0 快速开始

1. 下载源码

```
# git clone https://github.com/francinexue/xuefu.git
```

2. 使用pycharm，将xuefu作为项目根目录

```
xuefu -> 右键 -> mark directory as -> Source Root
```

3. 运行demo目录示例

```
# cd ./demo/backtest
# python csvDemo.py
```

1 pyalgotrade 简介

1.1 面向群体

pyalgotrade是美洲一人所写，github 地址：https://github.com/gbeced/pyalgotrade，适用python版本为2.7，面向用户为有一定编程经验，需要灵活进行回测操作，且希望完全本地运行，保障策略安全的金融从业人员，量化爱好者，大学生等群体。

1.2 用户群体

pyalgotrade本身具有事件驱动、回测、实盘扩展、talib支持等多个特点，对于在策略回测方面，相教于zipline更适合国内个人使用，相较于vnpy、quicklib等在策略回测方面有明显优势。如果只是为程序化交易的用户且希望界面友好的请选择vnpy，在友好性及技术支持方面vnpy要高于pyalgotrade，如果希望进行更加深入及个性化的研究，请选择pyalgotrade。

1.3 功能特点

pyalgotrade代码据说由金融从业20余年的人员编写，在代码质量方面极具简洁性，同时在灵活性、可扩展性、高效性方面优势非常明显，当然因为文档过少，所以对于非计算机科班出身的人员学习有一定难度，查阅文献太少，阅读源码太难。

1.4 主要内容

我在2015年中的时候接触该框架，由于pyalgotrade本身不支持pandas等，自己改了点源码写了些小demo放在该github上，希望能给初学者一些参考。github地址：https://github.com/francinexue/xuefu

由于当时初学python，时间比较久了，许多文法并不优美，但也正好可以让大家顺藤摸瓜，容易阅读，代码啰嗦的地方莫要见笑。

2 入门篇-从demo学起

> - 开发环境
>
>   `pycharm`
> - 操作系统
>
>   `windows&linux`双系统
> - 编译环境
>
>   有经验者可以使用`./dev/docker/`目录下的docker环境
>
>   无经验者使用`python2.7`即可
> - 包依赖
>
>   ```
>   * 为必需，～ 为非必需
>   pyalgotrade     *
>   pandas          *
>   lxml            *
>   numpy           *
>   matplotlib      *
>   tushare         *
>   pymongo         ～     -> mongodb,
>   sqlalchemy      ～     -> postgres
>   gmsdk           ～     掘金<http://www.myquant.cn/docs/api/python>
>   peakutils       ～
>   scipy           *
>   statsmodels     *
>   ta-lib          *
>   tesseract-ocr   ～     文本识别
>   easytrader      ～
>   keras, cuda, theano                          ～
>   ipython notebook                              ～
>   libthostmduserapi.so libthosttraderapi.so   ～    -> ctp
>
>
>
>   ```

2.1 跑通第一个例子，csvDemo

- 该例子主运行文件为`demo.backtest.csvDemo.py`,数据文件位于`api.stock.csv`

`python csvDemo.py`

- 结果包含两部分：

```
/usr/bin/python2.7 /root/PycharmProjects/xuefu/demo/backtest/csvDemo.py
2000-02-24 00:00:00 strategy [INFO] BUY at ￥63.19
2000-04-03 00:00:00 strategy [INFO] SELL at ￥78.62
2000-06-05 00:00:00 strategy [INFO] BUY at ￥79.25
2000-07-06 00:00:00 strategy [INFO] SELL at ￥71.75
2000-08-07 00:00:00 strategy [INFO] BUY at ￥80.88
2000-09-12 00:00:00 strategy [INFO] SELL at ￥83.00
2000-12-06 00:00:00 strategy [INFO] BUY at ￥31.19
2000-12-29 00:00:00 strategy [INFO] Final portfolio value: $1074351.19
```

