\*基本上底层要完美封装的比较花时间的工具

Jetty（目前只找到这么一款嵌入式http服务器）

Http client（以及想得到的能简化http请求的工具，包括json，xml解释器）

JDBC（数据库管理的通用组件）

线程封装和通信（只能想想，暂时不急）

\*分层方式

|  |  |
| --- | --- |
| 脚本层（能够解释简单投资逻辑，利用下几层对象，在限定权限基础上做投资决策）其实这东西比下一层好实现，而且可以和下层相互依赖 | |
| 应用层（投资分析逻辑，行情分析逻辑，监控实验室等，处于梦想阶段） | |
| 基础层（常时运行，多监听回调修改线程任务，或做相应单次处理，纯异步） | |
| 系统级逻辑层（主要是与行情有关的基础逻辑） | 用户级逻辑层（分用户的交易逻辑） |
| 底层（主要网络通信，线程，数据库，基础模型） | |

\*基础环境所用的模块

系统入口点（创建各工厂之后阻塞，监听主线程结束）

常时运行模块

监听http请求

投资工作室（做了个糖衣对象，上层开发的主入口点）

交易员（包括多种交易接口）

行情接口（包括多种请求方式）

\*脚本语言环境需要的注入内容（暂时完全同步吧，如果异步实在太难了）

常时运行设置接口

投资规则分析模型

自定义数据分析模型

主观的交易类对象

简单数据

\*和股票及用户交易行为有关的统计量模型

要想全很复杂，内容也很多，我以前写的飞马系统主要就针对期货，所以就简单写了一个对象，像这样

struct DepthMarketData

{

String name;

String time;

Double price;

Double volume;

…

}

那么问题来了，在管理不同品种时，这样特别冗长，主要问题是，数据有各种特质，就需要不同处理存储方式。

按品种分类，我这里想支持三种主要的，股票，期货，期权（这个不重要）

每个有各种的特色，股票分板块，还有各种经营内容，所以管理股票用的得是分层链表结构，便于消息收发，然后股票有财务报表信息需要存储（这个是静态的，也简单）；期货期权的品种就特别少，但是需要关注更多虚拟统计量，这个查一下数据类型就知道了。

按数据性质分的话，比如我之前的系统没有做很完善的数据处理机制，都是手工写代码算的

1. 分时平均价，应该很清楚，如果每次都调用均价指标，系统会爆炸，所以分时平均就可以用公式简化：ap(t)=(ap(t-1)\*(t-1)+p(t))/t好算很多
2. 最新价：lp(t)=lp(t-1)这个如果直接算好像也是挺好算
3. 最高最低价：mp(t) = max(p(t), mp(t-1))
4. 成交量：时段成交量uv(t)=tv(t)-tv(t-1)

我当时就主要遇到这个四种，因为只是为了完成k线和分时图才写的一些统计量，而且只针对期货，所以代码还不复杂，但在自定义投资策略中肯定不行。

所以在金融产品数据封装以前，先进行数据类型封装。之后所有的数据封装模型都要必须符合这一标准。

\*\*数据分类原则：

主要目的 实现对复杂统计量简化方法的封装，简化递推性模型处理的代码

用于描述的品种：

个股，期货，期权，实际指数

变量有关性：这个是我们能处理优化的公式

Inputs代表所有n时刻的量，前面的都是同类变量在之前时刻的值

1. 最简单的无关性
2. 时序相关性（变量只与同一类但不同时间的变量相关）
3. 横向相关性（变量只与同一时间但不同类变量相关）
4. 有限时移横向相关量（我也不知道自己在说什么，反正就是能够由2，3组合计算的量）
5. 复杂相关性（我不知道怎么做说真的）要么干脆继承一个类来实现

话说 股票数据有个问题，一种是瞬时数据模型，只与某时刻有关，比如实时价格，另一种是时段模型，只有在时间累积的情况下有意义比如成交量，均价什么的（这里的时段模型以分析周期为主，一般常用分析周期由1min,5min,1h,1d,5d,10d,30d,180d,365d），所以说不定两种不同的性质会影响数据的存储，不过除了效率问题上的，我暂时想不到逻辑上会发生什么问题

\*\*数据分类方法：

1. 无关量（如实时报价，不需要计算参数的量，但是与时间有关，主要是从外面直接抓取的为主），干脆称之为原始变量
2. 静态量（如股票代码，反映对象性质，完全不用变化）
3. 有限统计量（单变量的计算复杂度为O(1)）

所以这类的统计量可以用递推式或简单公式计算

在这个基础上可以进行进一步分类继承

时序相关量

初始化量（进行一次赋值永久继承）

更新量（每次采用inputs）

比较量（如果符合继承规则则继承，否则采用）

平均量，合计量（都是递推式可以解决的量）

关联分析量

比如期限差=期货价格-现货指数价格

\*暂时觉得两种基础量可以组合凑成所有大众经常用到的分析公式，如果有需要的话，可以继承进行计算。

1. 无限统计量（单变量的计算复杂度不为O(1)）

这类的统计量会极度影响数据效率，tm不想考虑

\*\*各自需要的接口方法

1. 无关量get/set/加一个时间引用
2. 静态量get/set好像和第一个差不多，我是觉得如果分开的话可能有性能提升的空间
3. 有限统计量

Get/set

Update(inputs)封装计算规则或替换规则（如果相同统计量的输入是一致的话，就可以用接口来批量处理了）

1. 无限统计量

纯calc计算方法

这里就产生了一个问题，到底这些方法当中可以access到多少变量呢，尽量越少越好，但是这样就造成了接口一致性的问题。这里有一个挺好的技术叫依赖注入，可是我完全不知道java自己写代码的话是如何实现的，所以暂时只能按多一点分类执行了。Java的lambda表达式功能还特别弱，暂时只考虑用runner对象，其他基本没法儿写，等java9吧。

\*底层的想法，主要的几块

数据存储支持

1. Mysql，redis数据底层封装
2. 自己设的数据缓存队列和统计模型
3. 支持注入的数据取出机制（需要考虑客户安全问题）
4. 好像还是首先要把需要存多少东西想清楚。。

主程序支持

线程管理工厂

由于线程数影响效率，封装runner进行线程合并

封装一个定时运行的线程类

资源管理

如ID管理什么的，慢慢完善就行了

网络支持

客户端模块

数据抓取，回调

http监听模块

\*程序基础层框架 主要以工厂模式管理各模块对象，

程序入口点

线程管理

网络监听管理

网络请求管理

数据库管理

行情数据管理

创建

创建

创建

创建

创建

投资管理

定时线程

和所有都有关系

创建/合并/销毁

注册回调

请求/查询

请求/查询

注册回调

创建

请求/查询

其实联系还有挺多的

\*几个基本的异步代码执行点为：

1. 线程管理模块的主方法阻塞监听请求回调（说不定可以加一个命令行）
2. 定时执行的线程的run（）
3. 网络监听到请求的异步回调
4. 网络请求返回数据的异步回调
5. 剩余部分都是同步的被执行代码

如果数据单元体量很大，可能还要考虑线程安全 这里先不管了