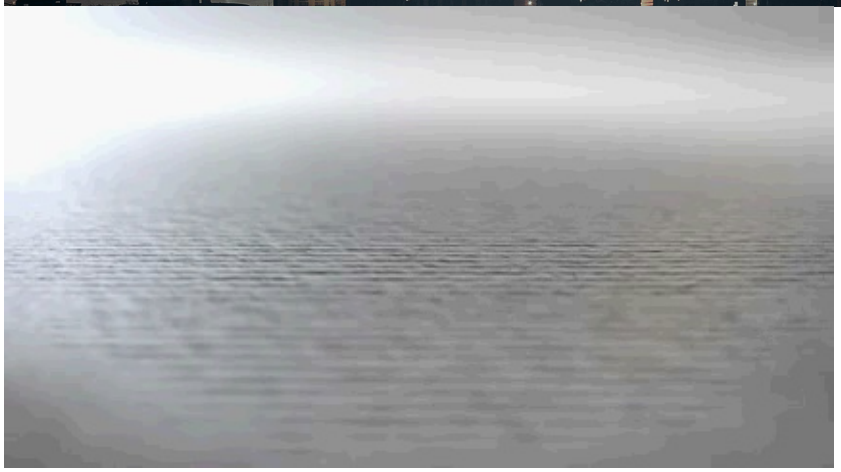
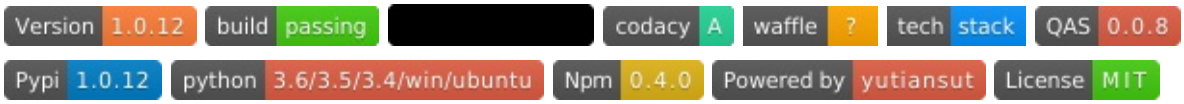
Table of Contents

QUANTAXIS 量化金融策略框架

[点击右上角Star和Watch来跟踪项目进展! 点击Fork来创建属于你的QUANTAXIS!]





> 欢迎加群讨论: 群链接

QUANTAXIS量化金融策略框架,是一个面向中小型策略团队的量化分析解决方案. 我们通过高度解耦的模块化以及标准化协议,可以快速的实现面向场景的定制化解决方案.QUANTAXIS是一个渐进式的开放式框架,你可以根据自己的需要,引入自己的数据,分析方案,可视化过程等,也可以通过RESTful接口,快速实现多人局域网/广域网内的协作.

- 1. 功能
- 1. 文档
- 1. 安装和部署
- 1. 更新
- 1. Docker
- 1. 使用说明
- 1. Jupyter示例
- 1. 开发计划
- 1. 常见问题FAQ
- 1. 项目捐赠
- 1. 回测Webkit插件概览
- 1. QUANTAXIS 标准化协议和未来协议

电脑配置推荐

推荐配置: 6代以上CPU+ 16/32GB DDR3/DDR4内存+ 256GB以上SSD硬盘

最低配置: 支持X64位的CPU

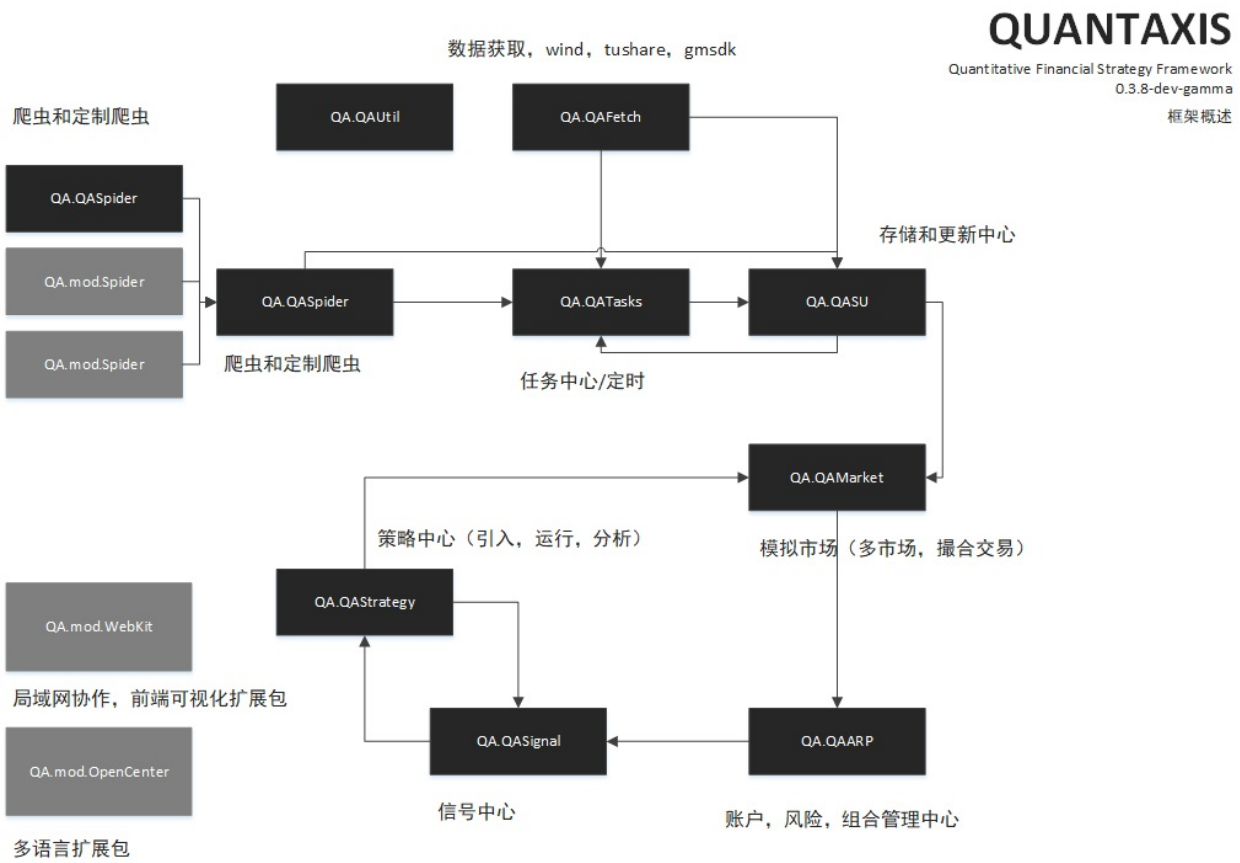
因为在存储本地数据的时候,需要存储超过2GB的本地数据,而32位的MONGODB最高只支持2GB左右的数据存储,因此最少需要一个X64位的CPU

如果SSD资源够用,尽量将数据存储在SSD中,增加 `wiretiger` 写盘的速度

如果是阿里云/腾讯云的服务器,请在最初的时候 选择64位的操作系统

1. 功能

======



已经实现：

- [x] 日线（自1990年）回测 [定点复权] (T+1)
- [x] 分钟线 [1min/5min/15min/30min/60min]回测 (T+1)
- [x] 股指期货日线(T+0)/指数日线/ETF日线
- [x] 股指期货分钟线(T+0) / 指数分钟线/ETF分钟线

[1min/5min/15min/30min/60min]
- [x] 期货日线/分钟线(期货指数/期货主连/期货合约)
- [x] 基于pytdx/tushare以及各种爬虫的数据源
- [x] 实时交易数据,实时tick
- [x] 基于Vue.js的前端网站
- [x] 自定义的数据结构QADataStruct
- [x] 指标计算QAIndicator
- [x] 板块数据(0.5.1新增)/同花顺,通达信板块
- [x] 基本面数据(部分 最新一期财务报表)
- [x] 行情分发
- [x] 循环回测
- [x] 回测管理优化(新增回测主题/版本号)

预计实现:

- [ ] 文档更新
- [ ] 期货回测
- [ ] 实盘
- [ ] 分析模块(行情分析/板块分析)
- [ ] 多数据库支持
- [ ] 权限管理
- [ ] 成交记录分析器

- QUANTAXIS 2018开发计划表

2. 文档

文档参见: book

下载文档手册

PDF | MOBI | EPUB

3. 安装和部署

```
git clone https://github.com/yutiansut/quantaxis --depth 1
```

参见 安装说明

4. 更新

参见 更新说明

5. Docker

参见 Docker

6. 使用说明

参见

- QUANTAXIS的使用示例
- QUANTAXIS回测API
- QUANTAXIS的数据结构
- QUANTAXIS指标系统
- QUANTAXIS的数据获取指南
- QUANTAXIS行情研究
- QUANTAXIS回测分析
- 常见策略整理

7. Jupyter示例

参见 Jupyter示例

8. 开发计划

参见 开发计划

9. 常见问题FAQ

参见 FAQ

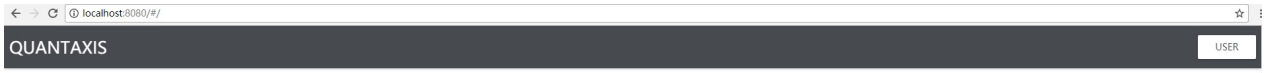
10. 项目捐赠

写代码不易...请作者喝杯咖啡呗?

(PS: 支付的时候 请带上你的名字/昵称呀 会维护一个赞助列表~)

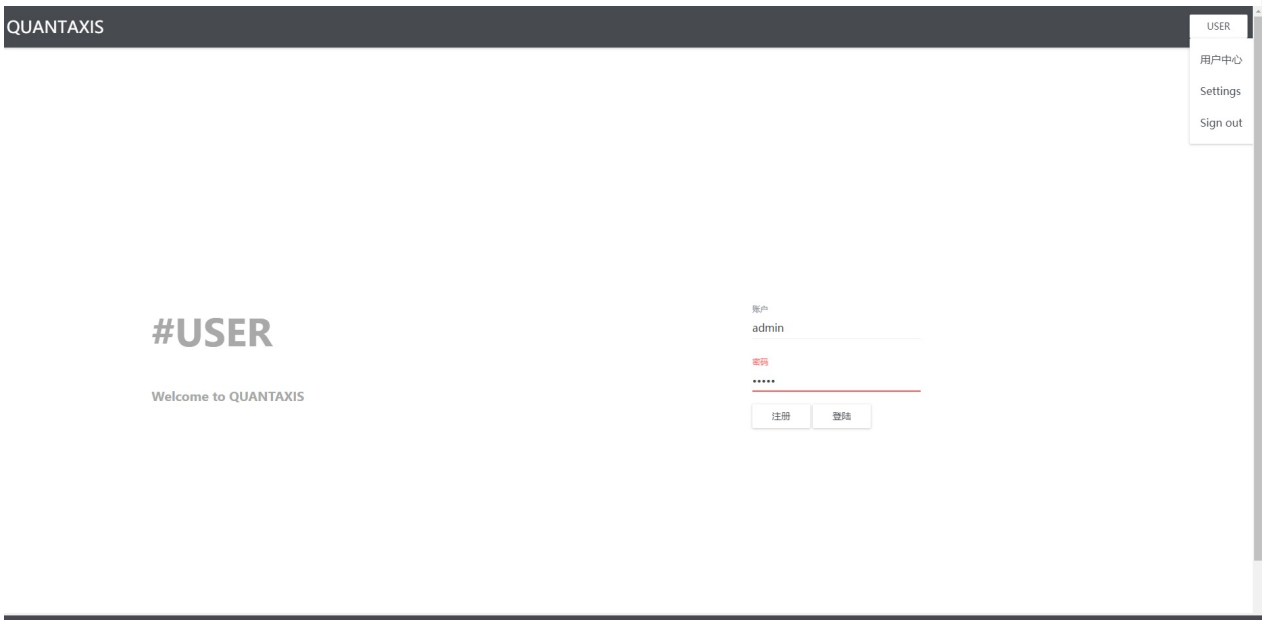
捐赠列表

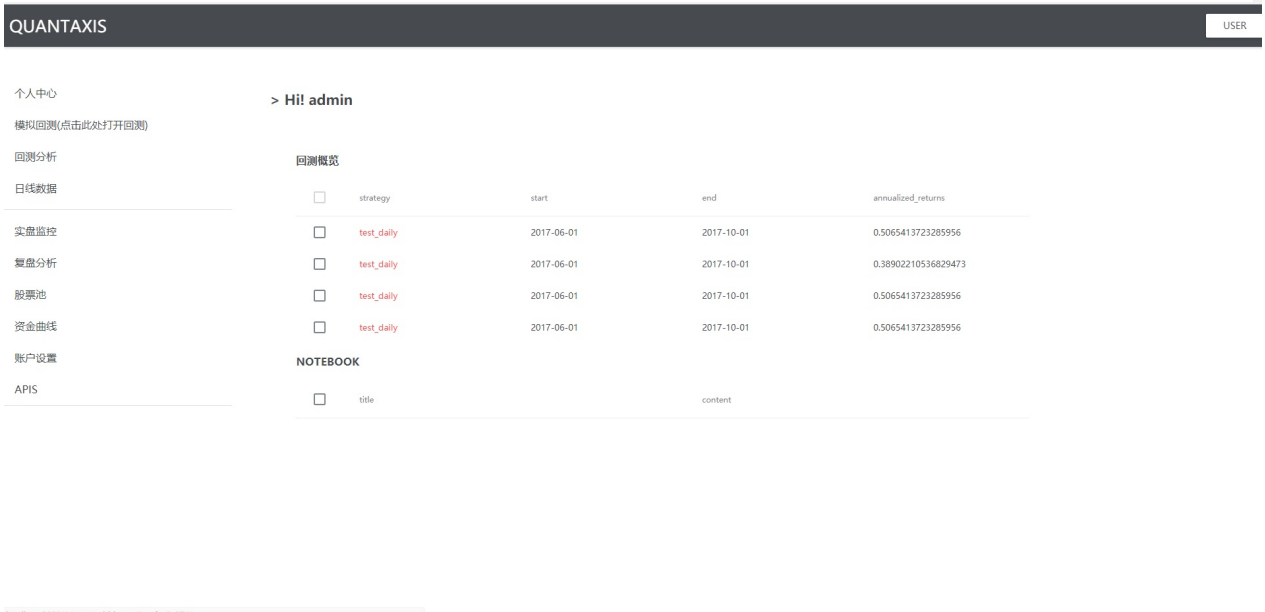
11. 回测Webkit插件概览

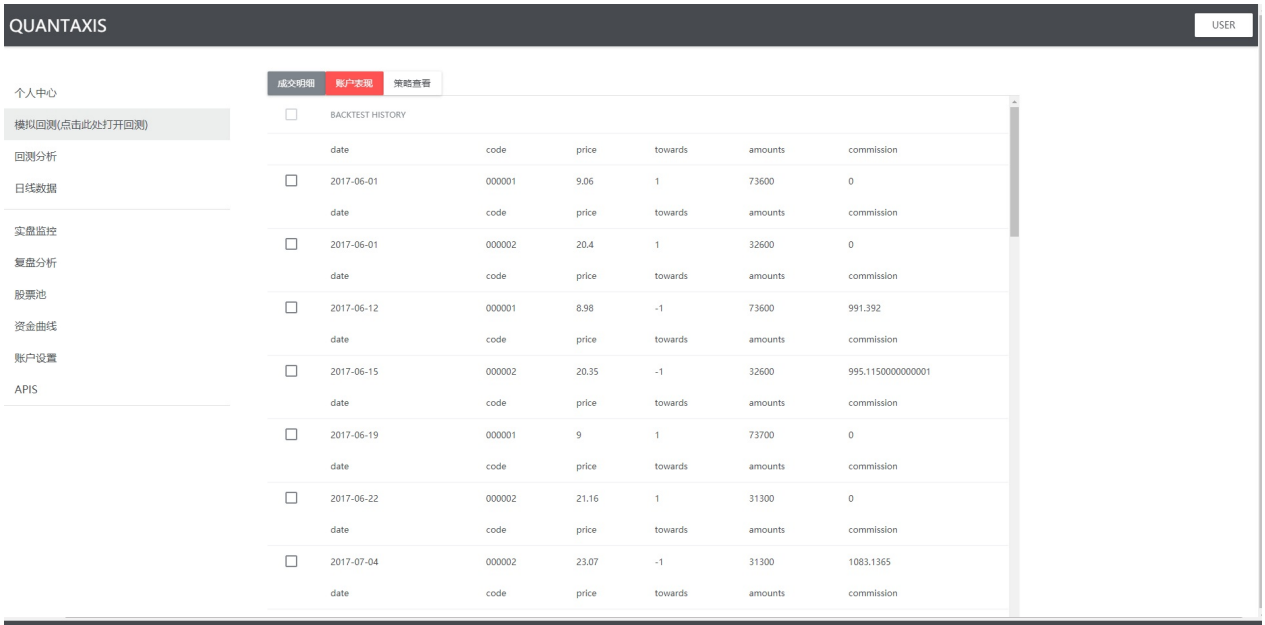


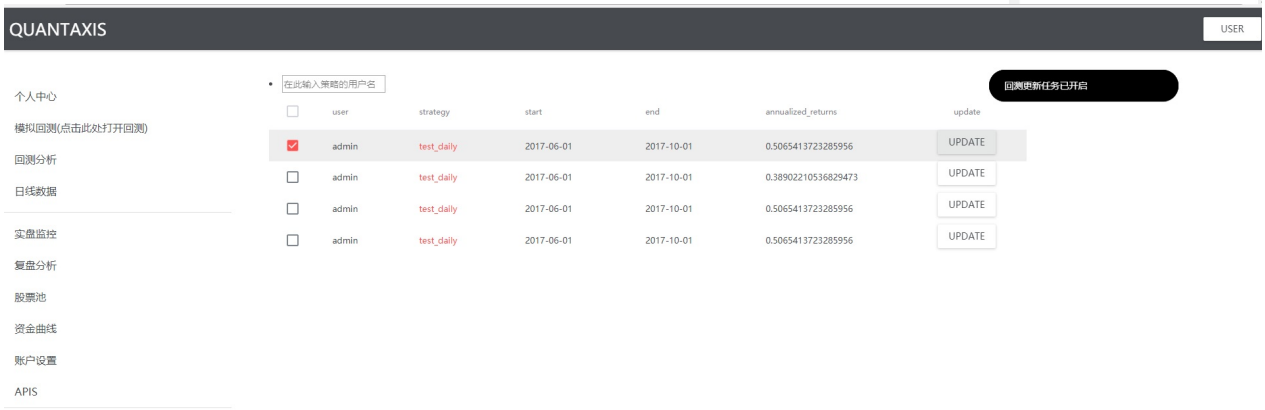
#QUANTAXIS

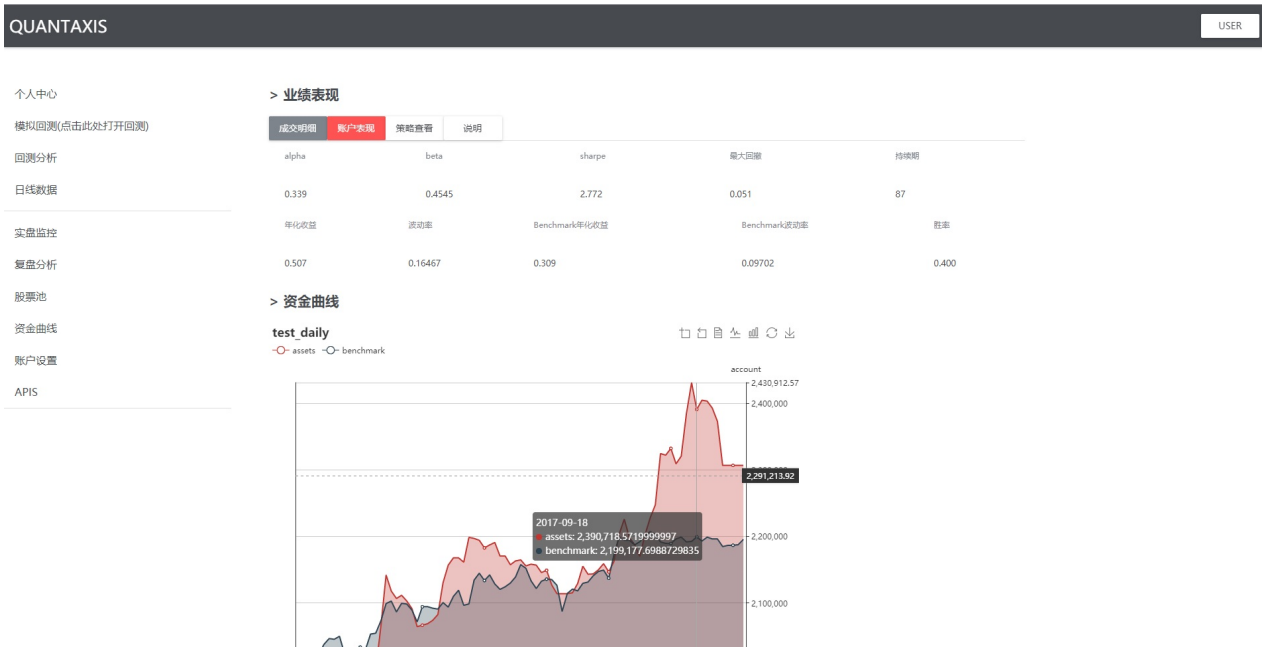
Intergrated Quantitative System | WebKit Visualization Soulution






