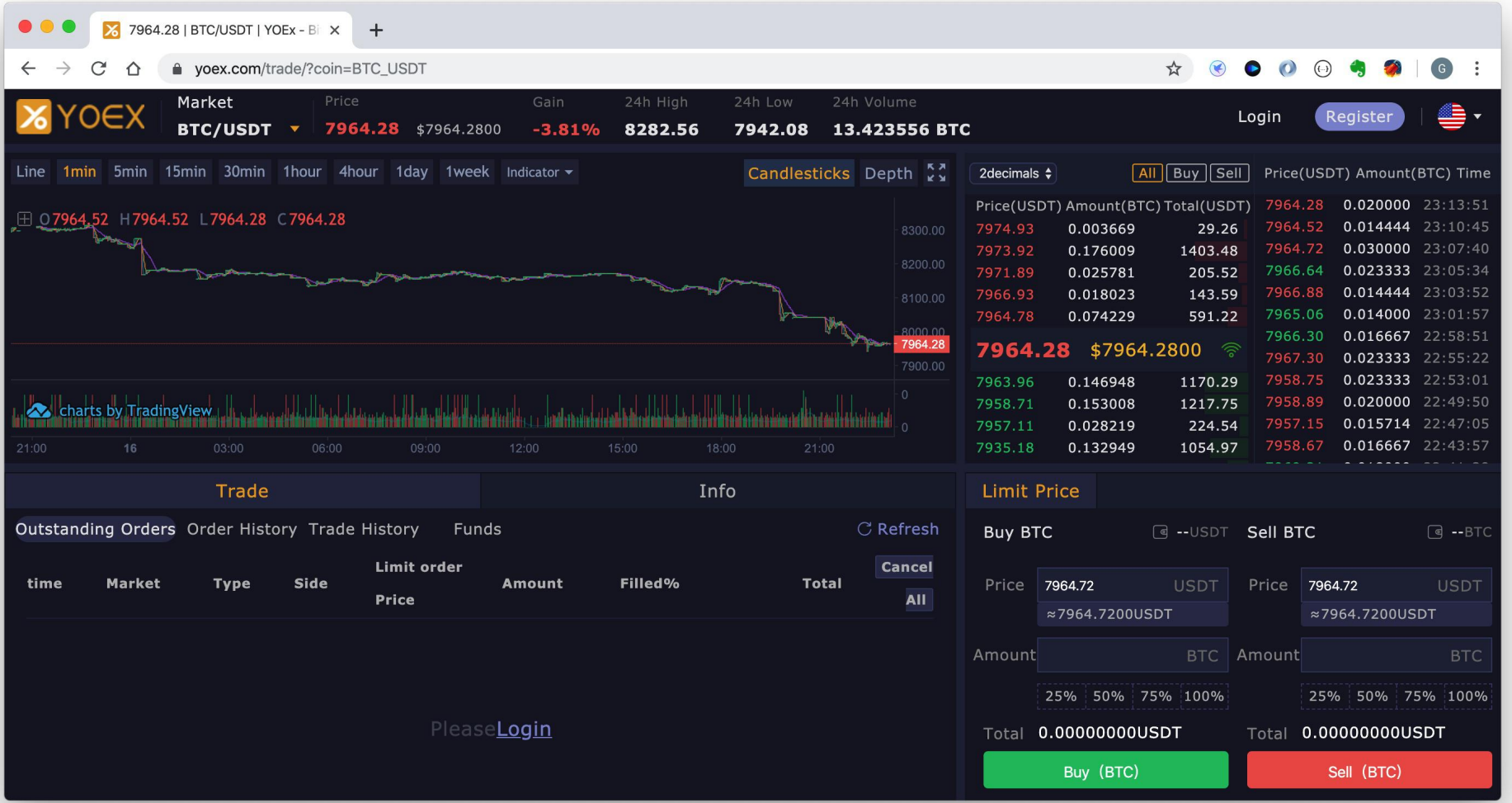
数字货币交易系统

Python实践

代少飞

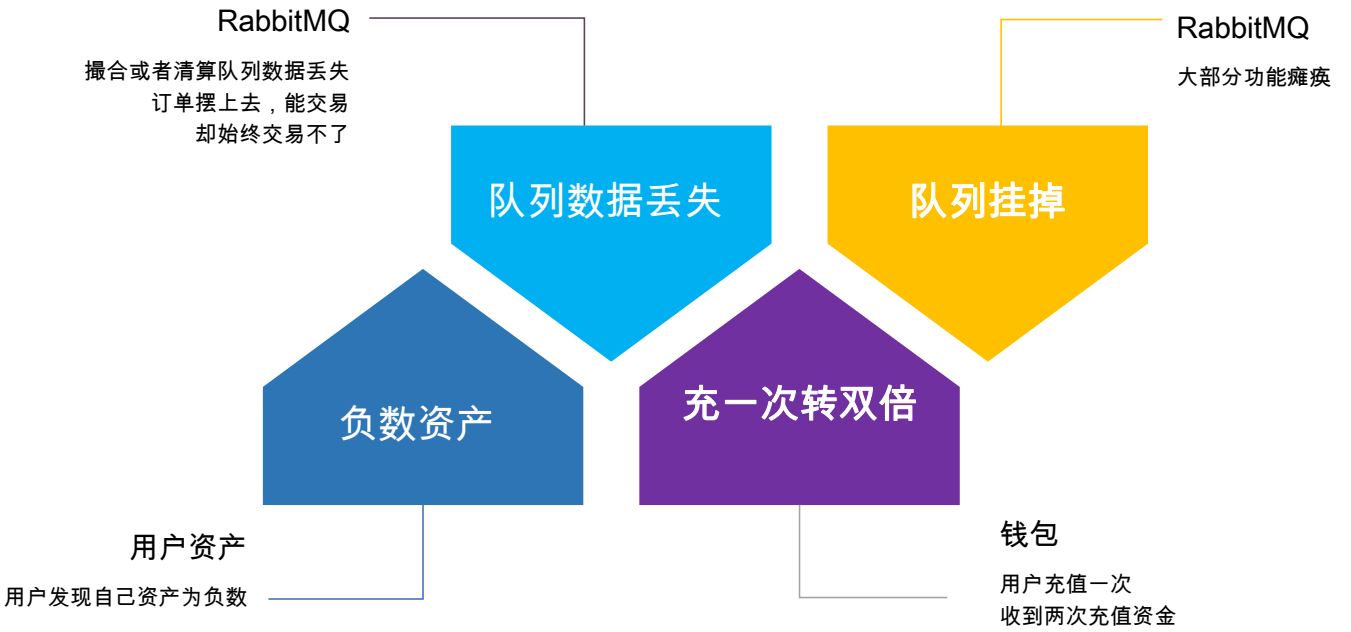
目录 CONTENTS


7964.28 | BTC/USDT | YOEx - Bi
yoex.com/trade/?coin=BTC_USDT
YOEX
Market
BTC/USDT
Price
7964.28
$7964.2800
Gain
-3.81%
24h High
8282.56
24h Low
7942.08
24h Volume
13.423556 BTC
Login
Register
Line
1min
5min
15min
30min
1hour
4hour
1day
1week
Indicator
Candlesticks
Depth
O7964.52 H7964.52 L7964.28 C7964.28
8300.00
8200.00
8100.00
8000.00
7964.28
7900.00
charts by TradingView
21:00
16
03:00
06:00
09:00
12:00
15:00
18:00
21:00
2decimals
All
Buy
Sell
Price(USDT) Amount(BTC) Total(USDT)
7974.93 0.003669 29.26
7973.92 0.176009 1403.48
7971.89 0.025781 205.52
7966.93 0.018023 143.59
7964.78 0.074229 591.22
7964.28 $7964.2800
7963.96 0.146948 1170.29
7958.71 0.153008 1217.75
7957.11 0.028219 224.54
7935.18 0.132949 1054.97
Price(USDT) Amount(BTC) Time
7964.28 0.020000 23:13:51
7964.52 0.014444 23:10:45
7964.72 0.030000 23:07:40
7966.64 0.023333 23:05:34
7966.88 0.014444 23:03:52
7965.06 0.014000 23:01:57
7966.30 0.016667 22:58:51
7967.30 0.023333 22:55:22
7958.75 0.023333 22:53:01
7958.89 0.020000 22:49:50
7957.15 0.015714 22:47:05
7958.67 0.016667 22:43:57
Trade
Info
Outstanding Orders
Order History
Trade History
Funds
Refresh
time
Market
Type
Side
Limit order Price
Amount
Filled%
Total
Cancel All
PleaseLogin
Limit Price
Buy BTC
--USDT
Sell BTC
--BTC
Price
7964.72
USDT
≈7964.7200USDT
Amount
BTC
25%
50%
75%
100%
Total
0.00000000USDT
Buy (BTC)
Sell (BTC)

