2017-4-23

Leftovers

详细设计文档

（迭代三）

目录**（按Ctrl+目录有链接）**

[1. 引言 3](#_Toc477600597)

[1.1编制目的 3](#_Toc477600598)

[1.2参考文献 3](#_Toc477600599)

[2. 逻辑视角 3](#_Toc477600600)

[1.系统结构的选取原因考虑 3](#_Toc477600601)

[2.系统的分层架构 3](#_Toc477600602)

[3.系统架构中的对象分类 4](#_Toc477600603)

[3.组合视角 5](#_Toc477600604)

[3.1开发包示意 5](#_Toc477600605)

[3.2物理部署 5](#_Toc477600606)

[4.接口视角 6](#_Toc477600607)

[4.1模块的职责 6](#_Toc477600608)

[4.2用户界面层的分解 6](#_Toc477600609)

[4.2.1.用户界面层模块的职责 7](#_Toc477600610)

[4.3.业务逻辑层的分解 7](#_Toc477600611)

[4.3.1业务逻辑层模块的职责 7](#_Toc477600612)

[4.3.2逻辑层的接口规范 7](#_Toc477600613)

[4.4.数据层的分解 9](#_Toc477600614)

[4.4.1数据层模块的职责 9](#_Toc477600615)

[5.信息视角 10](#_Toc477600616)

[1.VO PO定义的选择 10](#_Toc477600617)

[2.VO PO定义结果 11](#_Toc477600618)

**更新历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 作者 | 描述 | 日期 |
| v0.1 | 郭浩滨 | 确定基本模板 | 2017-3-4 |
| v0.2 | 全体人员 | 详细设计评审 | 2017-3-8 |
| V0.3 | 林志和 | 完善MarketBLService接口跟MarketDataService接口 | 2017-3-17 |
| V0.4 | 吴游杰 | 完善分层图+包图+界面跳转图 | 2017-3-17 |
| V0.5 | 郭浩滨 | 完善StockBLService接口跟StockDataService接口，VOPO视角等 | 2017-3-17 |
| V0.6 | 林志和 | 完善BackTestDataService跟UniverseDataService接口 | 2017-4-22 |
| V0.7 | 郭浩滨 | 完善BackTestBLService跟UniverseBLService接口 | 2017-4-23 |
| V0.8 | 吴游杰 | 界面跳转图 | 2017-4-23 |

# 引言

## 1.1编制目的

本报告详细完成对Quantour系统的概要设计，达到指导详细设计和开发的目的，同时实现测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户编写，是了解系统的导航。

## 1.2参考文献

1、IEEE标准

2、《需求规格说明文档》

3、《软件体系结构设计目标（IEEE标准）》

4、《软件工程与计算（卷三）》

# 逻辑视角

## 1.系统结构的选取原因考虑

由于系统是从Quantour网站通过网络传输协议获取数据，如果每次用户的请求都要与网站服务器交互，那么由于远程调用速度缓慢的原因系统反应的时间将比较长。所以我们计划将网络上的数据首先缓存到本机，根据服务器更新的频率刷新本机数据，用户访问数据的时候只与本机数据进行交互，提高系统效率。