QUANTAXIS
USER
个人中心
模拟回测(点击此处打开回测)
回测分析
日线数据
实盘监控
复盘分析
股票池
资金曲线
账户设置
APIS
成交明细
账户表现
策略查看
BACKTEST HISTORY
date code price towards amounts commission
2017-06-01 000001 9.06 1 73600 0
2017-06-01 000002 20.4 1 32600 0
2017-06-12 000001 8.98 -1 73600 991.392
2017-06-15 000002 20.35 -1 32600 995.1150000000001
2017-06-19 000001 9 1 73700 0
2017-06-22 000002 21.16 1 31300 0
2017-07-04 000002 23.07 -1 31300 1083.1365


QUANTAXIS
USER
个人中心
模拟回测(点击此处打开回测)
回测分析
日线数据
实盘监控
复盘分析
股票池
资金曲线
账户设置
APIS
在此输入策略的用户名
回测更新任务已开启
user strategy start end annualized_returns update
admin test_daily 2017-06-01 2017-10-01 0.5065413723285956 UPDATE
admin test_daily 2017-06-01 2017-10-01 0.38902210536829473 UPDATE
admin test_daily 2017-06-01 2017-10-01 0.5065413723285956 UPDATE
admin test_daily 2017-06-01 2017-10-01 0.5065413723285956 UPDATE


QUANTAXIS
USER
个人中心
模拟回测(点击此处打开回测)
回测分析
日线数据
实盘监控
复盘分析
股票池
资金曲线
账户设置
APIS
> 业绩表现
成交明细
账户表现
策略查看
说明
alpha beta sharpe 最大回撤 持续期
0.339 0.4545 2.772 0.051 87
年化收益 波动率 Benchmark年化收益 Benchmark波动率 胜率
0.507 0.16467 0.309 0.09702 0.400
> 资金曲线
test_daily
assets benchmark
account
2,430,912.57
2,400,000
2,291,213.92
2,200,000
2,100,000
2017-09-18
assets: 2,390,718.5719999997
benchmark: 2,199,177.6988729835

QUANTAXIS USER

个人中心
模拟回测(点击此处打开回测)
回测分析
日线数据
实盘监控
复盘分析
股票池
资金曲线
账户设置
APIS

test_daily

成交明细 账户表现 策略查看

Version: V1 Topic:EXAMPLE Last_modified_time:2017-12-15T18:01:08.872Z

```
# coding=utf-8

from datetime import datetime

import numpy as np

import QUANTAXIS as QA
from QUANTAXIS import QA_Backtest as QB


"""
写在前面:
===============QUANTAXIS BACKTEST STOCK_DAY中的变量
常量:
QB.backtest_type 回测类型 day/1min/5min/15min/30min/60min/index_day/index_1min/index_5min/index_15min/index_30min/index_60min/
QB.account.message  当前账户消息
QB.account.cash  当前可用资金
QB.account.hold  当前账户持仓
QB.account.history  当前账户的历史交易记录
QB.account.assets 当前账户总资产
QB.account.detail 当前账户的交易对账单
QB.account.init_assest 账户的最初资金
QB.strategy_gap 前推日期
QB.strategy_name 策略名称

QB.strategy_stock_list 回测初始化的时候  输入的一个回测标的
QB.strategy_start_date 回测的开始时间
QB.strategy_end_date  回测的结束时间

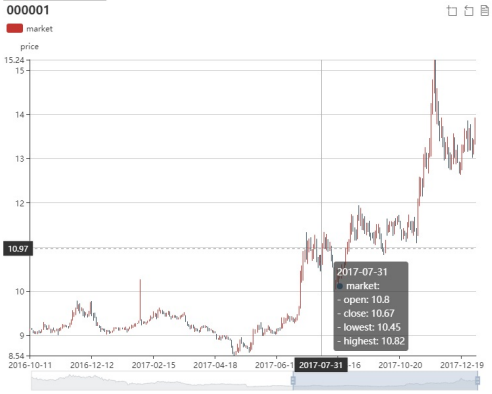
QB.setting.QA_setting_user_name = str('admin') #回测账户
QB.setting.QA_setting_user_password = str('admin') #回测密码
```

QUANTAXIS USER

个人中心
模拟回测(点击此处打开回测)
回测分析
日线数据
实盘监控
复盘分析
股票池
资金曲线
账户设置
APIS

> 日线数据查看

000001



QUANTAXIS USER

个人中心
模拟回测(点击此处打开回测)
回测分析
日线数据
实盘监控
复盘分析
股票池
资金曲线
账户设置
APIS

推送设置
暂时无法使用

通知设置

通知
实盘交易推送
回测推送

12. QUANTAXIS 标准化协议和未来协议

QUANTAXIS-Stardand-Protocol 版本号0.0.8

详情参见 QUANATXISProtocol

QUANTAXIS 捐赠列表

写代码不易...请作者喝杯咖啡呗?

(PS: 支付的时候 请带上你的名字/昵称呀 会维护一个赞助列表~)

==========

- 2017-9-25 沈乐
- 2017-9-25 许海涵
- 2017-9-27 吕少麟
- 2017-9-27 doskoi
- 2017-10-1 zz
- 2017-10-4 Hakase
- 2017-10-4 头上无毛
- 2017-10-5 庆斌
- 2017-10-09 成成
- 2017-10-09 Rainx 徐景
- 2017-10-09 Dice(steven)
- 2017-10-12 沈乐
- 2017-10-12 空空子

- 2017-10-12 宇清
- 2017-10-13 昊
- 2017-10-13 恒光
- 2017-10-30 Ims 黎明
- 2017-11-16 威
- 2017-11-26 SunnyBoy00
- 2017-12-05 威
- 2017-12-06 PdlMojoMoo
- 2017-12-07 威
- 2017-12-19 在云端
- 2017-12-21 双宏
- 2017-12-28 在云端
- 2018-02-03 *明龙
- 2018-02-16 周文扬

QUANTAXIS 的安装/部署/更新

部署问题:

- Windows/Linux(ubuntu)/Mac 已测试通过
- python3.6(开发环境) python2 回测框架不兼容(attention! 之后会逐步用更多高级语法) [*] 如果需要交易,请下载32位的python3.6
- nodejs 需要安装>7的版本,来支持es6语法
- mongodb是必须要装的
- 强烈推荐mongodb的可视化库 robomongo 百度即可下载

一个简易demo(需要先安装并启动mongodb,python版本需要大于3)

安装

- WINDOWS安装 参见 windows
- Ubuntu安装 参见 Ubuntu
- MAC 安装 参见 MAC
- 便捷版本 参见 Portable-QA

启动QUANTAXIS CLI 并进行数据的初始化存储

在命令行输入 quantaxis 进去quantaxis CLI

```
quantaxis> save all
quantaxis> save stock_block
quantaxis> save stock_info
```

随意新建一个目录:(不要跟QUANTAXIS文件夹在一个目录)

在命令行输入 quantaxis 进去quantaxis CLI

输入examples 在当前目录下生成一个示例策略

运行这个示例策略:

python backtest.py

一般而言 日线4个组合的回测(一年)在14-17秒左右 5min级别4个组合的回测(一年)在3-4分钟左右

启动QUANTAXIS_Webkit来查看回测的结果

启动网络插件(nodejs 版本号需要大于6,最好是7)

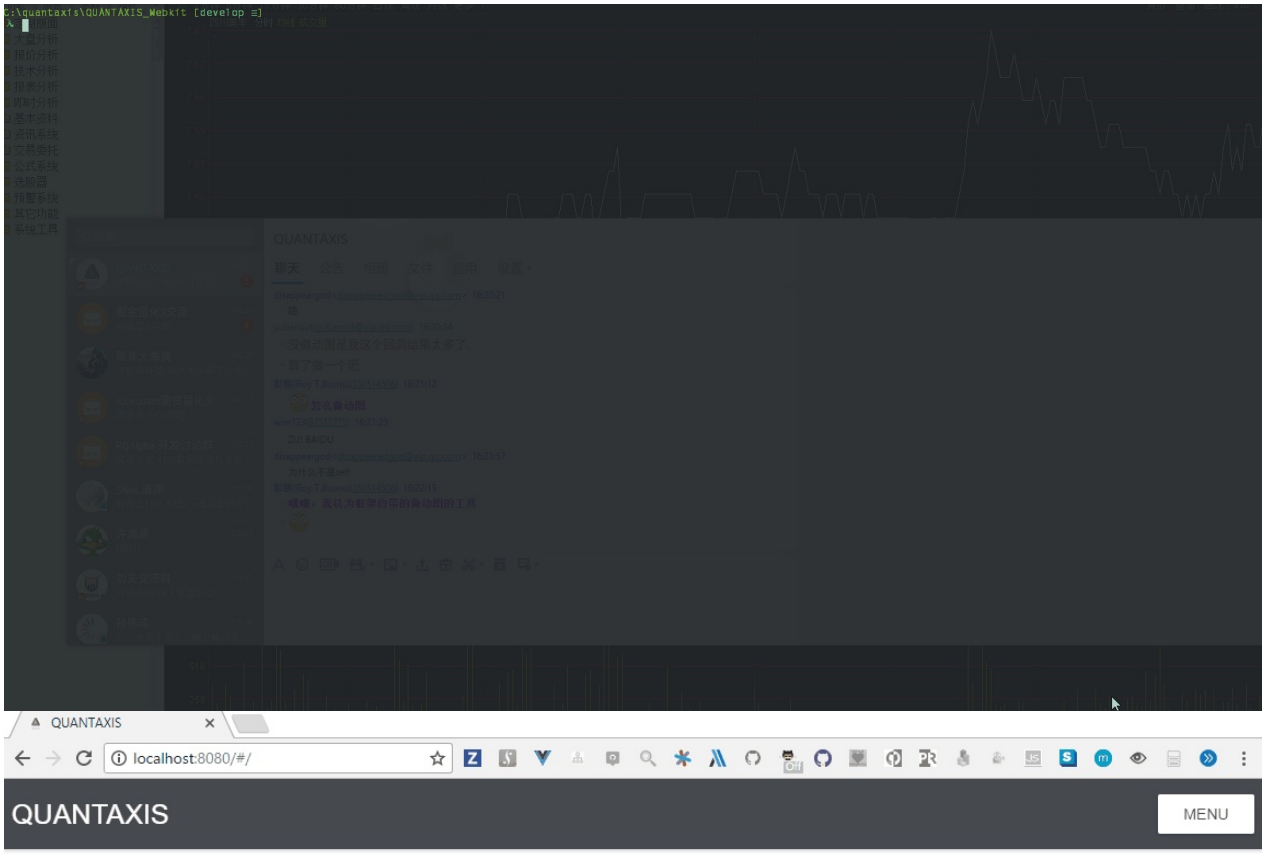
```
cd QUANTAXIS_Webkit
# 先启动后台服务器  在3000端口
cd backend
(sudo) forever start bin/www
cd ..
# 再启动前端服务器  在8080端口
cd web
(sudo) npm run dev 或者 forever start build/dev-server.js
```

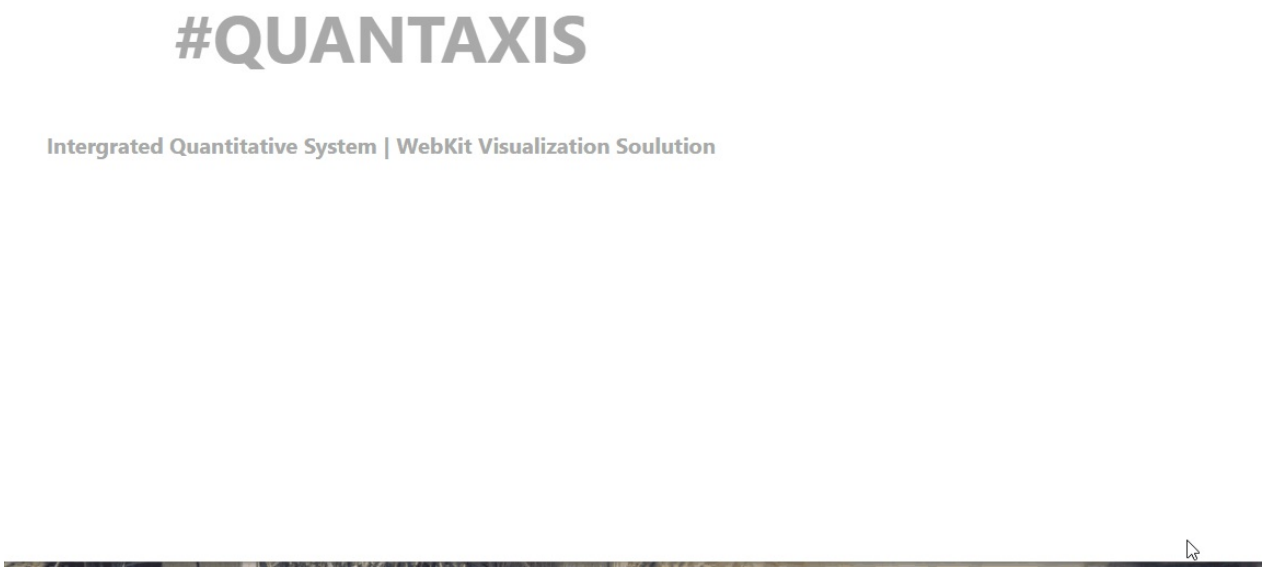
会自动启动localhost:8080网页端口,用账户名admin,密码admin登录 (注明: admin注册是在python的QUANTAXIS save all时候执行的)

另外 如果save all已经执行,依然登录不进去 点击插件状态 查看3000端口是否打开

登录后点击左上角 <模拟回测> 在模拟回测的选择界面的用户名搜索框输入回测的时候的用户名(默认是admin),回车

选择和你回测策略中名称一致的结果即可进入可视化界面





(web操作的图太大 github上无法显示, 可以点进链接查看)

更新QUANTAXIS

由于目前项目还在开发中,所以需要使用Git来更新项目:

点击右上角的Star和watch来持续跟踪项目进展~

常规更新:

```
cd QUANTAXIS
git pull
```

如果本地有进行更改,遇到更新失败:

(注意: 最好不要在本地修改该项目文件,如果需要做一些自定义功能,可以进fork[在项目的右上角])

```
git reset --hard origin/master
git pull
```

QUANTAXIS 的安装 WIN篇

1. 部署问题:

- Windows/Linux(ubuntu) 已测试通过
- python3.6(开发环境) python2 回测框架不兼容(attention! 之后会逐步用更多高级语法) [*] 如果需要交易,请下载32位的python3.6
- nodejs 需要安装>7的版本,来支持es6语法
- mongodb是必须要装的
- 强烈推荐mongodb的可视化库 robomongo 百度即可下载

一个简易demo(需要先安装并启动mongodb,python版本需要大于3)

2. 下载PYTHON(可以跳过)

QUANATXIS 支持的安装环境是python3以上 优先推荐3.6环境

在windows下,推荐使用ANACONDA集成环境来安装python[推荐Anaconda3-5.0.1-Windows-x86_64.exe]

(由于anaconda较大而官网的速度较慢,推荐去清华的anaconda镜像站下载)

清华镜像ANACONDA链接

3. 安装(可以跳过)

在安装ANACONDA的过程中,注意勾选 `add to path` 选项,将python的执行路径加入系统路径中

在安装完成后,可以使用 `python -V` 来验证是否成功

```
λ  python -V
Python 3.6.3 :: Anaconda, Inc.
```

4. 下载git

QUANTAXIS的代码托管在github,你需要经常用过 `git pull` 来更新代码,所以请勿直接在网站上下载zip压缩包

git 下载地址

同样,在安装的时候 选择 `add to path`

5. 使用git下载QUANTAXIS

打开命令行(推荐使用powershell) 选择你想要的目录 下载quantaxis

```
WIN键+R 在运行中输入 powershell 回车

cd C:\
git clone https://github.com/yutiansut/quantaxis --depth 1
```

6. 安装QUANTAXIS的依赖项

```
cd C:\quantaxis

python -m pip install -r requirements.txt -i https://pypi.douban
io.com/simple
python -m pip install tushare
python -m pip install pytdx
python -m pip install -e . -i https://pypi.doubanio.com/simple
```

完成以后 在命令行输入 `quantaxis` 即可进入QUANTAXIS的cli界面

```
λ  quantaxis
QUANTAXIS>> start QUANTAXIS
QUANTAXIS>> Selecting the Best Server IP of TDX
QUANTAXIS>> === The BEST SERVER ===
 stock_ip 115.238.90.165 future_ip 61.152.107.141
QUANTAXIS>> Welcome to QUANTAXIS, the Version is remake-version
QUANTAXIS>
```

7. 下载安装数据库

QUANTAXIS使用MONGODB数据库作为数据存储,需要下载数据库

下载地址 下载地址 MongoDB 64位 3.4.7

安装以后,需要在本地新建一个文件夹作为数据存储的文件夹,示例中,我们建在D盘

```
# 打开Powershell(Win键+R 在运行中输入Powershell)
cd D:
md data
# 然后在data目录下 新建一个data目录用于存放mongo的数据,log目录用于存放lo
g
cd data
md data
md log
# 到Mongo的程序文件夹下,使用命令
cd C:\Program Files\MongoDB\Server\3.4\bin
# 用mongod 命令安装
.\mongod.exe --dbpath  D:\data\data  --logpath D:\data\log\mongo
.log --httpinterface --rest --serviceName 'MongoDB' --install


# 如果你下载了3.6版本以上的mongodb 则使用

.\mongod.exe --dbpath  D:\data\data  --logpath D:\data\log\mongo
.log --serviceName 'MongoDB' --install
```

开启数据库服务

```
# 启动mongodb服务
net start MongoDB
```

8. 安装QUANTAXIS的web插件

QUANTAXIS使用了nodejs写了web部分的插件,所以需要下载nodejs

nodejs下载地址

注意: 需要下载的是nodejs8的版本,切勿下载9版本的nodejs

安装时也需要 `add to path`

安装完成后,在命令行输入 `node -v` 来查看是否安装成功

```
λ  node -v
v8.9.3
```

```
npm install cnpm -g
cnpm install forever -g

cd C:\quantaxis\QUANTAXIS_WEBKIT\backend
cnpm install

cd C:\quantaxis\QUANTAXIS_WEBKIT\web
cnpm install
```

9. 安装完成后 参见部署

部署

QUANTAXIS 的安装 Ubuntu篇

1. 一键部署

```
wget https://raw.githubusercontent.com/yutiansut/QUANTAXIS/master/config/ubuntu16.sh
sudo bash ./ubuntu16.sh
```

