扫地僧 Backtrader 给力教程系列一：

股票量化回测核心篇

作者：扫地僧，Email：saodiseng2020@163.com，微信：qtbhappy

纸上得来终觉浅，绝知此事要躬行！

——陆游

V1.1486

官网：https://quantcn.github.io/

QQ 答疑交流群

群名称:backtrader中国
群　号:1125384417

- 点这里在淘宝购买，或加微信 qtbhappy 联系购买本教程纸质版及源码。
- 若购买腾讯课堂视频教程“扫地僧 Backtrader 给力教程：量化回测核心篇（全集）”则免费赠送纸质版教程。
- 关注微信公众号 optMaster,查看高级专题
- 未来内容勘误及修订扩展请关注我们的知乎专栏：

目录

1. Backtrader 基础

本章介绍使用 Backtrader 回测投资策略的基本方法，以股票市场为例。本教程基于 windows 平台，采用的 Backtrader 版本是 1.9.74.123，采用的 python 程序编辑器是 visual studio code，采用的 python 版本为 python 3.6.8。

安装 Backtrader 只需要用命令 pip install backtrader 即可，如果速度太慢，可以使用如下安装命令，通过阿里云镜像安装：

pip install backtrader -i https://mirrors.aliyun.com/pypi/simple/

1.1. Backtrader 特性

Backtrader 是一款支持回测和实盘交易（中国实盘交易需要一定的定制）的功能丰富的 Python 框架，让用户专注于编写可重用的交易策略、指标和分析程序。它支持股票、期货、数字货币等市场的量化回测分析以及实盘交易。具有如下主要特性：

- 行情数据来源支持 csv 文件、在线来源、*pandas* 和 *blaze*
- 多数据多策略支持
- 支持混合不同时间粒度的数据
- 内置上百种技术指标
- 支持 TA-Lib 技术指标库
- 灵活定义佣金模式
- 集成的经纪行仿真
  - 订单类型：

    Market, *Close*, *Limit*, *Stop*, *StopLimit*, *StopTrail*, *StopTrailLimit*, *OCO*, bracket
  - 滑点 slippage
  - 对期货类的工具支持连续现金调整
- 交易日历支持

- 内置支持如下经纪行的实盘交易
  - Interactive Brokers
  - *Visual Chart*
  - *Oanda*

1.2. Backtrader 回测核心思想与“策略迭代表”

利用 Backtrader 进行量化投资的基本过程是这样的：

（1）提出策略。首先，提出一个投资策略，比如股票当日收盘时，股价上穿超过 5 日均线（即昨日收盘低于 5 日均线，本日收盘高于 5 日均线），则次日开盘买入 100 股（假设资金足够）；若收盘时股价跌破 5 日均线（即昨日收盘高于 5 日均线，本日收盘低于 5 日均线）且有持仓，则次日开盘全部抛出。

（2）在 Backtrader 中实现策略并进行回测。在 Backtrader 中实现该策略，为了验证该策略的有效性，要用历史行情数据测试该策略是否足够好。相当于回到过去的每一天，进行模拟交易，最后看效果如何。

（3）实盘交易：如果策略回测效果足够好，则投入实盘交易。

注：以上投资研究过程是简化的，实际上为了验证策略的有效性，还需要做更加精细的验证，但本节先做简化处理。

以下通过一个小案例来介绍 Backtrader 执行上述第（2）步回测的核心思想，同时介绍一些基本概念。以下是某股历史行情数据，记录了每日开盘、最高、最低、收盘和五日均线价格信息。

某股历史行情数据

Date	Open	High	Low	Close	5 Day Average	
1995-1-10	2.19	2.22	2.19	2.19		
1995-1-11	2.2	2.22	2.1	2.12		
1995-1-12	2.12	2.13	2.09	2.1		
1995-1-13	2.13	2.15	2.07	2.08		
1995-1-16	2.02	2.08	1.98	2.07	2.11	
1995-1-17	2.07	2.14	2.07	2.14	2.1	买入
1995-1-18	2.14	2.16	2.13	2.14	2.11	
1995-1-19	2.13	2.22	2.12	2.21	2.13	
1995-1-20	2.21	2.22	2.13	2.17	2.15	
1995-1-23	2.15	2.15	2.1	2.12	2.16	卖出
1995-1-24	2.11	2.14	2.09	2.12	2.15	