## 2.系统的分层架构

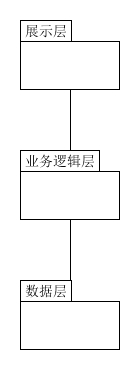
系统分为三个逻辑层次

1）展示层：用于向用户展示股票的详细信息

2）业务层：包含业务控制与逻辑处理的层次

3）数据层：用于与服务器端进行交互获取数据，并向业务层提供数据

图示如下：



## 3.系统架构中的对象分类

1）UI对象，负责处理系统数据的展现与用户的交互

2）Controller对象，控制用户的输入，调用BLService的服务

3）BLService对象，负责提供服务的抽象接口，获取从数据段封装好的PO，给界面提供需要的VO

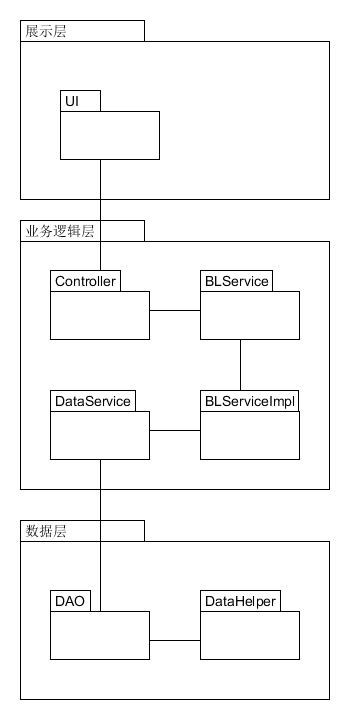
4）BLServiceImpl对象，负责实现BLService的接口

5）DataService对象，负责封装从DAO中获取的用户需要的指定信息

6）DAO对象，负责与网站数据库实体交互，获取数据

7）DataHelper对象，负责获取并处理数据

系统中的组件表示：



# 3.组合视角

## 3.1开发包示意

Quantour系统的最终开发包设计如表1所示

**表1** Quantour**系统的最终开发包设计**

|  |  |
| --- | --- |
| 开发（物理）包 | 依赖的其他开发包 |
| ui | controller |
| stockblservice | stockvo |
| marketblservice | marketvo |
| backtestblservice | backtest |
| stockdataservice | stockdatahelper |
| marketdataservice | marketdatahelper |
| backtestblservice | backtestdatahelper |
| stockdata | stockdataservice |
| marketdata | marketdataservice |
| backtestdata | backtestdataservice |
| util |  |

## 3.2物理部署

Quantour系统中客户端构件是放在客户端机上，服务器端构件是放在Quantour服务器端机器上。在客户端节点上，需要部署Java运行环境JRE。

# 4.接口视角

## 4.1模块的职责

客户端各层的职责分别如下表所示

**客户端各层的职责**

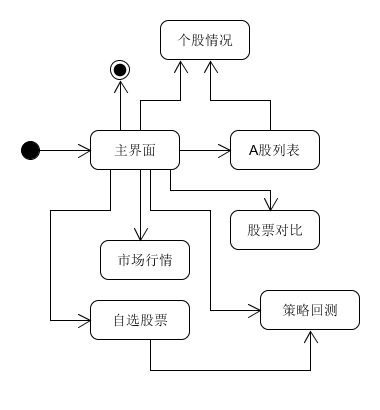
|  |  |
| --- | --- |
| 层 | 职责 |
| 用户界面层 | 基于窗口的进销存管理系统客户端用户界面 |
| 业务逻辑层 | 对于用户界面的输入进行响应并进行业务处理逻辑 |
| 数据层 | 从股票网站访问数据，持久化存入本地，提供指定数据给逻辑层使用 |

**层之间调用的接口**

|  |  |  |
| --- | --- | --- |
| 接口 | 服务调用方 | 服务提供方 |
| marketblservice  stockblservice | 客户端展示层 | 客户端业务逻辑层 |
| marketdataservice  stockdataservice | 客户端业务逻辑层 | 服务器端数据层 |

## 4.2用户界面层的分解

系统存在5个用户界面，他们之间的关系为：



### 4.2.1.用户界面层模块的职责

ComStockController主要用于负责界面的显示和界面之间的跳转工作。

Quantour包括以下几个类，负责各种股票信息图的展示功能。

StockInfoController负责读取股票一段时间内每天的股价，并以K线图的形式展现在界面上。

StockMarketController负责读取股票一段时间内的交易量信息，并以柱状图的形式展现在 界面上。

StockListController负责以表格的形式展示股票的信息。

StockCompareGraphController负责比较两只股票，并以折线图的形式显示在界面上

SelfSelectStockController负责自选股票池，并以卡片式显示在界面上。

StockStrategyController负责策略回测。

## 4.3.业务逻辑层的分解

### 4.3.1业务逻辑层模块的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| marketbl | 负责实现用户查询市场信息的功能 |
| stockbl | 负责实现用户查询股票信息的功能 |
| backtestbl | 负责实现用户进行策略回测的功能 |