上图即为买卖信息和收益率，累计收益率数据。

学习要点

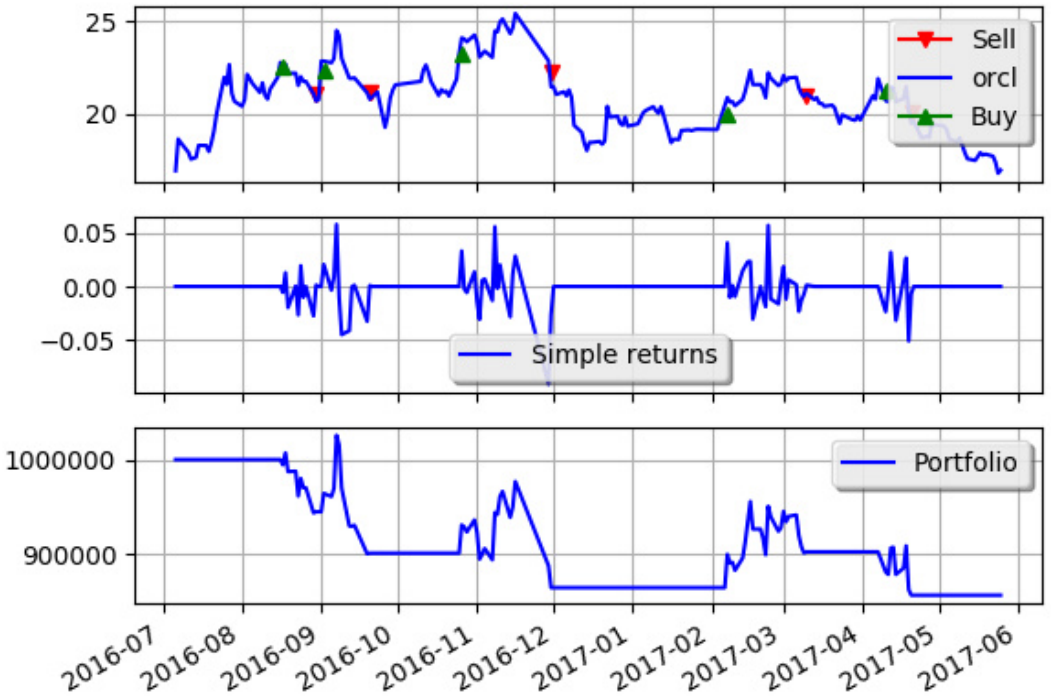
- 注意要点为csv的数据格式，csvDemo中直接加载的csv头部格式为 `Date,Open,High,Low,Close,Volume,Adj Close`
- pyal_utils 为我写的将回测后的各个指标如收益率、胜率、sharp比率、累计收益率等指标按照重新抽取出来np.array格式，可选。
- 买卖策略在`pandasDemo_run.py`中书写，onBar每次加载一条新数据（一行），self.__position变量记录是否持有仓位，初次学习可以直接使用 `if self.__position is None or not self.__position.isOpen():`来定义买入逻辑

`if self.__position is not None and not self.__position.exitActive():`定义卖出逻辑

2.2第二个例子学习，pandasDemo

本例子主要学习结合pandas的dataFrame加载数据，以及再讲解一些pyalgotrade的知识点。

- 主运行文件`demo.backtest.pandasDemo`,依赖文件为`cnx.dataFrameFeed`,其中cnx目录为我本地在pyalgotrade原作者基础上进行的一层封装。
- 运行结果如图



学习要点

- dataFramefeed支持的数据头部要注意，如下图

```
              open     high     close      low     volume    amount
 date
2017-05-25  16.95  17.15  16.99  16.84  2769759.0  47158227.0
2017-05-24  17.20  17.41  16.82  16.72  3383104.0  57300503.0
2017-05-23  17.75  17.99  17.41  17.36  1831028.0  32458338.0
2017-05-22  17.85  18.06  17.74  17.69  1413485.0  25259907.0
```

因数据级别不同，dataFramefeed 支持index为`str`类型的`%Y-%m-%d`或`%Y-%m-%d %H:%M:%S`以及`datetime64`两种类型，其他类型请转化数据格式

- 支持tushare，tushare读取的格式可以直接使用

```
import tushare as ts
dat = ts.get_h_data('600848')
dat.index.name = 'date'
feed = dataFramefeed.Feed()
feed.addBarsFromDataFrame("orcl", dat)
```

- run中使用了计算高点`highlow.High(self.__prices, N1, 3)`,而在onbar中使用的时候，不能有`None`,因此需要在第一行过滤。

