Backtrader 中文文档

框架的使用

让我们通过一系列的例子（从几乎什么都没有到完全成熟的策略），在使用 backtrader 的过程中大致一下解释 2 个基本概念。

Line

价格数据（Data Feeds）、技术指标（Indicators）和策略（Strategies）都是 *Line*。

“Line” 是由一系列的点组成的。典型的价格数据，通常由以下类别的数据组成：

- Open, High, Low, Close, Volume, OpenInterest
  开盘价、最高价、最低价、收盘价、成交量、持仓量

价格数据中的所有”Open” (开盘价)按时间组成一条 Line。所以，一组含有以上 6 个类别的价格数据，共有 6 条 Line。

如果我们也算上“DateTime”（时间，可以看作是一组数据的主键），一共有 7 条 Line。

下标 0

当访问一条 Line 的数据时，会默认指向下标为 *0* 的数据。

最后一个数据通过下标 *-1* 来访问。这也设计是为了符合 Python 的迭代器规则（一条 Line 可以被迭代，因此也是 iterable）。

在-1 之后是索引 0，它用于访问当前时刻。

假设在创建策略的过程中，在初始化期间创建具有简单移动平均值的策略：

```
self.sma = SimpleMovingAverage(.....)
```

访问此移动平均线的当前值的最简单方法：

```
av = self.sma[0]
```

在回测过程中，无需知道已经处理了多少条/分钟/天/月，”0”一直指向当前值。
遵循典型的 Python 传统，用下标 *-1* 来访问最后一个值：

```
previous_value = self.sma[-1]
```

当然，当然-2、-3 下标也是可以照常使用。

从 0 到 100: 示例
基本步骤

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import backtrader as bt
if __name__ == '__main__':
    cerebro = bt.Cerebro()
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    cerebro.run()
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

执行后输出为：

```
Starting Portfolio Value: 10000.00Final Portfolio Value: 10000.00
```

在这个例子中：

- 引入了 backtrader
- 实例化了 Cerebro 引擎
- 调用 了 *cerebro* 实例的 *run* 方法（里面会遍历数据）
- 输出了运行结果

虽然看起来不多，但让我们指出明确显示的内容：

- Cerebro 引擎在后台创建了一个 Broker (代理)实例
- 该 Broker 实例开始会有一些现金

引入 broker 经纪人的概念，可以简化用户的理解和使用。 如果用户未设置代理，则会设置默认的代理。

默认情况线，系统提供了 1 万块钱来开始交易。

设置起始现金

在金融领域，肯定只有“loser”（失败者）才从 1 万开始。 让我们多投点钱再次运行这个示例。

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import backtrader as bt
if __name__ == '__main__':
```

```
    cerebro = bt.Cerebro()
    cerebro.broker.setcash(100000.0)
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    cerebro.run()
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

执行完毕，结果变成：

```
Starting Portfolio Value: 1000000.00Final Portfolio Value: 1000000.00
```

加载价格数据

拥有现金很有趣，但所有这一切背后的目的是让一个自动化策略通过操作我们视为数据源的资产来增加现金，

...没有数据流 ->没有乐趣。让我们给程序喂点数据。

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import datetime  # For datetime objectsimport os.path  # To manage pathsim
port sys  # To find out the script name (in argv[0])
# Import the backtrader platformimport backtrader as bt
if __name__ == '__main__':
    # Create a cerebro entity
    cerebro = bt.Cerebro()
    # Datas are in a subfolder of the samples. Need to find where the scrip
t is
    # because it could have been called from anywhere
    modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
    datapath = os.path.join(modpath, '../../datas/orcl-1995-2014.txt')

    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        # Do not pass values before this date
        fromdate=datetime.datetime(2000, 1, 1),
        # Do not pass values after this date
        todate=datetime.datetime(2000, 12, 31),
        reverse=False)
```

```
    # Add the Data Feed to Cerebro
    cerebro.adddata(data)
    # Set our desired cash start
    cerebro.broker.setcash(100000.0)
    # Print out the starting conditions
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    # Run over everything
    cerebro.run()
    # Print out the final result
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

执行完后的输出是：

```
Starting Portfolio Value: 1000000.00Final Portfolio Value: 1000000.00
```

代码量略有增加，因为我们添加了：

- 加载我们的示例脚本里面价格数据文件的位置，
- 使用 datetime 来过滤*价格数据* 的起止范围

除此之外，价格数据流已创建并添加到 cerebro 引擎中。

输出没有改变，如果有的话，那才是奇迹。

注意:
Yahoo 的价格数据非主流，它是以时间倒序排列的。*reversed=True* 参数将顺序 *反转* 一次，这样就得到了我们想要的正序数据。

第一个策略

钱已经给了 broker （经纪人），价格数据 也已经载入了。万事俱备。
让我们给策略加点代码，让他打印出每天的"收盘价"。
DataSeries (K 线类的父类) 能够直接访问到 OHLC (开盘价、最高价、最低价、收盘价) 数据。这使我们打印数据很方便。

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
```

```
import datetime  # For datetime objectsimport os.path  # To manage pathsim
port sys  # To find out the script name (in argv[0])
# Import the backtrader platformimport backtrader as bt
# Create a Strateyclass TestStrategy(bt.Strategy):

    def log(self, txt, dt=None):
        ''' Logging function for this strategy'''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # Keep a reference to the "close" line in the data[0] dataseries
        self.dataclose = self.datas[0].close

    def next(self):
        # Simply log the closing price of the series from the reference
        self.log('Close, %.2f' % self.dataclose[0])

if __name__ == '__main__':
    # Create a cerebro entity
    cerebro = bt.Cerebro()

    # Add a strategy
    cerebro.addstrategy(TestStrategy)

    # Datas are in a subfolder of the samples. Need to find where the scrip
t is
    # because it could have been called from anywhere
    modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
    datapath = os.path.join(modpath, '../../datas/orcl-1995-2014.txt')

    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        # Do not pass values before this date
```

```
        fromdate=datetime.datetime(2000, 1, 1),
        # Do not pass values before this date
        todate=datetime.datetime(2000, 12, 31),
        # Do not pass values after this date
        reverse=False)

    # Add the Data Feed to Cerebro
    cerebro.adddata(data)

    # Set our desired cash start
    cerebro.broker.setcash(100000.0)

    # Print out the starting conditions
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())

    # Run over everything
    cerebro.run()

    # Print out the final result
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

运行后的输出结果为：

```
Starting Portfolio Value: 100000.002000-01-03T00:00:00, Close, 27.852000-0
1-04T00:00:00, Close, 25.392000-01-05T00:00:00, Close, 24.05.........2000-
12-26T00:00:00, Close, 29.172000-12-27T00:00:00, Close, 28.942000-12-28T00:
00:00, Close, 29.292000-12-29T00:00:00, Close, 27.41Final Portfolio Value:
 100000.00
```