在脚本运行中,遇到输入的时候 一律选 `yes` 或者 `y`

2. 手动部署

3. 换源

```
echo "deb-src http://archive.ubuntu.com/ubuntu xenial main restr
icted #Added by software-properties
deb http://mirrors.aliyun.com/ubuntu/ xenial main restricted
deb-src http://mirrors.aliyun.com/ubuntu/ xenial main restricted
 multiverse universe #Added by software-properties
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates main restri
cted
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-updates main re
stricted multiverse universe #Added by software-properties
deb http://mirrors.aliyun.com/ubuntu/ xenial universe
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates universe
deb http://mirrors.aliyun.com/ubuntu/ xenial multiverse
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates multiverse
deb http://mirrors.aliyun.com/ubuntu/ xenial-backports main rest
ricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-backports main
restricted universe multiverse #Added by software-properties
deb http://archive.canonical.com/ubuntu xenial partner
deb-src http://archive.canonical.com/ubuntu xenial partner
deb http://mirrors.aliyun.com/ubuntu/ xenial-security main restr
icted
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-security main r
estricted multiverse universe #Added by software-properties
deb http://mirrors.aliyun.com/ubuntu/ xenial-security universe
deb http://mirrors.aliyun.com/ubuntu/ xenial-security multiverse
 " | tee /etc/apt/sources.list.d/sources.list


apt-get update
```

4. 安装python

```
apt install software-properties-common

add-apt-repository ppa:jonathonf/python-3.6
apt-get update

apt-get install python3.6-dev
wget https://bootstrap.pypa.io/get-pip.py

python3.6 get-pip.py
```

5. 安装git

```
apt-get install git
```

6. 下载安装quantaxis

```
cd ~
git clone https://github.com/yutiansut/quantaxis
cd ~/quantaxis
python3.6 -m pip install -r requirements.txt -i https://pypi.dou
banio.com/simple
python3.6 -m pip install tushare
python3.6 -m pip install pytdx
python3.6 -m pip install -e .
```

7. 安装mongo

```
apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 293
0ADAE8CAF5059EE73BB4B58712A2291FA4AD5
# Ubuntu 16.04
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubun
tu xenial/mongodb-org/3.6 multiverse" | sudo tee /etc/apt/source
s.list.d/mongodb-org-3.6.list

# 更新
apt-get update
# 安装MongoDB
apt-get install -y mongodb-org --allow-unauthenticated
```

8. 安装nodejs

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
apt-get install -y nodejs
apt-get install npm
npm install npm -g #更新npm
npm install forever -g #安装一个全局的forever 用于之后启动
npm install cnpm -g
```

9. 安装QUANTAXIS_WEBKIT

```
cd ~/quantaxis/QUANTAXIS_Webkit/backend
npm install


cd ~/quantaxis/QUANTAXIS_Webkit/web
npm install
```

QUANTAXIS 的安装 MAC篇

- 1. python
- 1. QUANTAXIS
- 1. MONGODB
- 1. NODEJS
- 1. QUANTAXIS_WEBKIT

1. python

```
brew install python3
```

2. QUANTAXIS

```
git clone https://github.com/yutiansut/quantaxis --depth 1
cd quantaxis .
sudo python3.6 -m pip install -r requirements.txt -i https://pyp
i.doubanio.com/simple
sudo python3.6 -m pip install tushare
sudo python3.6 -m pip install pytdx

python3.6 -m pip install -e .
```

3. MONGODB

安装mongodb:

```
brew update
brew install mongodb
```

创建数据库文件:

```
sudo mkdir -p /data/db
```

然后需要输入你的密码

启动mongodb

```
sudo mongod
```

继续输入你的密码

4. NODEJS

```
brew install node

sudo npm install cnpm -g
sudo npm install forever -g
```

5. QUANTAXIS_WEBKIT

```
cd QUANTAXIS_WEBKIT\web
cnpm install

cd QUANTAXIS_WEBKIT\backend
cnpm install
```

使用Docker建立QUANTAXIS执行环境

1. QUANTAXIS的镜像

QUANTAXIS官方维护了2个镜像:

境外版本:

1. DOCKER网站下的 `yutiansut/quantaxis` 以及国内的加速版本 `registry.docker-cn.com/yutiansut/quantaxis`

境内版本/(享受阿里云内网加速):

1. 阿里云DOCKER= 上海镜像仓库的 `registry.cn-shanghai.aliyuncs.com/yutiansut/quantaxis`

其中 杭州的镜像是包括了 `node_modules` ,以及市场日线数据的镜像 比较大 适合只想一次性部署的同学们

阿里云上海仓库,DOCKER官网的镜像是同一份docker,包含了所有必需的程序 但是没有存储数据

	海外仓库	上海阿里云DOCKER仓库
描述	网速不好的轻量纯净版本	网速不好的轻量纯净版本
地址	国外(可以加速)	上海
系统	Ubuntu16.04	Ubuntu16.04
python入口名	python3.6	python
mongo	mongo 3.4 社区版	暂无
nodejs	nodejs 8.2.1	nodejs 8.9.3
sshserver	内置	暂无
QUANTAXIS 目录	/QUANTAXIS	/home/quantaxis
forever	有	有
web部分的依赖项	未安装	已安装
日线数据	未存储	未存储
版本号	V+数字版本	V+数字版本
JupyterNoteBook	暂不支持	支持

2. 1.获取QUANTAIS镜像

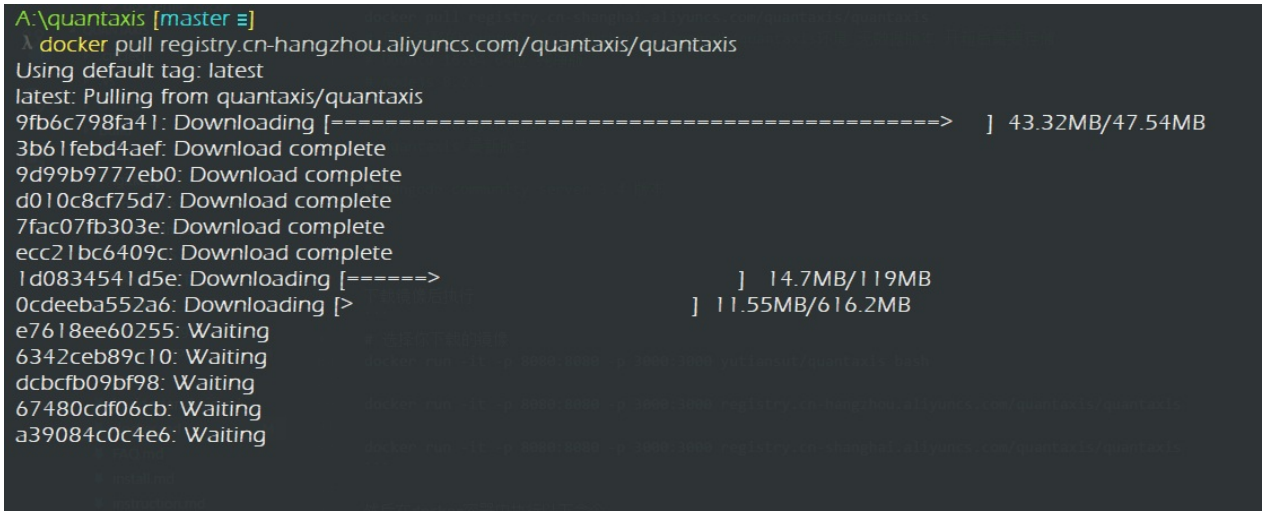
首先，到docker网站下载相应的版本，并创建账号（注意：登录docker账号才能下载镜像）

(如果国外网站下载速度过慢,windows版本的docker安装文件群共享有)

2.1. 1.1 执行以下命令获取镜像(2选1)

```
# 海外镜像(境外用户)
docker pull yutiansut/quantaxis


# 上海阿里云镜像(国内用户)
docker pull registry.cn-shanghai.aliyuncs.com/yutiansut/quantaxis
```



2.2. 1.2 运行镜像(2选1)

```
# 选择你下载的镜像
docker run -it -p 8080:8080 -p 3000:3000 yutiansut/quantaxis bas
h

# 带jupyter版本
docker run -it -e GRANT_SUDO=yes -p 8888:8888 -p 8080:8080 -p 30
00:3000 registry.cn-shanghai.aliyuncs.com/yutiansut/quantaxis
```

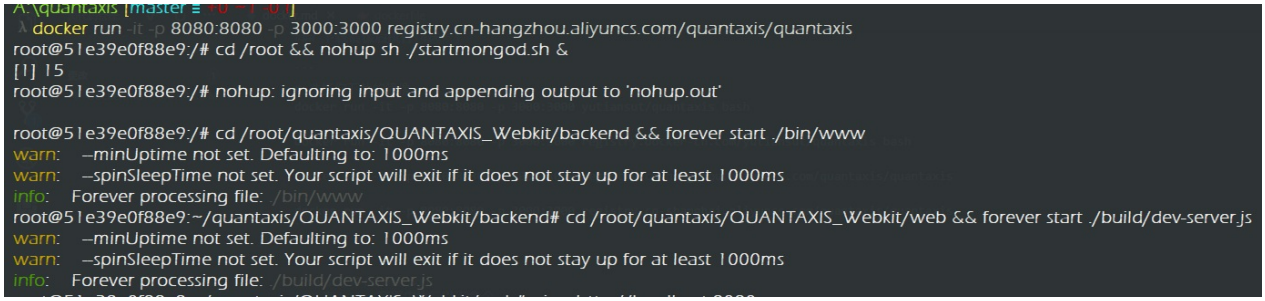
2.3. 1.3 在docker中执行命令(启动服务)

```
# 启动 mongodb
cd /home/quantaxis/config && nohup sh ./run_backend.sh &


# 启动 WEBKIT
cd /home/quantaxis/QUANTAXIS_Webkit/backend && forever start ./b
in/www

cd /home/quantaxis/QUANTAXIS_Webkit/web && forever start ./build
/dev-server.js

# 启动jupyter
cd /home/quantaxis/config && nohup sh ./startjupyter.sh &
```



2.4. 在浏览器中打开以下链接

```
http://localhost:8080
http://localhost:8888
```

2.5. 其他注意选项

1. docker 是可以通过ssh 连接的 `/etc/init.d/ssh start`
2. 多窗口

首先需要运行一个docker

```
A:\quantaxis [master ≡]
λ  docker run -it -p 8080:8080 -p 3000:3000 registry.cn-shanghai
.aliyuncs.com/yutiansut/quantaxis
root@f22b5357dc6e:/#
```

然后在别的命令行执行 `docker ps` 查询正在运行的docker的container_id

```
A:\Users\yutia
λ  docker ps


CONTAINER ID        IMAGE
            COMMAND             CREATED             STATUS
        PORTS                                              NAMES
f22b5357dc6e        registry.cn-shanghai.aliyuncs.com/yutiansut/
quantaxis   "bash"              21 seconds ago      Up 20 second
s       0.0.0.0:3000->3000/tcp, 0.0.0.0:8080->8080/tcp   boring_
panini
```

然后执行 `docker exec -it [CONTAINERID] /bin/bash` 进入

```
A:\Users\yutia
λ  docker exec -it  f22b5357dc6e /bin/bash
root@f22b5357dc6e:/#
```

QUANTAXIS更新

由于目前项目还在开发中,所以需要使用Git来更新项目:

点击右上角的Star和watch来持续跟踪项目进展~

常规更新:

```
cd QUANTAXIS
git pull
```

如果本地有进行更改,遇到更新失败:

(注意: 最好不要在本地修改该项目文件,如果需要做一些自定义功能,可以进fork[在项目的右上角])

```
git reset --hard origin/master
git pull
```

更新QUANTAXIS (python)

一般而言,本地开发模式 更新完代码就无需考虑pip install 重新安装的问题, 但如果涉及 setup.py的更新 则需要

```
cd QUANTAXIS
pip install -e .
```

更新QUANTAXIS_Webkit (nodejs)

当更新了package.json文件时,项目需要重新 npm install 来安装和更新依赖项

当出现报错时

```
cd QUANTAXIS_Webkit/web
npm install
```

一般可以解决问题

QUANTAXIS的核心数据结构以及方法

属性用@property装饰器装饰,进行懒运算 提高效率

- 1. 取一个股票的数据
- 1. 取多个股票的数据
- 1. 显示结构体的数据 .data
- 1. 显示结构体的开/高/收/低 .open/.high/.close/.low
- 1. 结构体拆分 splits()
- 1. 数据结构复权to_qfq()/to_hfq()
-

1. 数据透视 .pivot()

- 1. 数据的时间筛选.select_time(start,end)
- 1. 数据按时间往前/往后推 select_time_with_gap(time,gap,methods)
- 1. 选取某一个月份的数据 select_month(month)
- 1. 选取结构组里面某一只股票select_code(code)
- 1. 取某一只股票的某一个时间的bar(code,time,if_trade)
- 1. 画图 plot(code)
- 1. 统计学部分
    - 14.1. 平均价 price
    - 14.2. price均值 mean
    - 14.3. max/min
    - 14.4. 方差/样本方差 pvariance/variance
    - 14.5. 标准差/样本标准差 pstdev/stdev
    - 14.6. 调和平均数 mean_harmonic
    - 14.7. 众数 mode
    - 14.8. 振幅 amplitude
    - 14.9. 偏度 skew
    - 14.10. 峰度 kurt
    - 14.11. 百分比变化 pct_change
    - 14.12. 平均绝对偏差 mad
    - 14.13. 价格差分 price_diff

QA_DataStruct具有的功能:

- 数据容器
- 数据变换 [分拆/合并/倒序] split/merge/reverse
- 数据透视 pivot
- 数据筛选 select_time/select_time_with_gap/select_code/get_bar/select_month
- 数据复权 to_qfq/to_hfq

- 数据显示 show
- 格式变换 to_json/to_pandas/to_list/to_numpy/to_hdf
- 数据库式查询 query
- 画图 plot
- 计算指标 add_func
- 生成器 panel_gen(按时间分类的面板生成器)/security_gen(按股票分类的股票生成器)

QA_DataStruct_Stock_block

- (属性)该类下的所有板块名称 block_name
- 查询某一只股票所在的所有板块 get_code(code)
- 查询某一个板块下的所有股票 get_block(block)
- 展示当前类下的所有数据 show

我们可以通过

```
import QUANTAXIS as QA

# QA.QA_fetch_stock_day_adv
# QA.QA_fetch_stock_min_adv
# QA.QA_fetch_index_day_adv
# QA.QA_fetch_index_min_adv
```

day线的参数是code, start, end min线的参数是code, start, end, frequence='1min'

其中 code 可以是一个股票,也可以是一列股票(list)

1. 取一个股票的数据

```
QA.QA_fetch_stock_day_adv('000001','2017-01-01','2017-10-01')
In [5]: QA.QA_fetch_stock_day_adv('000001','2017-01-01','2017-10
-01')
Out[5]: QA_DataStruct_Stock_day with 1 securities
```

2. 取多个股票的数据

```
QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-01-01','2017
-10-01')
In [6]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-01-0
1','2017-10-01')
Out[6]: QA_DataStruct_Stock_day with 2 securities
```

3. 显示结构体的数据 .data

```
In [10]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').data
Out[10]:
                     code   open   high    low  close    volume
      date
date       code
2017-09-20 000001  000001  11.14  11.37  11.05  11.29  787154.0
2017-09-20
2017-09-21 000001  000001  11.26  11.51  11.20  11.46  692407.0
2017-09-21
2017-09-22 000001  000001  11.43  11.52  11.31  11.44  593927.0
2017-09-22
2017-09-25 000001  000001  11.44  11.45  11.18  11.29  532391.0
2017-09-25
2017-09-26 000001  000001  11.26  11.30  10.96  11.05  967460.0
2017-09-26
2017-09-27 000001  000001  11.01  11.08  10.90  10.93  727188.0
2017-09-27
2017-09-28 000001  000001  10.98  10.98  10.82  10.88  517220.0
2017-09-28
2017-09-29 000001  000001  10.92  11.16  10.86  11.11  682280.0
2017-09-29
2017-09-20 000002  000002  28.50  29.55  28.00  28.73  613095.0
2017-09-20
2017-09-21 000002  000002  28.50  29.06  27.75  28.40  536324.0
2017-09-21
2017-09-22 000002  000002  28.39  28.67  27.52  27.81  423093.0
2017-09-22
2017-09-25 000002  000002  27.20  27.20  26.10  26.12  722702.0
2017-09-25
2017-09-26 000002  000002  26.12  27.22  26.10  26.76  593044.0
2017-09-26
2017-09-27 000002  000002  27.00  27.28  26.52  26.84  367534.0
2017-09-27
2017-09-28 000002  000002  27.00  27.15  26.40  26.41  262347.0
2017-09-28
2017-09-29 000002  000002  26.56  26.80  26.00  26.25  345752.0
2017-09-29
```

4. 显示结构体的开/高/收/低 .open/.high/.close/.low

```
In [5]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-2
0','2017-10-01').high
Out[5]:
date        code
2017-09-20  000001    11.37
2017-09-21  000001    11.51
2017-09-22  000001    11.52
2017-09-25  000001    11.45
2017-09-26  000001    11.30
2017-09-27  000001    11.08
2017-09-28  000001    10.98
2017-09-29  000001    11.16
2017-09-20  000002    29.55
2017-09-21  000002    29.06
2017-09-22  000002    28.67
2017-09-25  000002    27.20
2017-09-26  000002    27.22
2017-09-27  000002    27.28
2017-09-28  000002    27.15
2017-09-29  000002    26.80
Name: high, dtype: float64
```

5. 结构体拆分 splits()

当一个DataStruct里面存在多个证券时,可以通过拆分的方法,将其变成多个DataStruct

```
In [3]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-2
0','2017-10-01').splits()
Out[3]:
[< QA_DataStruct_Stock_day with 1 securities >,
 < QA_DataStruct_Stock_day with 1 securities >]
```

6. 数据结构复权to_qfq()/to_hfq()

返回的是一个DataStruct,用.data展示返回的数据的结构

其中DataStruct.if_fq的属性会改变

```
In [4]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-2
0','2017-10-01').to_qfq().data

Out[4]:
                     code   open   high    low  close    volume
      date  \
date       code
2017-09-20 000001  000001  11.14  11.37  11.05  11.29  787154.0
2017-09-20
2017-09-21 000001  000001  11.26  11.51  11.20  11.46  692407.0
2017-09-21
2017-09-22 000001  000001  11.43  11.52  11.31  11.44  593927.0
2017-09-22
2017-09-25 000001  000001  11.44  11.45  11.18  11.29  532391.0
2017-09-25
2017-09-26 000001  000001  11.26  11.30  10.96  11.05  967460.0
2017-09-26
2017-09-27 000001  000001  11.01  11.08  10.90  10.93  727188.0
2017-09-27
2017-09-28 000001  000001  10.98  10.98  10.82  10.88  517220.0
2017-09-28
2017-09-29 000001  000001  10.92  11.16  10.86  11.11  682280.0
2017-09-29
2017-09-20 000002  000002  28.50  29.55  28.00  28.73  613095.0
2017-09-20
2017-09-21 000002  000002  28.50  29.06  27.75  28.40  536324.0
2017-09-21
2017-09-22 000002  000002  28.39  28.67  27.52  27.81  423093.0
2017-09-22
2017-09-25 000002  000002  27.20  27.20  26.10  26.12  722702.0
2017-09-25
2017-09-26 000002  000002  26.12  27.22  26.10  26.76  593044.0
2017-09-26
2017-09-27 000002  000002  27.00  27.28  26.52  26.84  367534.0
```

```
2017-09-27
2017-09-28 000002  000002  27.00  27.15  26.40  26.41  262347.0
2017-09-28
2017-09-29 000002  000002  26.56  26.80  26.00  26.25  345752.0
2017-09-29
```

```
                  preclose  adj
date       code
2017-09-20 000001       NaN  1.0
2017-09-21 000001     11.29  1.0
2017-09-22 000001     11.46  1.0
2017-09-25 000001     11.44  1.0
2017-09-26 000001     11.29  1.0
2017-09-27 000001     11.05  1.0
2017-09-28 000001     10.93  1.0
2017-09-29 000001     10.88  1.0
2017-09-20 000002       NaN  1.0
2017-09-21 000002     28.73  1.0
2017-09-22 000002     28.40  1.0
2017-09-25 000002     27.81  1.0
2017-09-26 000002     26.12  1.0
2017-09-27 000002     26.76  1.0
2017-09-28 000002     26.84  1.0
2017-09-29 000002     26.41  1.0
```

7. 数据透视 .pivot()