表中每一行记录在 Backtrader 中称为一个 bar（或 K 线），根据时间粒度不同，可以是日线 bar，周线 bar 或分钟线 bar 等，其实也就是一条 K 线相关的数据。假设用户已在 Backtrader 中实现了策略，行情数据也已加载到回测程序中，并且假设有初始资金 1000 元，买卖无佣金，Backtrader 是如何执行回测的呢？

我们把上表称为**策略迭代表**（它是理解 backtrader 迭代逻辑的核心概念，更复杂的涉及多周期、多股的策略迭代表参见 10.1.2，11.2），想象 Backtrader 首先将一个指针指向迭代表 1 月 16 号的 bar(因为从这一天才可以开始计算收盘价对 5 日均线的上穿或下穿)，触发策略类的 next 方法，该方法检查该日 bar 数据，发现收盘价未上穿 5 日均线无须动作。

然后指针迭代到 1 月 17 号的 bar,触发的 next 方法发现收盘价 2.14 上穿 5 日均线 2.10，则创建买入 100 股的订单。指针继续迭代到 1 月 18 日的 bar，买入 100 股的订单以开盘价 2.14 元执行，此时形成 100 股仓位，之后立即触发策略类 next 方法，该方法检查 18 日收盘价未发生上穿或下穿 5 日均线，所以不创建任何订单。指针继续迭代到 1 月 19 日的 bar，又触发 next 方法，无须需动作，直到指针顺次迭代到 1 月 23 日的 bar，触发的 next 方法检查到该日收盘下穿 5 日均线，故创建卖出 100 股订单。指针迭代到 1 月 24 日，以开盘价 2.11 元执行 100 股卖出订单，然后立即触发 next 方法，检查该日收盘价未发生上穿或下穿 5 日均线无须需动作。经过以上操作，原始的 1000 元最后变成了 907 元，说明该策略对这个案例是失败的。

这个小案例很好的说明了 Backtrader 回测的核心思想。Backtrader 会将行情数据加载到一个行情数据（Data Feed）对象，供用户编写的策略类使用。在策略类里，有一个核心

的方法 next，**每当指针移到一个新的行情 bar（该 bar 就成为当前 bar，对应的日期称为当前日期或当前时间）**，会自动触发 next 方法，用户可以在该方法中判断该 bar 的收盘价对 5 日均线的上穿下穿情况，进而执行各种动作。

1.3. Backtrader 回测基本步骤

了解了以上 Backtrader 回测的核心思想后，本节分析一个较为完整的代码案例，具体研究 Backtrader 回测的基本步骤。假设我们要回测一个策略，该策略根据日 k 线确定某单一股票的买卖时机：当收盘价上穿 5 日均线时，若无仓位，则市价买入 100 股；当收盘价下破 5 日均线时，则市价卖出 100 股，初始股票仓位为 0，初始资金 10000 元。