  `if self.__high.__len__() is not 3:return`

- 获取可用资金的方法为：

  `self.getBroker().getCash()`

2.3 第三个例子，porfolioDemo

本例子主要讲解多股票（合约如何回测）。pyalotrade不是特别擅长多合约回测，当然还是有这个功能的，而且比较简单。另外多因子选股不适合该框架；实现多股票回测可以有两种方式，一是在外部调用单个回测，这种做法可以使用多进程进行并发，另一种即在一个`backtestStrategy`里进行回测。核心代码如下：

```
#加载多个合约
instrument = ["lenovo", "mi"]
feed.addBarsFromCSV("lenovo", "../../api/stock/csv/600847.csv")
feed.addBarsFromCSV("mi","../../api/stock/csv/600848.csv")
strat = pdr.VWAPMomentum(feed, instrument, vwapWindowSize, threshold)
#在onbar中第一行使用for循环
for element in bars.getInstruments():
    return
```

学习要点

- pyalgotrade的数据类型和pandas的对应关系如下：

	pyalgotrade	pandas
0	Feed	Dict
1	Feed[instrument]==BarDataSeries	DataFrame
2	DataSeries	Series
3	Bar	Row

当然pyalgotrade内置类型和pandas并不通用，需要经过一定的转化。pyalgotrade自己内建各类型而不是用pandas的原因主要为：

- pyalgotrade整个框架采用事件驱动的模式，可以简单理解为模拟真实的交易环境，每次加载一条数据（相当于真实的情况下来一条新数据处理一条），这样的话就需要切片到当前Bar获取内部的细节，pandas的`ix`,`iloc`满足不了该功能，也不符合开发理念。
- pandas在数据处理方面功能十分强大，但也恰恰因此牺牲了很多效率，原作者考虑了该问题，因此进行了自己的封装。我在长期使用pandas的过程中也发现了效率问题，当然也可以有不少优化方案，这块后期再谈。
- 更详细的pyalgotrade数据类型的解析位于下一章

2.4 第四个例子，sqlDemo

本例子为从数据库中读写并进行回测操作，实际仍然以“桥接”的方式，而非“直连”，以dataFrame为桥，目前示例可对mongodb和postGres进行存取。

数据存取

- 存取postgres数据库代码位于`api.stock.histmd.to_sql_md.py`,核心代码不再讲解了，数据库的配置信息位于`~/constant.py`下。个人需自己安装postgres数据库，配置如下参数，`_PATH_CODE`为需要批量下载存储的代码数据列表，定期运行一下即更新数据到数据库中了。

```
"""
数据库常数
"""
_PATH_CODE_ = 'd:/data/code.csv'
_ENGINE_ = 'postgresql://postgres:root@localhost:5432/tushare'
```

数据源使用tushare，tushare相关行情都可存取。

```
INTERFACE AS FOLLOWS:

1. set_h_data(start = ct._START_,middle =
ct._MIDDLE_,autype="qfq",index=False,retry_count = 3,pause=0)
 #获取历史交易信息存入数据库中，默认从1994-2015年。若不设定默认取最近一年，其他参数与tushare相同
 #指数行情tushare其实可以查询，但未提供列表，因此自行构造
2. get_h_data(code)
 #根据代码从数据库读数据
3. set_realtime_quotes(code=['sh'],pause = 10)
 # 实时数据
4. set_stock_basics
```

- 另外说一句定期下载数据到csv文件的方法位于`api.stock.histmd.to_csv_md.py`,代码写的都太早了，也可以使用，存储目录位于`'../csv/'`,具体方法不在详述。
- 存取mongodb的代码位于`api.stock.histmd.to_mongodb_md`,这块代码功能较为完备，具有定期自动更新数据功能。

```
CONFIG@ ～/constant.py  配置信息

TUSHARE_FOUNDAMENTAL_METHOD_LIST_ = ['get_growth_data',
'get_operation_data']#财务获取列表
_MONGODB_ENGIN_ = 'mongodb://localhost:27017/';
_MONGODB_DATABASE_ = 'admin'

INTERFACE                接口

1. foundamental_dao.py  财务数据
2. k_data_dao.py        日线级别数据
3. k_index_dao.py       指数行情
4. k_min_dao.py         分钟数据###
```

代码示例

```
    # 从postgres 中加载
    if loadtype == 'pgs':
        from api.stock.histmd import to_postgres_md as tpd
        dat = tpd.get_h_data('600848')

    #从mongodb加载
    else:
        from api.stock.histmd.to_mongodb_md import tmd
        w = tmd.KdataDbCache()
        dat = w.get_k_data('300426', start='2015-02-05', end='2015-02-19')
```

