

编写 BOLL 心得体会

作者: [cpucuppcu](#)

原文链接: <https://ld246.com/article/1659538642399>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

关于接口编程

- 大佬讲如何实现

昨天你问我是怎么想出来的，我其实也是第一次把代码写得这么风骚，我觉得关键在于我不是面向逻辑去思考，而是面向接口去思考。
我会先确定一个接口设计对于用户来讲是足够简单易懂的，然后再想办法实现它

面向接口思考 这种思考模式需要大量看源码才能有吧

平时写多了就自然会有

拿今天咱们从MACD到BOLL的写法的改进，其实又是一次简化与创新

受益匪浅，十分感谢

关于实操演练

- 错误之处

大佬我遇见一个问题。。就这个问题昨天搞了好半天

```
return tv -> {  
    TimeSeriesValue v = MA(n).apply(tv);  
    TimeSeriesValue v0 = STD(MA(n),n).apply(tv);  
    v.setValue(v.getValue() + x * v0.getValue());  
    return v;  
};
```

我在里面调用的MA 和STD 每次进来都是从0开始的

命名搞错了，Boll

嘿嘿，名字一念看哈

MA是个闭包函数

所以一般都是从最外层把MA传进来就没这个问题

你现在每次调都会新建一个闭包对象

```
lower(MA(params.n),STD(MA(20),params.n),params.x)
```

所以我就这么写也是无奈之举

不，在Boll实例化时创建

- 原因

闭包函数不应该在另一个闭包函数内使用，我把另外两个接口函数的创建，放到实例化时就没问题了

```
private TimeSeriesUnaryOperator ma;  
private TimeSeriesUnaryOperator std;  
  
private Boll(int n, int x) {  
    this.n = n;  
    this.x = x;  
    this.ma = MA(n);  
    this.std = STD(n);  
}
```

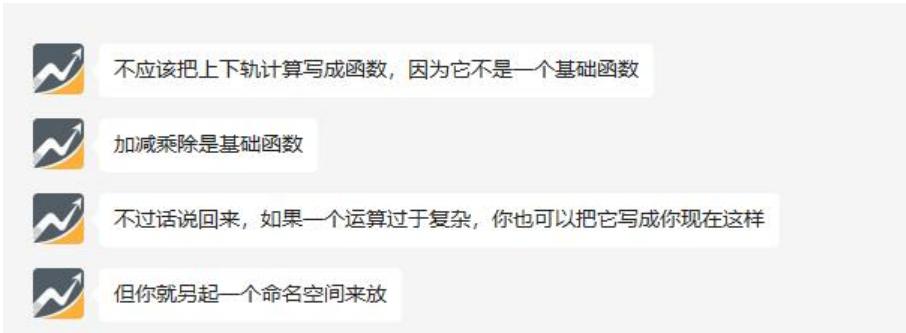
最终产物

-

```
public final class Boll {  
    private int n;  
    private int x;  
    private TimeSeriesUnaryOperator ma;  
    private TimeSeriesUnaryOperator std;  
  
    private Boll(int n, int x) {  
        this.n = n;  
        this.x = x;  
        this.ma = MA(n);  
        this.std = STD(n);  
    }  
  
    private static Boll create(int n , int x){  
        return new Boll(n,x);  
    }  
  
    public static Boll of(int n, int x){  
        return create(n, x);  
    }  
  
    public TimeSeriesUnaryOperator upper(){  
        return tv -> {  
            TimeSeriesValue v = ma.apply(tv);  
            TimeSeriesValue v0 = std.apply(tv);  
            v.setValue(v.getValue() + x * v0.getValue());  
            return v;  
        };  
    }  
  
    public TimeSeriesUnaryOperator lower(){  
        return tv -> {  
            TimeSeriesValue v = ma.apply(tv);  
            TimeSeriesValue v0 = std.apply(tv);  
            v.setValue(v.getValue() - x * v0.getValue());  
            return v;  
        };  
    }  
  
    public TimeSeriesUnaryOperator mid(){  
        return tv -> {  
            TimeSeriesValue v = ma.apply(tv);  
            return v;  
        };  
    }  
}
```

不同的命名空间内不能乱放

-



简化接口的写法

- ```
+ // Boll 上轨
+ this.bollupper = ctx.newIndicator("BOLL_UPPER",params.indicatorSymbol,params.t,
+ Indicator.ValueType.CLOSE,upper(MA(params.t),STD(MA(20),params.t),params.k));
+ // Boll 下轨
+ this.bolllower = ctx.newIndicator("BOLL_LOWER",params.indicatorSymbol,params.t,
+ Indicator.ValueType.CLOSE,lower(MA(params.t),STD(MA(20),params.t),params.k));
+ // Boll 中轨
+ this.bollMid = ctx.newIndicator("BOLL_MID",params.indicatorSymbol,params.t, Indicator.ValueType.CLOSE,MA(params.t));
}
```
- 当你真要封装时，这个就不应该这样写了

```
public void createBoll(BarData barData, BarData bollData) {
 Boll boll = new Boll();
 boll.setUpper(bollData);
 boll.setLower(bollData);
 boll.setMid(bollData);
}
```