1.3.1. 前复权还是后复权

所用行情数据来自一个 csv 文件（600000qfq.csv），其部分内容如下图，股票价格是已经前复权的价格。为了保证数据前后一致性，要采用复权数据进行回测。

	A	B	C	D	E	F	G	H	I	J	K	L
1		ts_code	trade_date	open	high	low	close	pre_close	change	pct_chg	vol	amount
2	4852	600000.SH	20000104	1.9649	2.0278	1.9468	2.0113	1.9468	0.0645	3.3131	44961	113946.8
3	4851	600000.SH	20000105	2.0113	2.0436	1.9783	1.9885	2.0113	-0.0228	-1.1336	52528	134465.4
4	4850	600000.SH	20000106	1.9806	2.0687	1.9704	2.0444	1.9885	0.0559	2.8112	62297	160059.8
5	4849	600000.SH	20000107	2.0687	2.1631	2.0546	2.1159	2.0444	0.0715	3.4974	213553	575751.1
6	4848	600000.SH	20000110	2.1238	2.1907	2.101	2.1435	2.1159	0.0276	1.3044	165397	450453.5
7	4847	600000.SH	20000111	2.1435	2.1474	2.0546	2.0609	2.1435	-0.0826	-3.8535	93908	251055.9
8	4846	600000.SH	20000112	2.0452	2.0452	1.9508	1.9759	2.0609	-0.085	-4.1244	352749	889867.3
9	4845	600000.SH	20000113	1.9665	1.9862	1.9508	1.9586	1.9759	-0.0173	-0.8756	79756	199244.3
10	4844	600000.SH	20000114	1.9571	1.9665	1.8808	1.9036	1.9586	-0.055	-2.8081	178619	434206.1
11	4843	600000.SH	20000117	1.8941	1.9224	1.8682	1.9193	1.9036	0.0157	0.8248	81015	194777
12	4842	600000.SH	20000118	1.9272	1.9327	1.8784	1.8988	1.9193	-0.0205	-1.0681	76933	185091.7
13	4841	600000.SH	20000119	1.8988	1.9106	1.8863	1.8981	1.8988	-0.0007	-0.0369	46584	112325.2
14	4840	600000.SH	20000120	1.8973	1.939	1.8957	1.9224	1.8981	0.0243	1.2802	51144	124668.2
15	4839	600000.SH	20000121	1.9279	1.9374	1.8941	1.9091	1.9224	-0.0133	-0.6918	81387	197523.2
16	4838	600000.SH	20000124	1.9036	1.9075	1.8839	1.902	1.9091	-0.0071	-0.3719	92503	222404.4
17	4837	600000.SH	20000125	1.902	1.9791	1.902	1.9342	1.902	0.0322	1.693	77423	191183
18	4836	600000.SH	20000126	1.935	1.9508	1.9122	1.9201	1.9342	-0.0141	-0.729	27377	67037.92
19	4835	600000.SH	20000127	1.9248	1.9492	1.9036	1.9342	1.9201	0.0141	0.7343	63350	154926.9
20	4834	600000.SH	20000128	1.935	1.9744	1.9319	1.9696	1.9342	0.0354	1.8302	123627	307633.6
21	4833	600000.SH	20000214	2.0452	2.1616	2.0058	2.1395	1.9696	0.1699	8.6261	237391	628039
22	4832	600000.SH	20000215	2.1395	2.1616	2.0176	2.02	2.1395	-0.1195	-5.5854	262677	693158.5
23	4831	600000.SH	20000216	2.0058	2.0098	1.9594	1.9822	2.02	-0.0378	-1.8713	190691	478863.3

特别提醒，**实际回测时，应该采用后复权数据**进行回测，这样得到的回测结果收益率相当于分红没有再投资的收益率。当有现金分红时，采用前复权回测的结果收益率很可能不对，而且有可能出现负数价格。本教程中许多案例采用前复权数据，只是为了方便看最后价格与实际价格一致，但实际回测时不能用前复权。