学习要点

- 这里讲一下plt，原作者封装了一层，可以加载`dataSeries`直接作图，与pandas中的`Series`&`DataFrame`直接作图原理相同。如回测完后，可以加载收益的数据并作图。

```
plt.getOrCreateSubplot("returns").addDataSeries("Simple returns",
returnsAnalyzer.getReturns())
```

2.5 第五个例子，minWithDayDemo

从该节往后，涉及到的内容应该慢慢会感兴趣，相对前面略复杂一些。本例子为日线和分钟线混合的例子，即既需要日线级别的指标数据，同时又需要更细粒度的分钟级别的数据。

例如，在交易软件中，日线macd的实时数值是在不停变化的，随着当前价格的改变而改变，当当日收盘时后，该值才定格。本例子中所计算的ENE指标即采用动态计算的方法，当然该版本较为原始，后面例子写法可能更好一些。

代码示例

首先在`minWithDay_run.py`文件，类`ENE_backtest`的`__init__`中，针对5分钟数据重采样为日线数据。

```
self.__feed_day = self.resampleBarFeed(frequency=bar.Frequency.DAY,
callback=self.resample_callback)

def resample_callback(self, some_number, bars):
   if 1 > 2:return
```

- 其中`resample_callback`为回调函数，在本例子中，当从5分钟重采样为日线时，每新来一个bar，凑够一个日线并生成日线的一行数据后即调用该函数，作用等同于日线的`onBar`方法。在本例子中，不需要回调函数，所以直接写了`if False:return`，举例来说，当分钟级别bar走到`2017-07-17 10:40`的时候，各数据走动如下：

```
data                          value                                  data@time
-------------------------------------------------------------------------------
----
bar.getDateTime()                                                    2017-07-17
10:40
self.__feed[instrument].getCloseDataSeries()[-1]                    2017-07-17
10:40
self.__feed_day[instrument].getCloseDataSeries()[-1]                2017-07-16
15:00
```

- 计算ENE指标的时候，采用的手工从日线数据当中抽取一部分数据，然后从当日分钟数据中取当前数据，最终进行合并计算，代码不再列举。

学习要点

- 数据预处理

我本地本来只有一分钟级别数据，无5分钟级别的数据，因此为了演示示例，先将1分钟csv数据重采样出5分中的数据。这块数据是在加载之前进行的。即加载的数据`feed.addBarsFromDataFrame("orcl", dat)`就已经是5分钟数据。

```
dat = pd.read_csv("../../api/stock/csv/600281SH_min.csv", index_col=
['datetime'], encoding='gbk')
feed = dataFramefeed.Feed(frequency=bar.Frequency.MINUTE)
feed.addBarsFromDataFrame("orcl", dat)
resample.resample_to_csv(feed, bar.Frequency.MINUTE * 5,
                                "../../api/stock/csv/600281SH_5min.csv")  # 这
样resample的数据是对的
```

该重采样方法为通用方法，能够将pandas的`dataFrame`进行重采样，**尽量不要用pandas自带的resample方法！**，关于重采样的分析在下一章再继续讲。

同时该resample方法位于`from pyalgotrade.tools import resample`，
与`BacktestingStrategy`里面的resample（位于`from pyalgotrade import strategy.BacktestingStrategy.resample`）不同，一静态生成，一动态计算，不要弄混。

- 如果需要计算纯日线级别的指标如MA，可以在`__init__`中写入`self.__day_ma = ma.SMA(self.__feed_day[instrument].getCloseDataSeries(),10)`计算，这样计算的数值全是用的日线的数据，当日5分钟级别的信息并没有动态计算。
- 对于股票，其持有期需要大于1天,该Demo下的代码写的不完善，用后面例子中的。

```
def onEnterOK(self):
      self.holdDay =
self.__position.getEntryOrder().getSubmitDateTime().replace(hour=9,minu
te=15,second=0)

def onBar(self,bars):
      if (bars.getDateTime() - self.holdDay) < timedelta(days=1):  # 持
有期大于一天
            return
```

2.6 第六个例子，indexBBCurveDemo

本例子是根据丁鹏-<量化投资策略与技术>书中描述的依据布朗运动计算牛熊线，判断大盘指数，并作为择时操作的依据，牛市、熊市、震荡市策略各有不同此处择时买卖策略仅是简单的超过牛线则买入，跌破熊线则卖出，有兴趣的可以深入研究下。