刚才程序的逻辑：

- 通过 cerebro.adddata(data) 加载多条价格数据。在策略类中的 __init__ 方法中，可以通过 self.datas 访问到所有的价格数据。
  self.datas 是一个标准的 Python 列表，内容就是加载的一条或多条价格数据，列表的顺序就是价格数据加载的顺序。
  self.datas[0] 即是加载的第一条价格数据，它被框架默认使用。
- 由于只需访问收盘价数据，于是使用 self.dataclose = self.datas[0].close 将第一条价格数据的 收盘价 Line 赋值给新变量。

- 现在开始循环处理价格数据，当经过一个 K 线柱的时候 next() 方法就会被调用一次。当然中间也会处理 技术指标 相关逻辑，后边会讲到。

给策略加点逻辑

让我们观察一下图线，来点实际的交易。

- 如果价格三连跌的话，买买买！

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import datetime  # For datetime objectsimport os.path  # To manage pathsim
port sys  # To find out the script name (in argv[0])
# Import the backtrader platformimport backtrader as bt
# Create a Strateyclass TestStrategy(bt.Strategy):

    def log(self, txt, dt=None):
        ''' Logging function fot this strategy'''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # Keep a reference to the "close" line in the data[0] dataseries
        self.dataclose = self.datas[0].close

    def next(self):
        # Simply log the closing price of the series from the reference
        self.log('Close, %.2f' % self.dataclose[0])

        if self.dataclose[0] < self.dataclose[-1]:
            # current close less than previous close
            if self.dataclose[-1] < self.dataclose[-2]:
                # previous close less than the previous close
                # BUY, BUY, BUY!!! (with all possible default parameters)
                self.log('BUY CREATE, %.2f' % self.dataclose[0])
```

```
                self.buy()


if __name__ == '__main__':
    # Create a cerebro entity
    cerebro = bt.Cerebro()
    # Add a strategy
    cerebro.addstrategy(TestStrategy)

    # Datas are in a subfolder of the samples. Need to find where the scrip
t is
    # because it could have been called from anywhere
    modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
    datapath = os.path.join(modpath, '../../datas/orcl-1995-2014.txt')

    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        # Do not pass values before this date
        fromdate=datetime.datetime(2000, 1, 1),
        # Do not pass values before this date
        todate=datetime.datetime(2000, 12, 31),
        # Do not pass values after this date
        reverse=False)
    # Add the Data Feed to Cerebro
    cerebro.adddata(data)
    # Set our desired cash start
    cerebro.broker.setcash(100000.0)
    # Print out the starting conditions
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    # Run over everything
    cerebro.run()
    # Print out the final result
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

运行之后的输出是:

```
Starting Portfolio Value: 100000.002000-01-03, Close, 27.852000-01-04, Clos
e, 25.392000-01-05, Close, 24.052000-01-05, BUY CREATE, 24.052000-01-06, Clo
se, 22.632000-01-06, BUY CREATE, 22.632000-01-07, Close, 24.37.........2000
-12-20, BUY CREATE, 26.882000-12-21, Close, 27.822000-12-22, Close, 30.06200
0-12-26, Close, 29.172000-12-27, Close, 28.942000-12-27, BUY CREATE, 28.9420
00-12-28, Close, 29.292000-12-29, Close, 27.41Final Portfolio Value: 99725.0
8 若干个买入操作被执行，我们的余额也在减少。有些重要的事情还没理清。

交易在何时以什么价格执行的？
```

在下一个例子中，将会打印执行的结果。

细心的可能会问，买了多少？买的什么？订单怎么被执行的？BackTrader 框架替我们做了这些事：

- 如果没有指定的话，self.datas[0] (即主价格数据) 即是标的物
- 交易数量由 仓位数量 参数默认指定了，默认指定为"1"了，后面例子我们会修改
- 订单被以"市价"成交了。 Broker （经纪人，之前提到过）使用了下一个 K 线柱的开盘价，因为你在当前柱的收盘提交的订单，下一柱的开盘价是他接触到的第一个价格
- 没有为订单设置佣金费，后边会加上

不止买入，还要卖出

知道了如何买入，就需要说一下卖出了：

Strategy 类有一个变量 position 保存当前持有的资产数量
buy() 和 sell() 会返回 被创建的订单 (还未被执行的)
订单状态改变后将会通知 Strategy 实例的 notify() 方法

"卖出" 逻辑也很简单：

- 5 个柱之后（在第 6 个时候执行）不管涨跌都卖
- 请注意，这里没有指定具体时间，而是指定的柱的数量。一个柱可能代表 1 分钟、1 小时、1 天、1 星期等等，这取决于你价格数据文件里一条数据代表的周期。
- 虽然我们心里知道每个柱代表一天，但策略不知道也不关心。

还有一条：

- 当还有头寸的时候，不再买入

注意没有柱的下标传给 next() 方法，那它是怎么知道已经经过了 5 个柱了呢？ 这里用了一个很 Python 的方式：调用 len() 获取它 Line 的长度。
交易发生时记下它的长度，后边比较大小，看是否经过了 5 个柱。

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import datetime  # For datetime objectsimport os.path  # To manage pathsimpo
rt sys  # To find out the script name (in argv[0])
# Import the backtrader platformimport backtrader as bt
# Create a Strateyclass TestStrategy(bt.Strategy):
    def log(self, txt, dt=None):
        ''' Logging function fot this strategy'''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # Keep a reference to the "close" line in the data[0] dataseries
        self.dataclose = self.datas[0].close

        # To keep track of pending orders
        self.order = None

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # Buy/Sell order submitted/accepted to/by broker - Nothing to do
            return

        # Check if an order has been completed
        # Attention: broker could reject order if not enough cash
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log('BUY EXECUTED, %.2f' % order.executed.price)
            elif order.issell():
                self.log('SELL EXECUTED, %.2f' % order.executed.price)
```

```
            self.bar_executed = len(self)

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')
        # Write down: no pending order
        self.order = None

    def next(self):
        # Simply log the closing price of the series from the reference
        self.log('Close, %.2f' % self.dataclose[0])
        # Check if an order is pending ... if yes, we cannot send a 2nd one
        if self.order:
            return
        # Check if we are in the market
        if not self.position:
            # Not yet ... we MIGHT BUY if ...
            if self.dataclose[0] < self.dataclose[-1]:
                    # current close less than previous close
                    if self.dataclose[-1] < self.dataclose[-2]:
                        # previous close less than the previous close
                        # BUY, BUY, BUY!!! (with default parameters)
                        self.log('BUY CREATE, %.2f' % self.dataclose[0])
                        # Keep track of the created order to avoid a 2nd order
                        self.order = self.buy()
        else:
            # Already in the market ... we might sell
            if len(self) >= (self.bar_executed + 5):
                # SELL, SELL, SELL!!! (with all possible default parameters)
                self.log('SELL CREATE, %.2f' % self.dataclose[0])
                # Keep track of the created order to avoid a 2nd order
                self.order = self.sell()
if __name__ == '__main__':
    # Create a cerebro entity
    cerebro = bt.Cerebro()
    # Add a strategy
```