1.3.2. 回测步骤

以下是回测程序代码：

first.py

```
from datetime import datetime
import backtrader as bt
import os.path  # 管理路径
import sys  # 发现脚本名字(in argv[0])
# 创建策略类
class SmaCross(bt.Strategy):
    # 定义参数
    params = dict(period=5  # 移动平均期数
                  )
    def __init__(self):
        # 移动平均线指标
        self.move_average = bt.ind.MovingAverageSimple(
            self.datas[0].close, period=self.params.period)
    def next(self):
        if not self.position.size:  # 还没有仓位
            # 当日收盘价上穿 5 日均线，创建买单，买入 100 股
            if self.datas[0].close[-1] < self.move_average.sma[
                    -1] and self.datas[0].close[0] > self.move_average.sma[0]:
                self.buy(size=100)
        # 有仓位，并且当日收盘价下破 5 日均线，创建卖单，卖出 100 股
        elif self.datas[0].close[-1] > self.move_average.sma[-1] and self.datas[
                0].close[0] < self.move_average.sma[0]:
            self.sell(size=100)
##########################
# 主程序开始
##########################
# 创建大脑引擎对象
cerebro = bt.Cerebro()
# 获取本脚本文件所在路径
modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
# 拼接得到数据文件全路径
datapath = os.path.join(modpath, './600000qfq.csv')
# 创建行情数据对象，加载数据
data = bt.feeds.GenericCSVData(
    dataname=datapath,
    datetime=2,  # 日期行所在列
    open=3,  # 开盘价所在列
```

```
    high=4,  # 最高价所在列
    low=5,  # 最低价所在列
    close=6,  # 收盘价所在列
    volume=10,  # 成交量所在列
    openinterest=-1,  # 无未平仓量列.(openinterest 是期货交易使用的)
    dtformat=('%Y%m%d'),  # 日期格式
    fromdate=datetime(2019, 1, 1),  # 起始日
    todate=datetime(2020, 7, 8))  # 结束日
cerebro.adddata(data)  # 将行情数据对象注入引擎
cerebro.addstrategy(SmaCross)  # 将策略注入引擎
cerebro.broker.setcash(10000.0)  # 设置初始资金
cerebro.run()  # 运行
print('最终市值: %.2f' % cerebro.broker.getvalue())
cerebro.plot(style='bar')
```

上述代码展示了 Backtrader 回测的基本步骤，解释如下:

（1） 创建策略类

程序一开始创建了策略类 SmaCross，实现了本节开头提到的策略，具体细节稍后讨论。

（2） 创建 Cerebro 大脑引擎对象

cerebro = bt.Cerebro 语句创建 **cerebro 引擎对象**（cebrebro 是大脑的意思），该对象负责协调回测涉及的各个组件的活动。

（3） 创建行情数据对象并注入引擎

data = bt.feeds.GenericCSVData 句创建行**情数据对象 data（后面简称数据对象，亦称数据馈送对象）**，该对象从一个 csv 文件获取行情数据，读取 2019 年 1 月 1 日到 2020 年 7 月 8 日的行情数据。然后用 cerebro.adddata 语句将数据对象注入引擎。注意该函数参数指出了开盘价、收盘价等在 csv 文件对应的列号（列从 0 开始编号）。如果列号-1，则说明该列在 csv 文件不存在。创建好的数据对象通过语句 cerebro.adddata(data) 注入引擎，即可被策略使用。

（4） 注入策略到引擎

cerebro.addstrategy 将策略注入引擎，引擎内将实例化策略对象。

（5） 设置初始资金

cerebro.broker.setcash 设置初始资金。以后还需要设置佣金、滑点等。

（6） 执行回测

cerebro.run 执行回测，最后 cerebro.broker.getvalue 输出结果市值。

注：MovingAverageSimple 类有两个别名：SMA, SimpleMovingAverage，可以用别名替换上面代码中的 MovingAverageSimple。

另外，还用 cerebro.plot(style='bar')画出 bar 图。此句还可写为 cerebro.plot(style='candlestick')，cerebro.plot()，读者可自行尝试观察效果。

注：运行上述程序，调用 cerebro.plot()命令时，若出现如下错误：

cannot import name 'warnings' from 'matplotlib.dates'

很可能是 matplotlib 版本不兼容。backtrader 与 matplot 3.3 不兼容，要降级到 3.2，可运行如下命令可降级：

pip uninstall matplotlib

pip install matplotlib==3.2.2

1.4. 策略类运行逻辑分析

在上述代码中，cerebro.run 命令会激发策略类 SmaCross 执行其回测逻辑。

（1） init 方法

首先执行策略类的初始化方法 init，该方法只执行一次，这里定义了一个 5 日移动平均线指标。

（2） next 方法

初始化方法执行完后，策略会在行情数据上从远到近进行迭代，每迭代到一个新 bar（该 bar 成为当前 bar，对应的日期成为当前日），就激活一次 next 方法。

在 next 方法中，检查如果没有仓位，再判断若前一日收盘价小于本日 5 日均线价，并且本日收盘价大于本日 5 日均线价（意味着收盘价上穿 5 日均线），则创建市价买单，买 100 股，该市价买单将在下一个 bar 开始的时候以该 bar 的开盘价执行。