```
In [6]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-2
0','2017-10-01').pivot('open')
Out[6]:
code         000001  000002
date
2017-09-20   11.14   28.50
2017-09-21   11.26   28.50
2017-09-22   11.43   28.39
2017-09-25   11.44   27.20
2017-09-26   11.26   26.12
2017-09-27   11.01   27.00
2017-09-28   10.98   27.00
2017-09-29   10.92   26.56
```

8. 数据的时间筛选.select_time(start,end)

```
In [10]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').select_time('2017-09-20','2017-09-25')
Out[10]: QA_DataStruct_Stock_day with 2 securities

In [11]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').select_time('2017-09-20','2017-09-25').data
Out[11]:
                     code   open   high    low  close    volume
      date
date       code
2017-09-20 000001  000001  11.14  11.37  11.05  11.29  787154.0
2017-09-20
2017-09-21 000001  000001  11.26  11.51  11.20  11.46  692407.0
2017-09-21
2017-09-22 000001  000001  11.43  11.52  11.31  11.44  593927.0
2017-09-22
2017-09-25 000001  000001  11.44  11.45  11.18  11.29  532391.0
2017-09-25
2017-09-20 000002  000002  28.50  29.55  28.00  28.73  613095.0
2017-09-20
2017-09-21 000002  000002  28.50  29.06  27.75  28.40  536324.0
2017-09-21
2017-09-22 000002  000002  28.39  28.67  27.52  27.81  423093.0
2017-09-22
2017-09-25 000002  000002  27.20  27.20  26.10  26.12  722702.0
2017-09-25
```

9. 数据按时间往前/往后推 select_time_with_gap(time,gap,methods)

time是你选择的时间 gap是长度 (int) methods有'<=','lte','<','lt','eq','==','>','gt','>=','gte'的选项

```
In [14]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').select_time_with_gap('2017-09-20',2,'gt')
Out[14]: QA_DataStruct_Stock_day with 2 securities

In [15]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').select_time_with_gap('2017-09-20',2,'gt').data
Out[15]:
                     code   open   high    low  close    volume
      date
date       code
2017-09-21 000001  000001  11.26  11.51  11.20  11.46  692407.0
2017-09-21
2017-09-22 000001  000001  11.43  11.52  11.31  11.44  593927.0
2017-09-22
2017-09-21 000002  000002  28.50  29.06  27.75  28.40  536324.0
2017-09-21
2017-09-22 000002  000002  28.39  28.67  27.52  27.81  423093.0
2017-09-22
```

10. 选取某一个月份的数据 select_month(month)

可以通过select_month 来选取某一个月份的数据

```
In [4]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-2
0','2017-10-01').select_month('2017-09')
Out[4]: < QA_DataStruct_Stock_day with 2 securities >
```

11. 选取结构组里面某一只股票select_code(code)

```
In [16]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').select_code('000001')
Out[16]: QA_DataStruct_Stock_day with 1 securities
In [17]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').select_code('000001').data
Out[17]:
                     code   open   high    low  close    volume
      date
date       code
2017-09-20 000001  000001  11.14  11.37  11.05  11.29  787154.0
2017-09-20
2017-09-21 000001  000001  11.26  11.51  11.20  11.46  692407.0
2017-09-21
2017-09-22 000001  000001  11.43  11.52  11.31  11.44  593927.0
2017-09-22
2017-09-25 000001  000001  11.44  11.45  11.18  11.29  532391.0
2017-09-25
2017-09-26 000001  000001  11.26  11.30  10.96  11.05  967460.0
2017-09-26
2017-09-27 000001  000001  11.01  11.08  10.90  10.93  727188.0
2017-09-27
2017-09-28 000001  000001  10.98  10.98  10.82  10.88  517220.0
2017-09-28
2017-09-29 000001  000001  10.92  11.16  10.86  11.11  682280.0
2017-09-29
```

12. 取某一只股票的某一个时间的 bar(code,time,if_trade)

第三个选项 默认是True

第三选项的意义在于,如果出现了停牌,参数如果是True 那么就会返回空值 而如果是False,就会返回停牌前最后一个交易日的值

```
In [18]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').get_bar('000001','2017-09-20',True)
Out[18]: QA_DataStruct_Stock_day with 1 securities

In [19]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').get_bar('000001','2017-09-20',True).data
Out[19]:
                     code   open   high    low  close    volume
      date
date       code
2017-09-20 000001  000001  11.14  11.37  11.05  11.29  787154.0
2017-09-20
```

13. 画图 plot(code)

如果是()空值 就会把全部的股票都画出来

```
In [20]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').plot()
QUANTAXIS>> The Pic has been saved to your path: .\QA_stock_day_
codepackage_bfq.html

In [21]: QA.QA_fetch_stock_day_adv(['000001','000002'],'2017-09-
20','2017-10-01').plot('000001')
QUANTAXIS>> The Pic has been saved to your path: .\QA_stock_day_
000001_bfq.html
```

14. 统计学部分

14.1. 平均价 price

为了统计学指标的需要, price=AVERAGE(open+high+low+close)

price是一个 `pd.Series` 类

```
In [7]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10
-01').price
Out[7]:
date        code
2017-09-20  000001    11.2125
2017-09-21  000001    11.3575
2017-09-22  000001    11.4250
2017-09-25  000001    11.3400
2017-09-26  000001    11.1425
2017-09-27  000001    10.9800
2017-09-28  000001    10.9150
2017-09-29  000001    11.0125
dtype: float64
```

14.2. price均值 mean

mean是price的均值

```
In [6]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10
-01').mean
Out[6]: 11.173125
```

14.3. max/min

max/min 分别是price序列的最大值和最小值

```
In [8]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10
-01').max
Out[8]: 11.424999999999999

In [9]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10
-01').min
Out[9]: 10.915000000000001
```

14.4. 方差/样本方差 pvariance/variance

分别是price的方差和样本方差

```
In [10]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-1
0-01').variance
Out[10]: 0.0367852678571427

In [11]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-1
0-01').pvariance
Out[11]: 0.03218710937499986
```

14.5. 标准差/样本标准差 pstdev/stdev

分别是price的总体标准差和样本标准差

```
In [12]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10-01').pstdev
Out[12]: 0.17940766253145338

In [13]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10-01').stdev
Out[13]: 0.19179485878704544
```

14.6. 调和平均数 mean_harmonic

price的调和平均数

```
In [14]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10-01').mean_harmonic
Out[14]: 11.170242242781745
```

14.7. 众数 mode

返回price的众数 (注意: price序列可能没有众数,因此可能会报错,内部处理后,返回None)

```
In [31]: QA.QA_fetch_stock_day_adv('000001','2017-01-20','2017-10-01').mode
Out[31]: 9.1375

In [31]: QA.QA_fetch_stock_day_adv('000001','2017-01-20','2017-10-01').mode
Out[31]: None
```

14.8. 振幅 amplitude

返回price的振幅

```
In [33]: QA.QA_fetch_stock_day_adv('000001','2017-01-20','2017-10-01').amplitude
Out[33]: 3.1325000000000003
```

14.9. 偏度 skew

返回price的偏度

```
In [35]: QA.QA_fetch_stock_day_adv('000001','2017-01-20','2017-10-01').skew
Out[35]: 0.70288041557825753
```

14.10. 峰度 kurt

返回price的峰度

```
In [37]: QA.QA_fetch_stock_day_adv('000001','2017-01-20','2017-10-01').kurt
Out[37]: -1.0703273213086726
```

14.11. 百分比变化 pct_change

返回price的百分比变化

```
In [40]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10-01').pct_change
Out[40]:
date        code
2017-09-20  000001         NaN
2017-09-21  000001    0.012932
2017-09-22  000001    0.005943
2017-09-25  000001   -0.007440
2017-09-26  000001   -0.017416
2017-09-27  000001   -0.014584
2017-09-28  000001   -0.005920
2017-09-29  000001    0.008933
dtype: float64
```

14.12. 平均绝对偏差 mad

返回price的平均绝对偏差

```
In [41]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10-01').mad
Out[41]: 0.16062499999999957
```

14.13. 价格差分 price_diff

返回价格的一阶差分

```
In [42]: QA.QA_fetch_stock_day_adv('000001','2017-09-20','2017-10-01').price_diff
Out[42]:
date        code
2017-09-20  000001       NaN
2017-09-21  000001    0.1450
2017-09-22  000001    0.0675
2017-09-25  000001   -0.0850
2017-09-26  000001   -0.1975
2017-09-27  000001   -0.1625
2017-09-28  000001   -0.0650
2017-09-29  000001    0.0975
dtype: float64
```

QUANTAXIS的指标系统

QUANTAXIS的核心数据结构有一个方法叫add_func(func,*args,*kwargs),作为一个指标入口,会返回一个和DataStruct中股票数量一致长度的list

QUANTAXIS有两种类型的指标:

- 基础指标(输入为Series的指标)
- 应用级指标(可应用于DataStruct的指标)

其中,基础指标是为了应用级指标做准备的,及对应于Series的分析和dataframe的分析的关系

基础类指标 [基本和同花顺/通达信一致]

```
import QUANTAXIS as QA
QA.MA(Series, N)
QA.EMA(Series, N)
QA.SMA(Series, N, M=1)
QA.DIFF(Series, N=1)
QA.HHV(Series, N)
QA.LLV(Series, N)
QA.SUM(Series, N)
QA.ABS(Series)
QA.MAX(A, B)
QA.MIN(A, B)
QA.CROSS(A, B)
QA.COUNT(COND, N)
QA.IF(COND, V1, V2)
QA.REF(Series, N)
QA.STD(Series, N)
QA.AVEDEV(Series, N)
QA.BBIBOLL(Series, N1, N2, N3, N4, N, M)
```

应用级指标 add_func(func)

```
import QUANTAXIS as QA
QA.QA_indicator_OSC(DataFrame, N, M)
QA.QA_indicator_BBI(DataFrame, N1, N2, N3, N4)
QA.QA_indicator_PBX(DataFrame, N1, N2, N3, N4, N5, N6)
QA.QA_indicator_BOLL(DataFrame, N)
QA.QA_indicator_ROC(DataFrame, N, M)
QA.QA_indicator_MTM(DataFrame, N, M)
QA.QA_indicator_KDJ(DataFrame, N=9, M1=3, M2=3)
QA.QA_indicator_MFI(DataFrame, N)
QA.QA_indicator_ATR(DataFrame, N)
QA.QA_indicator_SKDJ(DataFrame, N, M)
QA.QA_indicator_WR(DataFrame, N, N1)
QA.QA_indicator_BIAS(DataFrame, N1, N2, N3)
QA.QA_indicator_RSI(DataFrame, N1, N2, N3)
QA.QA_indicator_ADTM(DataFrame, N, M)
QA.QA_indicator_DDI(DataFrame, N, N1, M, M1)
QA.QA_indicator_CCI(DataFrame, N=14)
```

自己写一个指标:

比如 绝路航标

```
import QUANTAXIS as QA
def JLHB(data, m=7, n=5):
    """
    通达信定义
    VAR1:=(CLOSE-LLV(LOW,60))/(HHV(HIGH,60)-LLV(LOW,60))*80;
    B:SMA(VAR1,N,1);
    VAR2:SMA(B,M,1);
    绝路航标:IF(CROSS(B,VAR2) AND B<40,50,0);
    """
    var1 = (data['close'] - QA.LLV(data['low'], 60)) / \
        (QA.HHV(data['high'], 60) - QA.LLV(data['low'], 60)) * 80

    B = QA.SMA(var1, m)
    var2 = QA.SMA(B, n)
    if QA.CROSS(B,var2) and B[-1]<40:
        return 1
    else:
        return 0

# 得到指标
QA.QA_fetch_stock_day_adv('000001','2017-01-01','2017-03-31').to
_qfq().add_func(JLHB)
```

QUANTAXIS的数据获取部分

- 1. 从网上获取 | FROM WEBSITE
    - 1.1. 股票/日线 | STOCK_CN/DAY
    - 1.2. 股票/分钟线 | STOCK_CN/MIN
    - 1.3. 股票/权息数据 | STOCK_CN/XDXR
    - 1.4. 股票/列表 | STOCK_CN/LIST
    - 1.5. 指数/列表 | IDNEX_CN/LIST
    - 1.6. 指数/日线 | INDEX_CN/DAY
    - 1.7. 指数/分钟线 | INDEX_CN/MIN
    - 1.8. 最新交易价格STOCK | LAST PRICE
    - 1.9. 实时上下五档 STOCK_CN/QUOTATION
    - 1.10. 分笔数据 | STOCK_CN/TRANSACTION
    - 1.11. 版块数据 | STOCK_CN/BLOCK
- 1. 从数据库获取 | FROM LOCALHOST
    - 2.1. 股票日线 | STOCK_CN/DAY
    - 2.2. 股票分钟线 | STOCK_CN/MIN
    - 2.3. 指数/基金日线 | INDEX_CN,ETF_CN/DAY
    - 2.4. 指数/基金分钟线 | INDEX_CN,ETF_CN/MIN
    - 2.5. 板块 | STOCK_CN/ BLOCK

具体可以参见jupyter notebook

1. 从网上获取 | FROM WEBSITE

QA.QA*fetch_get* 系列:

从网上获取数据

1.1. 股票/日线 | STOCK_CN/DAY

```
QA.QA_util_log_info('日线数据')
QA.QA_util_log_info('不复权')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_day('00001','2017-01-01'
,'2017-01-31')

QA.QA_util_log_info('前复权')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_day('00001','2017-01-01'
,'2017-01-31','01')

QA.QA_util_log_info('后复权')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_day('00001','2017-01-01'
,'2017-01-31','02')

QA.QA_util_log_info('定点前复权')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_day('00001','2017-01-01'
,'2017-01-31','03')


QA.QA_util_log_info('定点后复权')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_day('00001','2017-01-01'
,'2017-01-31','04')
```

1.2. 股票/分钟线 | STOCK_CN/MIN

```
QA.QA_util_log_info('分钟线')
QA.QA_util_log_info('1min')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_min('000001','2017-07-0
1','2017-08-01','1min')

QA.QA_util_log_info('5min')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_min('000001','2017-07-0
1','2017-08-01','5min')

QA.QA_util_log_info('15min')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_min('000001','2017-07-0
1','2017-08-01','15min')

QA.QA_util_log_info('30min')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_min('000001','2017-07-0
1','2017-08-01','30min')

QA.QA_util_log_info('60min')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_min('000001','2017-07-0
1','2017-08-01','60min')
```

1.3. 股票/权息数据 | STOCK_CN/XDXR

```
QA.QA_util_log_info('除权除息')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_xdxr('00001')
```

1.4. 股票/列表 | STOCK_CN/LIST

```
QA.QA_util_log_info('股票列表')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_list('stock')
```

1.5. 指数/列表 | IDNEX_CN/LIST

```
QA.QA_util_log_info('指数列表')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_list('index')
```

1.6. 指数/日线 | INDEX_CN/DAY

```
QA.QA_util_log_info('指数日线')
data=QA.QAFetch.QATdx.QA_fetch_get_index_day('000001','2017-01-0
1','2017-09-01')
```

1.7. 指数/分钟线 | INDEX_CN/MIN

```
QA.QA_util_log_info('指数分钟线')
QA.QA_util_log_info('1min')
data=QA.QAFetch.QATdx.QA_fetch_get_index_min('000001','2017-07-0
1','2017-08-01','1min')


QA.QA_util_log_info('5min')
data=QA.QAFetch.QATdx.QA_fetch_get_index_min('000001','2017-07-0
1','2017-08-01','5min')


QA.QA_util_log_info('15min')
data=QA.QAFetch.QATdx.QA_fetch_get_index_min('000001','2017-07-0
1','2017-08-01','15min')
```

1.8. 最新交易价格STOCK | LAST PRICE

```
QA.QA_util_log_info('最后一次交易价格')
QA.QA_util_log_info('参数为列表')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_latest(['000001','000002'])


QA.QA_util_log_info('参数为一只股票')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_latest('000001')
```

1.9. 实时上下五档 STOCK_CN/QUOTATION

```
QA.QA_util_log_info('实时价格')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_realtime(['000001','000002'])
```

1.10. 分笔数据 | STOCK_CN/TRANSACTION

```
QA.QA_util_log_info('分笔成交')

历史分笔
data=QA.QAFetch.QATdx.QA_fetch_get_stock_transaction('000001','2001-01-01','2001-01-15')
实时分笔(当天)
data=QA.QAFetch.QATdx.QA_fetch_get_stock_transaction_realtime('000001')
```

1.11. 版块数据 | STOCK_CN/BLOCK

```
QA.QA_util_log_info('板块数据')
data=QA.QAFetch.QATdx.QA_fetch_get_stock_block()
```

2. 从数据库获取 | FROM LOCALHOST

QA.QA*fetch* 系列

从本地数据库获取数据

2.1. 股票日线 | STOCK_CN/DAY

```
QA_fetch_stock_day_adv(code,start,end)
```

2.2. 股票分钟线 | STOCK_CN/MIN

```
QA_fetch_stock_min_adv(code,start,end,frequence='1min') # freque
nce可以选1min/5min/15min/30min/60min
```

2.3. 指数/基金日线 | INDEX_CN,ETF_CN/DAY

```
QA_fetch_index_day_adv(code,start,end)
```

2.4. 指数/基金分钟线 | INDEX_CN,ETF_CN/MIN

```
QA_fetch_index_min_adv(code,start,end,frequence='1min') # freque
nce可以选1min/5min/15min/30min/60min
```

2.5. 板块 | STOCK_CN/ BLOCK

```
QA_fetch_stock_block_adv(code)
```

QUANTAXIS ANALYSIS 行情分析/研究

主要是针对行情的各种统计学特征/指标等分析,支持QA*DataStruct*系列的add_func()功能

接收DataFrame形式的行情以及QUANTAXIS.QADATA格式的行情

目前有:

(属性)

- 一阶差分
- 样本方差
- 方差
- 标准差
- 样本标准差
- 平均数
- 调和平均数
- 众数
- 振幅(极差)
- 偏度
- 峰度
- 百分比变化
- 平均绝对偏差

```
import QUANTAXIS as QA

data=QA.QA_fetch_stock_day_adv('600066','2013-12-01','2017-10-01'
) #[可选to_qfq(),to_hfq()]
s=QA.QAAnalysis_stock(data)
# s 的属性是( < QAAnalysis_Stock > )

s.open # 开盘价序列
s.close # 收盘价序列
s.high # 最高价序列
s.low # 最低价序列
s.vol  # 量
s.volume # 同vol
s.date  # 日期
s.datetime
s.index  # 索引
s.price  # 平均价(O+H+L+C)/4
s.mean # price的平均数
s.max  # price的最大值
s.min # price的最小值
s.mad # price的平均绝对偏差
s.mode  # price的众数(没啥用)
s.price_diff # price的一阶差分
s.variance # price的方差
s.pvariance # price的样本方差
s.stdev  # price的标准差
s.pstdev # price的样本标准差
s.mean_harmonic # price的调和平均数
s.amplitude  #price的振幅[极差]
s.skewnewss # price的峰度 (4阶中心距)
s.kurtosis  # price的偏度 (3阶中心距)
s.pct_change # price的百分比变化序列


s.add_func(QA.QA_indicator_CCI) # 指标计算, 和DataStruct用法一致
```

账户/组合/策略的说明 QAARP模块

@yutiansut

2018/1/26

在1.0版本以后,回测的策略是以继承账户类来进行的

账户/组合/策略的关系

```
{
  "User A":{
    "PortfolioA1":{
      "Account A" : "Strategy 1",
      "Account B" : "Strategy 2"
  },"Portfolio A2":{
      "Account C" : "Strategy 3"
  },
  "User B":{
    "Portfolio B1":{
      "Account D" : "Strategy 4"
    }
  }
}
```

```
# 在这里我们展示如何创建一个组合/创建账户/增加已有账户
import QUANTAXIS as QA
```

```
# 创建用户
userA=QA.QA_User()

# 创建两个组合 A1,A2
PortfolioA1=userA.new_portfolio()
PortfolioA2=userA.new_portfolio()

# A1里面增加两个策略(新建)
strategy1=PortfolioA1.new_account()
strategy2=PortfolioA1.new_account()

# 打印user的组合
In []: userA.portfolio_list
Out[]:
{'Portfolio_WpsaoQY6': < QA_Portfolio Portfolio_WpsaoQY6 with 0
Accounts >,
 'Portfolio_w5uJvtf7': < QA_Portfolio Portfolio_w5uJvtf7 with 2
Accounts >}