```
    cerebro.addstrategy(TestStrategy)
    # Datas are in a subfolder of the samples. Need to find where the script
is
    # because it could have been called from anywhere
    modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
    datapath = os.path.join(modpath, '../../datas/orcl-1995-2014.txt')
    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        # Do not pass values before this date
        fromdate=datetime.datetime(2000, 1, 1),
        # Do not pass values before this date
        todate=datetime.datetime(2000, 12, 31),
        # Do not pass values after this date
        reverse=False)
    # Add the Data Feed to Cerebro
    cerebro.adddata(data)
    # Set our desired cash start
    cerebro.broker.setcash(100000.0)
    # Print out the starting conditions
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    # Run over everything
    cerebro.run()
    # Print out the final result
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

运行之后的输出为:

```
Starting Portfolio Value: 100000.002000-01-03T00:00:00, Close, 27.852000-01
-04T00:00:00, Close, 25.392000-01-05T00:00:00, Close, 24.052000-01-05T00:00:
00, BUY CREATE, 24.052000-01-06T00:00:00, BUY EXECUTED, 23.612000-01-06T00:
00:00, Close, 22.632000-01-07T00:00:00, Close, 24.372000-01-10T00:00:00, Cl
ose, 27.292000-01-11T00:00:00, Close, 26.492000-01-12T00:00:00, Close, 24.9
02000-01-13T00:00:00, Close, 24.772000-01-13T00:00:00, SELL CREATE, 24.7720
00-01-14T00:00:00, SELL EXECUTED, 25.702000-01-14T00:00:00, Close, 25.1
8.........2000-12-15T00:00:00, SELL CREATE, 26.932000-12-18T00:00:00, SELL
EXECUTED, 28.292000-12-18T00:00:00, Close, 30.182000-12-19T00:00:00, Close,
 28.882000-12-20T00:00:00, Close, 26.882000-12-20T00:00:00, BUY CREATE, 26.
```

```
882000-12-21T00:00:00, BUY EXECUTED, 26.232000-12-21T00:00:00, Close, 27.82
2000-12-22T00:00:00, Close, 30.062000-12-26T00:00:00, Close, 29.172000-12-2
7T00:00:00, Close, 28.942000-12-28T00:00:00, Close, 29.292000-12-29T00:00:0
0, Close, 27.412000-12-29T00:00:00, SELL CREATE, 27.41Final Portfolio Value:
 100018.53
```

接下来，让我们设定一个常见的 0.1% 的费率，买卖都要收。
一行代码就能搞定:

```
cerebro.broker.setcommission(commission=0.001) # 0.001 即是 0.1%
```

我们想看看，加和不加手续费，结果有什么区别。

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import datetime  # For datetime objectsimport os.path  # To manage pathsimpo
rt sys  # To find out the script name (in argv[0])
# Import the backtrader platformimport backtrader as bt
# Create a Strateyclass TestStrategy(bt.Strategy):
    def log(self, txt, dt=None):
        ''' Logging function fot this strategy'''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # Keep a reference to the "close" line in the data[0] dataseries
        self.dataclose = self.datas[0].close
        # To keep track of pending orders and buy price/commission
        self.order = None
        self.buyprice = None
        self.buycomm = None

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # Buy/Sell order submitted/accepted to/by broker - Nothing to do
            return
        # Check if an order has been completed
        # Attention: broker could reject order if not enough cash
        if order.status in [order.Completed]:
```

```
            if order.isbuy():
                self.log(
                    'BUY EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f' %
                    (order.executed.price,
                     order.executed.value,
                     order.executed.comm))
                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            else:  # Sell
                self.log('SELL EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f'
%
                         (order.executed.price,
                          order.executed.value,
                          order.executed.comm))
            self.bar_executed = len(self)
        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')
        self.order = None
    def notify_trade(self, trade):
        if not trade.isclosed:
            return
        self.log('OPERATION PROFIT, GROSS %.2f, NET %.2f' %
                 (trade.pnl, trade.pnlcomm))

    def next(self):
        # Simply log the closing price of the series from the reference
        self.log('Close, %.2f' % self.dataclose[0])
        # Check if an order is pending ... if yes, we cannot send a 2nd one
        if self.order:
            return
        # Check if we are in the market
        if not self.position:
            # Not yet ... we MIGHT BUY if ...
            if self.dataclose[0] < self.dataclose[-1]:
                    # current close less than previous close
```

```
                if self.dataclose[-1] < self.dataclose[-2]:
                    # previous close less than the previous close
                    # BUY, BUY, BUY!!! (with default parameters)
                    self.log('BUY CREATE, %.2f' % self.dataclose[0])
                    # Keep track of the created order to avoid a 2nd order
                    self.order = self.buy()
        else:
            # Already in the market ... we might sell
            if len(self) >= (self.bar_executed + 5):
                # SELL, SELL, SELL!!! (with all possible default parameters)
                self.log('SELL CREATE, %.2f' % self.dataclose[0])
                # Keep track of the created order to avoid a 2nd order
                self.order = self.sell()


if __name__ == '__main__':
    # Create a cerebro entity
    cerebro = bt.Cerebro()
    # Add a strategy
    cerebro.addstrategy(TestStrategy)
    # Datas are in a subfolder of the samples. Need to find where the script
is
    # because it could have been called from anywhere
    modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
    datapath = os.path.join(modpath, '../../datas/orcl-1995-2014.txt')
    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        # Do not pass values before this date
        fromdate=datetime.datetime(2000, 1, 1),
        # Do not pass values before this date
        todate=datetime.datetime(2000, 12, 31),
        # Do not pass values after this date
        reverse=False)
    # Add the Data Feed to Cerebro
    cerebro.adddata(data)
```

```
    # Set our desired cash start
    cerebro.broker.setcash(100000.0)
    # Set the commission - 0.1% ... divide by 100 to remove the %
    cerebro.broker.setcommission(commission=0.001)
    # Print out the starting conditions
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    # Run over everything
    cerebro.run()
    # Print out the final result
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

