<SQuant>

软件构架文档

版本 <1.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2018.10.17 | 1.0 | 软件架构文档初稿 | 罗金宏，陈东仪 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 范围 4

1.3 定义、首字母缩写词和缩略语 4

1.4 参考资料 4

1.5 概述 4

2. 构架表示方式 4

3. 构架目标和约束 5

4. 用例视图 5

4.1 行情展示 5

4.2 交易下单 5

4.3 实时风控 5

4.4 算法交易 6

4.5 策略构建 6

5. 逻辑视图 6

5.1 概述 6

5.2 展示层 6

5.3 业务逻辑层 6

5.4 数据访问层 7

6. 进程视图 7

7. 部署视图 8

7.1 Dispatcher Server 8

7.2 Web Server 8

7.3 Database 8

8. 实施视图 9

9. 数据视图 10

10. 技术视图 10

11. 质量 10

11.1 易用性设计 10

11.2 可靠性设计 10

11.3 性能设计 11

11.4 可支持性设计 11

软件构架文档

# 简介

## 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

## 范围

本文档适用于高级软件开发课程第七小组正在开发的SQuant系统。本项目SQuant一站式量化交易平台，通过利用现有权威开源在线数据源，结合量化交易策略和机器学习方法，构建一个仿真证券交易决策系统。

## 定义、首字母缩写词和缩略语

专业术语：

1. 量化交易：量化交易是指以先进的数学模型替代人为的主观判断，利用计算机技术从庞大的历史数据中海选能带来超额收益的多种“大概率”事件以制定策略，极大地减少了投资者情绪波动的影响，避免在市场极度狂热或悲观的情况下作出非理性的投资决策。
2. 股票：股票是股份公司发行的所有权凭证，是股份公司为筹集资金而发行给各个股东作为持股凭证并借以取得股息和红利的一种有价证券。每股股票都代表股东对企业拥有一个基本单位的所有权。每家上市公司都会发行股票。
3. 期货： 与现货完全不同，现货是实实在在可以交易的货（商品），期货主要不是货，而是以某种大众产品如棉花、大豆、石油等及金融资产如股票、债券等为标的标准化可交易合约。因此，这个标的物可以是某种商品（例如黄金、原油、农产品），也可以是金融工具。

缩略词：

1. TWAP：时间加权平均价格，该算法的目标在于计算您的定单在提交之时至获得执行之间的时间加权平均价格。如果勾选允许在超过结束时间交易，执行时间结束后未或执行的定单会继续被执行。用户可以设定在达到特定提交时才交易定单。
2. VWAP：成交量加权平均价，是将多笔交易的价格按各自的成交量加权而算出的平均价，若是计算某一证券在某交易日的VWAP，将当日成交总值除以总成交量即可。VWAP可作为交易定价的一种方法，亦可作为衡量机构投资者或交易商的交易表现的尺度。

## 参考资料

1. SQuant系统软件需求规约，1.0版，高级软件开发课程第七小组
2. SQuant系统软件开发计划，1.0版，高级软件开发课程第七小组
3. RUP的软件架构文档模板。

## 概述

本文档余下部分按照以下结构组织：第二节介绍系统架构的表示方式；第三节介绍系统架构目标和约束；第四节到第十节分别展示系统的用例视图 、逻辑视图、进程视图、部署视图、数据视图、实施视图、技术视图；第十一节介绍系统的大小及性能。

# 构架表示方式

SQuant系统包括服务器端和Web端，Web端和服务器采用BS架构。

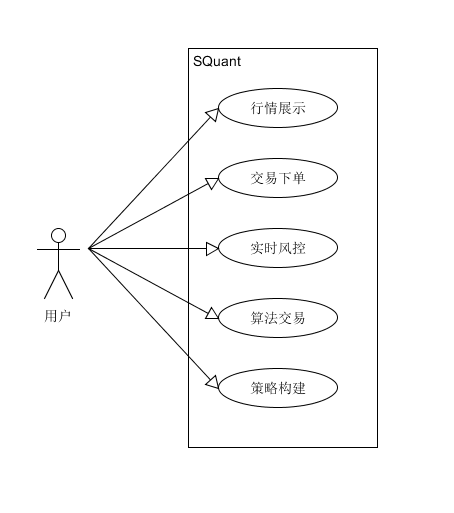
# 构架目标和约束

SQuant架构需要满足的非功能需求有：

1. 安全性：系统对外部公开的接口，除登录注册等接口外，需要进行安全验证，保证用户敏感信息的安全性。
2. 可用性：系统需要保证99.9%的可用率，错误恢复时间在2小时内。

