = new StringReader(str); JsonTextReader jsonReader = new
 JsonTextReader(sr); JsonSerializer serializer = new JsonSer
ializer(); JsonOne one = serializer.Deserialize<JsonOne>(jso
nReader); return one.data; } public List<AssetData> GetHisto
ryData() { string period = "DAY1"; DateTime dt = DateTime.No
w; string time = dt.Year + "-" + dt.Month.ToString().PadLeft
(2, '0') + "-" + dt.Day.ToString().PadLeft(2, '0') + "T00:00
:00.000Z"; string limit = "250"; return GetHistoryData(perio
d, time, limit); }}
```

Step2 网格交易的抽象结构 GridMethod。

```
public abstract class GridMethod{ // ?? public IDigitalCurre
ncy Asset { get; protected set; } // ????? public string Mar
ketId { get; protected set; } // ??????? public double BaseP
```

```
rice { get; protected set; } // ???????????????? public dou
ble BaseMoney { get; protected set; } // ????????????? pub
lic int Days { get; protected set; } // ????????????? pub
lic int DailyTurnover { get; protected set; } // ??????????
 public double GridAmplitudeUp { get; protected set; } // ??
????????? public double GridAmplitudeDown { get; protected s
et; } // ???????????? public int GridCount { get; protected
set; } // ???? public List<Order> OrderList { get; protected
 set; } // ?????????? protected abstract void CalculateAmpli
tude(); // ?????? protected abstract void CalculateGrid(); /
/ ?????? public abstract void CancelAllOrder(); // ?????? pu
blic abstract List<OrderResponse> CreateBidOrders(); // ????
?? public abstract List<OrderResponse> CreateAskOrders(); //
 ???????? public override string ToString() { string infor =
 "AssetId:" + Asset.AssetId + Environment.NewLine + "MarketI
d:" + MarketId + Environment.NewLine + "BasePrice:" + BasePr
ice + Environment.NewLine + "BaseMoney:" + BaseMoney + Envir
onment.NewLine + "Days:" + Days + Environment.NewLine + "Dai