1 常见问题

- 队列数据丢失
- 队列挂掉
- 负数资产
- 冲钱一次，转两倍资产

常见问题


RabbitMQ
撮合或者清算队列数据丢失
订单摆上去，能交易
却始终交易不了
队列数据丢失
RabbitMQ
大部分功能瘫痪
队列挂掉
负数资产
充一次转双倍
用户资产
用户发现自己资产为负数
钱包
用户充值一次
收到两次充值资金

RabbitMQ队列数据丢失

- 确保消息持久化，设置**durable=True,** 声明**exchange**，**queue** 持久化，**delivery_mode=2** 指明**message**为持久
- 处理完后才确认消息

```
self.channel.queue_declare(queue=queue_name, durable=True)
for message in message_list:
    message = json.dumps(message)
    self.channel.basic_publish(
        exchange='',
        routing_key=queue_name,
        body=message,
        properties=pika.BasicProperties(delivery_mode=2, ))
```

RabbitMQ挂掉

- 流程设计不合理，导致队列之间高频互传，
  最后**RabbitMQ**挂掉
- 使用**Redis**统计短时间相同订单传递次数
  或者在消息体里面添加计数字段，再做相应限制处理

资产为负

In [12]:
```
cush_start = 0.02
b = 0.1
c = 0.2
cush_end = float(cush_start) - float(b)*float(c)
cush_end
```

Out[12]:
```
-3.469446951953614e-18
```

资产为负

In [13]:
```
from decimal import Decimal
cush_end = Decimal(str(cush_start))-Decimal(str(b))*Decimal(str(c))
cush_end
```

Out[13]:
```
Decimal('0.00')
```

充一次转双倍

- 如果充值申请状态为**“start”**，充值完成状态为**“done”**，页面同时点两次，请求进入**Rabbitmq**队列就会处理两次
- 应该加入**“pending” (**处理中**)**的中间状态，然后就可以做校验，第一次充值请求，**update ** set ** pending** 返回值是**1**，去充值，如果再次请求（未完成）返回值是**0**，不充值，直到处理完成后才改成**“done”**

2 数据库锁问题

- 资产存MySQL有事务操作
- 可能会出现锁的问题

select for update 会锁住订单表的一行 (id int not null primary key)

进程1 先执行

```
import pymysql
import time

conn = pymysql.connect(host="localhost",port=3306,user="root",passwd="123456",db="ex_bank",charset="utf8")
cursor = conn.cursor(cursor=pymysql.cursors.DictCursor)
sql_1 = "select * from ex_orders where id=1 for update"
cursor.execute(sql_1)
time.sleep(10)
sql_2 = "select * from ex_orders where id=2 for update"
cursor.execute(sql_2)
```

进程2 再执行 (进程1执行后10s内)

```
import pymysql

conn = pymysql.connect(host="localhost",port=3306,user="root",passwd="123456",db="ex_bank",charset="utf8")
cursor = conn.cursor(cursor=pymysql.cursors.DictCursor)
sql_1 = "select * from ex_orders where id=2 for update"
cursor.execute(sql_1)
sql_2 = "select * from ex_orders where id=1 for update"
cursor.execute(sql_2)
```

死锁出现

进程2顺利执行

进程1会报“锁”的问题

```
Traceback (most recent call last):
  File "/Users/ghoti/PycharmProjects/ex_git/my_clearing/test/select_update_test.py", line 17, in <module>
  cursor.execute(sql_2)
  File "/Users/ghoti/.pyenv/versions/py365/lib/python3.6/site-packages/pymysql/cursors.py", line 170, in execute
  result = self._query(query)
  File "/Users/ghoti/.pyenv/versions/py365/lib/python3.6/site-packages/pymysql/cursors.py", line 328, in _query
  conn.query(q)
  File "/Users/ghoti/.pyenv/versions/py365/lib/python3.6/site-packages/pymysql/connections.py", line 516, in query
  self._affected_rows = self._read_query_result(unbuffered=unbuffered)
  File "/Users/ghoti/.pyenv/versions/py365/lib/python3.6/site-packages/pymysql/connections.py", line 727, in _read_query_result
  result.read()
  File "/Users/ghoti/.pyenv/versions/py365/lib/python3.6/site-packages/pymysql/connections.py", line 1066, in read
  first_packet = self.connection._read_packet()
  File "/Users/ghoti/.pyenv/versions/py365/lib/python3.6/site-packages/pymysql/connections.py", line 683, in _read_packet
  packet.check_error()
  File "/Users/ghoti/.pyenv/versions/py365/lib/python3.6/site-packages/pymysql/protocol.py", line 220, in check_error
  err.raise_mysql_exception(self._data)
  File "/Users/ghoti/.pyenv/versions/py365/lib/python3.6/site-packages/pymysql/err.py", line 109, in raise_mysql_exception
  raise errorclass(errno, errval)
pymysql.err.OperationalError: (1213, 'Deadlock found when trying to get lock; try restarting transaction')
```