这个创建没人看得懂
- 这个创建没人看得懂

```
this.bollUpper = ctx.newIndicator("BOLL_UP", symbol, Boll.of(n, x).upper());
this.bollMid = ctx.newIndicator("BOLL_UP", symbol, Boll.of(n, x).mid());
this.bollLower = ctx.newIndicator("BOLL_UP", symbol, Boll.of(n, x).low());
```



细节可以再斟酌  
一般.of是创建一个对象，上面的写法就创建了三个对象



所以这里有点不太好，不过不影响我的大意

## 闭包函数编写时不应该传入另一个闭包函数

- 案例

```
/**
 * 函数: STD
 * 说明: 估算标准差
 * 用法:STD(size)为收盘价的size日估算标准差
 * @param size
 * @return
 */
2个用法 + lazy
static TimeSeriesUnaryOperator STD(TimeSeriesUnaryOperator ma,int size){
 final double[] values = new double[size];
 final AtomicInteger cursor = new AtomicInteger();
 final AtomicDouble sumOfValues = new AtomicDouble();
 return tv ->{
 TimeSeriesValue applyMA =ma.apply(tv);
 long timestamp = tv.getTimestamp();
 double val = tv.getValue();
 double oldVal = values[cursor.get()];
 values[cursor.get()] = val;
 cursor.set(cursor.incrementAndGet() % size);
 /*
 * 平方差计算方法 前20个值 和 20值平均之间的差 (close - ma20) 平方,通过交换法 先 等于close的平方 - ma20的平方
 */
 sumOfValues.addAndGet(delta: Math.pow(val,2) - Math.pow(oldVal,2));
 val = (sumOfValues.get() - (Math.pow(applyMA.getValue(),2) * size)) ;
 //求平均值并且开平方
 val = Math.sqrt(val / size);
 return new TimeSeriesValue(val, timestamp);
 };
}
```



这个入参明显不对

不应该有这个ma的入参

STD(N), 才对

你参照MA函数思考一下

是不是

我传入当前均线的值就行么

但是这个标准差需要一个MA20的结果

准备要把MA函数实现复制一份到 STD



不是复制一份



是在STD内有一个数组



计算标准差



本来就已经有一些计算库是可以提供标准差的计算的

## ● 原用法

STD(MA(20),params.n)

修改思路：不是需要一个MA20的值吗，我弄个数组加起来除以20就行，本来values就是成的

```
static TimeSeriesUnaryOperator STD(int size){
 final double[] values = new double[size];
 final AtomicInteger cursor = new AtomicInteger();
 final AtomicDouble sumSTDOfValues = new AtomicDouble();
 final AtomicDouble sumMaOfValues = new AtomicDouble();

 return tv ->{
 long timestamp = tv.getTimestamp();
 double val = tv.getValue();
 double oldVal = values[cursor.get()];
 values[cursor.get()] = val;
 cursor.set(cursor.incrementAndGet() % size);
 /**
 * 平方差计算方法 前20个值 和 20值平均之间的差 (close - ma20)平方,通过交换法 先 等于 close的平方 减ma20的平方
 */
 sumMaOfValues.addAndGet(val - oldVal);
 sumSTDOfValues.addAndGet(Math.pow(val,2) - Math.pow(oldVal,2));
 val = (sumSTDOfValues.get() - (Math.pow(sumMaOfValues.get() / size ,2) * size)) ;
 //求平均值并且开平方
 val = Math.sqrt(val / size);
 return new TimeSeriesValue(val, timestamp);
 };
}
```

● 修改思路2.0 大佬的意思是方差函数没必要自己写，有现成的。。。就写这个手动方差计算，想了俩小时哎~~~

结果 引用一个MATH函数库 commons-math之常用科学计算（百分位、总体方差、中位数、变异系数、偏度系数、峰度系数<sup>1</sup>）

```
static TimeSeriesUnaryOperator STD(int size){
 final double[] values = new double[size];
 final AtomicInteger cursor = new AtomicInteger();
 return tv ->{
 long timestamp = tv.getTimestamp();
 values[cursor.get()] = tv.getValue();
 cursor.set(cursor.incrementAndGet() % size);
 double variance = new StandardDeviation().evaluate(values);
 };
}
```

```
 return new TimeSeriesValue(variance, timestamp);
 }
}
```

## 总结：

1. 函数位置要放对
  2. 确认接口设计是否简单易懂再想办法实现
  3. 对象里的闭包函数更好用
  4. 闭包入参要慎重
  5. 善用现成函数库
-