lyTurnover:" + DailyTurnover + Environment.NewLine + "GridAm
plitudeUp:" + GridAmplitudeUp + Environment.NewLine + "GridA
mplitudeDown:" + GridAmplitudeDown + Environment.NewLine + "
GridCount:" + GridCount; return infor; }}
```

Step3 等金额的网格策略 GridMethodEqualMoney。

该交易策略的每个网格所使用的网格宽度相同、金额相同，即是一种沉淀数字资产的交易策略。

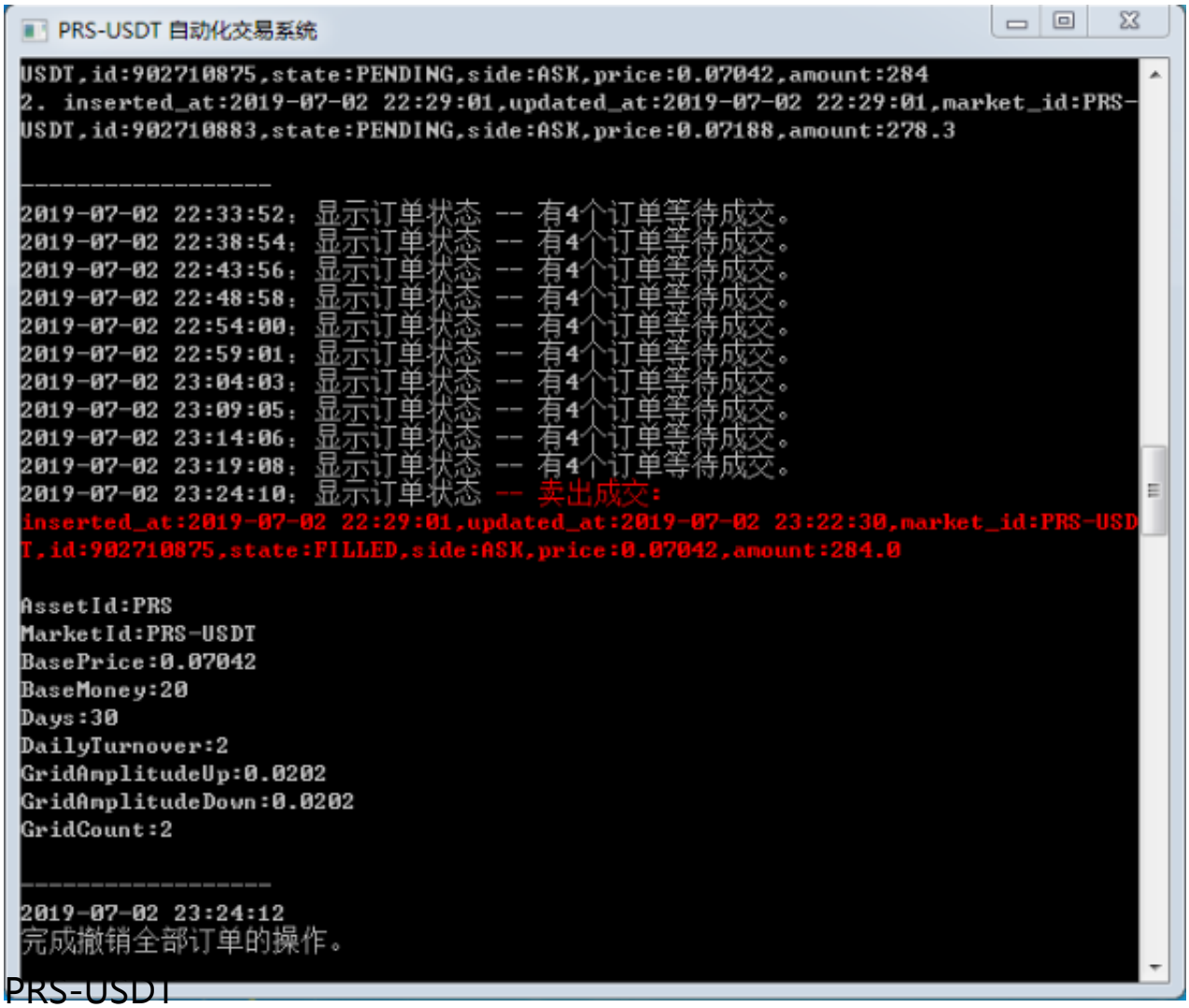
```
public sealed class GridMethodEqualMoney : GridMethod{ priva
te readonly IDigitalCurrencyUtility _digitalCurrencyUtility;
 public GridMethodEqualMoney(IDigitalCurrency asset, string
marketId, int days, int dailyTurnover, double baseMoney, dou
ble basePrice, int gridCount, IDigitalCurrencyUtility digita
lCurrencyUtility) { _digitalCurrencyUtility = digitalCurrenc
yUtility; Asset = asset; MarketId = marketId; BasePrice = ba
sePrice; BaseMoney = baseMoney; Days = days; DailyTurnover =
 dailyTurnover; GridCount = gridCount; //???????????? Calcul
ateAmplitude(); //?????? CalculateGrid(); } protected overri
de void CalculateAmplitude() { //????????????????????? List<
```

```
AssetData> lstTrade = Asset.GetHistoryData(); List<double> l
st = new List<double>(); int count = Days < lstTrade.Count -
 1 ? Days : lstTrade.Count - 1; for (int i = 0; i < count; i
++) { double zf = (lstTrade[i].high - lstTrade[i].low)/lstTr
ade[i + 1].close; lst.Add(zf); } double result = lst.Count =
= 0 ? 0.02 : Math.Round(lst.Average()/(2.0*DailyTurnover), 4
); result = Math.Max(0.02, result); GridAmplitudeUp = result
; GridAmplitudeDown = result; } protected override void Calc
ulateGrid() { OrderList = new List<Order>(); double price =
Math.Round(BasePrice, 6); double amount = Math.Round(BaseMon
ey/price, 3); Order order = new Order(price, amount); OrderL
ist.Add(order); for (int i = 0; i < GridCount; i++) { price
= Math.Round(price/(1.0 - GridAmplitudeUp), 6); amount = Mat