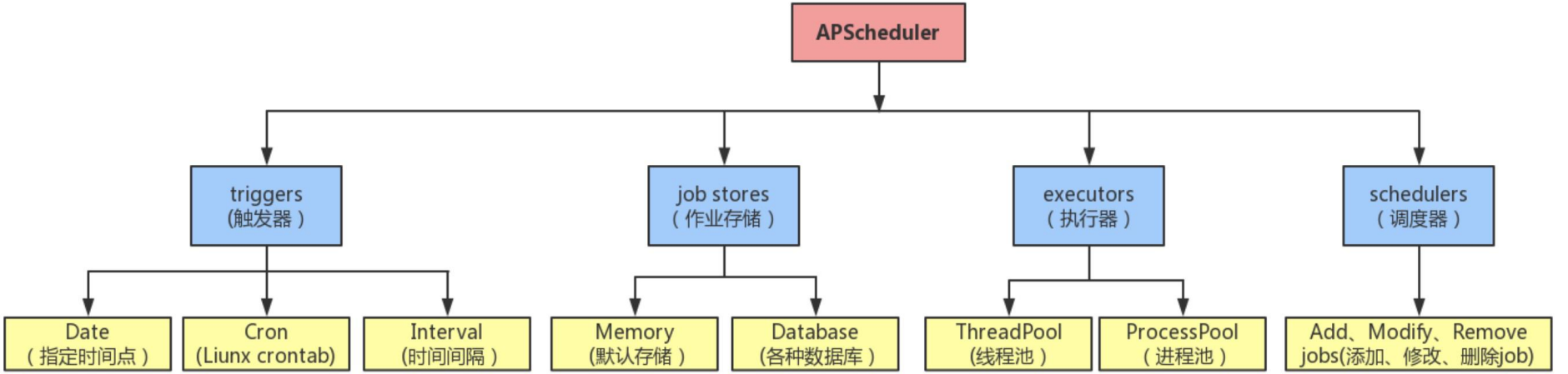
解决方法

- 直接将资产的所有操作放**redis**里面，利用**redis**的原子性处理事务

3 任务调度框架

- APScheduler

APscheduler架构



APScheduler优点

- 支持定时，定期，一次性任务
- 支持任务持久化，存储器支持广泛(**Memory**，**Postgres**，**MongoDB**，**Redis**，**ZooKeeper**，**SQLAlchemy**等)
- 调度器支持广泛(**Twisted**，**gevent**，**Tornado**，**asyncio**等)
- 可动态调整(添加，修改，删除)任务
- 远程调用支持**RPyC**(一个用作远程过程调用，同时也可以用作分布式计算的**Python**模块)，少量代码，轻松远程调用

```
from twisted.internet import reactor
from apscheduler.schedulers.twisted import TwistedScheduler

scheduler = TwistedScheduler()

scheduler.add_job(refresh_k_minute_val, 'interval', seconds=1,
                  max_instances=500, kwargs={'table': 'kline_1m_%s' % market_pair})
scheduler.add_job(update_market_val, 'interval', seconds=2, max_instances=500,
                  kwargs={'str_time': '', 'table': 'kline_1m_%s' % market_pair,
                          'table_k': 'market_info_%s' % market_pair})
scheduler.add_job(check_create_1m_history, 'interval', seconds=5,
                  max_instances=500, kwargs={'table': 'kline_1m_%s' % market_pair})

interval_instance_list = [interval_5m, interval_15m, interval_30m, interval_1h,
                          interval_4h, interval_1d, interval_1w, interval_1M]
for interval_instance in interval_instance_list:
    scheduler.add_job(update_k_val, 'interval', seconds=2, max_instances=500, kwargs={
                      'interval_instance': interval_instance})
    scheduler.add_job(check_create_1m_base_history, 'interval', minutes=2,
                      max_instances=500, kwargs={'interval_instance': interval_instance})

scheduler.start()
reactor.run()
```

4 监控

- 交易系统一行代码写错，可能损失好几百万

监控

- 首先每一笔资金变动都得有变动前后的值和原因
- 挂单精度是否正确
- 一个账户，一个币种，不算手续费，初略盘点
- 利用订单客观数据(挂单价格，数量，手续费)，复现交易，与交易记录进行对比
- 每个币种，整个资金池的流入和流出是否平衡
- 等等

THANK YOU

Ghoti

广东 深圳

扫一扫上面的二维码图案，加我微信