# 打印组合A1
In []: PortfolioA1
Out[]:  < QA_Portfolio Portfolio_w5uJvtf7 with 2 Accounts >


# 打印策略 本质是Account类
In []: strategy1
Out[]: < QA_Account Acc_qHL6dSC4>


# 创建一个策略 自定义on_bar事件
class Strategy3(QA.QA_Strategy):
  def __init__(self):
    super().__init__()

  def on_bar(self,event):
    print(event)

# 实例化该策略到strategy3
In []: strategy3=Strategy3()
```

```
# 打印strategy3
In []: strategy3
Out[]: < QA_Account Acc_AZPD18vS>

# 把该策略加载到A2组合中
In []: PortfolioA2.add_account(strategy3)

# 打印A2
In []: PortfolioA2
Out[]: < QA_Portfolio Portfolio_w7sx5Q31 with 1 Accounts >

# 打印组合列表

In []: userA.portfolio_list
Out[]:
{'Portfolio_WpsaoQY6': < QA_Portfolio Portfolio_WpsaoQY6 with 1
Accounts >,
 'Portfolio_w5uJvtf7': < QA_Portfolio Portfolio_w5uJvtf7 with 2
Accounts >}
```

创建自定义的策略

```
from QUANTAXIS.QAARP.QAStrategy import QA_Strategy
from QUANTAXIS.QAUtil.QAParameter import (AMOUNT_MODEL, MARKET_T
YPE,
                                          FREQUENCE, ORDER_DIREC
TION,
                                          ORDER_MODEL)
from QUANTAXIS.QAUtil.QALogs import QA_util_log_info

class MAStrategy(QA_Strategy):
    def __init__(self):
        super().__init__()
        self.frequence = FREQUENCE.DAY
        self.market_type = MARKET_TYPE.STOCK_CN

    def on_bar(self, event):
```

```
        sellavailable=self.sell_available
        try:
            for item in event.market_data.code:
                if sellavailable is None:

                    event.send_order(account_id=self.account_coo
kie,
                                     amount=100, amount_model=AM
OUNT_MODEL.BY_AMOUNT,
                                     time=self.current_time, cod
e=item, price=0,
                                     order_model=ORDER_MODEL.MAR
KET, towards=ORDER_DIRECTION.BUY,
                                     market_type=self.market_typ
e, frequence=self.frequence,
                                     broker_name=self.broker)

                else:
                    if sellavailable.get(item, 0) > 0:
                        event.send_order(account_id=self.account
_cookie,
                                         amount=sellavailable[it
em], amount_model=AMOUNT_MODEL.BY_AMOUNT,
                                         time=self.current_time,
 code=item, price=0,
                                         order_model=ORDER_MODEL
.MARKET, towards=ORDER_DIRECTION.SELL,
                                         market_type=self.market
_type, frequence=self.frequence,
                                         broker_name=self.broker
                                         )
                    else:
                        event.send_order(account_id=self.account
_cookie,
                                         amount=100, amount_mode
l=AMOUNT_MODEL.BY_AMOUNT,
                                         time=self.current_time,
 code=item, price=0,
                                         order_model=ORDER_MODEL
.MARKET, towards=ORDER_DIRECTION.BUY,
```

```
                                                market_type=self.market
_type, frequence=self.frequence,
                                                broker_name=self.broker
)

        except:
            pass
```

深入了解策略的组成

QA_Strategy 类完全继承 QA_Account, 因此,策略可以完全调用account类中的属性

```
self.history # dict形式 账户的历史交易
self.history_table #pd.Dataframe 形式 账户的交易表
self.cash # list格式 账户的现金表
self.cash_table #pd.Dataframe 形式 账户的现金流量表
self.hold #账户的最新持仓
self.latest_cash #账户的最近一次成功交易后的现金
self.trade # 账户的每日/分钟交易表
self.daily_cash # 账户的每日结算时的现金
self.daily_hold # 账户每日结算的持仓
self.current_time # 账户的当前时间

# 账户的on_bar事件
self.on_bar(self,event)

event 事件封装了数据和方法*(包括 所需的行情数据/下单接口)
```

风险分析模块

QA_Risk 是一个风险计算模块

```
R=QA.QA_Risk(ACCOUNT,benchmark_code='000300',benchmark_type=MARK
ET_TYPE.INDEX_CN)

#< QA_RISK ANALYSIS ACCOUNT-Acc_50wle3cY >

R()
# R() 是一个dataframe形式的表达结果
    account_cookie    annualize_return    max_dropback    portfo
lio_cookie    profit    time_gap    user_cookie        volatilit
y
0    Acc_50wle3cY    -0.000458            0.00012         Portfo
lio_oAkrKvj9    -0.000011    6        USER_l1CeBXog    64.696986

R.message

{'account_cookie': 'Acc_50wle3cY',
 'annualize_return': -0.0004582372482384578,
 'max_dropback': 0.00012000168002352033,
 'portfolio_cookie': 'Portfolio_oAkrKvj9',
 'profit': -1.1000154002127616e-05,
 'time_gap': 6,
 'user_cookie': 'USER_l1CeBXog',
 'volatility': 64.69698601944299}
```

组合视角 PORTFOLIO VIEW

QA_PortfolioView 是一个组合视角,只要输入account列表,就可以生成一个视角

```
accounts=[QA_Account1,QA_Account2,....]
PV=QA.QA_PortfolioView(accounts)
```

PV可以直接被加载到QA_Risk模块中

```
risk_pv=QA.QA_Risk(PV)

#calc result

risk.message
```

成交记录分析器和场景驱动说明:

考虑一个策略的产生和优化的场景,往往对于优化的行为难以做出有效的评估.

对于一个优化动作而言,往往其行为应该表现在成交记录中,对此我们做出如下的分析假设:

A. 优化开仓(选股/择时)的行为:

1. 开仓后是否导致过滤掉劣质股票(波动率高,收益率低)
2. 持有期内股票的回撤是否下降
3. 持有期内连续上涨的股票是否被错误过滤
4. 如果是正确的股票 是否被买在了半山腰(买入前前3日和前10日涨幅是否过大)
5. 是否是在行情震荡期买入(买入前3日和前10日的收益率方差)
6. 买入是在成交量波动放大的情况下还是在成交量波动减少的情况下买入的

通常, 一个较好的开仓优化行为有如下的特点:

对于选股

1. 不能错误的过滤较好表现的股票

2. 有效的减少了失误买入的情况(及持有期的最大回撤下降,最大回撤天数下降,持有期平均利润上升,持有期连续均涨幅上升)

对于择时

1. 不能在上涨尾期买入(前10日涨跌幅和前3日涨跌幅不易过大)
2. 不能在较为震荡和不明确行情下买入(前10日和前3日收益率方差不易过大)
3. 根据策略的不同 判断是否较好的反应了策略的需求(如在成交量萎缩的情况下买入)

B. 优化持仓期行为& 优化卖出时机:

1. 持仓期长度是否有预期的改变(如持仓时间过短)
2. 持有期利润是否上升
3. 持有期的最大连续涨幅是否抓住(最大连续涨幅是否上升)
4. 持有期的最大跌幅是否下降
5. 持有期的连续亏损时间是否下降
6. 卖出后是否出现较大的涨幅(卖早了)
7. 卖出后是否行情均方差上升了(卖出后的行情震荡)
8. 卖出后是否有巨额放量/巨额缩量的异常行为
9. 在每一日的组合内股票的盈亏数比例是否上升

通常, 一个较好的持仓/卖出优化行为有如下的特点:

对于选股:

持仓期内无法选股,但是可以做T调仓(暂时不支持分析)

对于择时:

1. 一个较好的持仓/卖出优化 应该具有提升持仓期平均利润的行为

2. 一个较好的持仓/卖出优化应该能抓住尽可能多的利润,减少卖出后的涨幅
3. 一个较好的持仓/卖出优化应该能减少巨额亏损的次数
4. 一个较好的持仓/卖出优化在总体上应该表现为每一日的组合内股票的盈亏数量比例上升
5. 一个较好的持仓卖出优化应该能较好的表达策略意图(如提升持仓时间等等)

C. 总体评价:

一个较好的优化行为应该有如下特点:

1. 表达优化的意图
2. 组合的alpha,beta,shape比率上升,最大回撤减少,胜率提高
3. 组合的每日股票盈亏数量比例提升

后期扩展用途:

1.实盘状态管理和风险判断

对于某一类型的策略,有一个策略的表现平均值,用以作为实盘的运行判断依据:

如果实盘的表现和回测的表现接近(买入前指标,卖出后指标,持仓期的平均指标),则实盘的预期和回测应较为相近,判断为风险正常期,如果在某一些指标上出现了较大的偏差,如卖出后的涨跌幅变大,说明出现了一些回测时未考虑到的条件和情况,需要进行重新的回测分析,判断为风险异常.

2.作为标准化的指标,进行机器学习的判断和分析

大量的指标可以作为机器学习的维度,进行一些非线性的数据采集,判断这一类型的策略的总体表现

指标	用途	
持有期长度	B1	
持有期利润	B2,A3	

持有期每日收益方差	A1,B2	
持有期最大连续涨跌幅	A3,B3	
持有期最大连续上涨天数占比	A3,B3	
持有期最大连续回撤	A1,B4	
持有期最大连续回撤天数占比	A1,B4	
持有期行情的成交量方差	A1,B1	
持有期行情的行情方向改变次数	A1,B1	
买入前3日涨跌幅	A4	
买入前3日成交量方差	A5	
买入前3日行情收益方差	A5	
买入前10日涨跌幅	A4	
买入前10日成交量方差	A5	
买入前10日收益方差	A5	
卖出后3日涨跌幅	B6	
卖出后3日成交量方差	B7,B8	
卖出后3日行情收益方差	B7	
卖出后10日涨跌幅	B6	
卖出后10日成交量方差	B7,B8	
卖出后10日收益方差	B7	
总体情况分析	对应情况	
alpha	C2	
beta	C2	
shape	C2	
最大回撤	C2	
持续期	C2	
年化收益	C2	
波动率	C2	
胜率	C2	
收益期望值	C2	

每日组合内股票的盈利数/组合数	C3,B9	

QUANTAXIS 的后台api

quantaxis 采用前后端分离的模式开发,所以对于后端而言 是一个可以快速替换/语言随意的部分.只需要按照规则设置好REST的url即可

1. 后端的标准和格式规范

1.1. 基础标准

quantaxis的后台可以用 nodejs(express/koa), python(flask/django/tornado), go 等等语言实现

quantaxis的后台部分主要是作为微服务,给前端(web/client/app)等提供标准化的查询/交互接口

2. 命名格式

quantaxis的后台命名格式

http://ip:port/功能(backtest/marketdata/user/..)/细分功能(info/query_code/..)

example:

```
http://localhost:3000/backtest/info_cookie?cookie=xxxxx  ==>  按backtest的cookie查询backtest的信息
```

3. 后端的实现方式和注意事项

3.1. 跨域支持

因为是前后端分离的模式, 需要对于url采取跨域允许

跨域在python中的实现

3.1.1. Flask

```
@app.route("/status")
def status():
    rst = make_response(jsonify('200'))
    rst.headers['Access-Control-Allow-Origin'] = '*'
    rst.headers['Access-Control-Allow-Methods'] = 'PUT,GET,POST,
DELETE'
    allow_headers = "Referer,Accept,Origin,User-Agent"
    rst.headers['Access-Control-Allow-Headers'] = allow_headers
    return rst
```

3.1.2. Tornado

```
class BaseHandler(tornado.web.RequestHandler):

    def set_default_headers(self):
        print ("setting headers!!!")
        self.set_header("Access-Control-Allow-Origin", "*") # 这
个地方可以写域名
        self.set_header("Access-Control-Allow-Headers", "x-reque
sted-with")
        self.set_header('Access-Control-Allow-Methods', 'POST, G
ET, OPTIONS')

    def post(self):
        self.write('some post')

    def get(self):
        self.write('some get')

    def options(self):
        # no body
        self.set_status(204)
        self.finish()
```

跨域在nodejs中的实现

3.1.3. express

```
router.get('*', function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "X-Requested-With")
;
  res.header("Access-Control-Allow-Methods", "PUT,POST,GET,DELET
E,OPTIONS");
  res.header("X-Powered-By", ' 3.2.1')
  res.header("Content-Type", "application/json;charset=utf-8");
  next();
});
```

3.2. 权限

后台服务需要保护好隐私不被泄露,避免路径攻击和端口暴露等问题

4. 必须实现的部分

4.1. 用户管理 /user

4.1.1. 登陆

```
http://[ip]:[port]/users/login?name=[]&password=[]
```

4.1.2. 注册

```
http://[ip]:[port]/users/signup?name=[]&password=[]
```

4.2. 回测部分 /backtest

4.2.1. 回测概览(列表查询)

```
http://[ip]:[port]/backtest/list?user=[]
```

4.2.2. 单个回测结果查询()

```
http://[ip]:[port]/backtest/info?cookie=[]
```

4.3. 行情查询部分 /marketdata & /data

功能性的API,分别代表着 日线/分钟线/实时(5档)/分笔数据

4.3.1. URI总规则 GENERAL URI RULE

```
总URI为 http://[ip]:[port]/[market_type]/[frequence]?code=[]&star
t=[]&end=[]
```

4.3.2. 股票日线 STOCK DAY

```
http://[ip]:[port]/marketdata/stock/day?code=[]&start=[]&end=[]
```

当不给定结束日期的时候,返回的就是直到当前的数据

4.3.3. 股票分钟线 STOCK MINDATA

```
http://[ip]:[port]/marketdata/stock/min?code=[]&start=[]&end=[]
```

当不给定结束日期的时候,返回的就是直到当前的数据

4.3.4. 股票实时上下五档 STOCK REALTIME 5-ASK/BID

```
http://[ip]:[port]/marketdata/stock/realtime?code=[]
```

实时返回股票的L1上下五档的行情数据

4.3.5. 股票分笔数据 STOCK TRANSACTION

```
http://[ip]:[port]/marketdata/stock/transaction?code=[]&start=[]
&end=[]
```

code 指的是具体的股票代码 start 指的是分笔开始的时间 end 指的是分笔结束的时间

4.3.6. 股票财务数据

```
http://[ip]:[port]/marketdata/stock/info?code=[]&time=[]
```

code 指的是具体的股票 time 指的是时间段

如 code=000001 time=2018Q1 指的是000001的2018年1季度

time的格式为: YEAR['YYYY']+Q+times1,2,3,4

4.3.7. 期货日线

```
http://[ip]:[port]/marketdata/future/day?code=[]&start=[]&end=[]
```

4.3.8. 期货分钟线

```
http://[ip]:[port]/marketdata/future/min?code=[]&start=[]&end=[]
```

4.4. 实时行情推送 /quotation

