法律声明

本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

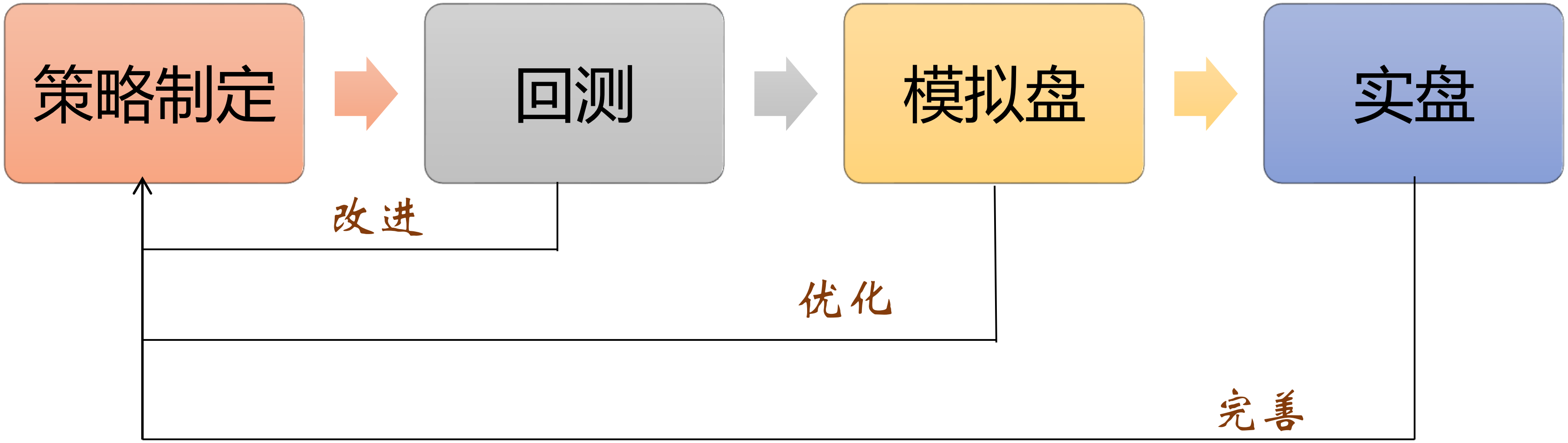
第七课

交易决策子系统的实现：模拟撮合和实盘交易接口

系统化构建量化交易体系：

模块2：**搭建自己的股票回测及交易平台**

一个量化策略的进阶之路

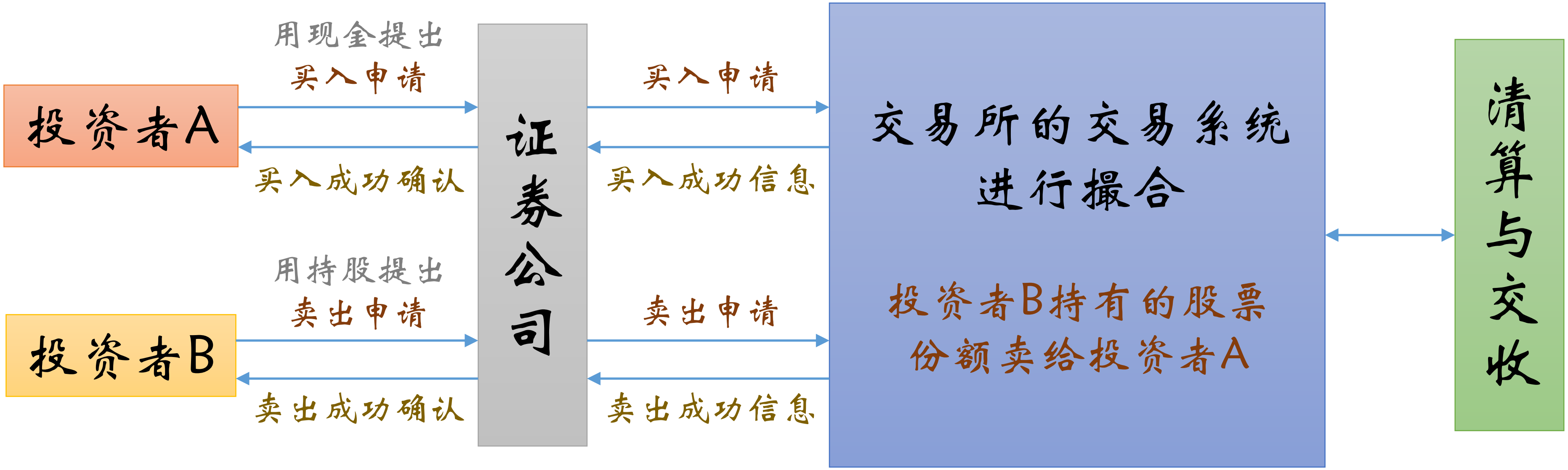


内容介绍

- 模拟撮合模块的实现和单元测试
- 策略和模拟盘对接实现
- 常见的实盘对接方式
- 如何缩小回测和实盘之间的差异
- 实盘策略运行过程中的日志分析

模拟撮合

实盘委托成交过程示意图



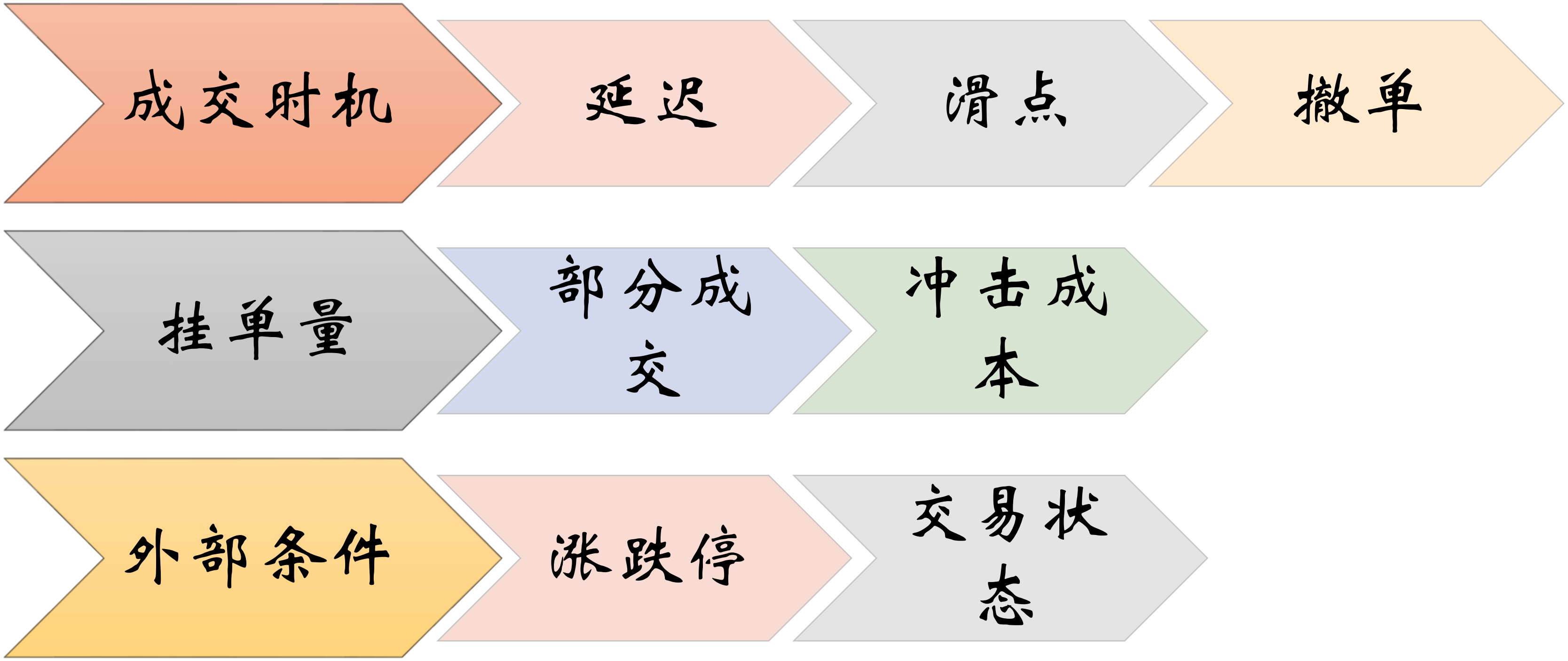
简单模拟盘的实现方法(1)