h.Round(BaseMoney/price, 3); order = new Order(price, amount
); OrderList.Add(order); } price = Math.Round(BasePrice, 6);
 for (int i = 0; i < GridCount; i++) { price = Math.Round(pr
ice*(1.0 - GridAmplitudeDown), 6); amount = Math.Round(BaseM
oney/price, 3); order = new Order(price, amount); OrderList.
Add(order); } OrderList = OrderList.OrderBy(a => a.Price).To
List(); } public override void CancelAllOrder() { _digitalCu
rrencyUtility.CancelAllOrder(MarketId); } public override Li
st<OrderResponse> CreateBidOrders() { List<Order> lst = Orde
rList.GetRange(0, GridCount); return _digitalCurrencyUtility
.CreateBidOrders(lst, MarketId); } public override List<Orde
rResponse> CreateAskOrders() { List<Order> lst = OrderList.G
etRange(GridCount + 1, GridCount); return _digitalCurrencyUt
ility.CreateAskOrders(lst, MarketId); }}
```

总结

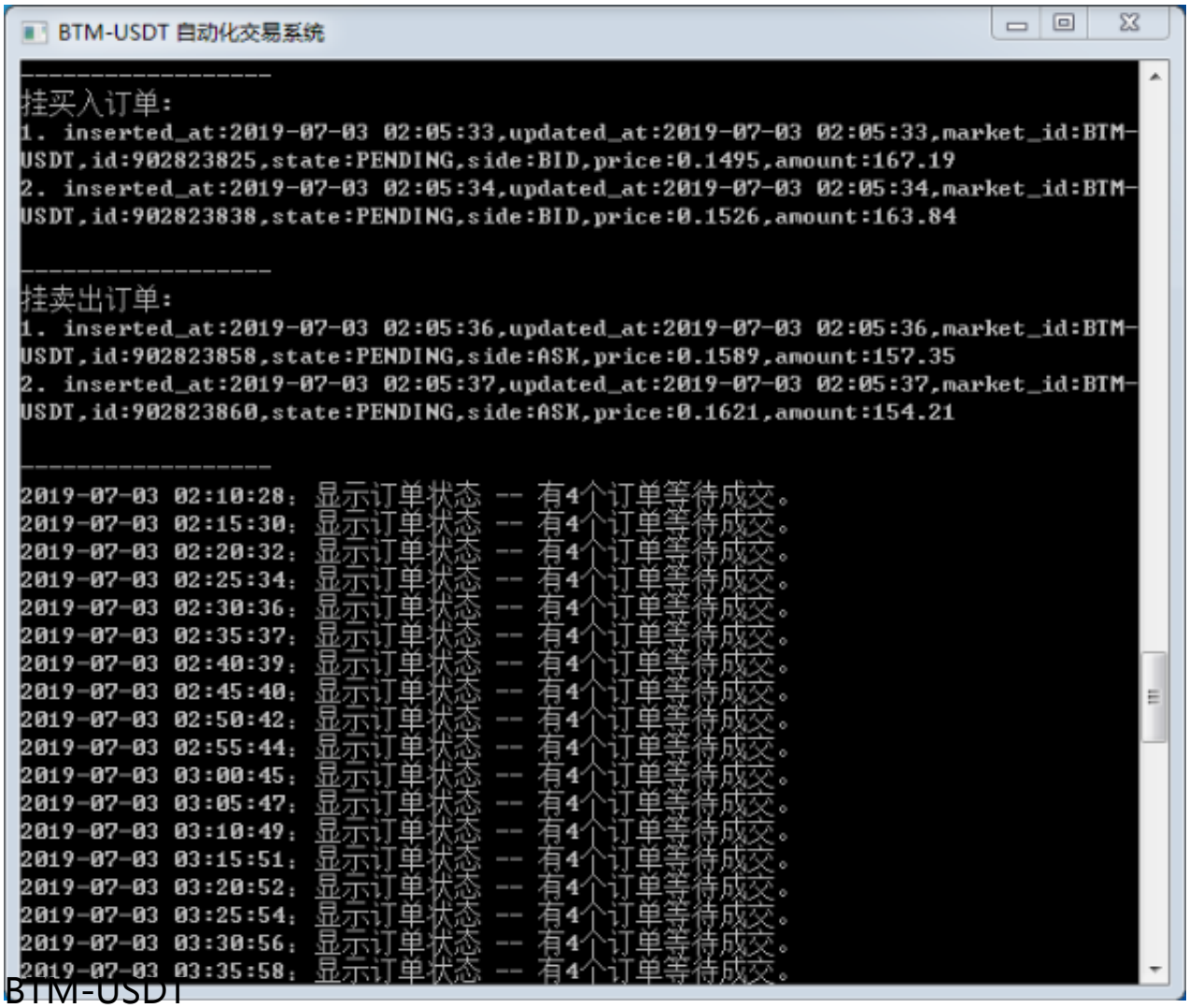
到此为止，网格交易策略的实现框架就基本介绍完了，下面是我利用该框架对BTC、EOS、BTM、PRS、ONE等 进行网格交易的截图。

ONE-USDT 交易对



PRS-USDT

EOS-USDT 交易对



BTM-USDT

该策略的回测和评估需要积累一定数据才能进行，我们后面再来介绍。

网格交易看起来似乎简单，其实里面包含了很复杂的问题，比如市场趋势向上、向下、横盘震荡时我们如何调整向上、向下网格的宽度；如何确定每个网格买入、卖出的数字资产数量（金字塔、倒金字塔、同量……）；资金固定的情况下如何进行资金分配等等。这些都是需要考虑和写程序根据不同的场景进行自适应调整的。慢慢来啊！今天就到这里吧！See You！

相关图文：

- 如何利用 C# 实现 K 最邻近算法？
- 如何利用 C# 实现 K-D Tree 结构？
- 如何利用 C# + KDTree 实现 K 最邻近算法？
- 如何利用 C# 对神经网络模型进行抽象？

- 如何利用 C# 实现神经网络的感知器模型？
- 如何利用 C# 实现 Delta 学习规则？
- 如何利用 C# 爬取带 Token 验证的网站数据？
- 如何利用 C# 向 Access 数据库插入大量数据？
- 如何利用 C# 开发「桌面版百度翻译」软件！
- 如何利用 C# 开发「股票数据分析软件」（上）
- 如何利用 C# 开发「股票数据分析软件」（中）
- 如何利用 C# 开发「股票数据分析软件」（下）
- 如何利用 C# 爬取「财报说」中的股票数据？
- 如何利用 C# 爬取 One 持有者返利数据！
- 如何利用 C# 爬取Gate.io交易所的公告！
- 如何利用 C# 爬取BigOne交易所的公告！
- 如何利用 C# 爬取「猫眼电影：热映口碑榜」及对应影片信息！
- 如何利用 C# 爬取「猫眼电影专业版：票房」数据！
- 如何利用 C# 爬取「猫眼电影：最受期待榜」及对应影片信息！
- 如何利用 C# 爬取「猫眼电影：国内票房榜」及对应影片信息！
- 如何利用 C# + Python 破解猫眼电影的反爬虫机制？