```
/quotation 推送指的是 建立一个websocket链接:
```

1. user login [Handler]
2. auth []
3. send_req [front end/client]
4. make connection

5. data transport

QUANTAXIS-Strategy

CTA Strategy Framework Review | 策略框架综述

趋势策略

日内趋势策略(tick级别的回测)

日内交易策略要求所有开仓头寸都必须在日内交易时段结束前平仓出局,这种策略下资金暴露在风险中的时间最短,能获得稳定的利润收益,但也要求所选择的投资品种在日内有着较大的波动和成交量,这种策略更多应用于豆粕,螺纹钢,橡胶等商品

菲阿里四价策略

菲阿里四价策略是一种通道突破方法,用过计算四个核心价格作为两条通道线的支撑,前一交易日最低价,前一交易日最高价,前一交易日收盘价和当前开盘价.

参数设置

上轨:上一交易日最高价 下轨:上一交易日最低价

交易逻辑

当价格突破上轨时,买入开仓,当价格突破下轨时,卖出开仓

这种交易策略会遇到一定程度的假突破问题，由于多空的博弈，在突破位会出现很大的阻力，有一定程度的随机回落的可能。需要进行一定的止盈止损策略进行限制

横盘突破策略

横盘是指价格波动幅度较小,没有明显的上涨或者下跌趋势,这时候市场的多空力量大致均衡.当出现横盘突破,及多空力量被打破的时候,表示一方动能较强,后市价格继续朝突破方向运动的趋势性更强.

当价格在过去30根K线的高低点围绕中轴上下0.5%的范围内波动时: 上轨=过去30根K线的最高价 下轨=过去30根K线的最低价

如果价格突破上下轨则表示当前价格波动较大,形成一个入场信号,也就是说: 当价格突破上轨时,买入开仓 当价格突破下轨时,卖出开仓

唐奇安通道策略

唐奇安的主要思想是寻找一定时间(X)内出现的最高价和最低价 上轨:过去X天内的最高点 下轨:过去X天内的最低嗲 如果价格突破上下轨则表示价格运动较为强势,释放入场信号 当价格突破上轨时,买入开仓 当价格突破下轨时,卖出开仓

R-Breaker策略

R-B是一个结合了日内趋势追踪和反转的策略,核心是通过前一交易日的最高.最低,收盘价计算出6个重要的价格指标.及突破买入价,观察卖出价,反转卖出价,反转买入价,观察买入价,突破卖出价.

参数设置

观察卖出价=最高价+0.35*(收盘价-最低价)

观察买入价=最低价-0.35*(最高价-收盘价)

反转卖出价=1.07/2*(*最高价+最低价*)-0.07*最低价

反转买入价=1.07/2*(*最高价+最低价*)-0.07*最高价

突破买入价=观察卖出价+0.25*(观察卖出价-观察买入价)

突破卖出价=观察买入价-0.25*(观察卖出价-观察买入价)

交易逻辑

1. 空仓情况下,盘中价格跌破突破卖出价,采取趋势策略,即在该点做空.
2. 空仓情况下,盘中价格超过突破买入价,采取趋势策略,即在该点做多.
3. 持多单时,当日最高价超过观察卖出价后,盘中价格回落并且跌破发转卖出价,采取反转策略,即在该点反手做空
4. 持空单时,当日最低价低于观察买入价后,盘中价格反弹并且超过反转买入价,采取反转策略,即在该点反手做多.
5. 设定止盈止损条件
6. 收盘前进行平仓

Dual Thrust策略

在开盘价的基础上,确定一个非对称的上下轨道，利用前N日的最高，最低和收盘价格去确定一个合理的震荡区间，将上下轨到开盘价的距离设置为震荡区间的一定比例，一旦当前价格突破了这个比例，则产生入场或离场信号。

参数设置

记前N日的最高价的最大值为HH,前N日的最低价的最小值为LL

记前N日的收盘价的最大值为HC,前N日的收盘价的最小值为LC

震荡区间为Range

Range = MAX{HH-LC,HC-LL};

BuyLine = Open + K_s×Range;

SellLine = Open - K_x*Range;

交易逻辑

当价格突破上轨BuyLine时：买多开仓|平空反向做多

当价格突破下轨SellLine时,买空开仓|平多反向做空

Dual Thrust 策略中震荡区间的选取为前 N 日的最值,使得其在一定时期内保持相对稳定,参数K_s 和K_x 用来调节多头和空头触发条件难易。当K_s 较大时,空头较容易被触发;当K_x 较大时,多头较容易被触发。参数的调节可以通过参考数据测试得到的最优参数并结合主观分析得到。

ATR策略

Average True Range 真实波动范围指标，用于衡量市场波动的程度，是显示市场变化率的指标。目前，主要用于衡量价格的波动，并不能直接反应价格走向以及趋势的稳定性，仅仅表明价格波动的程度。

参数设置

- X1: 当前交易日的价差 PriceMax-PirceMin
- X2: 前一个交易日收盘价与当前交易日的最高价的波幅
- X3: 前一交易日的收盘价与当前交易日的最低价的波幅

  真实波幅TR=MAX{X1,X2,X3}

  TR=MAX{Max(High-Low),Abs(Ref(Close,1)-High),Abs(Ref(Close,1)-Low)}

  ATR 是TR的N日移动平均 AR=MA(TR)

  一般而言，N=14

交易逻辑

- 日内平仓
- 日内ATR突破基于当
- 价格突破上轨，买入开仓

- 价格跌穿下轨，卖出开仓

King-keltner策略

参数设置

- 中心价=MA((H+L+C)/3,40) 中心价是最高价，最低价，收盘价三者平均后的40日移动平均线
- 计算真实价格区间（TrueRange），等于本日最高价-本日最低价，本日最高价-昨日收盘价，本日最低价-昨日收盘价的三者的最大值

  TR=MAX{Max(High-Low),Abs(Ref(Close,1)-High),Abs(Ref(Close,1)-Low)}
- 计算价格上下轨(upBand,dnBand),其中multi为固定参数，可设置初始值为1

  上轨=中心价+multi× MA(TR,40)

  下轨=中心价-multi× MA(TR,40)
- 平仓价格=中心价

交易逻辑

- 买入开仓：今日中心价大于昨日中心价，且价格突破上轨
- 卖出开仓：今日中心价小于昨日中心价，且价格突破下轨
- 买入平仓：今日价格向下突破平仓价格
- 卖出平仓：今日价格向上突破平仓价格

其中，平仓条件既是止损条件，也是止盈条件（移动止盈）

该策略的核心是对于multi参数和MA的天数的优化问题，风险在于，震荡时期的频繁交易会因为止损过多而导致较大回撤

实证检验部分

不同商品期货的单边回测结果

组合策略的相关性与回测表现

日间交易策略

均线策略

均线策略的内在思想是短期和长期趋势不同的涨跌变化对交易有不同的指导意义。均线策略中的均线体现了价格的变化,常用的均线有简单移动平均、加权移动平均、指数移动平均

MACD策略

布林带通道策略

布林策略Bollinger Bandit

参数设置

- 计算布林带的上下边界，等于收盘价的50日移动平均价加减1.25倍标准差
    - 上边界=MA(收盘价,50)+StdDev(收盘价,50)×1.25
    - 下边界=MA(收盘价,50)-StdDev(收盘价,50)×1.25
- 计算计步器，等于当日收盘价-30日前收盘价
    - 如果今日未平仓，则计算天数-1，直到递减到10
    - 如果今日平仓，将计算天数还原到50
- 计算止损价，等于收盘价的移动平均，计算天数初始值为50

  止损价=MA(收盘价，计算天数)

交易逻辑

- 买入开仓：计步器大于0，当前价格大于上边界
- 卖出开仓：计步器小于0，且当前价格小于下边界
- 卖出平仓：当止损价大于上边界且当前价格小于止损价
- 买入平仓：当止损价小于下边界且当前价格大于止损价

套利策略

我们并没有使用传统的跨期套利，跨品种套利和跨市场套利去对套利算法进行区分，而是使用关联套利和内因套利去区分这几种套利方法。

关联套利

关联套利是指套利对象之间没有必然的内因约束，但价格受共同因素所主导，但受影响的程度不同，通过两种对象对同一影响因素表现不同而建立的套利关系称之为关联套利

关联套利：套利对象之间基差的大小不对买卖力量产生负反馈作用,多数的情况下形成一个发散性的蛛网.

盈利原理：通过追逐比价(或差价)趋势的办法，将错误操作的损失通过止损控制在一定的限度之内，将正确操作的利润尽量放大,从而达到“赚多赔少”的总体目标

关联套利理论基础:

1.供需关系决定了商品的价格趋势;供需关系紧张的程度决定了价格趋势的强度.

2.价格趋势在没有受到新的力量作用时,会保持原来的方向.（牛顿第一定律）

3.基本面不会在一天之内改变.(如农产品供应的年周期性和消费的广泛性;对于工业品而言,也存在一个较长的生产周期)

农产品跨品种套利

[替代性品种，并无内因关系，只是供求关系]

大豆与玉米、玉米与小麦之间、不同油脂类品种之间

基本金属跨品种套利

铜、铝、锌之间的套利

金融衍生品跨市跨品种套利

不同国家的股票指数套利

内因套利

内因套利是指当商品期货投资对象间价格关系因某种原因过分背离时，通过内在纠正力量而产生的套利行为。

内因套利：套利对象之间基差的大小负反馈于买卖力量,形成一个收敛性的蛛网.

盈利原理：谨慎选择有限波动差价(或比价)两端的极端机会，提高胜算率来保证交易的成功，即使出现意外，也可以能够通过现货处理、向后延期等办法来抑制亏损

1.存在基差收缩的机制(内因)

2.市场是有效的(特别是期货市场)

核心要点：

1 寻找导致目前价格关系过分背离的原因。

2 分析未来能够纠正价格关系恢复的内在因素。

步骤：

第一步:选择经过有效性检验的、并且有内因约束的套利对象，确定套利追踪目标。

比如:大豆期现套利;大豆,豆粕,豆油的远近合约套利;大豆与豆粕,豆油三者的压榨套利;连豆和CBOT黄大豆之间跨市套利等等.

第二步:建立上述套利对象的历史比价（差价）数据库，并每日更新。

第三步:将当前比价（差价）分别导入各套利对象的比价（差价）区间，用数理的方法鉴别出当前比价（差价）在区间中所处位置，并计算该比价（差价）在历史上所出现的概率。

第四步:通过数理分择,判定基差偏离程度和套利机会的大小.

第五步:内因佐证分析. 建立各影响因子的数据资料库,通过多因素分析方法来分析寻找导致目前价格关系过分背离的原因,分析未来能够纠正价格关系恢复的内在因素。

- 进口成本
- 现货走势图
- 运费变化图

- 升贴水变化图
- 仓单变化
- 压榨利润
- 政策因素
- 需求方面
- 供给方面
- 经济周期
- 政治因素
- 自然因素
- 金融因素

第六步:按照内因套利的五大原则,对套利外部环境进行评估，再次鉴别市场的有效性以及头寸的可持续性。

第七步:进入资金管理和风险控制的实际操作阶段。

期现套利

期现套利（Arbitrage）是利用同一种商品在期货市场与现货市场之间的不合理的价差进行的套利行为。

两个投资组合，若其未来现金流完全相同，则现值一定相等，否则将出现套利机会：买入现值较低的投资组合，同时卖出较高的投资组合，并持有到期，必定可获得无风险利润。

同一种商品的期货合约价格与其现货价格之间存在着无套利机会的定价关系，这种关系通常称为持有成本定价。所谓持有成本，是指商品的储藏成本加上为资产融资所需支付的利息再扣掉持有资产带来的收入。

同一品种的跨期套利

单一农产品品种的跨期套利

同一种商品的不同交割月份的期货合约价格之间也存在着无套利机会的定价关系。当远期合约的价格超过无套利区间的上边界时，可以从事正向套利操作；而跌过无套利区间的下边界时，可以从事反向套利操作。

套利步骤：

1。通过计算无套利区间，建立套利机会每日跟踪系统。 相邻合约间跨期套利的持仓成本＝间隔期内的商品仓储费用＋交易交割手续费＋套利期内资金占压成本（贷款利息）＋增值税（（交割结算价－买入价格）×税率）

2。当满足套利条件时，开始做市场有效性检验，比如，具备不具备逼仓条件，市场容量，交易群体调查等等。

3。展期条件和展期收益评估。

4。风险评估和风险预警措施制定。

5。实施操作。

同一品种的跨市场套利

国内外大豆进口套利

> 套利公式：
>
> 进口利润=国内大豆价格-进口成本价
>
> 进口成本=（CBOT期价+海岸升水+海运费）×汇率+港杂费
>
> 进口大豆关税1%，增值税13%。
>
> 港杂费100元/吨左右。

单位换算：

1蒲式耳大豆=60磅=60×0.4536公斤=27.216公斤=0.027216吨

1美分/蒲式耳=0.01美元/0.027216吨=0.36743美元/吨；

产业链套利

黑色产业链

钢产业 **(**螺纹钢，铁矿，焦炭**)**

炼钢生产的成本主要是生铁，废钢，合金，电极，耐火材料，辅助材料，电能，维护检修和其他费用。中国目前主要的炼钢设备为转炉和电炉，基于冶炼原理的不同，转炉和电炉在主要的原料（生铁、废钢）配比有一定的差异，转炉工艺一般需

配置10%的废钢，而电炉工艺废钢的使用量则占到80%。

结合国内钢铁企业的平均情况，炼铁工艺中影响总成本的主要因素是原料（铁矿石、焦炭）成本，而包括辅料、燃料、人工费用在内的其他费用与副产品回收进行冲抵后仅占总成本的10%左右，而炼钢工艺中因为耗电量的增加、合金的加入以及维检费用的上升使得除主要原料外的其他费用占到炼钢总成本的18%左右。炼铁、炼钢工艺中的其他费用波动不大。

套利逻辑

> 钢成本/吨=1.6吨铁精粉+0.5吨焦炭+生铁费+钢胚费+轧材费+其他费用
>
> 螺纹钢期货价格=1.6×铁矿石期货价格+0.5×焦炭期货价格+其他成本

在实际的期货市场上，由于价格的波动，出现了无套利等式左右不等的情况，此时的价差及为钢厂的利润，这也是内因策略的利润核心。跟随钢厂利润，我们可以认为，当钢厂利润较高，钢厂就会进行产量的调整，提高开工率并且在原材料市场上进行大量购买，导致原材料市场的供小于求，铁矿石和焦炭价格上涨，导致钢厂的利润下降；反之，如果钢厂利润下降，钢厂会动态调整最优产量从而影响了上下游产业的价格，铁矿石和焦炭价格也会下降，同时在产品市场上，由于钢厂的供给减少，钢材的价格也会上升，恢复了钢厂的利润平衡。

期货品种	螺纹钢	铁矿石	焦炭
保证金	9%	10%	15%
合约乘数	10吨/手	100吨/手	100吨
最小变动单位	1元/吨	0.5元/吨	0.5元/吨

策略设置

- 开仓条件：价差在[10日均值+标准差，10日均值+1.2×标准差],有回归趋势
- 平仓条件：回归到十日均值进行平仓

- 止损： 5%，止损后10日内不开仓
- 换仓： 主力合约一般是1,5,9 在主力合约到来的前一个月进行换仓
- 滑点： 一个最小变动价位

炼焦产业 (焦煤，焦炭)

煤焦加工套利包括三种模式，分别为独立焦化企业模式、煤矿企业模式以及自有焦化厂的钢铁企业模式。煤焦加工套利的难点在于配煤炼焦工艺的确定、产成品质量及升贴水的确定、以及副产品构成及其价值的确定。一级冶炼焦的配煤比例是主焦煤占比 35%、1/3 焦煤占比 25%、气煤占比 12%、肥煤占比 18%、瘦煤占比 10%。企业为了降低炼焦的成本，一般是提高价格较低的瘦煤的比重而降低主焦煤比重，因此企业为了应对市场变化带来的成本压力会改变原料的配比，那么炼焦成本也是在变化的，配煤的比例是动态变化的，并不是固定的。

套利逻辑

参考一般的炼焦工艺，平均 1.3 吨炼焦煤加工产生 1 吨焦炭和若干副产品。自焦煤期货上市以来，期货焦炭指数/焦煤指数的比价均值却高达 1.37，同时较长时间维在 1.38 以上，指数最高比价曾达到 1.45。因此，我们最终确定炼焦利润的公式为： 炼焦利润=焦炭期货价格-1.4*焦煤期货价格-其他成本

上述等式的系数是参考基本面的逻辑来确定的，与钢厂利润套利类似，等式中固定的系数与实际生产的系数存在差异，实际生产的系数也难以把握，因此通过量化的手段来确定各品种之间的关系也是一种备选方案。

期货合约	焦煤	焦炭
保证金	15%	15%
合约乘数	60吨/手	100吨/手
最小变动单位	0.5元/吨	0.5元/吨

策略设置

开仓条件：价差在 10 日均值加 1 倍标准差和 1.2 倍标准差之间，且有回归趋势开仓。 平仓条件：回归到 10 日均值进行平仓。 止损：设置的止损为 2%，止损后 10 天内不开仓。 换仓：主力合约一般为 1,5,9 月份，因此我们在主力合约到期前一个多月进行主力合约的换仓。 滑点：一个最小变动价位

能源化工产业链

甲醇制PP利润套利 (甲醇，聚丙烯)

套利逻辑

甲醇既可自用又可外售的特征是甲醇制 PP 套利可行的重要原因。根据理论生产成本，3 吨甲醇另加 800 元加工费用可制得 1 吨聚丙烯。具体到期货价格的层面，在无套利机会的情况下可以得到如下等式： 聚丙烯期货价格=3*甲醇期货价格+800

期货品种	甲醇	聚丙烯**PP**
保证金	7%	7%
合约乘数	10吨/手	5吨
最小变动单位	1元/吨	1元/吨

策略设置

开仓条件：价差超越 5 日均值加 1 倍标准差进行开仓。 平仓条件：回归到 5 日均值进行平仓。 止损：设置的止损为 5%，止损后 10 天内不开仓。 换仓：主力合约一般为 1,5,9 月份，因此我们在主力合约到期前一个多月进行主力合约的换仓。 滑点：一个最小变动价位。

农产品期货产业链

大豆提油套利 (大豆，豆粕，豆油)

套利逻辑

大豆提油套利的模式主要是在美豆和国内豆粕豆油间展开，具体的路径是进口美豆后在国内压榨出售。因此要关注的点是美豆成本、豆粕价格和豆油价格。美豆的成本涉及到增值税、关税、汇率、港杂费和运费等因素。计算美豆到厂成本主要公式如下：

大船舱底完税价格=到岸价格（1+增值税）（1+关税）*美元兑人民币汇率

进口大豆到厂成本=大船舱底完税价格+港杂费+运费

进口大豆到厂成本即为美豆的成本。相对于煤焦钢产业链，大豆压榨的技术参数较为稳定，国内的大豆压榨工艺显示大豆压榨时油粕产出比例存在如下的关系：

100%进口大豆=19.2%豆油+78.6%豆粕+2.2.%损耗

因此，进口大豆压榨的利润公式可以根据美豆成本以及压榨的技术参数得出。

进口大豆压榨利润=豆粕价格78.6%+豆油价格19.2%-进口大豆到厂成本-其他费用

鸡蛋利润套利 (鸡蛋，豆粕，玉米)

套利逻辑

鸡蛋作为日常消费品，需求比较稳定。一般，每年在端午节（5 月）和中秋节（9 月）会出现两个需求小高峰，春节前后（2 月）会出现需求低谷。最近几年由于集约养殖比例增大以及鸡蛋期货上市，鸡蛋价格的季节性波动已经越来越不明显。鉴于鸡蛋需求稳定的特征，鸡蛋价格一般由产蛋成本决定。 蛋鸡养殖的主要成本包括鸡苗、饲料、水、电、人工和防疫等费用。需要注意的是，除了鸡蛋出售收入，还有鸡粪以及蛋鸡淘汰以后变卖的收入。一只蛋鸡从鸡苗到产蛋到退役，整个过程经

历 17 个月。蛋鸡在生命周期内产蛋 37.5 斤，消耗饲料约 122.7 斤，鸡苗成本约 3 元，防疫费支出 3 元，水电费支出 0.5 元，鸡粪获利 3.4 元，淘汰鸡出售约 16.4 元。 据此，我们测算出：

鸡蛋盈亏平衡点=(饲料费用+鸡苗成本+防疫费+水电费-鸡粪收入-淘汰鸡收入)/37.5

上式中鸡苗成本、防疫费、水电费、鸡粪收入和淘汰鸡收入变化不大，可以使用常数替代，主要的变量是饲料价格。蛋鸡饲料的典型配方是 62%的玉米、31%的豆粕和 7%的预混料(麦麸、磷酸氢钙、石粉、食盐、其他添加剂等)，预混料价格大约在 2.5 元/斤。那么，饲料价格的公式可以表示为：

一斤饲料的价格=0.62玉米价格 + *0.31*豆粕价格+ 0.175

依据鸡蛋的盈亏平衡点计算公式和饲料价格的公式，可以得出联系期货价格的等式。

鸡蛋盈亏平衡点(元/吨) = 2.02864玉米*(元/吨)* + *1.01432*豆粕(元/吨) + 437.2

鸡蛋的利润波动曲线回归周期仍然比较长,并且没有明显的周期性规律,总体来看,价差要经历差不多一年的时间才会恢复到 0 附近的水平。因此如果设定固定价差来进行套利,开仓次数将很有限,而且价差设置越大开仓次数越少。如果价差设置过小,则需要很长时间才会回归,并且会有很大的回撤。因此我们仍然采用之前的在均值上加减标准差的方式进行开仓,在价差恢复到均 值时进行平仓。

策略设置

开仓条件:价差超越 10 日均值加 2 倍标准差进行开仓。 平仓条件:回归到 10 日均值进行平仓。 止损:设置的止损为 5%,止损后 10 天内不开仓。 换仓:主力合约一般为 1,5,9 月份,因此我们在主力合约到期前一个多月进行主力合约的换仓。 滑点:一个最小变动价位。

风险控制

止损策略

技术指标

成本损失

趋势研判

策略组合和资金曲线

马科维茨的均值方差分析

策略相关性以及风险收益比

机器学习的应用

参数优化

群参数优化算法

PSO粒子群优化算法

通过群体中个体之间的协作和信息共享来寻找最优解·

鸟被抽象为没有质量和体积的微粒(点)，并延伸到N维空间，粒子i在N维空间的位置表示为矢量Xi＝(x1，x2，…，xN)，飞行速度表示为矢量Vi＝(v1，v2，…，vN)。每个粒子都有一个由目标函数决定的适应值(fitness value)，并且知道自己到目前为止发现的最好位置(pbest)和现在的位置Xi。这个可以看作是粒子自己的飞行经验。除此之外，每个粒子还知道到目前为止整个群体中所有粒子发现的最好位置(gbest)(gbest是pbest中的最好值)，这个可以看作是粒子同伴的经验。粒子就是通过自己

的经验和同伴中最好的经验来决定下一步的运动。 PSO初始化为一群随机粒子(随机解)。然后通过迭代找到最优解。在每一次的迭代中，粒子通过跟踪两个“极值”(pbest，gbest)来更新自己。在找到这两个最优值后，粒子通过下面的公式来更新自己的速度和位置。

$$ v*{i}=v*{i} + c*{1} rand()(pbest*{i} -x*{i}) +c*{2}rand()(gbest_{i}-x_i) (1)$$

$$x*{i}=x*{i}+v_{i} (2)$$

> 在公式（1），（2）中，i=1,2,3,...,N是此群中粒子的总数

$$ v_{i} 是粒子的速度$$

$$ rand() 介于（0,1）的随机数$$

$$ x_{i}粒子的当前位置$$

$$ c*{1},c*{2} 学习因子 一般设置为2$$

$$ v*{i}的最大值为 V*{max} $$

公式(1)的第一部分称为【记忆项】，表示上次速度大小和方向的影响；公式(1)的第二部分称为【自身认知项】，是从当前点指向粒子自身最好点的一个矢量，表示粒子的动作来源于自己经验的部分；公式(1)的第三部分称为【群体认知项】，是一个从当前点指向种群最好点的矢量，反映了粒子间的协同合作和知识共享。粒子就是通过自己的经验和同伴中最好的经验来决定下一步的运动。

以上面两个公式为基础，形成了PSO的标准形式。

$$v*{i}=\omega *v*{i}+c*{i}rand()(pbest*{i}-x*{i})+c_2×rand()*(gbest*{i}-x_{i}) (3)$$

$$ \omega 为惯性因子 值为非负$$

公式(2)和 公式(3)被视为标准PSO算法。

标准PSO算法流程

1）初始化一群微粒(群体规模为N)，包括随机位置和速度；

2）评价每个微粒的适应度；

3）对每个微粒，将其适应值与其经过的最好位置pbest作比较，如果较好，则将其作为当前的最好位置pbest；

4）对每个微粒，将其适应值与其经过的最好位置gbest作比较，如果较好，则将其作为当前的最好位置gbest；

5）根据公式(2)、(3)调整微粒速度和位置；

6）未达到结束条件则转第2）步

迭代终止条件根据具体问题一般选为最大迭代次数Gk或(和)微粒群迄今为止搜索到的最优位置满足预定最小适应阈值。

公式(2)和(3)中pbest和gbest分别表示微粒群的局部和全局最优位置。

当C1＝0时，则粒子没有了认知能力，变为只有社会的模型(social-only)：

被称为全局PSO算法。粒子有扩展搜索空间的能力，具有较快的收敛速度，但由于缺少局部搜索，对于复杂问题比标准PSO 更易陷入局部最优。

当C2＝0时，则粒子之间没有社会信息，模型变为只有认知(cognition-only)模型：

被称为局部PSO算法。由于个体之间没有信息的交流，整个群体相当于多个粒子进行盲目的随机搜索，收敛速度慢，因而得到最优解的可能性小。

参数分析 参数：群体规模N，惯性因子 ，学习因子c1和c2，最大速度Vmax，最大迭代次数Gk。

群体规模N：一般取20～40，对较难或特定类别的问题可以取到100～200。

最大速度Vmax：决定当前位置与最好位置之间的区域的分辨率(或精度)。如果太快，则粒子有可能越过极小点；如果太慢，则粒子不能在局部极小点之外进行足够的探索，会陷入到局部极值区域内。这种限制可以达到防止计算溢出、决定问题空

间搜索的粒度的目的。

权重因子：包括惯性因子和学习因子c1和c2。使粒子保持着运动惯性，使其具有扩展搜索空间的趋势，有能力探索新的区域。c1和c2代表将每个粒子推向pbest和gbest位置的统计加速项的权值。较低的值允许粒子在被拉回之前可以在目标区域外徘徊，较高的值导致粒子突然地冲向或越过目标区域。

参数设置：

1) 如果令c1＝c2＝0，粒子将一直以当前速度的飞行，直到边界。很难找到最优解。 2) 如果＝0，则速度只取决于当前位置和历史最好位置，速度本身没有记忆性。假设一个粒子处在全局最好位置，它将保持静止，其他粒子则飞向它的最好位置和全局最好位置的加权中心。粒子将收缩到当前全局最好位置。在加上第一部分后，粒子有扩展搜索空间的趋势，这也使得的作用表现为针对不同的搜索问题，调整算法的全局和局部搜索能力的平衡。较大时，具有较强的全局搜索能力；较小时，具有较强的局部搜索能力。

3) 通常设c1＝c2＝2。Suganthan的实验表明：c1和c2为常数时可以得到较好的解，但不一定必须等于2。Clerc引入收敛因子(constriction factor) K来保证收敛性。

通常取为4.1,则K＝0.729.实验表明，与使用惯性权重的PSO算法相比，使用收敛因子的PSO有更快的收敛速度。其实只要恰当的选取和c1、c2，两种算法是一样的。因此使用收敛因子的PSO可以看作使用惯性权重PSO的特例。

恰当的选取算法的参数值可以改善算法的性能。