特殊约束有：

1. 开发工具：服务器端使用PyCharm开发，Web端使用VS Code

****

# 用例视图

**图1 系统用例图**

## 行情展示

用户可以通过股票或者期货代码或名称查询相应产品的行情信息，需要提供实时行情查询、订阅及日内分钟线查询服务，历史行情数据查询服务和参考数据查询服务，并以合适的图形化方式进行展示，方便用户进行分析。

## 交易下单

通过委派下单的方式来尽可能接近实盘交易。

## 实时风控

通过对策略的实时监控，应对极端市场环境变化，运行止损策略，来保证收益。用户在界面上可以自主选择是否启用这一功能。

## 算法交易

算法交易作为系统的一个主要模块，可以为用户提供强大的算力支持，使用一些常见的交易算法如TWAP、VWAP、篮子算法等等，帮助用户完成委托下单。通过算法自动发单，尽可能减少交易操作对市场的影响，获得更好的交易价格。

## 策略构建

包括经典策略、自定义可视化策略构建、机器学习方法构建策略。

# 逻辑视图

## 概述

**图2 系统逻辑视图**

系统采用三层架构风格，分为展示层、业务逻辑层、数据访问层，表示层负责向用户展示界面，并将用户请求发送给业务逻辑层；业务逻辑层处理本系统的核心业务，主要分为五大功能模块，调用数据访问层提供的持久化操作查询金融数据、用户数据、策略详情等商业实体，并将处理结果反馈至展示层；数据访问层负责执行数据库持久性操作。

## 展示层

浏览器端为用户提供了可视化界面，是与用户交互的接口。负责系统的前端业务逻辑， 将用户请求转发给业务逻辑层，并显示经过业务逻辑层处理的业务。

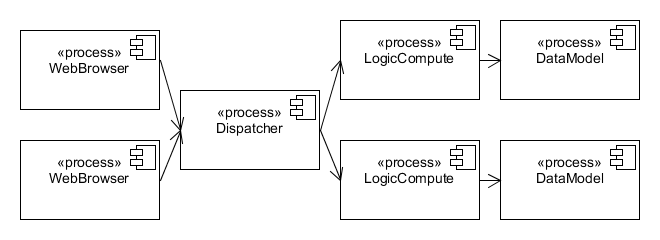
## 业务逻辑层

负责对用户的http请求作出响应，处理本系统的核心业务逻辑，转发用户请求，共有五大功能模块：行情展示服务、交易下单服务、实时风控服务、算法交易服务、策略构建服务，详情可见本文件第四节。

其中，行情展示模块依赖金融数据服务，包括对数据进行筛选过滤，第三方金融数据，类型包括实时行情数据，历史行情数据和相关参考数据等；策略构建模块可以基于机器学习模型，通过策略定制来自定义策略，并使用回测分析引擎进行性能分析；算法交易模块采用TWAP、VWAP算法、机器学习等模型，通过系统的发单服务，实现算法的启动；交易下单模块使用发单服务实现订单的委派；风控模块通过风控模型来实现。

## 数据访问层

是本系统的数据业务维护模块。数据库存储存储基础金融数据如期货股票代码、用户的基本信息、用户自定义策略等。本系统使用MySQL关系型数据库存储信息，通过Django框架提供的对象关系映射（OR映射）完成对数据库的访问，同时提供事务支持。MySQL数据库作为一个快速、多线程编程的数据库，不仅满足本系统访问速度的要求，而且具有强大的查询功能，可以在同一查询中混用来自不同数据库的表，查询方便快捷。



# 进程视图

**图3 系统进程视图**

1. Dispatcher：接受来自Web Browser的请求，将他们分发给适当的服务器上的LogicCompute进程进行处理。对为LogicCompute进程提供注册功能、发现功能。
2. LogicCompute：对Dispatcher分派的请求进行实际的处理，并返回请求结果。
3. DataModel：与数据库进行交互，提供数据库的读写操作，为LogicCompute进程提供数据支持。