- □ 模拟盘开始运行，接收日线行情
- □ 同时接收量化策略执行中发出的买入或卖出的交易委托指令
  - ■ 委托价格（和委买委买量）由量化策略自行计算，模拟盘不做时机校验
- □ 判断以下条件是否满足：
  - ■ 虚拟账户中的现金余额充足（买委托）或当日可卖股数充足（卖委托）
  - ■ 价格落在有效区间（当日K线最高价和最低价区间内）
- □ 如果满足成交条件，则更新虚拟账户中的现金余额和股票持仓，并返回成交回报消息；否则拒绝该委托单
- □ 重复上述循环，直至系统结束运行

简单模拟盘的实现方法(2)

- 模拟盘接收实时行情（1分钟K线或tick快照行情）
- 同时接收量化策略执行中发出的买入或卖出的交易委托指令
- 如果是无效指令（无效代码、价格或委买委卖量）则拒绝，否则
- 判断以下条件是否满足：
  - 虚拟账户中的现金余额充足（买委托）或当日可卖股数充足（卖委托）
  - 价格落在有效区间（当根K线高低价之间，或盘口五档委托价范围内）
- 如果满足成交条件，则更新虚拟账户中的现金余额和股票持仓，并返回成交回报消息；否则，继续等待可成交的时机，或撤单
- 重复上述循环，直至系统结束运行

简单模拟盘的不足



成交时机

延迟

滑点

撤单

挂单量

部分成交

冲击成本

外部条件

涨跌停

交易状态

设计目标

- □ 撮合成交价与tick级行情同步
- □ 支持“部分”成交
- □ 实际成交量与五档盘口挂单量相关
- □ 体现滑点和冲击成本效果
- □ 支持全部或部分撤单（如有部分成交）
- □ 有效应对盘中临时停牌动作
- □ 正确判断涨跌停价位的成交条件

Tick数据样例

时间	2018-03-15 14:59:04		
证券代码	600000	证券名称	浦发银行
交易所	沪A	交易状态	交易
昨收盘价	12.39	成交量	17287774.0
今开盘价	12.35	成交额	213972089.0
今最高价	12.41	委比	34.18
今最低价	12.34	换手率	0.0006
最新价	12.38	量比	0.7274
涨停价	13.63	涨跌额	-0.01
跌停价	11.15	涨跌幅%	-0.0008
卖五价	12.43	卖五量	61400.0
卖四价	12.42	卖四量	473398.0
卖三价	12.41	卖三量	177146.0
卖二价	12.40	卖二量	228798.0
卖一价	12.39	卖一量	178500.0
买一价	12.38	买一量	570273.0
买二价	12.37	买二量	168500.0
买三价	12.36	买三量	258100.0
买四价	12.35	买四量	800658.0
买五价	12.34	买五量	484200.0

买卖委托单

委托单号	20180315_0012
委托时间	2018-03-15 09:37:23
证券代码	600000
交易方向	Buy (or Sell)
委托价格	12.39
委托数量	800

成交回报记录

委托单号	20180315_0012
委托时间	2018-03-15 09:37:23
成交编号	20180315_0016
成交时间	2018-03-15 09:39:31
证券代码	600000
交易方向	Buy (or Sell)
委托价格	12.39
委托数量	800
成交价格	12.38
成交数量	200 (本次成交股数)
累计成交	600

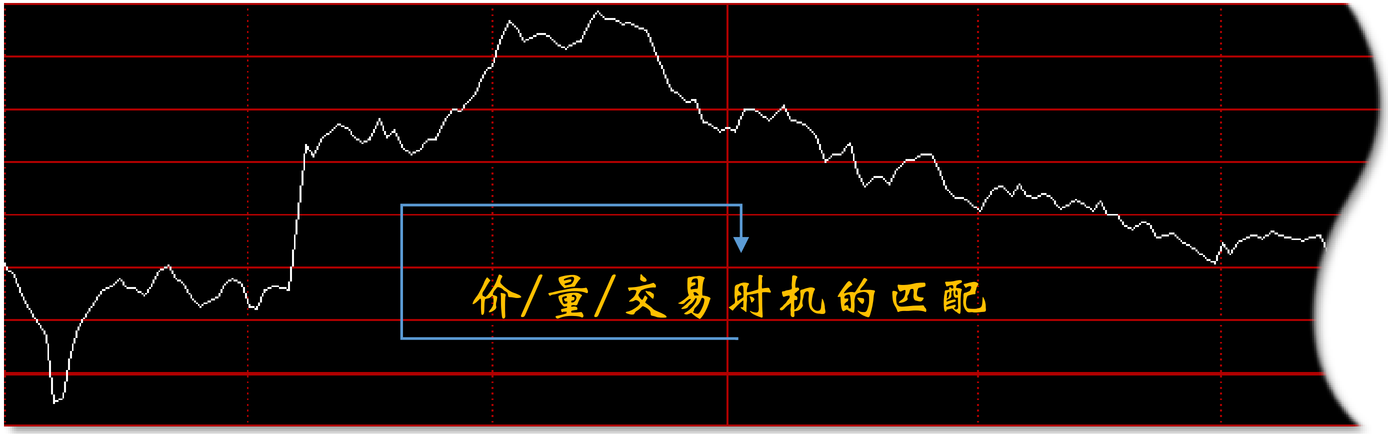
沪深交易所的撮合规则

- 委托单类型：限价指令
  - 按限定价格或更好价格成交的指令
- 当有买入价≥卖出价时，自动撮合成交
  - 优先级：价格优先、时间优先
- 撮合成交价等于买入价(BP)、卖出价(SP)和前一成交价(CP)
  三者中居中的那一个价格
  - 当BP≥SP≥CP，则最新成交价=SP
  - 当BP≥CP≥SP，则最新成交价=CP
  - 当CP≥BP≥SP，则最新成交价=BP