```
class bird:
"""
speed:速度
position:位置
fit:适应度
lbestposition:经历的最佳位置
lbestfit:经历的最佳的适应度值
"""
def __init__(self, speed, position, fit, lBestPosition, lBestFit)
:
    self.speed = speed
    self.position = position
    self.fit = fit
    self.lBestFit = lBestPosition
    self.lBestPosition = lPestFit
```

```
import random

class PSO:
    """
    fitFunc:适应度函数
    birdNum:种群规模
    w:惯性权重
    c1,c2:个体学习因子，社会学习因子
    solutionSpace:解空间，列表类型：[最小值，最大值]
    """
    def __init__(self, fitFunc, birdNum, w, c1, c2, solutionSpac
e):
        self.fitFunc = fitFunc
        self.w = w
        self.c1 = c1
        self.c2 = c2
        self.birds, self.best = self.initbirds(birdNum, solution
Space)

    def initbirds(self, size, solutionSpace):
        birds = []
        for i in range(size):
```

```
            position = random.uniform(solutionSpace[0], solution
Space[1])
            speed = 0
            fit = self.fitFunc(position)
            birds.append(bird(speed, position, fit, position, fi
t))
        best = birds[0]
        for bird in birds:
            if bird.fit > best.fit:
                best = bird
        return birds,best

    def updateBirds(self):
        for bird in self.birds:
            # 更新速度
            bird.speed = self.w * bird.speed + self.c1 * random.
random() * (bird.lBestPosition - bird.position) + self.c2 * rand
om.random() * (self.best.position - bird.position)
            # 更新位置
            bird.position = bird.position + bird.speed
            # 跟新适应度
            bird.fit = self.fitFunc(bird.position)
            # 查看是否需要更新经验最优
            if bird.fit > bird.lBestFit:
                bird.lBestFit = bird.fit
                bird.lBestPosition = bird.position

    def solve(self, maxIter):
        # 只考虑了最大迭代次数，如需考虑阈值，添加判断语句就好
        for i in range(maxIter):
            # 更新粒子
            self.updateBirds()
            for bird in self.birds:
                # 查看是否需要更新全局最优
                if bird.fit > self.best.fit:
                    self.best = bird
```

GA遗传算法

GA遗传算法:

```
from __future__ import division
import numpy as np
import random
import math
import copy
import matplotlib as mpl
import matplotlib.pyplot as plt
import time


class GA(object):
    def __init__(self, maxiter, sizepop, lenchrom, pc, pm, dim,
lb, ub, Fobj):
        """
        maxiter：最大迭代次数
        sizepop：种群数量
        lenchrom：染色体长度
        pc：交叉概率
        pm：变异概率
        dim：变量的维度
        lb：最小取值
        ub：最大取值
        Fobj：价值函数
        """
        self.maxiter = maxiter
        self.sizepop = sizepop
        self.lenchrom = lenchrom
        self.pc = pc
        self.pm = pm
        self.dim = dim
        self.lb = lb
        self.ub = ub
        self.Fobj = Fobj

    # 初始化种群：返回一个三维数组，第一维是种子，第二维是变量维度，第三维是
编码基因
    def Initialization(self):
```

```
        pop = []
        for i in range(self.sizepop):
            temp1 = []
            for j in range(self.dim):
                temp2 = []
                for k in range(self.lenchrom):
                    temp2.append(random.randint(0, 1))
                temp1.append(temp2)
            pop.append(temp1)
        return pop

    # 将二进制转化为十进制
    def b2d(self, pop_binary):
        pop_decimal = []
        for i in range(len(pop_binary)):
            temp1 = []
            for j in range(self.dim):
                temp2 = 0
                for k in range(self.lenchrom):
                    temp2 += pop_binary[i][j][k] * math.pow(2, k
)
                temp2 = temp2 * (self.ub[j] - self.lb[j]) / (mat
h.pow(2, self.lenchrom) - 1) + self.lb[j]
                temp1.append(temp2)
            pop_decimal.append(temp1)
        return pop_decimal

    # 轮盘赌模型选择适应值较高的种子
    def Roulette(self, fitness, pop):
        # 适应值按照大小排序
        sorted_index = np.argsort(fitness)
        sorted_fitness, sorted_pop = [], []
        for index in sorted_index:
            sorted_fitness.append(fitness[index])
            sorted_pop.append(pop[index])

        # 生成适应值累加序列
        fitness_sum = sum(sorted_fitness)
        accumulation = [None for col in range(len(sorted_fitness
))]
```

```
        accumulation[0] = sorted_fitness[0] / fitness_sum
        for i in range(1, len(sorted_fitness)):
            accumulation[i] = accumulation[i - 1] + sorted_fitne
ss[i] / fitness_sum

        # 轮盘赌
        roulette_index = []
        for j in range(len(sorted_fitness)):
            p = random.random()
            for k in range(len(accumulation)):
                if accumulation[k] >= p:
                    roulette_index.append(k)
                    break
        temp1, temp2 = [], []
        for index in roulette_index:
            temp1.append(sorted_fitness[index])
            temp2.append(sorted_pop[index])
        newpop = [[x, y] for x, y in zip(temp1, temp2)]
        newpop.sort()
        newpop_fitness = [newpop[i][0] for i in range(len(sorted
_fitness))]
        newpop_pop = [newpop[i][1] for i in range(len(sorted_fit
ness))]
        return newpop_fitness, newpop_pop

    # 交叉繁殖：针对每一个种子，随机选取另一个种子与之交叉。
    # 随机取种子基因上的两个位置点，然后互换两点之间的部分
    def Crossover(self, pop):
        newpop = []
        for i in range(len(pop)):
            if random.random() < self.pc:
                # 选择另一个种子
                j = i
                while j == i:
                    j = random.randint(0, len(pop) - 1)
                cpoint1 = random.randint(1, self.lenchrom - 1)
                cpoint2 = cpoint1
                while cpoint2 == cpoint1:
                    cpoint2 = random.randint(1, self.lenchrom - 1
)
```

```
                cpoint1, cpoint2 = min(cpoint1, cpoint2), max(cp
oint1, cpoint2)
                newpop1, newpop2 = [], []
                for k in range(self.dim):
                    temp1, temp2 = [], []
                    temp1.extend(pop[i][k][0:cpoint1])
                    temp1.extend(pop[j][k][cpoint1:cpoint2])
                    temp1.extend(pop[i][k][cpoint2:])
                    temp2.extend(pop[j][k][0:cpoint1])
                    temp2.extend(pop[i][k][cpoint1:cpoint2])
                    temp2.extend(pop[j][k][cpoint2:])
                    newpop1.append(temp1)
                    newpop2.append(temp2)
                newpop.extend([newpop1, newpop2])
        return newpop

    # 变异：针对每一个种子的每一个维度，进行概率变异，变异基因为一位
    def Mutation(self, pop):
        newpop = copy.deepcopy(pop)
        for i in range(len(pop)):
            for j in range(self.dim):
                if random.random() < self.pm:
                    mpoint = random.randint(0, self.lenchrom - 1
)
                    newpop[i][j][mpoint] = 1 - newpop[i][j][mpoi
nt]
        return newpop

    # 绘制迭代-误差图
    def Ploterro(self, Convergence_curve):
        mpl.rcParams['font.sans-serif'] = ['Courier New']
        mpl.rcParams['axes.unicode_minus'] = False
        fig = plt.figure(figsize=(10, 6))
        x = [i for i in range(len(Convergence_curve))]
        plt.plot(x, Convergence_curve, 'r-', linewidth=1.5, mark
ersize=5)
        plt.xlabel(u'Iter', fontsize=18)
        plt.ylabel(u'Best score', fontsize=18)
        plt.xticks(fontsize=18)
        plt.yticks(fontsize=18)
```

```
        plt.xlim(0, )
        plt.grid(True)
        plt.show()

    def Run(self):
        pop = self.Initialization()
        errolist = []
        for Current_iter in range(self.maxiter):
            print("Iter = " + str(Current_iter))
            pop1 = self.Crossover(pop)
            pop2 = self.Mutation(pop1)
            pop3 = self.b2d(pop2)
            fitness = []
            for j in range(len(pop2)):
                fitness.append(self.Fobj(pop3[j]))
            sorted_fitness, sorted_pop = self.Roulette(fitness,
pop2)
            best_fitness = sorted_fitness[-1]
            best_pos = self.b2d([sorted_pop[-1]])[0]
            pop = sorted_pop[-1:-(self.sizepop + 1):-1]
            errolist.append(1 / best_fitness)
            if 1 / best_fitness < 0.0001:
                print("Best_score = " + str(round(1 / best_fitne
ss, 4)))
                print("Best_pos = " + str([round(a, 4) for a in
best_pos]))
                break
        return best_fitness, best_pos, errolist


if __name__ == "__main__":
    # 价值函数，求函数最小值点 -> [1, -1, 0, 0]
    def Fobj(factor):
        cost = (factor[0] - 1) ** 2 + (factor[1] + 1) ** 2 + fac
tor[2] ** 2 + factor[3] ** 2
        return 1 / cost
    starttime = time.time()
    a = GA(100, 50, 10, 0.8, 0.01, 4, [-1, -1, -1, -1], [1, 1, 1
, 1], Fobj)
    Best_score, Best_pos, errolist = a.Run()
```

```
    endtime = time.time()
    print("Runtime = " + str(endtime - starttime))
    a.Ploterro(errolist)
```

蚁群算法

果蝇算法

分类器

SVM支持向量机

原理

支持向量机的原理大致可以理解为特征在无穷维度的线性分类，通过核函数映射，我们不断的增加特征将其分开并进行还原。

算法

```
# -*- coding: utf-8 -*-
"""
SVC
"""
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model,cross_validation,svm

def load_data_classfication():
    '''
    加载用于分类问题的数据集

    :return: 一个元组，用于分类问题。元组元素依次为：训练样本集、测试样本集、训练样本集对应的标记、测试样本集对应的标记
    '''
    iris=datasets.load_iris()# 使用 scikit-learn 自带的 iris 数据集
    X_train=iris.data
    y_train=iris.target
```

```
    return cross_validation.train_test_split(X_train, y_train,te
st_size=0.25,
        random_state=0,stratify=y_train) # 分层采样拆分成训练集和测
试集，测试集大小为原始数据集大小的 1/4

def test_SVC_linear(*data):
    '''
    测试 SVC 的用法。这里使用的是最简单的线性核

    :param data:  可变参数。它是一个元组，这里要求其元素依次为：训练样本
集、测试样本集、训练样本的标记、测试样本的标记
    :return: None
    '''
    X_train,X_test,y_train,y_test=data
    cls=svm.SVC(kernel='linear')
    cls.fit(X_train,y_train)
    print('Coefficients:%s, intercept %s'%(cls.coef_,cls.interce
pt_))
    print('Score: %.2f' % cls.score(X_test, y_test))
def test_SVC_poly(*data):
    '''
    测试多项式核的 SVC 的预测性能随 degree、gamma、coef0 的影响.

    :param data:  可变参数。它是一个元组，这里要求其元素依次为：训练样本
集、测试样本集、训练样本的标记、测试样本的标记
    :return: None
    '''
    X_train,X_test,y_train,y_test=data
    fig=plt.figure()
    ### 测试 degree ####
    degrees=range(1,20)
    train_scores=[]
    test_scores=[]
    for degree in degrees:
        cls=svm.SVC(kernel='poly',degree=degree)
        cls.fit(X_train,y_train)
        train_scores.append(cls.score(X_train,y_train))
        test_scores.append(cls.score(X_test, y_test))
    ax=fig.add_subplot(1,3,1) # 一行三列
    ax.plot(degrees,train_scores,label="Training score ",marker=
```

```
'+' )
    ax.plot(degrees,test_scores,label= " Testing  score ",marker=
'o' )
    ax.set_title( "SVC_poly_degree ")
    ax.set_xlabel("p")
    ax.set_ylabel("score")
    ax.set_ylim(0,1.05)
    ax.legend(loc="best",framealpha=0.5)

    ### 测试 gamma ，此时 degree 固定为 3####
    gammas=range(1,20)
    train_scores=[]
    test_scores=[]
    for gamma in gammas:
        cls=svm.SVC(kernel='poly',gamma=gamma,degree=3)
        cls.fit(X_train,y_train)
        train_scores.append(cls.score(X_train,y_train))
        test_scores.append(cls.score(X_test, y_test))
    ax=fig.add_subplot(1,3,2)
    ax.plot(gammas,train_scores,label="Training score ",marker='
+' )
    ax.plot(gammas,test_scores,label= " Testing  score ",marker=
'o' )
    ax.set_title( "SVC_poly_gamma ")
    ax.set_xlabel(r"$\gamma$")
    ax.set_ylabel("score")
    ax.set_ylim(0,1.05)
    ax.legend(loc="best",framealpha=0.5)
    ### 测试 r ，此时 gamma固定为10 ， degree 固定为 3######
    rs=range(0,20)
    train_scores=[]
    test_scores=[]
    for r in rs:
        cls=svm.SVC(kernel='poly',gamma=10,degree=3,coef0=r)
        cls.fit(X_train,y_train)
        train_scores.append(cls.score(X_train,y_train))
        test_scores.append(cls.score(X_test, y_test))
    ax=fig.add_subplot(1,3,3)
    ax.plot(rs,train_scores,label="Training score ",marker='+' )
    ax.plot(rs,test_scores,label= " Testing  score ",marker='o'
```

```
)
    ax.set_title( "SVC_poly_r ")
    ax.set_xlabel(r"r")
    ax.set_ylabel("score")
    ax.set_ylim(0,1.05)
    ax.legend(loc="best",framealpha=0.5)
    plt.show()
def test_SVC_rbf(*data):
    '''
    测试 高斯核的 SVC 的预测性能随 gamma 参数的影响

    :param data:  可变参数。它是一个元组，这里要求其元素依次为：训练样本
集、测试样本集、训练样本的标记、测试样本的标记
    :return: None
    '''
    X_train,X_test,y_train,y_test=data
    gammas=range(1,20)
    train_scores=[]
    test_scores=[]
    for gamma in gammas:
        cls=svm.SVC(kernel='rbf',gamma=gamma)
        cls.fit(X_train,y_train)
        train_scores.append(cls.score(X_train,y_train))
        test_scores.append(cls.score(X_test, y_test))
    fig=plt.figure()
    ax=fig.add_subplot(1,1,1)
    ax.plot(gammas,train_scores,label="Training score ",marker='
+' )
    ax.plot(gammas,test_scores,label= " Testing  score ",marker=
'o' )
    ax.set_title( "SVC_rbf")
    ax.set_xlabel(r"$\gamma$")
    ax.set_ylabel("score")
    ax.set_ylim(0,1.05)
    ax.legend(loc="best",framealpha=0.5)
    plt.show()
def test_SVC_sigmoid(*data):
    '''
    测试 sigmoid 核的 SVC 的预测性能随 gamma、coef0 的影响.
```

```
    :param data:  可变参数。它是一个元组，这里要求其元素依次为：训练样本集、测试样本集、训练样本的标记、测试样本的标记
    :return: None
    '''
    X_train,X_test,y_train,y_test=data
    fig=plt.figure()

    ### 测试 gamma ，固定 coef0 为 0 ####
    gammas=np.logspace(-2,1)
    train_scores=[]
    test_scores=[]

    for gamma in gammas:
        cls=svm.SVC(kernel='sigmoid',gamma=gamma,coef0=0)
        cls.fit(X_train,y_train)
        train_scores.append(cls.score(X_train,y_train))
        test_scores.append(cls.score(X_test, y_test))
    ax=fig.add_subplot(1,2,1)
    ax.plot(gammas,train_scores,label="Training score ",marker='+' )
    ax.plot(gammas,test_scores,label= " Testing  score ",marker='o' )
    ax.set_title( "SVC_sigmoid_gamma ")
    ax.set_xscale("log")
    ax.set_xlabel(r"$\gamma$")
    ax.set_ylabel("score")
    ax.set_ylim(0,1.05)
    ax.legend(loc="best",framealpha=0.5)
    ### 测试 r，固定 gamma 为 0.01 ######
    rs=np.linspace(0,5)
    train_scores=[]
    test_scores=[]

    for r in rs:
        cls=svm.SVC(kernel='sigmoid',coef0=r,gamma=0.01)
        cls.fit(X_train,y_train)
        train_scores.append(cls.score(X_train,y_train))
        test_scores.append(cls.score(X_test, y_test))
    ax=fig.add_subplot(1,2,2)
    ax.plot(rs,train_scores,label="Training score ",marker='+' )
```