运行后的结果为:

```
Starting Portfolio Value: 100000.002000-01-03T00:00:00, Close, 27.852000-01
-04T00:00:00, Close, 25.392000-01-05T00:00:00, Close, 24.052000-01-05T00:00:
00, BUY CREATE, 24.052000-01-06T00:00:00, BUY EXECUTED, Price: 23.61, Cost:
23.61, Commission 0.022000-01-06T00:00:00, Close, 22.632000-01-07T00:00:00,
 Close, 24.372000-01-10T00:00:00, Close, 27.292000-01-11T00:00:00, Close, 2
6.492000-01-12T00:00:00, Close, 24.902000-01-13T00:00:00, Close, 24.772000-
01-13T00:00:00, SELL CREATE, 24.772000-01-14T00:00:00, SELL EXECUTED, Price:
 25.70, Cost: 25.70, Commission 0.032000-01-14T00:00:00, OPERATION PROFIT, G
ROSS 2.09, NET 2.042000-01-14T00:00:00, Close, 25.18.........2000-12-15T00:
00:00, SELL CREATE, 26.932000-12-18T00:00:00, SELL EXECUTED, Price: 28.29, C
ost: 28.29, Commission 0.032000-12-18T00:00:00, OPERATION PROFIT, GROSS -0.0
6, NET -0.122000-12-18T00:00:00, Close, 30.182000-12-19T00:00:00, Close, 28.
882000-12-20T00:00:00, Close, 26.882000-12-20T00:00:00, BUY CREATE, 26.8820
00-12-21T00:00:00, BUY EXECUTED, Price: 26.23, Cost: 26.23, Commission 0.032
000-12-21T00:00:00, Close, 27.822000-12-22T00:00:00, Close, 30.062000-12-26
T00:00:00, Close, 29.172000-12-27T00:00:00, Close, 28.942000-12-28T00:00:00,
 Close, 29.292000-12-29T00:00:00, Close, 27.412000-12-29T00:00:00, SELL CRE
ATE, 27.41Final Portfolio Value: 100016.98
```

在继续之前，让我们看看这些带有盈亏的操作:

```
2000-01-14T00:00:00, OPERATION PROFIT, GROSS 2.09, NET 2.042000-02-07T00:00:
00, OPERATION PROFIT, GROSS 3.68, NET 3.632000-02-28T00:00:00, OPERATION PRO
FIT, GROSS 4.48, NET 4.422000-03-13T00:00:00, OPERATION PROFIT, GROSS 3.48,
NET 3.412000-03-22T00:00:00, OPERATION PROFIT, GROSS -0.41, NET -0.492000-04
-07T00:00:00, OPERATION PROFIT, GROSS 2.45, NET 2.372000-04-20T00:00:00, OPE
RATION PROFIT, GROSS -1.95, NET -2.022000-05-02T00:00:00, OPERATION PROFIT,
GROSS 5.46, NET 5.392000-05-11T00:00:00, OPERATION PROFIT, GROSS -3.74, NET
-3.812000-05-30T00:00:00, OPERATION PROFIT, GROSS -1.46, NET -1.532000-07-0
5T00:00:00, OPERATION PROFIT, GROSS -1.62, NET -1.692000-07-14T00:00:00, OPE
```

```
RATION PROFIT, GROSS 2.08, NET 2.012000-07-28T00:00:00, OPERATION PROFIT, GR
OSS 0.14, NET 0.072000-08-08T00:00:00, OPERATION PROFIT, GROSS 4.36, NET 4.2
92000-08-21T00:00:00, OPERATION PROFIT, GROSS 1.03, NET 0.952000-09-15T00:0
0:00, OPERATION PROFIT, GROSS -4.26, NET -4.342000-09-27T00:00:00, OPERATION
 PROFIT, GROSS 1.29, NET 1.222000-10-13T00:00:00, OPERATION PROFIT, GROSS -2.
98, NET -3.042000-10-26T00:00:00, OPERATION PROFIT, GROSS 3.01, NET 2.952000
-11-06T00:00:00, OPERATION PROFIT, GROSS -3.59, NET -3.652000-11-16T00:00:0
0, OPERATION PROFIT, GROSS 1.28, NET 1.232000-12-01T00:00:00, OPERATION PROF
IT, GROSS 2.59, NET 2.542000-12-18T00:00:00, OPERATION PROFIT, GROSS -0.06,
NET -0.12
```

“NET”那列的净收益加起来是:

```
15.83
```

但系统最后的余额是:

```
2000-12-29T00:00:00, SELL CREATE, 27.41Final Portfolio Value: 100016.98
```

很明显 15.83 不等于 16.98。其实没发生什么错误，净收益“NET”指的是已经落到口袋里的钱。
造成差别的原因是，最后一天还持有头寸。其实卖单已经发出去了，但还没来得及执行…
Broker 版本的净收益率是按照 2000-12-29 收盘价算的。实际应该按下一个交易日 2001-01-02 价格算:

```
2001-01-02T00:00:00, SELL EXECUTED, Price: 27.87, Cost: 27.87, Commission 0.
032001-01-02T00:00:00, OPERATION PROFIT, GROSS 1.64, NET 1.592001-01-02T00:
00:00, Close, 24.872001-01-02T00:00:00, BUY CREATE, 24.87Final Portfolio Val
ue: 100017.41
```

加起来之前的净收益:

```
15.83 + 1.59 = 17.42
```

这个净收益率 17.42 和最后的余额 100017.41 就对上了（忽略小数点误差）。

自定义策略：技术指标参数

在实践中，一般不将参数硬编码到策略中。Parameters（参数） 就是用来处理这个的。
参数的定义像这样:

```
params = (('myparam', 27), ('exitbars', 5),)
```

这个 tuple 嵌套看着不方便，格式化一下:

```
params = (
    ('myparam', 27),
    ('exitbars', 5),)
```

将策略添加到引擎的时候，可以指定刚才定义的参数:

```
# 添加策略 cerebro.addstrategy(TestStrategy, myparam=20, exitbars=7)
```

注意下面的 setsizing 方法已经被弃用。这里还保留是因为还有一些老示例在用。方法已改为下面这种:
cerebro.addsizer(bt.sizers.FixedSize, stake=10)
请参考 sizers 章节。
在策略类中使用买卖数量参数很容易，它们被保存在 “params” 参数里。例如，参数已经传入，在策略类里的 __init__ 方法中这样调用就可以了:

```
# 根据传入的参加设置买卖数量 self.sizer.setsizing(self.params.stake)
```

也可以直接将买卖数量传入 buy 和 sell 方法。
卖出的逻辑改为:

```
# 已经持有，可以卖出了 if len(self) >= (self.bar_executed + self.params.exitbars):
```

代码修改为:

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import datetime  # For datetime objectsimport os.path  # To manage pathsimport sys  # To find out the script name (in argv[0])
# Import the backtrader platformimport backtrader as bt


# Create a Strateyclass TestStrategy(bt.Strategy):
    params = (
        ('exitbars', 5),
    )
    def log(self, txt, dt=None):
        ''' Logging function fot this strategy'''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))


    def __init__(self):
        # Keep a reference to the "close" line in the data[0] dataseries
        self.dataclose = self.datas[0].close
```

```
        # To keep track of pending orders and buy price/commission
        self.order = None
        self.buyprice = None
        self.buycomm = None

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # Buy/Sell order submitted/accepted to/by broker - Nothing to do
            return
        # Check if an order has been completed
        # Attention: broker could reject order if not enough cash
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(
                    'BUY EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f' %
                    (order.executed.price,
                     order.executed.value,
                     order.executed.comm))
                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            else:  # Sell
                self.log('SELL EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f'
%
                         (order.executed.price,
                          order.executed.value,
                          order.executed.comm))
            self.bar_executed = len(self)
        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')
        self.order = None

    def notify_trade(self, trade):
        if not trade.isclosed:
            return
```