2.7 第七个例子，talibDemo

调用ta-lib的例子，talib是应用非常广泛的指标计算工具，内置了非常多的指标。具体安装步骤请参考：https://github.com/mrjbq7/ta-lib,其中Windows和Linux平台都需要自己编译并重新设置一下configure的目录。

pyalgotrade自己对talib进行了一层封装，因此可以使用其内建的talib方法，内建方法支持`DataSeries`、`BarDataSeries`数据格式直接结算；也可以使用原生Ta-lib方法，原生方法支持的数据格式为`np.array`

- 使用内建talib方法如下：

```
from pyalgotrade.talibext import indicator
self.__aroondown,self.__aroonup =
indicator.Adx(self.__feed[self.__instrument],len(self.__feed[self.__instrum
ent]),20)
```

> - 这个例子中Adx指标需要使用`high`、`low`、`close`三个指标进行计算，因此传入`BarDataSeires`，即`self.__feed[self.__instrument]`
> - 第二个参数`count`指的是返回的数据的最大长度，因为pyalgotrade基础的数据格式为定长`List`，即`ListDeque`,当来一个新数据后，达到定长则剔除掉最后一个数据，相当于数据结构中的`Queue`，此处就是相当于初始化返回值的长度，**建议均使用`len(self.__feed[self.__instrument])`**。
> - 第三个参数`timeperiod`才是需要计算的窗口大小，本例子中为计算20日的adx指标。

- 使用原生talib的方法本例子中没写，在bollingBandDemo中有示例。
- 另外常见指标如macd，kdj等等，以及其他指标可以在外部计算好之后传入里面，具体的针对pandas的`DataFrame`计算指标的方法位于`~/utils/formular.py`。本例子中kdj的指标计算即通过传入计算好的来回测。当然feed加载的过程中有一个extra_DataSeries，可以直接`self.getExtraDataSeries('kdj')`来获取预先加载的指标。

学习要点

- talib计算kdj指标的结果与当前国内软件的计算指标不同，所以不建议使用，建议自己计算。同时自己测试的布林带、macd、adx等指标是相同的；线性回归函数`ta.LINARREG`与通达信的`FORCAST`函数有出入，较后者更灵敏。
- **无论是使用内建talib指标还是原生talib，计算方法都需要在onBar中调用，而非如前面的ma指标写在`__init__`中。**
- 文件代码注释列举了其他内容。

```
#from stackoverflow we knows that when use talib in pyalgotrade,talibext
only calculate once,it won't calculate new result when new bar is
feed,so the indicator should be call in every onBars call.
#another caution is the talibext's parameter 'count' starts from 0 to
-1,as soon as
we must use count = len(barDs)
# the result of STOCH()  kdj is different with TongHuaShun, Reason
unknown,Maybe 'MaType'
#slowk,slowd =
talib.STOCH(dat.high.values,dat.low.values,dat.close.values,14,3,0,3,0)
 equals to
indicator.STOCH(self.__feed[self.__instrument],len(self.__feed[self.__in
strument]),
                                        14,3,0,3,0)
#at last trans the df of the kdj from utils.formular into
backtestStrategy
```

2.8 第八个例子，bollingDemo

这个例子整合了日线分钟线混合计算，talib计算等功能，较为全面

- talib原生函数计算的方法如下：

```
import talib as ta
self.day_closeDataList.append(bar.getClose())
closeDataArray = np.array(self.day_closeDataList)
self.day_closeDataList.pop(-1)
self.UPPER,self.MIDDLE,self.LOWER =
ta.BBANDS(closeDataArray.astype('float'),20,2,2,0)
```

学习要点

- `self.day_closeDataList`为提取`dataSeries`的数据列，`dataSeries`对象包含2个`SequenceDataSeires`,一个时间列，一个数据列，虽然为内部对象，但是对于python仍然可以从外部进行访问内部对象和方法。

```
self.day_closeDataList =
self.day_feed[instrument].getCloseDataSeries()._SequenceDataSeries__val
ues.data()
```

- talib原生函数计算载入的是`np.array`,而且是float类型，一定不要忘记`closeDataArray.astype('float')`
- 提取出的`self.day_closeDataList`与pyalgotrade的`self.__closeDataSeries`共享同一片地址，而非复制，因此在计算实时指标，添加完当前bar的数据后，不要忘记再将该新增数据取出来。