```
    ax.plot(rs,test_scores,label= " Testing  score ",marker='o'
)
    ax.set_title( "SVC_sigmoid_r ")
    ax.set_xlabel(r"r")
    ax.set_ylabel("score")
    ax.set_ylim(0,1.05)
    ax.legend(loc="best",framealpha=0.5)
    plt.show()
if __name__=="__main__":
    X_train,X_test,y_train,y_test=load_data_classfication() # 生
成用于分类问题的数据集
    test_SVC_linear(X_train,X_test,y_train,y_test) # 调用 test_SV
C_linear
    # test_SVC_poly(X_train,X_test,y_train,y_test) # 调用 test_SV
C_poly
    # test_SVC_rbf(X_train,X_test,y_train,y_test) # 调用 test_SVC
_rbf
    # test_SVC_sigmoid(X_train,X_test,y_train,y_test) # test_SVC
_sigmoid test_SVC_linear
```

应用

分类

通过支持向量机，我们可以对于日内策略进行一个简单的改进DEMO：

在日内策略中，很重要的一个问题就是如何识别真实突破：

很多时候，如DualHurst策略，R-breaker策略等等，对于突破的识别仅限于价格的简单关系，并没有理论的支撑，经常会出现一些由于价格波动导致的假突破而发生错误的交易信号。

支持向量机可以对于小样本的多维度向量进行快速的分析和分类，由于基于非线性的高维空间分类的方法，其分类精度超过了传统的分类方法，并可以对于非线性数据有着较好的适应度。

[简单的支持向量机原理示意]

我们可以通过对于前N日的开高收低，量，大单持仓等进行维度建模，先进行数据分解，再进行标准化，最后进行合成信号进行分类。

首先对于数据进行小波或者EMD分解，找到特定的周期特征部分，然后对于其特征部分进行前向标准化，放入维度矩阵，进行样本内训练，之后进行分类。

关于迁移学习，对于不同的商品期货品种进行训练，从而达到迁移学习的效果

预测

神经网络以及深度学习

分类树算法**C5.0**以及随机森林

聚类算法

信号分解算法

小波分析

EMD算法

迁移学习和增量算法

增强学习**Reinforcements**

QUANTAXIS的 任务计划

以下是QUANTAXIS在2018年需要主要实现的任务和计划

数据方面

- [ ] 财务数据 需要整理
- [ ] 期货部分的数据整理
- [ ] 期货部分的数据存储
- [ ] 比特币/虚拟货币部分的数据获取接口
- [ ] 比特币/虚拟货币的实时行情
- [ ] 一些基于爬虫的数据

功能方面

- [ ] 新版回测部分的账户优化
- [ ] 网站后台的重构
- [ ] 前端打包页面的适配
- [ ] 模糊查询/文本处理部分的优化
- [ ] 实时处理部分的优化
- [x] 账户管理的优化

文档方面

- [ ] 基础部分文档的更新
- [ ] 函数注释的更新

网站部分

- [ ] 基于tornado的后台重写
- [ ] 前端功能/界面的优化补充
- [ ] websocket优化

分析部分

- [ ] 更多基础指标函数的实现
- [ ] simulate生成器实现
- [ ] 更加人性化的分析功能

对接部分

- [ ] 实盘易对接
- [ ] tradex对接
- [ ] ccxt对接
- [ ] puppet对接

QUANTAXIS FAQ

1. 1.QUANTAXIS web部分下载后npm install 安装失败/报错/警告

1. `npm install` 命令访问的是国外服务器 有时候会出现下载速度较慢的问题, 解决方案:

```
npm install cnpm -g # 需要有管理员权限
cnpm install
```

1. 如果安装时出现 `fsevent` 的warning 无需担心 这个是MAC平台下的npm包,windows/linux均会有此waring

1. 如果你的nodejs版本>9 请降级你的nodejs版本 一般 安装8系列的nodejs都不会有问题

2. 2.QUANTAXIS web 无法启动

1. 首先检查是否8080端口被占用

```
netstat -ano | findstr 8080
tasklist | findstr 8080
```

1. 其次 IE浏览器和基于IE内核的(包括且不限于)360浏览器,QQ浏览器均无法正常支持, 请替换为chrome以及基于chrome内核的浏览器版本打开
2. 最后 如果在阿里云上部署了QUANTAXISWEB 无法打开远程地址的8080均为正常现象,请尽量本地部署

3. 3.QUANTAXIS web 启动后点击登录/注册按钮无反应

1. 首先检查是否进行了初始化的 `save all / save x` 命令 该命令会在数据库中创建一个admin的用户
2. 其次检查 `localhost:3000` 端口是否开启,及backend部分代码是否开启

开启backend的代码是

```
cd QUANTAXIS\QUANTAXIS_Webkit\backend
forever start bin\www
```

4. 4.如何更新数据

在命令行输入 `quantaxis` 进入QUANTAXIS CLI

1. 如果在初始化的时候是存储日线数据的 输入 `save all` 来更新数据
2. 如果在初始化时存储了分钟线数据: 输入 `save x` 来更新数据

QUANTAXIS-Protocol

标准化协议QAS/未来协议QAF

- 当前版本:0.0.9
- 协议最后修改日期:2017-12-04
- 项目版本:QUANTAXIS 0.5.25

QAS-00x Intro 简介

QAS-001-x About

QAS-001-1 关于QUANTAXIS

QUANTAXIS量化金融策略框架,是一个面向中小型策略团队的量化分析解决方案.我们通过高度解耦的模块化以及标准化协议,可以快速的实现面向场景的定制化解决方案.QUANTAXIS是一个渐进式的开放式框架,你可以根据自己的需要,引入自己的数据,分析方案,可视化过程等,也可以通过RESTful接口,快速实现多人局域网/广域网内的协作.

QAS-001-2 关于QUANTAXIS-Standard-Protocol

QUANTAXIS-Standard-Protocol(下称QAS)是为了规范化和标准化QUANTAXIS的数据获取,数据存储,模拟市场交易,以及标准化输出而建立的协议.通过遵守QAS协议,你可以快速的实现定制化需求,当然,你也可以在QAS的基础上增加自己的团队标准.

QAS-001-3 关于QUANTAXIS-Future-Protocol

QUANTAXIS-Future-Protocol(下称QAF)是一些目前尚未实现或未添加的功能/想法,通过issue提交和pull request,你可以参与到QAF的定制与完善中来.

QAS-002-x Module 模块

QUANTAXIS是一个渐进式框架,所以会有基础模块和扩展模块之分

QAS-002-1 Basical Module 基础模块

- 数据获取类 QAFetch QAFetch 主要是从固定的API获取数据,包括且不限于(Tushare,Wind,Gmsdk)等等,你也可以引入自己的数据模块接口.
- 数据存储/更新类 QASU
- 市场机制类 QAMarket
- 回测类 QABacktest
- 账户/风险/组合管理类 QAARP
- 工具类 QAUtil

QAS-002-2 Extension Module 扩展模块

- 爬虫类 QASpider
- 信号/事件驱动类 QASignal
- 任务机制/异步类 QAEngine
- 命令行扩展类 QACmd

QAS-002-3 Mod 插件模块

- QUANTAXIS-Webkit 插件
- QUANTAXIS-OpenCenter 插件

QAS-003-x Criterion 规范

QAS-003-1 模块命名方式

模块的命名

- QA+首字母大写的方法 (QASpider/QAFetch/QAMarket/QASignal/QAEngine/QAUtil)
- QA+纯大写字母的缩写(QASU,QAARP)

> 一般而言,模块是不修改的,当然,如果你需要深度定制你的模块名,则不仅需要遵守QAS#003-1协议,还需要修改"**init**.py",才能使你的模块生效

QAS-003-2 类命名方式

类的命名

- QA+ 大写字母类+小写字母 (QA_Account,QA_Risk,QA_Market,QA_Order,QA_Backtest)

QAS-003-3 函数命名方式

函数的命名

- QA模块名小写字母的函数 (QA_util_date_stamp)

函数的命名除了QA,后面的模块已经函数全部小写,同时,以简单易懂,名词-动词形式为主

主要是为了区分和类的关系

> 但如果模块是缩写形式,则缩写部分还是大写,如 QA_SU_save_stock_day

```
QA_Order  这个是类
QA_util_time_stamp 这个是函数
```

QAS-003-4 数据库/表命名方式

由于多数据源的问题,我们规定了基础指标和扩展指标,方便用户的不同权限的自定义需求

比如说,如果你只有免费的数据源通道,那么你可以选择基础的指标(pytdx,wind大奖章)

如果你具有wind机构版权限,那么你可以在免费的指标上扩展你的指标,但仍然保持原有的基础指标名不变

```
Client: ip:port
DataBase: quantaxis
Collections:
- trade_date,stock_list,options_list
- stock_day,future_day,options_day
- stock_tick,future_tick,options_tick
- log_signal
- user_setting,user_trade_history

Basical Key:
- date
- date_stamp
- open *stock
- high *stock
- low  *stock
- close *stock
- code
- name
```

QAS-003-5 RESTful 通信命名方式

QAS-10x QAFetch 数据获取类

QAS-101-x QA*fetch_get*

QA_fetch_get 系列是数据从外部获取的方法,一般而言,这个系列的函数方法封装的都是api

在使用这个fetch_get系列的时候,一般要指定数据源,比如是wind,或者tushare,或者Gmsdk等等

在QUANTAXIS.QAFetch 可以直接用的

```
from QUANTAXIS.QAFetch import *
```

- QA_fetch_get_stock_day
- QA_fetch_get_stock_realtime
- QA_fetch_get_stock_indicator
- QA_fetch_get_trade_date

在QUANTAXIS.QAFetch.QATushare中可以用的

```
from QUANTAXIS.QAFetch.QATushare import *
```

- QA_fetch_get_stock_day
- QA_fetch_get_stock_realtime
- QA_fetch_get_stock_info
- QA_fetch_get_stock_tick
- QA_fetch_get_stock_list
- QA_fetch_get_trade_date

QAS-102x-x QAS*fetch*

这个没有get_头的,是从本地数据库中获取数据

一般是本地的交易回测引擎和策略会使用这个api

```
from QUANTAXIS.QAFetch.QAQuery import *
```

- QA_fetch_stock_day
- QA_fetch_trade_date
- QA_fetch_stock_info
- QA_fetch_stocklist_day
- QA_fetch_index_day

QAS-20x QASU(save/update)数据存储/更新

QAS-201-x QA*SU_save*

这个系列是QUANTAXIS的存储数据的api方法,使用的数据库是mongodb

QAS-*202-x QA_SU_update*

QAS-203-x QA_SU_user

QAS-30x QAMarket 市场机制类

QAS-301-x QA_Market

QAS-302-x QA_Order

bid是一个标准报价包

```
    bid={
        'price':float(16),
        'date':str('2015-01-05'),
        'amount':int(10),
        'towards':int(1),
        'code':str('000001'),
        'user':str('root'),
        'strategy':str('example01'),
        'status':'0x01',
        'order_id':str(random.random())
        }
```

QAS-40x QABacktest 回测类

- QA_Backtest QA_Backtest 里面有4个抽象类:

```
account=QA_Account()
market=QA_Market()
bid=QA_Order()
setting=QA_Setting()
```

QAS-50x QAARP(account/risk/portfolio) 账户/风险/组合管理类

- QA_Account
- QA_Risk
- QA_Portfolio

QAS-501-x QA_Account

QAS-501-x QA_Risk

QAS-501-x QA_Portfolio

QAS-60x QAUtil 工具类

- QA*util*

QAS-70x QASpider 爬虫类

QAS-80x QASignal 信号/事件驱动类

QAS-90x QAEngine 任务/事件队列

QATASK 给了5种不同场景下的解决方案:

- QA_Event 主要负责的是事件的一对多的分发和订阅
- QA_Thread 主要负责的是维护一个函数句柄队列,可以理解为一个生产者消费者模型
- QA_Engine_Center 主要负责的是一个对外的兼容接口,无论是socket,还是zeromq,celery,rabbitmq,redis等等
- QA_Multi_Processing 主要是一个多线程和多进程的
- QA_Schedule 主要是一个定时/延时任务机制

QAS-901 QA_Event

QAS-902 QA_Thread

QAS-903 QA_Engine_Center

QAS-904 QA_MultiProcessing

QAS-905 QA_schedule

QAS-100x QACmd 命令行扩展类

QUANTAXIS 重构文档

	Content
DATE	2017-12-25
Author	yutiansut

重构目标

重构账户类、市场类、订单类 用于支持

- 多账户回测管理
- 组合管理、风险控制
- 期货回测
- 卖空规则
- 优化状态恢复
- 模拟盘
- 实盘对接

ACCOUNT重构

1. 账户的修改部分：

修改了account的创建方式和组合方式

```
{USER}-{PORTFOLIO}-{ACCOUNT}-{ORDER}-{MARKET}
```

account 作为一个最小的账户单元,具有

1. 独立的下单属性*(此前在backtest中撮合)
2. 独立的账户更新规则

3. 可以被快速分配到portfolio上(基于account_cookie)

account 通过 account.message进行数据通信

```
account.message记录了account的所有状态 用于

回测时和backtest通信
实盘时的message单元
组合管理和portfolio通信

快速存储-快速恢复account状态
```

1. account的属性

```
strategy_name=''  # 策略名称
user=''  # 用户
market_type=MARKET_TYPE.STOCK_CN  # market_type
hold=[['date', 'code', 'price', 'amount', 'order_id', 'trade_id'
]]  #list
sell_available=[['date', 'code', 'price', 'amount', 'order_id',
'trade_id']] #list
init_assest=1000000 #int
order_queue=pd.DataFrame() #pd.Dataframe
cash=[] #list
history=[] #history
detail #list
assets #list
account_cookie=None  # account_cookie
```

QUANTAXIS 的事件框架:

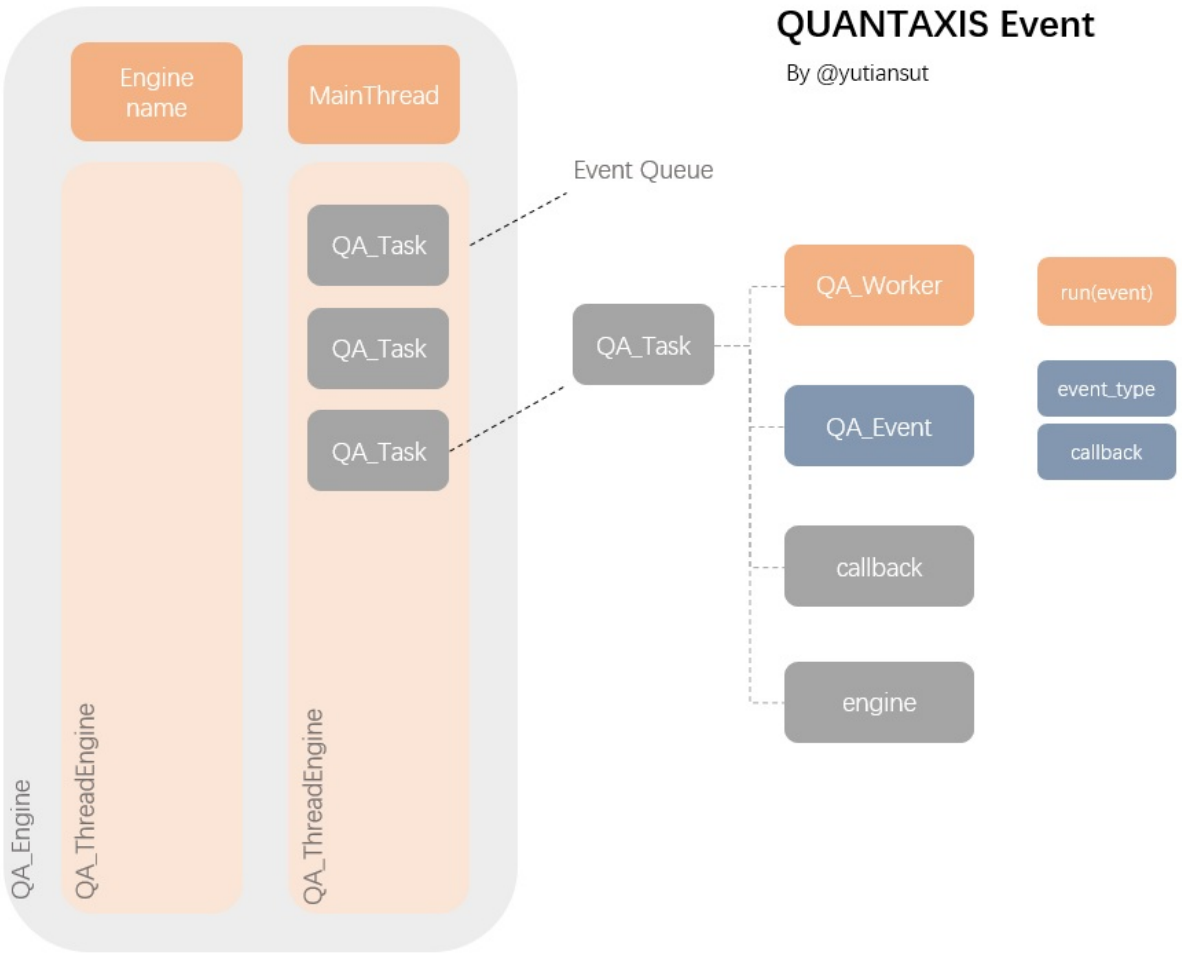
QUANTAXIS的事件框架是一个多线程架构:

```
QUANTAXIS/QAENGINE

QAENGINE分三个部分

- QAEvent
- QATask
- QAThread
(可扩展 ProcessEngine/AsyncioEngine)
```

事件的核心可以简单理解为一个带队列的线程(进程/协程),将事件分类,做成生产者消费者模式,通过队列传递任务



QATASK - 存在于队列中的标准单元

```
QA_Task(worker, event, engine=None, callback=False)

QA_Task 是被event_queue.put()中的内容

worker参数指的是 QA_Worker 需要用worker类以及继承了worker类去实例化

event指的是QA_Event event会指定事件的类别,事件的任务,事件的参数以及事件的回调

engine参数指的是在多线程引擎中,使用哪个线程去执行这个task,默认是None,及当前的主线程

callback 是回调函数,该函数不能有参数 ```if callback: callback() else :pass
```

```
## QAEVNET - 可扩展的事件任务

```QUANTAXIS
QA_Event是一个可以被继承的基础类,用于给QA_Worker指定事件任务

QA_Event(event_type=None, func=None, message=None, callback=Fals
e, *args, **kwargs)

QA_Event 必须要有的是event_type

func,message 一个是函数句柄,一个是消息句柄 可有可无
callback 是回调函数 根据自定义的worker去适配

除此以外 QA_Event可以通过**kwargs传入任何参数
```

QA_Worker - 指定执行事件的对象

```
QA_Worker是执行事件的对象,在QUANTAXIS内部 QA_Account,QA_Broker这些功
能性的类都是继承自QA_Worker

QA_Worker以及继承的类需要实现run方法,如

from QUANTAXIS.QAEngine.QAEvent import QA_Worker

class SelfDesignedWorker(QA_Worker):
    def __init__(self):
        super().__init__()

    def run(self,event):
        if event.event_type is ....:
            [do something]
        elif event.event_type is ...:
            [do another thing]
```

QA_Thread - 可自定义线程的带队列方法
```

QA_Thread是一个继承threading的带队列线程对象

通过threadengine 可以创建一个自定义的线程出来,使用event_queue来向线程推送任务(QA_Task),如果没有任务推送,线程会在后台等待

QA_Engine - 管理和分配任务的线程对象

QAEngine是一个用于管理threadengine的分派器,可以自定义创建QAThreadEngine,向指定线程推送任务 QA_Task的engine参数

参考示例:

代码在示例文件中

```
import QUANTAXIS as QA

"""
在这里 我们演示两种方法
1. 直接通过QA_Thread 创建一个事件线程做任务
2. 通过QA_Engine 来创建一个QA_Thread 来分派事件
"""

thread = QA.QA_Thread()  # 创建一个QA_Thread
engine = QA.QA_Engine()  # 创建一个QA_Engine
engine.start()  # engine 开启

engine.create_kernal('backtest')  # engine创建一个叫 backtest的线程
engine.start_kernal('backtest')  # engine 启动该线程

# 创建一个类,继承QA_Worker


class job(QA.QA_Worker):
    def __init__(self):
        super().__init__()
```

```
    def run(self, event):
        if event.event_type is 'selfdesign':
            print(vars(event))
            if event.callback:
                event.callback(event.message)
        else:
            print('unknown/unsupport event type')


jobx = job()  # 实例化这个类

# 创建一个event
event = QA.QA_Event(event_type='selfdesign', message='ssss', cal
lback=print)

# 创建一个标准task
task = QA.QA_Task(event=event, worker=jobx, engine='backtest')

# task有result方法
print(task.result)

thread.start()  # 开启thread 线程

thread.put(task)  # 向thread线程推送任务

engine.run_job(task)  # 向engine推送任务
```

QUANTAXIS MARKET

在0.6版本以后,QA_MARKET从成交功能转化成交易前置,然后解耦出来用于成交的QA_Broker和用于撮合的QA_Dealer

QUANTAXIS的market前置可以理解为一个类似于没有界面的交易客户端,通过注册账户和注册broker 来连接

在backtest中,使用时间轴(一个生成器)去推动数据更新和时间前进,在真实环境中,就是真实的时间和行情驱动前进