```
        self.log('OPERATION PROFIT, GROSS %.2f, NET %.2f' %
                 (trade.pnl, trade.pnlcomm))

    def next(self):
        # Simply log the closing price of the series from the reference
        self.log('Close, %.2f' % self.dataclose[0])
        # Check if an order is pending ... if yes, we cannot send a 2nd one
        if self.order:
            return
        # Check if we are in the market
        if not self.position:
            # Not yet ... we MIGHT BUY if ...
            if self.dataclose[0] < self.dataclose[-1]:
                    # current close less than previous close
                    if self.dataclose[-1] < self.dataclose[-2]:
                        # previous close less than the previous close
                        # BUY, BUY, BUY!!! (with default parameters)
                        self.log('BUY CREATE, %.2f' % self.dataclose[0])
                        # Keep track of the created order to avoid a 2nd order
                        self.order = self.buy()

        else:
            # Already in the market ... we might sell
            if len(self) >= (self.bar_executed + self.params.exitbars):
                # SELL, SELL, SELL!!! (with all possible default parameters)
                self.log('SELL CREATE, %.2f' % self.dataclose[0])
                # Keep track of the created order to avoid a 2nd order
                self.order = self.sell()
if __name__ == '__main__':
    # Create a cerebro entity
    cerebro = bt.Cerebro()
    # Add a strategy
    cerebro.addstrategy(TestStrategy)
    # Datas are in a subfolder of the samples. Need to find where the script
is
```

```
    # because it could have been called from anywhere
    modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
    datapath = os.path.join(modpath, '../../datas/orcl-1995-2014.txt')
    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        # Do not pass values before this date
        fromdate=datetime.datetime(2000, 1, 1),
        # Do not pass values before this date
        todate=datetime.datetime(2000, 12, 31),
        # Do not pass values after this date
        reverse=False)
    # Add the Data Feed to Cerebro
    cerebro.adddata(data)
    # Set our desired cash start
    cerebro.broker.setcash(100000.0)
    # Add a FixedSize sizer according to the stake
    cerebro.addsizer(bt.sizers.FixedSize, stake=10)
    # Set the commission - 0.1% ... divide by 100 to remove the %
    cerebro.broker.setcommission(commission=0.001)
    # Print out the starting conditions
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    # Run over everything
    cerebro.run()
    # Print out the final result
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

运行后的输出为:

```
Starting Portfolio Value: 100000.002000-01-03T00:00:00, Close, 27.852000-01
-04T00:00:00, Close, 25.392000-01-05T00:00:00, Close, 24.052000-01-05T00:00:
00, BUY CREATE, 24.052000-01-06T00:00:00, BUY EXECUTED, Size 10, Price: 23.6
1, Cost: 236.10, Commission 0.242000-01-06T00:00:00, Close, 22.63.........2
000-12-20T00:00:00, BUY CREATE, 26.882000-12-21T00:00:00, BUY EXECUTED, Siz
e 10, Price: 26.23, Cost: 262.30, Commission 0.262000-12-21T00:00:00, Close,
 27.822000-12-22T00:00:00, Close, 30.062000-12-26T00:00:00, Close, 29.17200
0-12-27T00:00:00, Close, 28.942000-12-28T00:00:00, Close, 29.292000-12-29T0
```

```
0:00:00, Close, 27.412000-12-29T00:00:00, SELL CREATE, 27.41Final Portfolio
Value: 100169.80
```

为了显示改变已生效，输出中显示了买卖数量。
买卖数量改为了原来的 **10** 倍，盈亏也变为了原来的 **10** 倍，变为了 **169.80** 。

添加技术指标

之前我提到过 **indicators**（技术指标），下一步就该添加他们了，要做的肯定比前边“三连跌”这种复杂点。
借用 **PyAlgoTrade** 这个框架的一个使用移动平均线的例子：

- 收盘价高于平均价的时候，以市价买入
- 持有仓位的时候，如果收盘价低于平均价，卖出
- 只有一个待执行的订单

大多数代码不用改变，在 **__init__** 方法中加入移动平均的实例化:

```
self.sma = bt.indicators.MovingAverageSimple(self.datas[0], period=self.par
ams.maperiod)
```

当然买入卖出的逻辑依赖平均价，具体代码如下。

注意
起始金额为 1000 元，无手续费，这和 PyAlgoTrade 保持一致。

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import datetime  # For datetime objectsimport os.path  # To manage pathsimpo
rt sys  # To find out the script name (in argv[0])
# Import the backtrader platformimport backtrader as bt


# Create a Strateyclass TestStrategy(bt.Strategy):
    params = (
        ('maperiod', 15),
    )


    def log(self, txt, dt=None):
```

```
        ''' Logging function fot this strategy'''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # Keep a reference to the "close" line in the data[0] dataseries
        self.dataclose = self.datas[0].close

        # To keep track of pending orders and buy price/commission
        self.order = None
        self.buyprice = None
        self.buycomm = None

        # Add a MovingAverageSimple indicator
        self.sma = bt.indicators.SimpleMovingAverage(
            self.datas[0], period=self.params.maperiod)

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # Buy/Sell order submitted/accepted to/by broker - Nothing to do
            return

        # Check if an order has been completed
        # Attention: broker could reject order if not enough cash
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(
                    'BUY EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f' %
                    (order.executed.price,
                     order.executed.value,
                     order.executed.comm))

                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            else:  # Sell
```

```
                self.log('SELL EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f'
%
                         (order.executed.price,
                          order.executed.value,
                          order.executed.comm))

            self.bar_executed = len(self)

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')

        self.order = None

    def notify_trade(self, trade):
        if not trade.isclosed:
            return

        self.log('OPERATION PROFIT, GROSS %.2f, NET %.2f' %
                 (trade.pnl, trade.pnlcomm))

    def next(self):
        # Simply log the closing price of the series from the reference
        self.log('Close, %.2f' % self.dataclose[0])

        # Check if an order is pending ... if yes, we cannot send a 2nd one
        if self.order:
            return

        # Check if we are in the market
        if not self.position:

            # Not yet ... we MIGHT BUY if ...
            if self.dataclose[0] > self.sma[0]:

                # BUY, BUY, BUY!!! (with all possible default parameters)
```

```
                self.log('BUY CREATE, %.2f' % self.dataclose[0])

                # Keep track of the created order to avoid a 2nd order
                self.order = self.buy()

        else:

            if self.dataclose[0] < self.sma[0]:
                # SELL, SELL, SELL!!! (with all possible default parameters)
                self.log('SELL CREATE, %.2f' % self.dataclose[0])

                # Keep track of the created order to avoid a 2nd order
                self.order = self.sell()

if __name__ == '__main__':
    # Create a cerebro entity
    cerebro = bt.Cerebro()

    # Add a strategy
    cerebro.addstrategy(TestStrategy)

    # Datas are in a subfolder of the samples. Need to find where the script
is
    # because it could have been called from anywhere
    modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
    datapath = os.path.join(modpath, '../../datas/orcl-1995-2014.txt')