仿真撮合的工作机制

Tick级行情数据
（最新价+五档盘口）

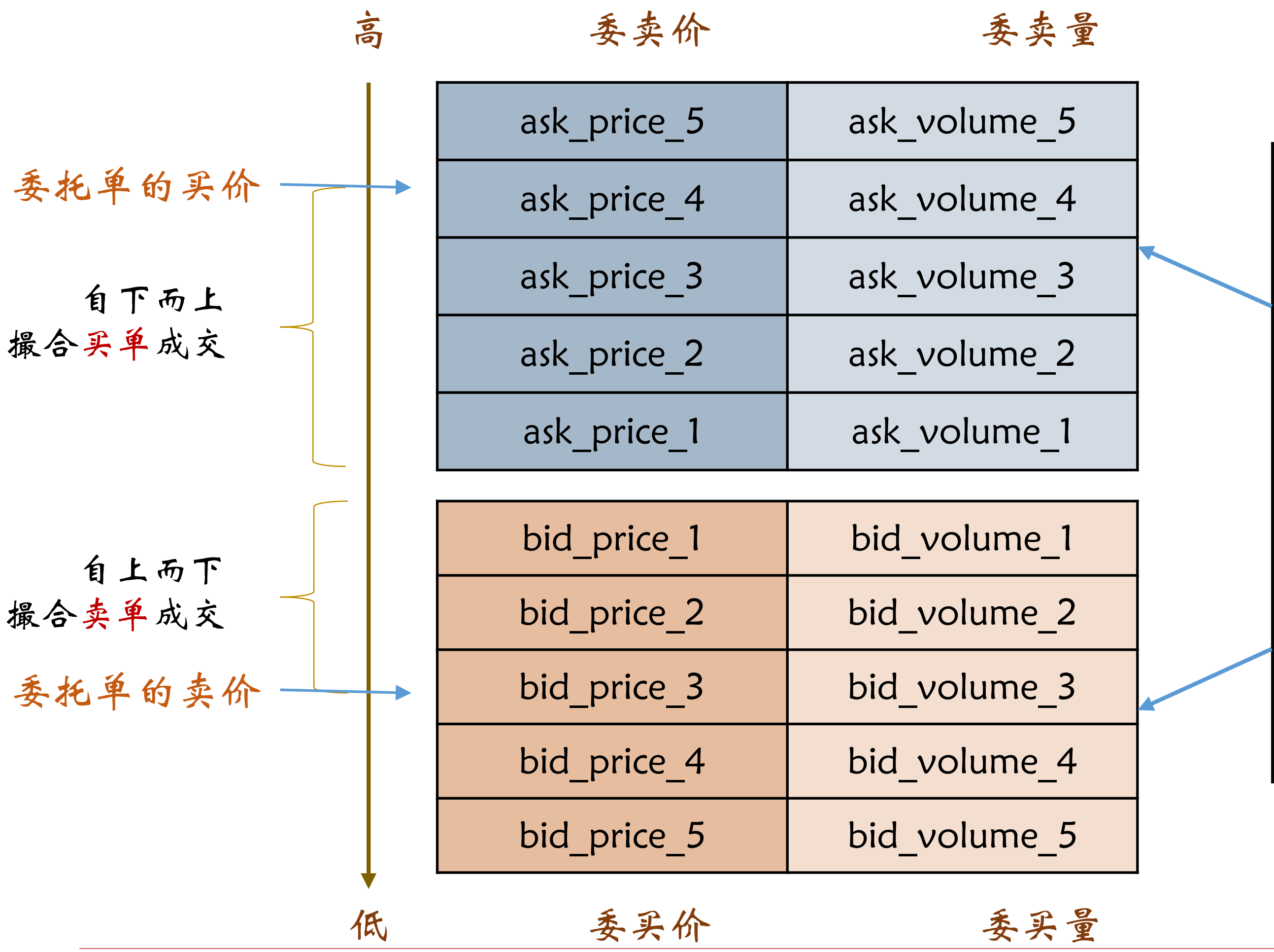
交易委托单
（买入/卖出指令）



价/量/交易时机的匹配

撮合成交记录
（成交价、成交股数…）

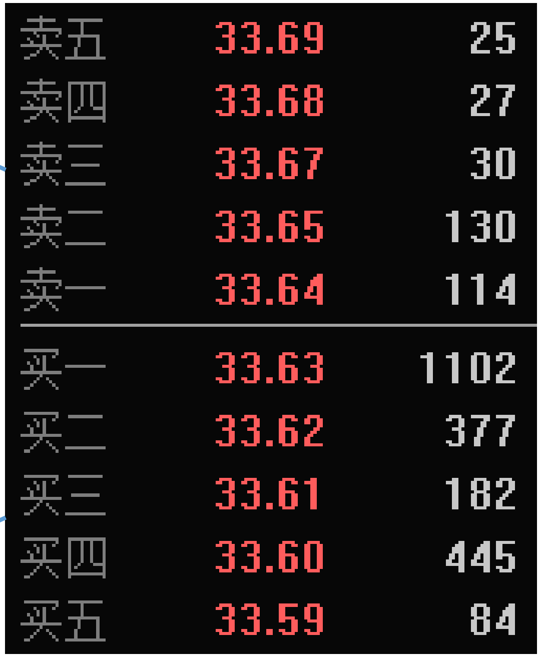
基本算法原理



委卖价	委卖量
ask_price_5	ask_volume_5
ask_price_4	ask_volume_4
ask_price_3	ask_volume_3
ask_price_2	ask_volume_2
ask_price_1	ask_volume_1

委买价	委买量
bid_price_1	bid_volume_1
bid_price_2	bid_volume_2
bid_price_3	bid_volume_3
bid_price_4	bid_volume_4
bid_price_5	bid_volume_5

盘口报价样例



卖五	33.69	25
卖四	33.68	27
卖三	33.67	30
卖二	33.65	130
卖一	33.64	114
买一	33.63	1102
买二	33.62	377
买三	33.61	182
买四	33.60	445
买五	33.59	84

运行过程(1)

- 模拟盘源源不断地、实时地接收tick级行情数据
- 同时接收运行中的量化策略发出的买入或卖出股票的委托单
- 每当收到一个新的tick时，取出它对应的股票代码code
  - 根据此code，筛选出所有未成交的、与此code有关的买卖委托单
  - 先按委托价格和委托时间排序：
    - 对所有买入委托单，按委托价格从大向小排序（高价优先）
    - 对所有卖出委托单，按委托价格从小向大排序（低价优先）
    - 同等价格条件下，按委托时间从前向后排序（先委托者优先）
  - 然后从前往后，顺序地对每一个委托单进行处理：
    - （接下页）

运行过程(2)

- □ 如果它是一个买入委托单（buy）：
  - ■ 检查它的买入报价是否落在该tick的五档委卖价范围内，如在，位置为：

$$i^b = \arg\max_i\left(ask\_price_i \mid ask\_price_i \le quote^b\right)$$

- □ 如果它是一个卖出委托单（sell）：
  - ■ 检查它的卖出报价是否落在该tick的五档委买价范围内，如在，位置为：

$$i^s = \arg\min_i(bid\_price_i \mid bid\_price_i \ge quote^s)$$

其中：

- *是五档委卖价，是五档委买价*
- *是委托单报价，上标表示，上标表示*
- *表示符合撮合条件的最大或最小挂单价所在的位置*

无效委托单的处理

- 以下情况视为无效委托单，将被直接拒绝
  - 无效的股票代码
  - 非交易日（或临时停牌状态下）挂单
  - 非交易时段（或不接受委托时段）挂单
  - 无效的委托价格
  - 无效的委托数量
  - 委买股数不是100的倍数（整手）