若已有仓位，并且本日收盘价下穿 5 日均线，则创建市价卖单，卖 100 股，该卖单将在下一个 bar 开始的时候（进入 next 方法前）以该 bar 的开盘价执行。

以上就是策略类的基本逻辑，非常简单清晰。还有一点需要指出的是，next 方法中会自动跳过不必要的 bar，比如，由于使用的是 5 日均线，因此前 4 个 bar 被跳过，从第 5 个

bar 才有 5 日均线值，next 方法会自动从第 5 个 bar 开始迭代，简化了用户编程工作。此外，上述订单类型默认都是市价订单，要想创建其他类型订单，请参考第 8 章。

1.5. 核心概念：line

以下解释策略中用到的一些重要概念，特别是线(line)相关的概念。

1.5.1. self.datas[0]

这是策略类的一个重要属性，它指向 cerebro 注入的第一个行情数据对象（如果注入了两个，则 self.datas[1]指向第二个数据对象），因此，它也是行情数据对象，可以设想其指向的数据如下表所示（注意,表中每一行是一个 bar）。

	A	B	C	D	E	F	G
1		ts_code	trade_date	open	high	low	close
2	4852	600000.SH	20000104	1.9649	2.0278	1.9468	2.0113
3	4851	600000.SH	20000105	2.0113	2.0436	1.9783	1.9885
4	4850	600000.SH	20000106	1.9806	2.0687	1.9704	2.0444
5	4849	600000.SH	20000107	2.0687	2.1631	2.0546	2.1159
6	4848	600000.SH	20000110	2.1238	2.1907	2.101	2.1435
7	4847	600000.SH	20000111	2.1435	2.1474	2.0546	2.0609
8	4846	600000.SH	20000112	2.0452	2.0452	1.9508	1.9759
9	4845	600000.SH	20000113	1.9665	1.9862	1.9508	1.9586
10	4844	600000.SH	20000114	1.9571	1.9665	1.8808	1.9036
11	4843	600000.SH	20000117	1.8941	1.9224	1.8682	1.9193
12	4842	600000.SH	20000118	1.9272	1.9327	1.8784	1.8988
13	4841	600000.SH	20000119	1.8988	1.9106	1.8863	1.8981
14	4840	600000.SH	20000120	1.8973	1.939	1.8957	1.9224
15	4839	600000.SH	20000121	1.9279	1.9374	1.8941	1.9091
16	4838	600000.SH	20000124	1.9036	1.9075	1.8839	1.902
17	4837	600000.SH	20000125	1.902	1.9791	1.902	1.9342
18	4836	600000.SH	20000126	1.935	1.9508	1.9122	1.9201
19	4835	600000.SH	20000127	1.9248	1.9492	1.9036	1.9342
20	4834	600000.SH	20000128	1.935	1.9744	1.9319	1.9696
21	4833	600000.SH	20000214	2.0452	2.1616	2.0058	2.1395
22	4832	600000.SH	20000215	2.1395	2.1616	2.0176	2.02
23	4831	600000.SH	20000216	2.0058	2.0098	1.9594	1.9822

1.5.2. 线 line

表中每一列在 Backtrader 中称为一条线 **line**，线由一系列数据点组成。比如 close 线由一系列收盘价构成。这里有 close line（收盘线）、open line（开盘线）、high line（最高价线）、low line（最低价线）等。

1.5.3. 含线对象（含 line 对象）

含有一条或多条线的对象称为含线对象，含线对象有一个类似列表的属性 lines，其中含有一条或多条线。

行情数据对象（或称数据馈送对象 data feed）如 self.datas[0]，指标对象如 self.move_average 都是含 line 对象，策略自己 self 也是含 line 对象。

简写：Backtrader 允许将 self.datas[0]写成 self.data。

1.5.4. 线对象（line 对象）

含线对象的 lines 列表中含有指向具体 line 数据的 line 对象，可通过索引访问其中的线对象（亦可简称线），如 self.datas[0].lines[0]就可访问含线对象中的 0 号线。