    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        # Do not pass values before this date
        fromdate=datetime.datetime(2000, 1, 1),
        # Do not pass values before this date
        todate=datetime.datetime(2000, 12, 31),
        # Do not pass values after this date
```

```
        reverse=False)

    # Add the Data Feed to Cerebro
    cerebro.adddata(data)

    # Set our desired cash start
    cerebro.broker.setcash(1000.0)

    # Add a FixedSize sizer according to the stake
    cerebro.addsizer(bt.sizers.FixedSize, stake=10)

    # Set the commission
    cerebro.broker.setcommission(commission=0.0)

    # Print out the starting conditions
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())

    # Run over everything
    cerebro.run()

    # Print out the final result
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

让我们 仔细地 看一下出现在下面日志中的第一条记录：

```
不再是新千年的第一个交易日 2000-01-03 了。
变成了 2000-01-24 ，怎么回事呢？
```

是因为，框架根据新代码做出了改变： The missing days are not missing. The platform has adapted to the new circumstances:

```
 在策略中我们加入了移动平均技术指标。
```

移动平均需要有个均线周期参数，程序根据这个参数回看计算前边的 X 条价格数据然后进行开仓判断，例子中周期是 15

2000-01-24 就是第 15 天
backtrader 框架假定策略加入这个技术指标是有正当理由的，比如 做开平仓的决策 。框架不会在数据没到位的时候就进行下一步。
在技术指标产生第一条数据之后，next 方法第一个被调用
在示例中只有一个技术指标，其实策略支持添加多个技术指标

运行后的输出为:

```
Starting Portfolio Value: 1000.002000-01-24T00:00:00, Close, 25.552000-01-2
5T00:00:00, Close, 26.612000-01-25T00:00:00, BUY CREATE, 26.612000-01-26T00:
00:00, BUY EXECUTED, Size 10, Price: 26.76, Cost: 267.60, Commission 0.00200
0-01-26T00:00:00, Close, 25.962000-01-27T00:00:00, Close, 24.432000-01-27T0
0:00:00, SELL CREATE, 24.432000-01-28T00:00:00, SELL EXECUTED, Size 10, Pric
e: 24.28, Cost: 242.80, Commission 0.002000-01-28T00:00:00, OPERATION PROFIT,
 GROSS -24.80, NET -24.802000-01-28T00:00:00, Close, 22.342000-01-31T00:00:
00, Close, 23.552000-02-01T00:00:00, Close, 25.462000-02-02T00:00:00, Close,
 25.612000-02-02T00:00:00, BUY CREATE, 25.612000-02-03T00:00:00, BUY EXECUT
ED, Size 10, Price: 26.11, Cost: 261.10, Commission 0.00.........2000-12-20T
00:00:00, SELL CREATE, 26.882000-12-21T00:00:00, SELL EXECUTED, Size 10, Pri
ce: 26.23, Cost: 262.30, Commission 0.002000-12-21T00:00:00, OPERATION PROFI
T, GROSS -20.60, NET -20.602000-12-21T00:00:00, Close, 27.822000-12-21T00:0
0:00, BUY CREATE, 27.822000-12-22T00:00:00, BUY EXECUTED, Size 10, Price: 28.
65, Cost: 286.50, Commission 0.002000-12-22T00:00:00, Close, 30.062000-12-2
6T00:00:00, Close, 29.172000-12-27T00:00:00, Close, 28.942000-12-28T00:00:0
0, Close, 29.292000-12-29T00:00:00, Close, 27.412000-12-29T00:00:00, SELL C
REATE, 27.41Final Portfolio Value: 973.90
```

一个盈利系统被改变之后开始亏损了...还是在手续费率设置为 0 的情况下。看来像他们说的 *简单* 添加一个 *技术指标* 并不是万能的。

注意：同样的交易逻辑和数据，和 PyAlgoTrade 输出的结果并不完全一致，当然只是轻微不一致。最可疑的原因是因为：小数点
处理"调整后价格"（分红、拆股后调整）时，PyAlgoTrade 并不对小数点进行四舍五入。
在对价格进行调整后， backtrader 的数据引擎将 Yahoo 价格数据的价格小数点缩减到 2 位。虽然输出看起来差不多，但积少成多结果就不同了。
将价格小数点缩减到 2 位是合理的，一般交易所只允许价格保留小数点后面 2 位。

注意：从 1.8.11.99 版本开始，backtrader 的 Yahoo 数据引擎可以设置是否做小数点位数保留，还可以设置保留多少位。

可视化：绘图

文字日志虽然能看到细节，但人们还是喜欢看可视化的东西，所以有必要将结果绘制成图表。

绘图很容易使用，只需添加一行代码:

```
cerebro.plot()
```

这行代码要放在 cerebro.run() 之后。
为方便使用，框架做了下面这些自动化的事情：

将添加第二条指数移动平均线，默认将使用数据进行绘制（就像第 1 条）。
将添加第三条加权移动平均线，在单独区域绘制（也许看起来不合理）
将添加一条 Stochastic （慢） ，使用默认参数。
将添加一条 MACD ，使用默认参数。
将添加一条 RSI 指标，使用默认参数。
将添加一条 RSI 指标的简单移动平均线，使用默认参数（将和 RSI 一起被绘制）。
将添加一条 ATR 指标，修改了默认参数以避免被绘制。

上面添加的这些指标，等于在策略类的 __init__ 方法中添加了以下语句:

```
# 需要绘制的指标 bt.indicators.ExponentialMovingAverage(self.datas[0], period
=25)bt.indicators.WeightedMovingAverage(self.datas[0], period=25).subplot =
 Truebt.indicators.StochasticSlow(self.datas[0])bt.indicators.MACDHisto(sel
f.datas[0])rsi = bt.indicators.RSI(self.datas[0])bt.indicators.SmoothedMovi
ngAverage(rsi, period=10)bt.indicators.ATR(self.datas[0]).plot = False
```

注意：即使 指标 没有被显式地声明为成员变量（如 self.sma = MovingAverageSimple…），
它们还是会被自动注册到策略类中，并影响开始执行 next 的最小周期，而且会被绘制。
在例子中，只有 RSI 的指标被赋予了一个 rsi 的变量，供后边为它创建移动平均线使用。

现在程序变成了这样：

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import datetime  # For datetime objectsimport os.path  # To manage pathsimpo
rt sys  # To find out the script name (in argv[0])
# Import the backtrader platformimport backtrader as bt


# Create a Strateyclass TestStrategy(bt.Strategy):
    params = (
        ('maperiod', 15),
    )
```

```
def log(self, txt, dt=None):
    ''' Logging function fot this strategy'''
    dt = dt or self.datas[0].datetime.date(0)
    print('%s, %s' % (dt.isoformat(), txt))

def __init__(self):
    # Keep a reference to the "close" line in the data[0] dataseries
    self.dataclose = self.datas[0].close