交易状态的处理

- 如果接收到买卖委托单时，对应的股票代码处于停牌状态，则直接拒绝
  - （参考上页）
- 如果已成功委托，但暂未成交，而期间发生了盘中临时停牌，则继续等待交易时机
  - 接收到的tick数据中有停复牌标志，可用来确定该股票的交易状态
- 收盘时刻如果仍有未成交的委托单，则撮合交易仿真系统主动撤单，并给策略运行主体发送通知消息

涨跌停价的处理

- 如果是封住涨停状态，则暂不成交买单
  - 定义：盘口买1价等于涨停价，卖1量（及其上各档委卖量）为零
- 如果是封住跌停状态，则暂不成交卖单
  - 定义：盘口卖1价等于跌停价，买1量（及其下各档委买量）为零
- 直到涨跌停状态打开，才按前面介绍的运行过程，继续进行正常的撮合交易仿真动作

模拟盘 – 初始化

```
# 是否支持部分成交
_enable_partial_filling = True

TimeSlot = Enum("TimeSlot", "NA new_day pre_trade trade_hours post_trade day_close")


class DealsMaker(threading.Thread):
    def __init__(self):
        super(DealsMaker, self).__init__()
        self.halt_dm = None
        self.inner_feeds = None
        self.feeds_sender = None
        self.simu_tick_receiver = None

        self.current_date = None
        self.current_slot = TimeSlot.trade_hours
        self.day_flip_flop = None
        self.is_noon_time = None
        self.order_seq_in_day = None

        self.latest_ticks_info = None
        self.active_orders = None
        self.expired_orders = None

        self._feeds_queue = queue.Queue()
        self._orders_queue = queue.Queue()
        self._ticks_queue = queue.Queue()
```

模拟盘 – 启动和停止

```
def run(self):

    # create and run a feeds-processor instance
    self.inner_feeds = queue.Queue()
    self.feeds_sender = FeedsProcessor(self.inner_feeds)
    self.feeds_sender.start()

    self.simu_tick_receiver = SimuTickReceiver(self._ticks_queue)
    self.simu_tick_receiver.start()

    # start the deal-making service
    self.halt_dm = threading.Event()
    while not self.halt_dm.is_set():
        self.mark_time_slot_in_day()
        self.work_on_orders_and_ticks()
    self._feeds_queue.empty()

    # notify the feeds-processor to stop as well
    self.inner_feeds.put_nowait(None)
    self.feeds_sender.join()
    self.simu_tick_receiver.join()

def stop_dm(self, wait=True):
    if self.halt_dm is None:
        return
    self.halt_dm.set()
    if wait and self.is_alive():
        self.join()

    if self.simu_tick_receiver is None:
        self.simu_tick_receiver.stop()
```

模拟盘 – 下单

```
def new_order(self, op, code, num, price):
    """
    下单
    :param op: 交易类型，buy - 买入， sell - 卖出
    :param code: 股票代码
    :param num: 委托股数
    :param price: 期望价格
    """
    order = {
        'op': op,
        'code': code,
        'num': num,
        'price': price
    }
    print('用户下单，方向：%s，股票代码：%s，委托量：%7d，价格：%7.2f'
          % (op, code, num, price), flush=True)
    self._orders_queue.put_nowait(order)
```

模拟盘 – 处理委托单和Tick

```
def work_on_orders_and_ticks(self):
    """
    处理接收到的委托单和tick数据
    """
    # time.sleep(0.01)

    try:
        if self._orders_queue.empty() is False:
            order = self._orders_queue.get(timeout=1)
            if len(order) > 0:
                self.on_receive_order(order)
    except Exception:
        traceback.print_exc()

    # time.sleep(0.01)
    count = 0
    try:
        if self._ticks_queue.empty() is False:
            tick = self._ticks_queue.get(timeout=1)
            if len(tick) > 0:
                    self.on_receive_tick(tick)
                    count += 1
    except Exception:
        traceback.print_exc()

    time.sleep(0.5)

    if self.current_slot is TimeSlot.trade_hours and len(self.active_orders) > 0 and count > 0:
        self.do_deals_matching()
```

模拟盘 – 生成委托单ID

```
def mark_time_slot_in_day(self):
    """
    维持日内的交易时间
    """
    if self.current_date is None:
        self.current_date = '2018-07-19'
        self.current_slot = TimeSlot.trade_hours

        self.on_slot_new_day()


# ------------------------

@property
def next_order_id(self):
    _order_id = "{}_{:0>4}".format(self.current_date.replace("-", ""), self.order_seq_in_day)
    self.order_seq_in_day += 1
    return _order_id

def on_slot_new_day(self):
    print("** new_day", flush=True)

    self.day_flip_flop = 0
    self.is_noon_time = False
    self.order_seq_in_day = 1
    self.latest_ticks_info = dict()

    self.active_orders = dict()
    self.expired_orders = dict()
```

模拟盘 — 收盘处理

```python
def on_slot_day_close(self):
    """
    收盘事件的处理
    """
    print("** day_close", flush=True)

    for ptr in self.active_orders.values():
        ptr["state"] = "closed"
        self.persistent_record("order", ptr)

        one_order = ptr["raw_order"]
        obj = dict(
            msg_type="closed",
            message="已关闭",
            reason="",
            order_id=ptr["order_id"],
            code=one_order["code"],
            # more info
            submit_time=ptr["submit_time"],
            revoke_time=datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            revoke_num=ptr["raw_order"]["num"] - ptr["acc_num"]
        )
        self.push_feed_to_queue(obj)
    self.expired_orders.update(self.active_orders)
    self.active_orders = dict()
```

模拟盘 — 委托单处理

```python
def on_receive_order(self, one_order):
    print("## received:", one_order, flush=True)
    success, extra = self.validate_format(one_order)
    if not success:
        obj = dict(
            msg_type="invalid",
            message="无效单",
            reason=extra,
        )
        self.push_feed_to_queue(obj)
        return

    success, extra = self.validate_timeslot()
    if not success:
        obj = dict(
            msg_type="rejected",
            message="拒绝",
            reason=extra
        )
        self.push_feed_to_queue(obj)
        return

    if one_order["op"] in ["buy", "sell"]:
        success, extra = self.check_available_ticks(one_order)
        if not success:
            obj = dict(
                msg_type="rejected",
                message="拒绝",
                reason=extra
            )
            self.push_feed_to_queue(obj)
            return
        self.register_new_order(one_order)
```