更常见的是通过线名称直接访问线对象，例如 self.datas[0].lines.close 可访问收盘线对象。

如果想知道含线对象具体含有哪些线，可通过 lines 的方法 getlinealiases 获取，如 self.datas[0].lines.getlinealiases()通常返回元组('close', 'low', 'high', 'open', 'volume', 'openinterest', 'datetime')，这就是行情数据对象 self.datas[0]含的所有线名称。

self.move_average.lines.getlinealiases()返回('sma')，只含一根线 sma。

策略自身所含 line 由 self.lines.getlinealiases()得到，返回('datetime',)，即策略自身只含一条 datetime 线。

在 next 方法中，可用 len(self)得到当前已经处理的 bar 的数目，它实际上统计的是 datetime 线到从最早到当前 bar 的长度。

简写：在策略类中，用名称访问数据对象的线时，允许省略 lines，因此 self.datas[0].lines.close 可简写为 self.datas[0].close，还可进一步简写为 self.data.close，省略 lines 是推荐的写法。用名称访问指标的线对象时，也可省略 lines，如 self.move_average.lines.sma 可简写为 self.move_average.sma。

1.5.5. 在 next 方法中通过索引访问 line 中的点

可在策略类的 next 方法中，通过索引访问线中的点，例如 self.datas[0].close[0]访问 close 线中当前 bar 的收盘价，索引-1 访问上一个 bar 的收盘价，1 访问下一个 bar 的收盘价。注意，在 next 方法中不能访问线的整体。

1.5.6. 在 next 方法中访问 datetime 线中的点

如 1.5.4 所述，策略本身含有 datetime 线，线上的每个点代表一个日期时间，在 next 方法中，通过 self.datetime[0]访问策略迭代表当前 bar 的日期时间(注意：**默认为该 bar 的结束时间**)，但该值是 float 类型，可以用 backtrader 的函数 bt.num2date(self.datetime[0])转成普通的 python 日期时间，转换后的值类似 2020-07-03 23:59:59.999989。这样处理比较麻烦，建议通过另一种简单的方法访问当前日期时间，无需转换，即 self.datetime.datetime(0)，而 self.datetime.datetime(-1)可以取得上一根 bar 的日期时间。

对行情数据对象的 datetime 线，也可进行类似操作。总结如下表。

针对策略	
访问日期时间 self.datetime.datetime(0)	取得年月日 self.datetime.datetime(0).year self.datetime.datetime(0).month self.datetime.datetime(0).day
访问日期 self.datetime.date(0)	self.datetime.date(0).year self.datetime.date(0).month self.datetime.date(0).day
针对行情数据对象	
self.datas[0].datetime.datetime(0)	self.datas[0].datetime.datetime(0).year self.datas[0].datetime.datetime(0).month self.datas[0].datetime.datetime(0).day
self.datas[0].datetime.date(0)	self.datas[0].datetime.date(0).year

	self.datas[0].datetime.date(0).month self.datas[0].datetime.date(0).day

这里要**注意一个特殊点，最后用的是圆括号，而不是方括号**。

1.5.7. 在 init 中访问 line 整体，并进行线整体运算

在 init 方法中，可用类似 self.datas[0].close 访问线的整体。在 init 方法中，经常对不同线或含线对象进行整体的加减等操作，结果变量可以当作 line 对象使用。比如，在 init 中，可以定义如下变量：

```
self.dif = self.data.close - self.move_average.sma
```

则 dif 可以当作线对象使用，其值是收盘价和 5 日均线的差价系列。这种整体操作是一种矢量化操作，性能极快，通常用于构造新指标。此后在 next 中，可以像一般线对象那样用 self.dif[0]访问 dif 的当前值。

注意，不能在 init 方法中访问线中的点。

1.5.8. init 方法中，含线对象可当作其默认线对象用

含线对象 lines 列表里的第一根线是其默认线，例如 self.datas[0]的 lines 列表里第一根线收盘线 close 是默认线。在策略的 init 方法中，含线对象 self.data 或 self.datas[0]可替换其默认线对象 self.data[0].close。所以如下三行代码是等效的：

```
self.move_average = bt.ind.MovingAverageSimple(self.datas[0].close, self.params.period =5)
self.move_average = bt.ind.MovingAverageSimple(self.datas[0], self.params.period =5)
self.move_average = bt.ind.MovingAverageSimple(self.data, self.params.period =5)
```