### 4.3.2逻辑层的接口规范

marketbl**包的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| MarketBLService.getAllStocks() | 语法 | Public java.util.List<DefaultStockVO> getAllStocks() |
| 前置条件 | 系统启动市场模块 |
| 后置条件 | 系统返回市场中的全部股票 |
| MarketBLService.getMarketInfo(java.time.LocalDate date) | 语法 | Public MarketVO getMarketInfo(java.time.LocalDate date) |
| 前置条件 | 系统启动市场模块 |
| 后置条件 | 系统返回某天的市场行情信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| MarketDataService. findLatestMIsOfAllStocks() | 获取所有股票的最近一个交易日的市场条目 | |
| MarketDataService.findMarketItemsByTime(java.time.LocalDate date) | 获得某天的市场条目列表 | |

stockbl**包的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| stockbl. findStockByName(String name, LocalDate beginTime, LocalDate endTime) | 语法 | public StockVO findStockByName(String name, LocalDate beginTime, LocalDate endTime) throws StockNotFoundException, TimeException; |
| 前置条件 | 已输入格式正确的起止时间与股票名称 |
| 后置条件 | 系统返回相应股票一段时间内的股票信息 |
| stockbl. findStockByCode(String code, LocalDate beginTime, LocalDate endTime) | 语法 | public StockVO findStockByCode(String code, LocalDate beginTime, LocalDate endTime) throws StockNotFoundException, TimeException; |
| 前置条件 | 已输入格式正确的起止时间与股票代码 |
| 后置条件 | 系统返回相应股票一段时间内的股票信息 |
| Stockbl. findSuggestions(String query) | 语法 | public List<String> findSuggestions(String query) throws StockNotFoundException; |
| 前置条件 | 用户输入查询词 |
| 后置条件 | 系统返回相应的股票名称建议 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| stockdataservice. findStockPOByCode(String code) | 系统返回相应代码的股票信息列表 | |
| stockdataservice. findCodeByName(String name) | 系统返回相应名称的股票代码 | |
| stockdataservice. findCodeName() | 系统返回所有的股票代码和名称的映射关系 | |