模拟盘 – 委托单处理

```
def validate_format(self, order):
    """
    验证交易指令格式的正确
    :param order: 交易指令
    :return: True - 格式正确, False - 格式错误
    """
    op = order.get("op")
    if op is None or op not in ["buy", "sell"]:
        return False, "无效指令"

    required_fields = set(["op"])
    if op in ["buy", "sell"]:
        required_fields.update(["code", "num", "price"])

    provided_fields = set(order.keys())
    if not required_fields.issubset(provided_fields):
        return False, "缺少必填字段"

    if op in ["buy", "sell"]:
        code = order.get("code")
        if code is None or type(code) is not str or code.strip() == "":
            return False, "证券代码格式错误"

        try:
            if type(order["num"]) is not int:
                order["num"] = int(order["num"])
            if type(order["price"]) is not float:
                order["price"] = float(order["price"])
        except Exception as e:
            return False, "数量或价格字段格式错误"

        if order["num"] <= 0 or order["price"] <= 0.0:
            return False, "申报数量或价格取值无效"
        if order["op"] == "buy" and order["num"] % 100 != 0:
            return False, "申买数量不是100的倍数"

    return True, None
```

```python
def validate_timeslot(self):
    if self.current_slot is not TimeSlot.pre_trade and \
            self.current_slot is not TimeSlot.trade_hours \
            or self.is_noon_time:
        return False, "非交易时段"

    return True, None

def check_available_ticks(self, one_order):
    ptr = self.latest_ticks_info.get(one_order["code"])
    if ptr is None:
        return False, "证券代码无效或暂无行情数据"

    return True, None
```

模拟盘 – 验证状态

```python
def register_new_order(self, one_order):
    order_id = self.next_order_id  # allocate a new order_id sequentially
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    self.persistent_record("request", one_order)

    # add to active orders' queue
    ptr = dict(
        order_id=order_id,
        submit_time=timestamp,
        raw_order=one_order,
        state="active",
        last_deal_time=None,
        acc_num=0,
        num_deals_made=0
    )
    self.active_orders[order_id] = ptr
    self.persistent_record("order", ptr)
```

模拟盘 – 添加新委托

模拟盘

```
 # send feedback form to client
    obj = dict(
        msg_type="accepted",
        message="已报",
        reason="",
        order_id=order_id,
        code=one_order["code"],
        submit_time=timestamp
    )
    self.push_feed_to_queue(obj)

def push_feed_to_queue(self, obj):
    try:
        self.inner_feeds.put_nowait(("feed", obj))
    except asyncio.QueueFull as e:
        print("Warning: inner queue for feeds-sending is full!", flush=True)

def persistent_record(self, category, obj):
    try:
        self.inner_feeds.put_nowait((category, obj))
    except asyncio.QueueFull as e:
        print("Warning: inner queue for feeds-sending is full!", flush=True)
```

模拟盘

```python
def on_receive_tick(self, one_tick):
    # print("处理Tick: %s, %s" % (one_tick['code'], one_tick['time']), flush=True)
    if one_tick["code"][0] > "6":
        return

    # ordinary codes
    tick_time = one_tick['time']
    if self.current_slot not in [TimeSlot.pre_trade, TimeSlot.trade_hours] or \
            self.current_slot is TimeSlot.trade_hours and self.is_noon_time:
        return  # in case ticks arrives before index, and ignore noon-time ticks
    if self.current_slot is TimeSlot.trade_hours and not self.is_noon_time and \
            (tick_time < "09:30:00" or tick_time >= "15:00:00" or "11:30:00" <= tick_time < "13:00:00"):
        # print("  -- ignored non-sync ticks:", the_tm, flush=True)
        return  # ignore non-trading hours in case of non-sync ticks

    code = one_tick["code"]
    ptr = self.latest_ticks_info.get(code)
    if ptr is None:
        ptr = dict(
            day_high=one_tick["high"],
            day_low=one_tick["low"],
            new_high=False,  # reaches new highest price compared w/ previous ticks in day
            new_low=False,  # reaches new lowest price compared w/ previous ticks in day
            prev_point=one_tick["price"],
            raw_tick=one_tick
        )
        self.latest_ticks_info[code] = ptr
    else:
        ptr["new_high"] = (one_tick["high"] > ptr["day_high"])
        ptr["new_low"] = (one_tick["low"] < ptr["day_low"])
        ptr["prev_point"] = ptr["raw_tick"]["price"]
        ptr["raw_tick"] = one_tick
        ptr["day_high"] = one_tick["high"]
        ptr["day_low"] = one_tick["low"]
```

模拟盘

```
def do_deals_matching(self):
    buy_list = [ptr for ptr in self.active_orders.values() if ptr["raw_order"]["op"] == "buy"]
    sell_list = [ptr for ptr in self.active_orders.values() if ptr["raw_order"]["op"] == "sell"]
    completed = list()

    if len(buy_list) > 0:
        buy_list.sort(key=lambda x: (x["raw_order"]["code"], -x["raw_order"]["price"], x["order_id"]))  #
H->L
        for ptr in buy_list:
            code = ptr["raw_order"]["code"]
            one_code_tick = self.latest_ticks_info.get(code)  # may not exist due to reloading from db
            if one_code_tick is None:
                continue
            fully_filled = self.try_a_match("buy", ptr, one_code_tick)
            if fully_filled:
                completed.append(ptr)

    if len(sell_list) > 0:
        sell_list.sort(key=lambda x: (x["raw_order"]["code"], x["raw_order"]["price"], x["order_id"]))  #
L->H
        for ptr in sell_list:
            code = ptr["raw_order"]["code"]
            one_code_tick = self.latest_ticks_info.get(code)  # may not exist due to reloading from db
            if one_code_tick is None:
                continue
            fully_filled = self.try_a_match("sell", ptr, one_code_tick)
            if fully_filled:
                completed.append(ptr)

    # move completed orders from active queue to expired queue
    for ptr in completed:
        order_id = ptr["order_id"]
        self.expired_orders[order_id] = self.active_orders.pop(order_id)
```

模拟盘

```
def try_a_match(self, op, order_ptr, tick_ptr):
    one_order = order_ptr["raw_order"]
    one_tick = tick_ptr["raw_tick"]
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    prev_point = tick_ptr["prev_point"]
    latest = one_tick["price"]
    highest = max(prev_point, latest)
    if tick_ptr["new_high"]:
        highest = max(highest, tick_ptr["day_high"])
    lowest = min(prev_point, latest)
    if tick_ptr["new_low"]:
        lowest = min(lowest, tick_ptr["day_low"])
    quotation = one_order["price"]

    fully_filled = False
    deal_price = None

    if _enable_partial_filling:
        try:
            pending_quotes = [
                (one_tick["a5_p"], one_tick["a5_v"]),
                (one_tick["a4_p"], one_tick["a4_v"]),
                (one_tick["a3_p"], one_tick["a3_v"]),
                (one_tick["a2_p"], one_tick["a2_v"]),
                (one_tick["a1_p"], one_tick["a1_v"]),
                (one_tick["b1_p"], one_tick["b1_v"]),
                (one_tick["b2_p"], one_tick["b2_v"]),
                (one_tick["b3_p"], one_tick["b3_v"]),
                (one_tick["b4_p"], one_tick["b4_v"]),
                (one_tick["b5_p"], one_tick["b5_v"]),
            ]
        except Exception:
            print("-- invalid tick detected:", one_tick, flush=True)
            return False
```

模拟盘

```
feasible = False
seq = None
if op == "buy":
    if quotation >= lowest:
        ub = highest if quotation > highest else quotation
        lb = lowest
        it = filter(lambda x: lb <= x[0] <= ub, pending_quotes)
        candidates = [pv for pv in it]
        seq = reversed(candidates)  # L->H
        feasible = True
else:  # "sell"
    if quotation <= highest:
        ub = highest
        lb = lowest if quotation < lowest else quotation
        it = filter(lambda x: lb <= x[0] <= ub, pending_quotes)
        candidates = [pv for pv in it]
        seq = candidates  # H->L
        feasible = True
```