上面的 MovingAverageSimple 类的第一个参数要求线对象，但用含线对象替换也行。

也可省略第一个参数，写成：

self.move_average = bt.ind.MovingAverageSimple(self.params.period =5)，这样会默认使用 self.datas[0].close 作为其第一个参数。

1.5.9. next 中含线对象作默认线当前值用，线对象可作当前值用

比如，在 next 方法中，含线对象 self.data、self.datas[0]以及线对象 self.data.close、self.datas[0].close 都可替换 self.datas[0].close[0]。而 self.move_average、self.move_average.sma 可替换 self.move_average.sma[0]。如下 next 中的代码是等效的：

```
if self.data.close[0] < self.move_average.sma[0]
if self.data.close < self.move_average.sma
if self.data < self.move_average
```

可见后面的写法减少代码量，**推荐在 next 中采用后面那种最简洁的代码形式**。如果采用简化的写法，前述 first.py 的策略类代码可以写成如下较简洁的形式。

shortcut.py

```
from datetime import datetime
import backtrader as bt
import os.path  # 管理路径
import sys  # 发现脚本名字(in argv[0])
# 创建策略类
class SmaCross(bt.Strategy):
    # 定义参数
    params = dict(period=5  # 移动平均期数
                  )

    def __init__(self):
        # 移动平均线指标
        self.move_average = bt.ind.MovingAverageSimple(
            self.data, period=self.params.period)
    def next(self):
        if not self.position:  # 还没有仓位
            # 当日收盘价上穿 5 日均线，创建买单，买入 100 股
            if self.data.close[
                    -1] < self.move_average[-1] and self.data > self.move_average:
                self.buy(size=100)
        # 有仓位，并且当日收盘价下破 5 日均线，创建卖单，卖出 100 股
        elif self.data.close[
                -1] > self.move_average[-1] and self.data < self.move_average:
            self.sell(size=100)
##########################
```

```
# 主程序开始
#########################
# 创建大脑引擎对象
cerebro = bt.Cerebro()
# 获取本脚本文件所在路径
modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
# 拼接得到数据文件全路径
datapath = os.path.join(modpath, './600000qfq.csv')
# 创建行情数据对象，加载数据
data = bt.feeds.GenericCSVData(
    dataname=datapath,
    datetime=2,  # 日期行所在列
    open=3,  # 开盘价所在列
    high=4,  # 最高价所在列
    low=5,  # 最低价所在列
    close=6,  # 收盘价所在列
    volume=10,  # 成交量所在列
    openinterest=-1, # 无未平仓量列.(openinterest 是期货交易使用的)
    dtformat=('%Y%m%d'),  # 日期格式
    fromdate=datetime(2019, 1, 1),  # 起始日
    todate=datetime(2020, 7, 8))  # 结束日
cerebro.adddata(data)  # 将行情数据对象注入引擎
cerebro.addstrategy(SmaCross)  # 将策略注入引擎
cerebro.broker.setcash(10000.0)  # 设置初始资金
cerebro.run()  # 运行
print('最终市值: %.2f' % cerebro.broker.getvalue())
```

1.5.10. 线的长度

如前所述，线由一系列的数据点组成，策略在策略迭代表上迭代过程中，点的个数动态增长，实际上，点的个数就是策略迭代到当前 bar 的 bar 数，也称为线的长度。在策略 next 方法中可用函数 len(含线对象)取得当前该含线对象的线长。这里的含线对象可以是数据馈送对象、策略对象自己或者指标对象。

对数据馈送对象，如果数据是预加载的（即一次性将所有数据都注入到策略，而不是动态加载数据），则可用方法 buflen 取得其总长度，因此，在策略类的 next 方法中，代码 len(self.datas[0])是当前已处理的数据长度，会动态增长，而 self.datas[0].buflen()是数据线的总长度，不会变化。