backtestbl**包的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| backtestbl. runMomentum(LocalDate beginTime, LocalDate endTime, Benchmark benchmark, String[] universeNames, double capitalBase, int formRate, int refreshRate) | 语法 | public void runMomentum(LocalDate beginTime, LocalDate endTime, Benchmark benchmark, String[] universeNames, double capitalBase, int formRate, int refreshRate) throws StockNotFoundException; |
| 前置条件 | 已输入格式正确的回测条件 |
| 后置条件 | 系统进行动量策略回测 |
| backtestbl. runMeanRevision(LocalDate beginTime, LocalDate endTime, Benchmark benchmark, String[] universeNames, double capitalBase, int pma, int refreshRate) | 语法 | public void runMeanRevision(LocalDate beginTime, LocalDate endTime, Benchmark benchmark, String[] universeNames, double capitalBase, int pma, int refreshRate) throws StockNotFoundException; |
| 前置条件 | 已输入格式正确的回测条件 |
| 后置条件 | 系统进行均值回归回测 |
| backtestbl. getBacktestResultOverview() | 语法 | public BacktestResultOverviewVO getBacktestResultOverview(); |
| 前置条件 | 用户已经进行策略回测 |
| 后置条件 | 获取回测总体结果 |
| backtestbl. getChangingWP(RateType rateType, int rate) | 语法 | public List<ChangingRateVO> getChangingWP(RateType rateType, int rate); |
| 前置条件 | 用户已经进行策略回测 |
| 后置条件 | 对于不同的形成期/持有期（均值回归只有形成期pma），获得超额收益率与年胜率与另一个变量的关系 |
| backtest. getReturns() | 语法 | public ReturnDistributionVO getReturns(); |
| 前置条件 | 用户已经进行策略回测 |
| 后置条件 | 获得每个周期的收益率 |
| backtest. getPositionRecord() | 语法 | public List<PositionRecordVO> getPositionRecord(); |
| 前置条件 | 用户已经进行策略回测 |
| 后置条件 | 获得持仓记录 |
| backtest. createNewUniverse(String universeName, List<String> universe) | 语法 | public void createNewUniverse(String universeName, List<String> universe) throws DuplicateName; |
| 前置条件 | 无 |
| 后置条件 | 创建新的股票池 |
| backtest. deleteUniverse(String universeName) | 语法 | public void deleteUniverse(String universeName); |
| 前置条件 | 用户已经进行创建股票池 |
| 后置条件 | 删除股票池 |
| backtest. getAllUniverses() | 语法 | public List<UniverseVO> getAllUniverses(); |
| 前置条件 | 无 |
| 后置条件 | 获取所有自建的股票池 |
| backtest. updateUniverse(String universeName, List<String> universe) | 语法 | public void updateUniverse(String universeName, List<String> universe); |
| 前置条件 | 用户已经创建股票池 |
| 后置条件 | 更新股票池 |
| backtest. getStockAttributes(String code) | 语法 | public StockAttributesVO getStockAttributes(String code) throws StockNotFoundException; |
| 前置条件 | 用户提供了有效的股票代码 |
| 后置条件 | 获取股票的属性 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| backtestdataservice.getStockLastPriceFromMemory(String code, LocalDate date) | 根据股票代码和日期返回股票距离日期最后一次的价格 | |
| backtestdataservice.getStockPrice(String code, LocalDate date, int n) | 根据股票代码和日期返回股票最近n天（包括当天）的价格 | |
| backtestdataservice.getListStockPrice(List<String> codes, LocalDate date, int n) | 系统根据股票代码列表和日期返回股票列表最近n天（包括当天）的价格 | |
| backtestdataservice.getBoardStockPrice(Board board, LocalDate date, int n) | 根据板块和日期返回该板块所有股票最近n天（包括当天）的价格 | |
| backtestdataservice.getSectionStockPrice(String section, LocalDate date, int n) | 根据行业和日期返回该行业所有股票最近n天（包括当天）的价格 | |
| backtestdataservice.getStockName(String code) | 根据股票代码返回股票名称 | |
| backtestdataservice.getTradeDays(LocalDate beginTime, LocalDate endTime) | 根据起始日期和结束日期返回中间的交易日列表（正序） | |
| backtestdataservice.getSections() | 返回所有的行业 | |
| backtestdataservice.getStockCodesByBoard(Board board) | 根据板块返回板块内的股票代码列表 | |
| backtestdataservice.getPricesByBenchmark(Benchmark benchmark, LocalDate beginTime, LocalDate endTime) | 根据基准返回一段时间（交易日）内基准的价格指数 | |
| backtestdataservice.getAveragePricesByCodesList(List<String> codes, LocalDate beginTime, LocalDate endTime) | 根据股票代码列表返回一段时间（交易日）内它们的平均价格 | |
| backtestdataservice.getAllStockCodes() | 返回所有股票代码 | |
| backtestdataservice.getStcokCodesBySection(String section) | 据行业返回该行业的股票代码列表 | |
| backtestdataservice.getStockCodesByUniverseName(String universe) | 根据自定义的股票池返回股票池内的股票代码 | |
| backtestdataservice.getSectionByStockCode(String code) | 根据股票代码返回股票行业 | |
| universedataservice.addNewUniverse(String universeName, List<String> universe) | 创建新的股票池 | |
| universedataservice.delUniverse(String universeName) | 删除对应的股票池 | |
| universedataservice.findAllUniverses() | 返回所有自建的股票池 | |
| universedataservice.updateUniverse(String universeName, List<String> universe) | 更新某个股票池 | |