模拟盘

```
if feasible:
    rest_num = one_order["num"] - order_ptr["acc_num"]
    for p, v in seq:
        deal_num = min(rest_num, v)
        rest_num -= deal_num

        # update queue info
        order_ptr["last_deal_time"] = timestamp
        order_ptr["acc_num"] += deal_num
        order_ptr["num_deals_made"] += 1
        self.persistent_record("order", order_ptr)

        filled_in_one_time = (deal_num == one_order["num"])

        # send a feed back
        obj = dict(
            msg_type="partial_filled" if not filled_in_one_time else "filled",
            message="部分成交" if not filled_in_one_time else "全部成交",
            reason="",
            order_id=order_ptr["order_id"],
            submit_time=order_ptr["submit_time"],
            # more info
            op=one_order["op"],
            code=one_order["code"],
            claim_price=one_order["price"],
            claim_num=one_order["num"],
            deal_seq_no=order_ptr["order_id"] + "_" + str(order_ptr["num_deals_made"]),
            deal_price=p,
            deal_num=deal_num,
            deal_time=timestamp,
            acc_num=order_ptr["acc_num"]
        )
```

模拟盘

```
 self.push_feed_to_queue(obj)
            self.persistent_record("deal", obj)

            if rest_num == 0:
                order_ptr["state"] = "done"
                self.persistent_record("order", order_ptr)
                fully_filled = True
                break

else:  # fully_filling only
    if op == "buy":
        if quotation >= lowest:
            if quotation > highest:
                deal_price = highest
            else:
                deal_price = quotation
            fully_filled = True
    else:  # "sell"
        if quotation <= highest:
            if quotation < lowest:
                deal_price = lowest
            else:
                deal_price = quotation
        fully_filled = True
```

模拟盘

```
 if fully_filled:
        # update queue info
        order_ptr["last_deal_time"] = timestamp
        order_ptr["acc_num"] = one_order["num"]
        order_ptr["num_deals_made"] += 1
        order_ptr["state"] = "done"
        self.persistent_record("order", order_ptr)

        # send a feed back
        obj = dict(
            msg_type="filled",
            message="全部成交",
            reason="",
            order_id=order_ptr["order_id"],
            submit_time=order_ptr["submit_time"],
            # more info
            op=one_order["op"],
            code=one_order["code"],
            claim_price=one_order["price"],
            claim_num=one_order["num"],
            deal_seq_no=order_ptr["order_id"] + "_" + str(order_ptr["num_deals_made"]),
            deal_price=deal_price,
            deal_num=one_order["num"],
            deal_time=timestamp,
            acc_num=one_order["num"]
        )
        self.push_feed_to_queue(obj)
        self.persistent_record("deal", obj)

return fully_filled
```

模拟盘 – Feed处理

```python
class FeedsProcessor(threading.Thread):
    """
    单独的日志处理线程
    """

    def __init__(self, inner_feeds):
        super(FeedsProcessor, self).__init__()
        self.inner_feeds = inner_feeds

    def run(self):
        while True:
            item = self.inner_feeds.get()
            if item is None:
                break
            category, original_record = item
            print(original_record, flush=True)
```

模拟盘 – 从文件中加载测试数据

```
class SimuTickReceiver(threading.Thread):
    def __init__(self, tick_queue):
        super(SimuTickReceiver, self).__init__()
        self.tick_queue = tick_queue
        self.running = True

    def run(self):
        ticks_file = os.path.join(sys.path[0], 'ticks')

        # 时间从开盘开始模拟，替换掉原始数据的时间
        tick_time = datetime.strptime('09:30:00', '%H:%M:%S')
        while self.running:
            with open(ticks_file) as contents:
                for line in contents:
                    tick = json.loads(line)
                    tick['time'] = tick_time.strftime('%H:%M:%S')

                    self.tick_queue.put_nowait(tick)
                    tick_time += timedelta(seconds=1)
                    time.sleep(0.01)

            time.sleep(3)

    def stop(self):
        self.running = False
```

模拟盘 – 从文件中加载测试数据

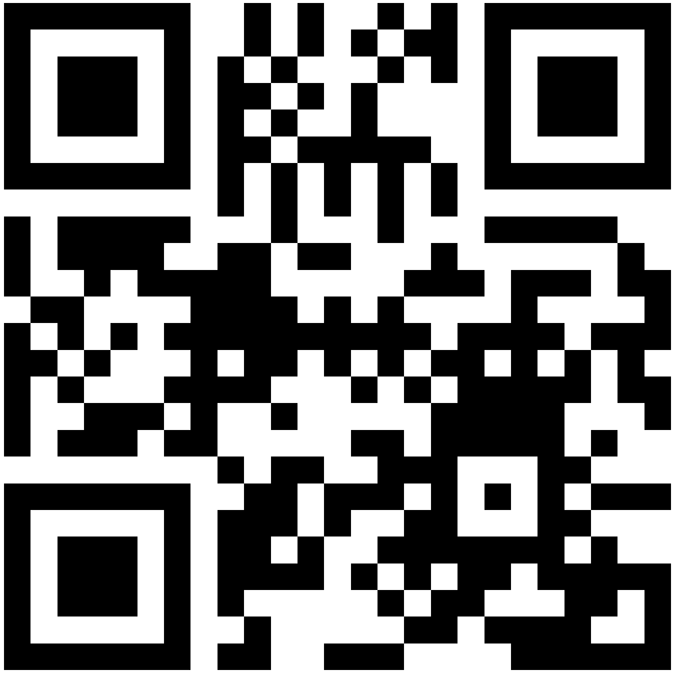
```
class SimuTickReceiver(threading.Thread):
    def __init__(self, tick_queue):
        super(SimuTickReceiver, self).__init__()
        self.tick_queue = tick_queue
        self.running = True

    def run(self):
        ticks_file = os.path.join(sys.path[0], 'ticks')

        # 时间从开盘开始模拟，替换掉原始数据的时间
        tick_time = datetime.strptime('09:30:00', '%H:%M:%S')
        while self.running:
            with open(ticks_file) as contents:
                for line in contents:
                    tick = json.loads(line)
                    tick['time'] = tick_time.strftime('%H:%M:%S')

                    self.tick_queue.put_nowait(tick)
                    tick_time += timedelta(seconds=1)
                    time.sleep(0.01)

            time.sleep(3)

    def stop(self):
        self.running = False
```

模拟盘 — 入口

```
def main():
    the_dm = DealsMaker()
    print("The Deals Maker is running, enter 'info' or 'exit' to debug or stop it...", flush=True)
    the_dm.start()
    while True:
        cmd = input().strip()
        if cmd.startswith('order: '):
            orders = cmd[7:].split(',')
            the_dm.new_order(orders[0], code=orders[1], num=int(orders[2]), price=float(orders[3]))
        elif cmd == "exit":
            break
    the_dm.stop_dm()


if __name__ == "__main__":
    main()
```