# 部署视图

**图4 系统部署视图**

SQuant系统主要分为Dispatcher Server、Web Server、Database三部分。

## Dispatcher Server

Dispatcher Server负责完成Web Server的注册、发现功能，并将浏览器请求分发到相应的Web Server服务器进行处理，从而实现多个Web Server服务器的负载均衡功能。多用户同时访问同一个Web Server服务器，可能对该服务器造成一定的压力，因此使用服务注册发现的机制进行负载均衡，减小单个服务器的压力。同时，多个Web Server服务器也能提高系统服务的可靠性。

## Web Server

Web Server主要负责实际处理用户的请求，通过调用Web Server中的相应的逻辑代码或访问Database获取数据，从而完成用户的访问请求，将计算结果返回给浏览器。

## Database

Database主要对数据进行持久化存储，提供持久化数据的访问。Database主要存储一些用户信息、用户策略等，读数据比写数据操作更多，不容易造成数据库访问瓶颈。因此，这里仅使用单个数据库实例。若后续发现数据库读写操作称为系统主要瓶颈，使用读写分离的分布式数据库来实现。

# 实施视图

**图5 系统实施视图**

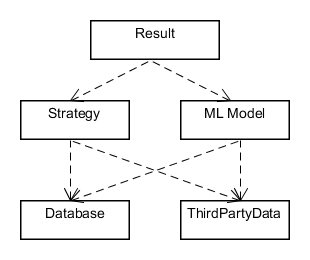
服务器端设计以实现各模块间的低耦合及模块内部的高内聚为导向，采用Django Web的MTV开发框架。服务器端分为Template模版层、View视图层以及Model模型层。Model层是最下面的托管数据的层级，它将系统中的各个角色都抽象成对象，并从数据库中抽取对应的数据；View层结合了一个Web应用程序中许多的逻辑，它们是链接到一个或多个定义URL上的函数，这些函数都返回一个HTTP对象，它扮演了常规MVC结构中Controller的角色，负责协调Models和Templates之间的关系；Template层则用于渲染HTML页面，它通常就是一些输出动态值的经过特殊格式化的HTML文本。Model层到数据库的映射有Django框架提供的对象关系映射（ORM）完成，除此之外，Django还提供了Template引擎，可以大大提高开发的速度。

客户端向服务端的请求采用Restful设计风格，力求访问的灵活性。

服务器端访问MySQL数据库以及第三方数据进行数据交互。

同时在系统层面上设计了安全保障模块、缓存模块和日志模块，为系统平台的安全和高效保驾护航。

# 数据视图



**图6 系统数据视图**

该图描述了数据在系统处理加工的过程。

原始数据：主要来自于存储用户信息和策略信息的Database和存储市场行情信息的Third Party DataCore这两部分。

业务逻辑层：经典Strategy和用户自定义的Strategy以及训练好的机器学习模型利用这些原始数据进行计算，并得出结果。

展示层：Result将得到的结果按易于用户理解的图形化方式展示给用户。

# 技术视图

本系统的技术选型如下：

1. 编程语言：Python、Java Script、HTML、CSS
2. 前端技术框架：VueJs
3. Web服务框架：Django（Python）
4. 数据库：MySQL
5. 机器学习编程平台：Tensorflow、Keras

# 质量

## 易用性设计

需求：用户在第一次使用即可掌握本产品大部分功能的使用。

设计策略：网站界面设计尽量符合用户使用习惯，页面设计多使用图标。

## 可靠性设计

需求：

* 最小故障间隔时间：不超过7天。
* 全年故障停运时间：不超过24小时。
* 故障平均修复时间：不超过30分钟。

设计策略：

* 使用三层架构，不同Tier之间实现充分解耦。一旦出现故障，可以快速定位并进行修复。
* 在开发阶段，进行充分的测试，例如单元测试、模块测试、集成测试、系统测试，尽可能保证系统可靠。
* 在开发过程中编写规范的文档，便于后续故障的定位恢复。

## 性能设计

需求：

* 网站最大支持并发数：100
* 单个操作的响应时间不应该超过5秒。

设计策略：

* 对高频访问数据进行缓存，减少数据访问时间。
* 对比较耗时的操作提前计算并存储结果，操作时直接读取数据，减小响应时间。

## 可支持性设计

需求：

* 前端代码根据当前主流前端代码规范进行编写。
* Python代码根据Python的标准风格规范进行编写。

设计策略：

编程时参照Google 编程规范，提交代码时，请组内其他同学进行Code Review。保证代码的可读性和规范性。