## 4.4.数据层的分解

数据层主要给业务逻辑层提供数据访问服务，主要包括对于服务器端数据的获取，对本地用户信息的查询。

### 4.4.1数据层模块的职责

数据层模块职责如下表所示

**数据层模块职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| marketdataservice | 提供对用户查询市场信息的服务 |
| Stockdataservice | 提供对用户查询、比较股票信息的服务 |
| backtestdataservice | 提供回测所需要的数据 |

**marketdataservice包的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| MarketDataService. findLatestMIsOfAllStocks() | 语法 | public List<MarketItemPO> findLatestMIsOfAllStocks() |
| 前置条件 | 无 |
| 后置条件 | 系统所有股票的最近一个交易日的市场条目 |
| MarketDataService.findMarketItems  ByTime(java.time.LocalDate date) | 语法 | public List<MarketItemPO> findMarketItemsByTime(LocalDate date) |
| 前置条件 | 输入正确的日期 |
| 后置条件 | 系统某天的市场条目列表 |

**stockdataservice包的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| stockdataservice. findStockPOByCode(String code) | 语法 | public StockPO findStockPOByCode(String code) throws StockNotFoundException; |
| 前置条件 | 无 |
| 后置条件 | 系统返回相应代码的股票信息 |
| stockdataservice. findCodeByName(String name) | 语法 | public String findCodeByName(String name) throws StockNotFoundException; |
| 前置条件 | 无 |
| 后置条件 | 系统返回相应名称的股票代码 |
| stockdataservice. findCodeName() | 语法 | public Map<String, String> findCodeName() throws StockNotFoundException; |
| 前置条件 | 无 |
| 后置条件 | 系统返回所有股票代码-名称的映射关系 |