休息一下

5分钟后回来



策略和模拟盘的对接实现

我们已经完成的任务

- 数据获取
  - 行情、基本面（基本信息和财务数据）
- 数据异常处理
  - 复权因子、涨跌停价、交易状态、停牌K线
- 因子计算
- 策略制定
  - 股票池
- 策略回测
  - 信号计算
  - 仓位管理
  - 止盈止损
- 模拟盘

策略和模拟盘的对接

- 交易日盘后，15:30
  - 完成数据获取
  - 完成数据异常处理
  - 计算因子
  - 回测（回测的结束日期为今天）
  - 运行日志分析脚本
  - 找到第二日需要交易的股票
- 第二日开盘，9:30
  - 启动模拟盘
  - 将头一日的待交易的票作为委托送入模拟盘

全任务脚本

```python
# -*- coding: utf-8 -*-

from datetime import datetime

from data.daily_crawler import DailyCrawler
from data.finance_report_crawler import FinanceReportCrawler
from data.fixing.daily_fixing import DailyFixing
from strategy.strategy_module import Strategy
from factor.factor_module import FactorModule

# 获取当前日期
current_date = datetime.now().strftime('%Y-%m-%d')


def crawl_data():
    """
    抓取数据
    """
    dc = DailyCrawler()
    dc.crawl_index(begin_date=current_date, end_date=current_date)
    dc.crawl(begin_date=current_date, end_date=current_date)

    fc = FinanceReportCrawler()
    fc.crawl_finance_report()
    fc.crawl_finance_summary()
```

全任务脚本

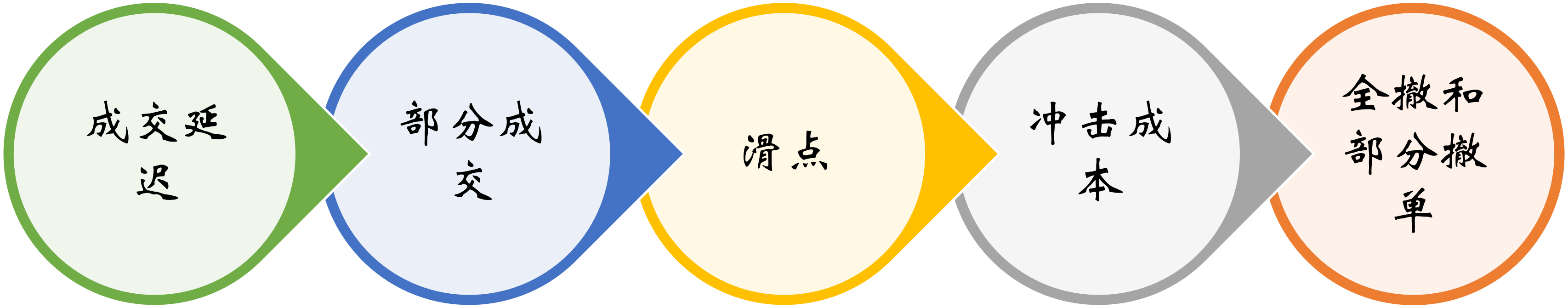
```
def fixing_data():
    “”“
    修复数据
    ”“”
    df = DailyFixing()
    # 计算复权因子和前收
    df.fill_au_factor_pre_close(current_date, current_date)
    # 计算涨停和跌停
    df.fill_high_limit_low_limit(current_date, current_date)
    # 填充缺失的K线
    df.fill_daily_k_at_suspension_days(current_date, current_date)
    # 填充交易状态
    df.fill_is_trading_between(current_date, current_date)


def compute_factor():
    “”“
    计算所有因子
    ”“”
    fc = FactorModule()
    fc.compute()


def get_today_candidates():
    “”“
    通过回测获得今天的备选股，logs/candidates.log
    """
    strategy = Strategy('low_pe_strategy')
    strategy.backtest()


if __name__ == '__main__':
    crawl_data()
    fixing_data()
    compute_factor()
    get_today_candidates()
```

常见的实盘对接方式

程序化实盘的途径

- □ 本地（图表/后台交易/券商插件）
  - 全字塔、MC8s、TB、WH8、TS、MQ/OQ、MT4、…
- □ 云端（SaaS/券商定制版）
  - 聚宽JoinQuant、优矿Uqer.io、米筐RiceQuant、Apama、BotVS、…
- □ SDK/API+UI（金融终端/Web）
  - 万得Wind、东财Choice、掘金量化gmsdk、…
- □ 开源框架（多基于Python）
  - PyCTP、VN.PY、QuickLib、…
- □ 底层接口+自己编写（策略框架+交易指令+接口调用）
  - CTP、Web Service、EasyTrader、…

如何缩小回测与实盘之间的差异


成交延迟
部分成交
滑点
冲击成本
全撤和部分撤单

实盘策略运行过程中的日志分析

日志系统的主要环节

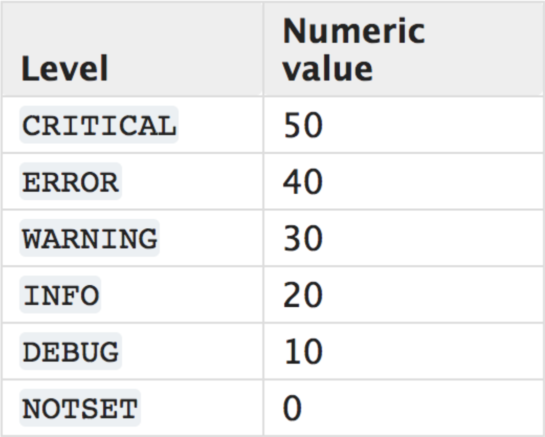
- □ 生产
- □ 采集
- □ 分析
- □ 报告

日志的分类

- 业务类
  - 股票池的筛选逻辑
  - 信号的计算过程
  - 交易的决策过程
- 系统类
  - 调试日志
  - 错误日志
  - 性能日志
  - 一般运行日志

Python的日志框架和使用

- logging
- 配置
  - 名称
  - 级别（Level）
  - 消息格式
  - 处理方式
- 项目内调用



Level	Numeric value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
NOTSET	0

日志文件配置

```
# -*- coding: utf-8 -*-

import logging.handlers
import sys


class QuantLogger:
    def __init__(self, name):
        # 业务日志的配置
        self.logger = logging.getLogger(name)

        self.logger.setLevel(logging.INFO)

        format = logging.Formatter('[%(asctime)s[%(name)s] %(levelname)s: %(message)s')
        handler = logging.handlers.TimedRotatingFileHandler(sys.path[2] + '/logs/' + name +
'.log', 'D')
        handler.setFormatter(format)

        self.logger.addHandler(handler)

        # 错误日志的配置
        self.errorLogger = logging.getLogger("ERROR")
        self.errorLogger.setLevel(logging.ERROR)
        errorFormatter = logging.Formatter('[%(asctime)s[' + name + '] %(levelname)s:
%(message)s')
        errorHandler = logging.handlers.TimedRotatingFileHandler(sys.path[2] + '/logs/error.log',
'D')
        errorHandler.setFormatter(errorFormatter)

        self.errorLogger.addHandler(errorHandler)
```