  `self.day_closeDataList.pop(-1)`
- **本例子中，由于使用重采样的日线数据计算实时计算布林带，而第一日内，并无日线数据，即`self.day_closeDataList`为`None`，不能传入talib进行计算，必需将第一日数据过滤，否则报错。**

```
 def onBars(self, bars):
        if not self.initState:  # 如果日线MA均线有数值了才开始计算（此处过滤掉
了前59天的数据）
            if self.day_ma60.__len__() == 0:
                return
            if self.day_ma60[-1] is None:
                return
            else:
                self.initState = True
```

2.9 第九个例子，tickDemo

pyalgotrade适用于低频率的数据回测，建议为1分钟级别及以上的数据，为了支持tick级别，我给重新封装了层`tickBar` `tickBarDataSeries`等，使得tick级别能"凑活"使用，做这些工作的目的也是为了统一回测和实盘代码。**由于tick级交易的理念和分钟日线级别的就不同，所以无法使用原生的`enterPosition`或`marketOrder`等方法进行买卖**，我自己写了一下回测模块买卖的封装，限于时间原因未进行测试，感兴趣的可以继续这块工作。

- tick级别回测示例，数据须包含至少`ap1,av1,bp1,bv1,datetime`五个数据，其中`ap1`读取为pyalgotrade的`ap`，同理包括`av1,bp1,bv1`，其余的数据项自动加载不用管。`ap -> askPrice,bp -> bidPrice`

2.10 第十个例子，tushareDemo

这个例子为实时行情测试的例子，通过自动载入tushare行情进行测试。其核心为tushareLiveFeed，之前网上有人写了一个，感觉写的比较啰嗦，我又给重写了遍，策略模块与回测模式一样。

```
    strat = LiveDemo_run
    liveFeed = tushareLiveFeed.LiveFeed(['600848'], '5',
preload_start='2017-01-01')
    brk = backtesting.Broker(1000,liveFeed)
    strat = strat(liveFeed, brk,['600848'],3)
    strat.run()
```

LiveFeed参数如下,由于有些指标需要提前计算，所以添加一**参数`preload_start`预加载前面一段时间的数据。**

```
    :param identifiers: codes
    :param ktype: 同tushare一样，ktype：数据类型，D=日k线 W=周 M=月 5=5分钟 15=15
分钟 30=30分钟 60=60分钟，默认为D
    :param preload_start:若需要预加载前面的数据，则设置开始时间，同tushare的start，
str类型
    :param apiCallDelay:后面每5分钟后调用ts的延时，一般用不到30秒
    :param maxLen:
```

- **若需实盘，可以在这个例子的onBar下将pyalgotrade的broker替换为实盘接口即可。**

2.11 第十一个例子 tushareTickDemo

本例子实现tushare的tick级别livefeed读写，与tushare格式兼容，tick级别的与前面的不一样的地方在于策略需求少，就不预先加载数据了。

- 经测试，tushare 请求一条数据平均耗时0.3秒

2.12 第十二个例子，ctpDemo

ctp接口的python封装由我的好友@liqi同学所做，当时使用vnpy以及pyctp的封装接口的时候发现很多问题，可能vnpy的用户知道一些容易卡死的问题（ps，我没用过），就是底层封装没做好。其他问题我也说不好，后来同学自己给手工封装了遍，所有原生`C++`函数一个没落，应该是目前国内最强封装吧。有需求的可以研究下。

- 本demo和其他的关联不大，主要将自己测试过程中的经验分享出来，应该能帮助初学者省至少一个月半个月的时间。看懂了这个demo，使用ctp接口进行买卖应该非常容易了。
- **这个demo可以单独拿出来作为tick级期货交易的父类，也可以和pyalgotrade结合成为期货实盘接口。**

3 进阶篇-pyalgotrade深度解析

3.1 pyalgotrade数据结构

pyalgotrade从上往下的基本数据结构如下：

	datatype	dataStructure	description
0	bar	Struct	cell
1	SequenceDataSeries	ListDeque	限长列表
2	DataSeries	SequenceDataSeries	2个SequenceDataSeries
3	BarDataSeries	DataSeries	类dataFrame含有OHCL的DataSeries
4	BarFeed	DataSeries	带行切片的BarDataSeries
5	CsvFeed	..	更高级封装
6	DataFrameFeed	..	更高级封装

其他有空再写吧

4 其他篇 数据分析、接口等