**backtestdataservice包的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| backtestdataservice.getStockLastPriceFromMemory(String code, LocalDate date) | 语法 | public double getStockLastPriceFromMemory(String code, LocalDate date) throws StockItemNotFoundException |
| 前置条件 | 股票代码存在 |
| 后置条件 | 系统根据股票代码和日期返回股票距离日期最后一次的价格 |
| backtestdataservice.getStockPrice(String code, LocalDate date, int n) | 语法 | public double[] getStockPrice(String code, LocalDate date, int n) throws StockItemNotFoundException |
| 前置条件 | 股票代码存在，股票没有停牌 |
| 后置条件 | 系统根据股票代码和日期返回股票最近n天（包括当天）的价格 |
| backtestdataservice.getListStockPrice(List<String> codes, LocalDate date, int n) | 语法 | public Map<String, Double[]> getListStockPrice(List<String> codes, LocalDate date, int n) |
| 前置条件 | 股票代码存在，股票没有停牌 |
| 后置条件 | 系统根据股票代码列表和日期返回股票列表最近n天（包括当天）的价格 |
| backtestdataservice.getBoardStockPrice(Board board, LocalDate date, int n) | 语法 | public Map<String, Double[]> getBoardStockPrice(Board board, LocalDate date, int n) |
| 前置条件 | 板块存在，板块内股票没有停牌 |
| 后置条件 | 系统根据板块和日期返回该板块所有股票最近n天（包括当天）的价格 |
| backtestdataservice.getSectionStockPrice(String section, LocalDate date, int n) | 语法 | public Map<String, Double[]> getSectionStockPrice(String section, LocalDate date, int n) |
| 前置条件 | 行业存在，行业内没有停牌 |
| 后置条件 | 系统根据行业和日期返回该行业所有股票最近n天（包括当天）的价格 |
| backtestdataservice.getStockName(String code) | 语法 | public String getStockName(String code) |
| 前置条件 | 股票代码存在 |
| 后置条件 | 系统根据股票代码返回股票名称 |
| backtestdataservice.getTradeDays(LocalDate beginTime, LocalDate endTime) | 语法 | public List<LocalDate> getTradeDays(LocalDate beginTime, LocalDate endTime) |
| 前置条件 | 无 |
| 后置条件 | 系统根据起始日期和结束日期返回中间的交易日列表（正序） |
| backtestdataservice.getSections() | 语法 | public List<String> getSections() |
| 前置条件 | 无 |
| 后置条件 | 系统返回所有的行业 |
| backtestdataservice.getStockCodesByBoard(Board board) | 语法 | public List<String> getStockCodesByBoard(Board board) |
| 前置条件 | 板块存在 |
| 后置条件 | 系统根据板块返回板块内的股票代码列表 |
| backtestdataservice.getPricesByBenchmark(Benchmark benchmark, LocalDate beginTime, LocalDate endTime) | 语法 | public TreeMap<LocalDate, Double> getPricesByBenchmark(Benchmark benchmark, LocalDate beginTime, LocalDate endTime) |
| 前置条件 | 基准存在 |
| 后置条件 | 系统根据基准返回一段时间（交易日）内基准的价格指数 |
| backtestdataservice.getAveragePricesByCodesList(List<String> codes, LocalDate beginTime, LocalDate endTime) | 语法 | public TreeMap<LocalDate, Double> getAveragePricesByCodeList(List<String> codes, LocalDate beginTime, LocalDate endTime) |
| 前置条件 | 无 |
| 后置条件 | 系统根据股票代码列表返回一段时间（交易日）内它们的平均价格 |
| backtestdataservice.getAllStockCodes() | 语法 | public List<String> getAllStockCodes() |
| 前置条件 | 无 |
| 后置条件 | 系统返回所有股票代码 |
| backtestdataservice.getStcokCodesBySection(String section) | 语法 | public List<String> getStockCodesBySection(String section) |
| 前置条件 | 行业存在 |
| 后置条件 | 系统根据行业返回该行业的股票代码列表 |
| backtestdataservice.getStockCodesByUniverseName(String universe) | 语法 | public List<String> getStockCodesByUniverseName(String universe) |
| 前置条件 | 股票池存在 |
| 后置条件 | 系统根据自定义的股票池返回股票池内的股票代码 |
| backtestdataservice.getSectionByStockCode(String code) | 语法 | public String getSectionByStockCode(String code) |
| 前置条件 | 股票代码存在 |
| 后置条件 | 系统根据股票代码返回股票行业 |
| universedataservice.addNewUniverse(String universeName, List<String> universe) | 语法 | public void addNewUniverse(String universeName, List<String> universe) throws DuplicateName |
| 前置条件 | 股票池名称不与已有的重复 |
| 后置条件 | 系统创建新的股票池 |
| universedataservice.delUniverse(String universeName) | 语法 | public void delUniverse(String universeName) |
| 前置条件 | 股票池名称存在 |
| 后置条件 | 系统删除对应的股票池 |
| universedataservice.findAllUniverses() | 语法 | public List<UniversePO> findAllUniverses() |
| 前置条件 | 无 |
| 后置条件 | 系统返回所有自建的股票池 |
| universedataservice.updateUniverse(String universeName, List<String> universe) | 语法 | public void updateUniverse(String universeName, List<String> universe) |
| 前置条件 | 股票池名称存在 |
| 后置条件 | 系统更新某个股票池 |

# 5.信息视角

## 1.VO PO定义的选择

系统的PO、VO类就是对应的相关的实体类，PO是数据层与逻辑层交互的实体，VO是逻辑层与界面交互的实体。

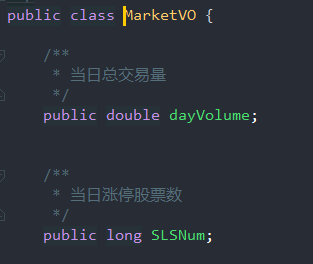
## 2.VO PO定义结果

由上我们定义的VO PO结果如下：

PO就是相关的实体类

VO是封装用于在界面上显示用户指定字段信息的数据。具体为：

1、



2、