日志文件配置

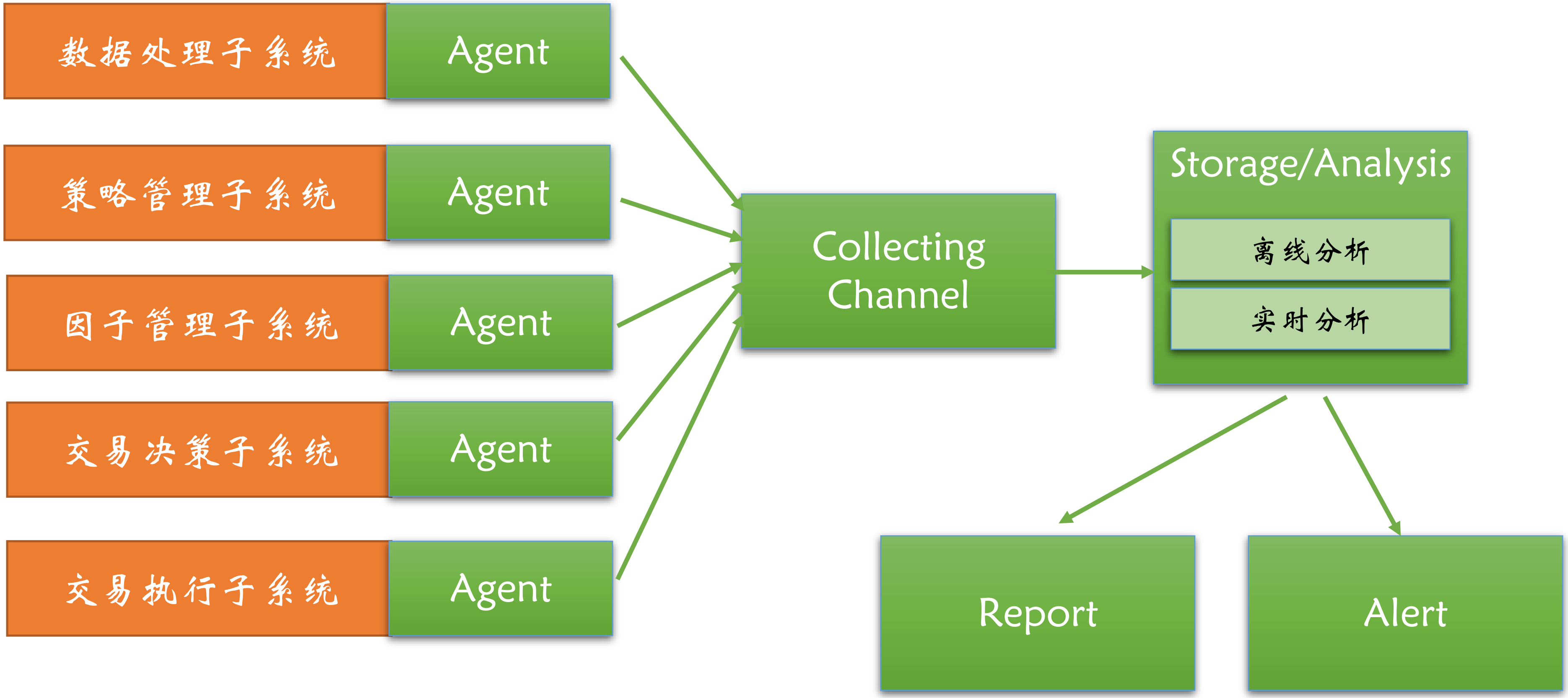
```
    # 调试日志的配置
    self.debugLogger = logging.getLogger("DEBUG")
    self.debugLogger.setLevel(logging.DEBUG)
    debugFormatter = logging.Formatter('[%(asctime)s[' + name + '] %(levelname)s:
%(message)s')
    debugHandler = logging.handlers.TimedRotatingFileHandler(sys.path[2] +
'/logs/debug.log', 'D')
    debugHandler.setFormatter(debugFormatter)

    self.debugLogger.addHandler(debugHandler)

def info(self, message, *args):
    self.logger.info(message, *args)

def error(self, message, *args):
    self.errorLogger.error(message, *args, exc_info=True)

def debug(self, message, *args):
    self.debugLogger.debug(message, *args)
```

日志的使用 – backtest.py

```
class Backtest:
    def __init__(self, strategy_option, begin_date=None, end_date=None):
        self.strategy_option = strategy_option
        if begin_date is None:
            self.begin_date = self.strategy_option.begin_date()
        else:
            self.begin_date = begin_date

        if end_date is None:
            self.end_date = self.strategy_option.end_date()
        else:
            self.end_date = end_date

        self.dm = DataModule()
        self.code_daily_cache = dict()

        self.logger = QuantLogger('backtest')
```

```
# 按照日期一步步回测
for _date in all_dates:
    self.logger.info('开始回测，日期：' + _date)
```

分布式日志分析的架构



数据处理子系统
Agent
策略管理子系统
Agent
因子管理子系统
Agent
交易决策子系统
Agent
交易执行子系统
Agent
Collecting Channel
Storage/Analysis
离线分析
实时分析
Report
Alert

总结

- 模拟撮合的逻辑和实现
- 策略和模拟盘的对接方式
- 缩小回测与实盘之间的差异的方式
- 实盘中日志分析

课后练习

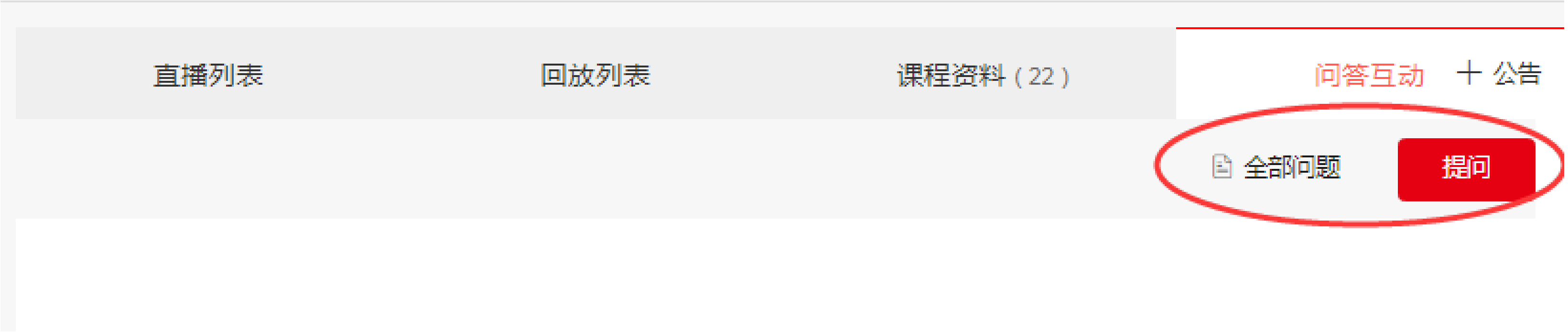
- 完善模拟盘：
  - 接入TuShare的Tick数据；（附代码）
  - 通过课上演示的方式发送委托单，观察运行结果（附结果图）

下节课预告

- 题目：技术型和基本面因子的编写
  - 编程语言和运行平台：Python、MongoDB

问答互动

在所报课的课程页面，
1、点击“全部问题”显示本课程所有学员提问的问题。
2、点击“提问”即可向该课程的老师和助教提问问题。



联系我们

小象学院：互联网新技术在线教育领航者

- 微信公众号：**小象学院**

THANKS