    # To keep track of pending orders and buy price/commission
    self.order = None
    self.buyprice = None
    self.buycomm = None

    # Add a MovingAverageSimple indicator
    self.sma = bt.indicators.SimpleMovingAverage(
        self.datas[0], period=self.params.maperiod)

    # Indicators for the plotting show
    bt.indicators.ExponentialMovingAverage(self.datas[0], period=25)
    bt.indicators.WeightedMovingAverage(self.datas[0], period=25,
                                        subplot=True)
    bt.indicators.StochasticSlow(self.datas[0])
    bt.indicators.MACDHisto(self.datas[0])
    rsi = bt.indicators.RSI(self.datas[0])
    bt.indicators.SmoothedMovingAverage(rsi, period=10)
    bt.indicators.ATR(self.datas[0], plot=False)

def notify_order(self, order):
    if order.status in [order.Submitted, order.Accepted]:
        # Buy/Sell order submitted/accepted to/by broker - Nothing to do
        return

    # Check if an order has been completed
```

```
        # Attention: broker could reject order if not enough cash
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(
                    'BUY EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f' %
                    (order.executed.price,
                     order.executed.value,
                     order.executed.comm))

                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            else:  # Sell
                self.log('SELL EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f'
%
                         (order.executed.price,
                          order.executed.value,
                          order.executed.comm))

            self.bar_executed = len(self)

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')

        # Write down: no pending order
        self.order = None

    def notify_trade(self, trade):
        if not trade.isclosed:
            return

        self.log('OPERATION PROFIT, GROSS %.2f, NET %.2f' %
                 (trade.pnl, trade.pnlcomm))

    def next(self):
        # Simply log the closing price of the series from the reference
```

```
        self.log('Close, %.2f' % self.dataclose[0])

        # Check if an order is pending ... if yes, we cannot send a 2nd one
        if self.order:
            return

        # Check if we are in the market
        if not self.position:

            # Not yet ... we MIGHT BUY if ...
            if self.dataclose[0] > self.sma[0]:

                # BUY, BUY, BUY!!! (with all possible default parameters)
                self.log('BUY CREATE, %.2f' % self.dataclose[0])

                # Keep track of the created order to avoid a 2nd order
                self.order = self.buy()

        else:

            if self.dataclose[0] < self.sma[0]:
                # SELL, SELL, SELL!!! (with all possible default parameters)
                self.log('SELL CREATE, %.2f' % self.dataclose[0])

                # Keep track of the created order to avoid a 2nd order
                self.order = self.sell()

if __name__ == '__main__':
    # Create a cerebro entity
    cerebro = bt.Cerebro()

    # Add a strategy
    cerebro.addstrategy(TestStrategy)
```

```
    # Datas are in a subfolder of the samples. Need to find where the script
is
    # because it could have been called from anywhere
    modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
    datapath = os.path.join(modpath, '../../datas/orcl-1995-2014.txt')

    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        # Do not pass values before this date
        fromdate=datetime.datetime(2000, 1, 1),
        # Do not pass values before this date
        todate=datetime.datetime(2000, 12, 31),
        # Do not pass values after this date
        reverse=False)

    # Add the Data Feed to Cerebro
    cerebro.adddata(data)

    # Set our desired cash start
    cerebro.broker.setcash(1000.0)

    # Add a FixedSize sizer according to the stake
    cerebro.addsizer(bt.sizers.FixedSize, stake=10)

    # Set the commission
    cerebro.broker.setcommission(commission=0.0)

    # Print out the starting conditions
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())

    # Run over everything
    cerebro.run()

    # Print out the final result
```

```
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())


    # Plot the result
    cerebro.plot()
```

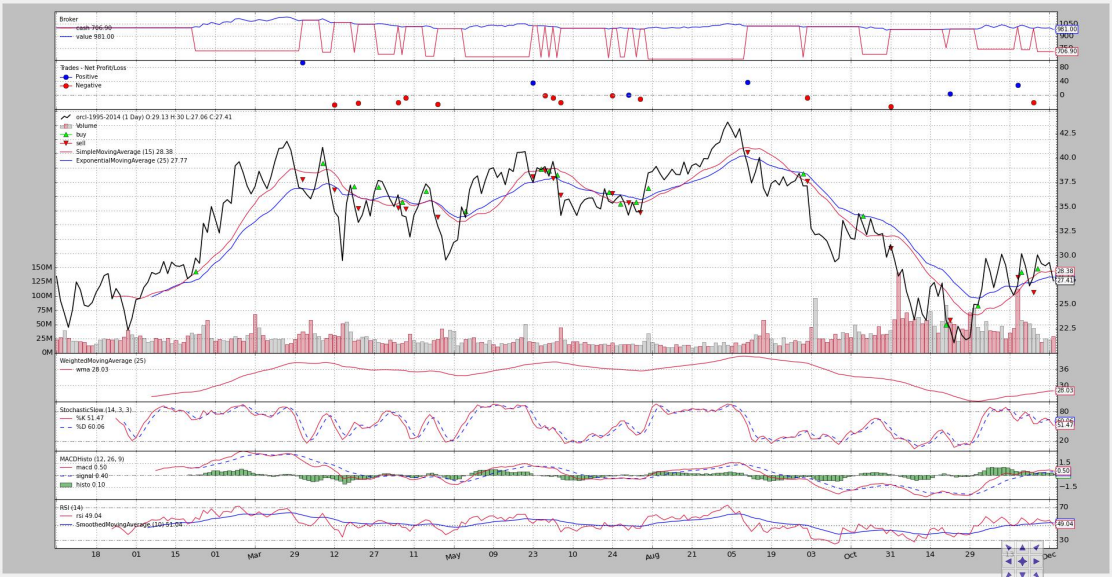
执行后的输出结果为:

```
Starting Portfolio Value: 1000.002000-02-18T00:00:00, Close, 27.612000-02-2
2T00:00:00, Close, 27.972000-02-22T00:00:00, BUY CREATE, 27.972000-02-23T00:
00:00, BUY EXECUTED, Size 10, Price: 28.38, Cost: 283.80, Commission 0.00200
0-02-23T00:00:00, Close, 29.73.........2000-12-21T00:00:00, BUY CREATE, 27.
822000-12-22T00:00:00, BUY EXECUTED, Size 10, Price: 28.65, Cost: 286.50, Co
mmission 0.002000-12-22T00:00:00, Close, 30.062000-12-26T00:00:00, Close, 2
9.172000-12-27T00:00:00, Close, 28.942000-12-28T00:00:00, Close, 29.292000-
12-29T00:00:00, Close, 27.412000-12-29T00:00:00, SELL CREATE, 27.41Final Po
rtfolio Value: 981.00
```

虽然策略逻辑没有变，但回测结果却变了。这是由于执行的 bar 的数量发生了变化。

注意：
前面提到过，框架会等待所有指标数据到位之后，才会运行 next 函数。 在上例中， MACD 是最后一个数据到位的指标（它的 3 条线都完成了输出）。
所以第一笔下单已经不是 2000 年 1 月份了，而是 2000 年 2 月份末。

图表如下：



参数调优

许多交易书籍都会说每个市场、每只股票（或期货等等）都有不同的节奏，也就是说没有一个参数能适应所有。
在之前的例子里，策略里使用的默认参数是 **15**。这个参数可以被更换并进行测试，以评估什么值更适合于市场。

注意
大量文献讨论了关于优化的优缺点。一般建议都会指向同一方向：不要过度优化。如果策略不理想， 而在拟合上下功夫，则可能产生一个在回测数据上非常优秀的参数，但这个参数在将来表现可能并不好。

修改了代码，以测试移动平均线的最优周期参数。为保持清新，删除了所有买入、卖出的输出。
修改后的例子：

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import datetime  # For datetime objectsimport os.path  # To manage pathsimpo
rt sys  # To find out the script name (in argv[0])


# Import the backtrader platformimport backtrader as bt


# Create a Strateyclass TestStrategy(bt.Strategy):
    params = (
        ('maperiod', 15),
        ('printlog', False),
    )

    def log(self, txt, dt=None, doprint=False):
        ''' Logging function fot this strategy'''
        if self.params.printlog or doprint:
            dt = dt or self.datas[0].datetime.date(0)
            print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # Keep a reference to the "close" line in the data[0] dataseries
        self.dataclose = self.datas[0].close
```

```
        # To keep track of pending orders and buy price/commission
        self.order = None
        self.buyprice = None
        self.buycomm = None

        # Add a MovingAverageSimple indicator
        self.sma = bt.indicators.SimpleMovingAverage(
            self.datas[0], period=self.params.maperiod)

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # Buy/Sell order submitted/accepted to/by broker - Nothing to do
            return

        # Check if an order has been completed
        # Attention: broker could reject order if not enough cash
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(
                    'BUY EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f' %
                    (order.executed.price,
                     order.executed.value,
                     order.executed.comm))

                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            else:  # Sell
                self.log('SELL EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f'
%
                         (order.executed.price,
                          order.executed.value,
                          order.executed.comm))

            self.bar_executed = len(self)
```

```
        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')

        # Write down: no pending order
        self.order = None

    def notify_trade(self, trade):
        if not trade.isclosed:
            return

        self.log('OPERATION PROFIT, GROSS %.2f, NET %.2f' %
                 (trade.pnl, trade.pnlcomm))

    def next(self):
        # Simply log the closing price of the series from the reference
        self.log('Close, %.2f' % self.dataclose[0])

        # Check if an order is pending ... if yes, we cannot send a 2nd one
        if self.order:
            return

        # Check if we are in the market
        if not self.position:

            # Not yet ... we MIGHT BUY if ...
            if self.dataclose[0] > self.sma[0]:

                # BUY, BUY, BUY!!! (with all possible default parameters)
                self.log('BUY CREATE, %.2f' % self.dataclose[0])

                # Keep track of the created order to avoid a 2nd order
                self.order = self.buy()

        else:
```

```
            if self.dataclose[0] < self.sma[0]:
                # SELL, SELL, SELL!!! (with all possible default parameters)
                self.log('SELL CREATE, %.2f' % self.dataclose[0])

                # Keep track of the created order to avoid a 2nd order
                self.order = self.sell()

    def stop(self):
        self.log('(MA Period %2d) Ending Value %.2f' %
                 (self.params.maperiod, self.broker.getvalue()), doprint=Tru
e)

if __name__ == '__main__':
    # Create a cerebro entity
    cerebro = bt.Cerebro()

    # Add a strategy
    strats = cerebro.optstrategy(
        TestStrategy,
        maperiod=range(10, 31))

    # Datas are in a subfolder of the samples. Need to find where the script
is
    # because it could have been called from anywhere
    modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
    datapath = os.path.join(modpath, '../../datas/orcl-1995-2014.txt')

    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        # Do not pass values before this date
        fromdate=datetime.datetime(2000, 1, 1),
        # Do not pass values before this date
        todate=datetime.datetime(2000, 12, 31),
```

```
        # Do not pass values after this date
        reverse=False)

    # Add the Data Feed to Cerebro
    cerebro.adddata(data)

    # Set our desired cash start
    cerebro.broker.setcash(1000.0)

    # Add a FixedSize sizer according to the stake
    cerebro.addsizer(bt.sizers.FixedSize, stake=10)

    # Set the commission
    cerebro.broker.setcommission(commission=0.0)

    # Run over everything
    cerebro.run()
```

这次没有调用 addstrategy ，而是用 optstrategy 函数将策略添加到 Cerebro 。 传入的是要测试的一系列值，而不是单个值。
在策略类中添加了 stop 方法，它将在每轮回测之后被调用。 我们用它来打印回测结束之后的资产余额（之前在 Cerebro 做的）。
框架将为策略测试每个参数值，下面是输出结果:

```
2000-12-29, (MA Period 10) Ending Value 880.302000-12-29, (MA Period 11) End
ing Value 880.002000-12-29, (MA Period 12) Ending Value 830.302000-12-29, (M
A Period 13) Ending Value 893.902000-12-29, (MA Period 14) Ending Value 896.
902000-12-29, (MA Period 15) Ending Value 973.902000-12-29, (MA Period 16) E
nding Value 959.402000-12-29, (MA Period 17) Ending Value 949.802000-12-29,
(MA Period 18) Ending Value 1011.902000-12-29, (MA Period 19) Ending Value 1
041.902000-12-29, (MA Period 20) Ending Value 1078.002000-12-29, (MA Period
21) Ending Value 1058.802000-12-29, (MA Period 22) Ending Value 1061.502000-
12-29, (MA Period 23) Ending Value 1023.002000-12-29, (MA Period 24) Ending
Value 1020.102000-12-29, (MA Period 25) Ending Value 1013.302000-12-29, (MA
Period 26) Ending Value 998.302000-12-29, (MA Period 27) Ending Value 982.20
2000-12-29, (MA Period 28) Ending Value 975.702000-12-29, (MA Period 29) End
ing Value 983.302000-12-29, (MA Period 30) Ending Value 979.80
```

结果：

周期参数在 18 以下的亏损（在没有手续费的情况下）。
周期参数在 18 至 26 之间的盈利。
周期参数大于 26 的又会亏损。

对这个策略来说，最优的参数是：

回看周期 20，本金 1000，盈利 78 元，收益率 7.8%。

注意
在上面的例子中，移除了多余的用来绘图的指标，数据开始回测的时间仅取决于我们添加的简单移动平均线。所以周期为 15 的回测结果和之前的有轻微不同。

总结

上面的教程，我们从一个头开始，一步步搭建了一个能运行的回测系统，并且具备绘制结果和优化参数功能。
除此之外，还能做一些提供胜率的事情：

自定义指标
    创建自定义指标很容易，绘制它们同样简单
下单数量
    资金管理是交易成功的关键之一
委托单类型（限价单、止损单、限价止损单）

英文文档