­

南京大学软件学院

AnyQuant项目

**迭代二**

**设计文档**

王梦麟 潘凌**伟**

王嘉琛 谭昕玥

目录

[1. 引言 4](#_Toc448572006)

[1.1编制目的 4](#_Toc448572007)

[1.2参考文献 4](#_Toc448572008)

[2. 逻辑视角 4](#_Toc448572009)

[1.系统结构的选取原因考虑 4](#_Toc448572010)

[2.系统的分层架构 4](#_Toc448572011)

[3.系统架构中的对象分类 5](#_Toc448572012)

[4.从结构设计到开发包的转换选择 6](#_Toc448572013)

[3.组合视角 6](#_Toc448572014)

[3.1开发包示意 6](#_Toc448572015)

[3.2物理部署 7](#_Toc448572016)

[4.接口视角 8](#_Toc448572017)

[4.1模块的职责 8](#_Toc448572018)

[4.2用户界面层的分解 8](#_Toc448572019)

[4.2.1.用户界面层模块的职责 9](#_Toc448572020)

[4.3.业务逻辑层的分解 10](#_Toc448572021)

[4.3.1业务逻辑层模块的职责 10](#_Toc448572022)

[4.3.2 逻辑层接口定义的思考过程 10](#_Toc448572023)

[4.3.3逻辑层的接口规范 12](#_Toc448572024)

[4.4.数据层的分解 19](#_Toc448572025)

[4.4.1数据层模块的职责 19](#_Toc448572026)

[4.4.2 数据层模块的接口规范 19](#_Toc448572027)

[5.信息视角 20](#_Toc448572028)

[5.1 VO PO定义的选择过程 20](#_Toc448572029)

[5.2 VO PO定义结果 21](#_Toc448572030)

更新历史

|  |  |  |
| --- | --- | --- |
| 姓名 | 时间 | 更新细节 |
| 谭昕玥，王梦麟 | 2016.3.2 | 创建文档 |
| 谭昕玥，王梦麟 | 2016.3.4 | 完善文档内容，添加目录 |
| 谭昕玥，王梦麟 | 2016.3.4 | 完善评审后的文档 |
| 王梦麟 | 2016.3.6 | 增加界面跳转图与VO定义 |
|  |  |  |
| 谭昕玥 | 2016.3.25 | 增加迭代二设计 |
| 王梦麟 | 2016.3.25 | 补充迭代二图表相关功能点的接口定义 |
| 王梦麟 | 2016.3.30 | 合并原本被覆盖的设计文档 |
| 谭昕玥 | 2016.4.9 | 单股分析模块接口设计变动 |
| 王梦麟 | 2016.4.14 | 修改逻辑层的接口规范 |
| 谭昕玥，王嘉琛，王梦麟 | 2016.4.15 | 修改包的定义结果并添加接口定义的思考过程 |

# 引言

## 1.1编制目的

本报告详细完成对AnyQuant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AnyQuant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

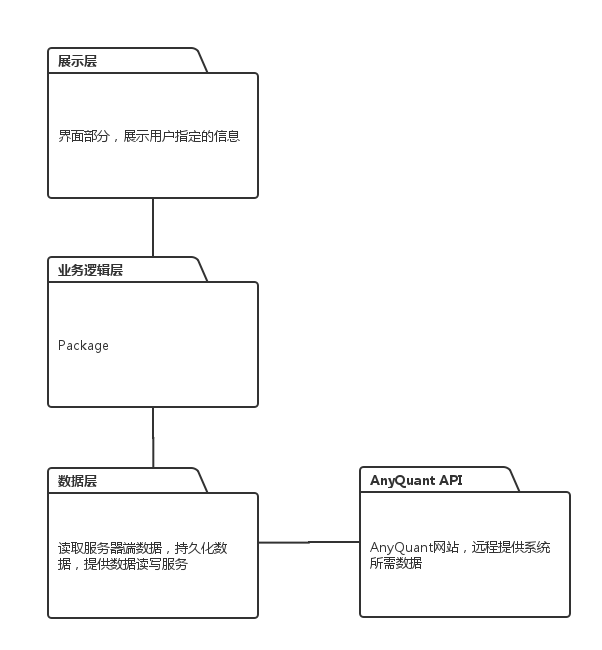
系统分为三个逻辑层次

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

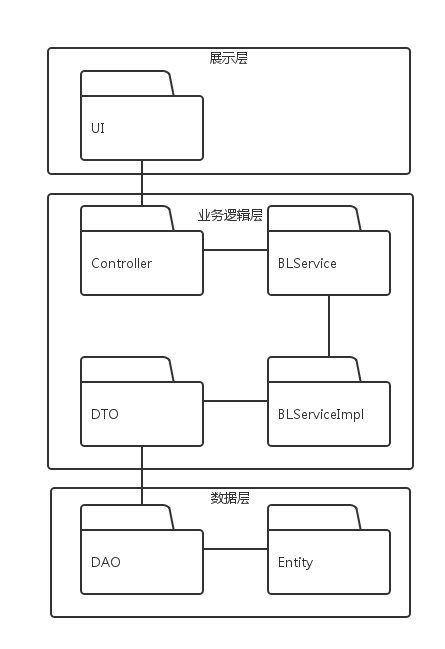
4）BLServiceImpl对象，负责实现BLService的接口

5）DTO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）Entity对象，从网站数据库获取的数据实体

系统中的组件表示：



## 4.从结构设计到开发包的转换选择

系统架构中的Entity对象是存储在AnyQuant开源网站上的，由于我们计划将Entity数据持久化保存到本机上并持续更新，DAO对象在大多数情况下也就可以只与本地数据交互。

DAO对象分裂为DataService与Data包，分别负责提供数据层的接口给逻辑层使用和实现数据层功能。DataService负责读取网站上的数据，把数据持久化保存至本机，并提供给逻辑层使用。

DTO对象的责任应该是封装从DAO也就是数据层中获取的批量数据的接口，我们这里将其封装为po对象，提供给逻辑层与数据层供共同使用。

由于网站上的JSON对象有一套简单明了的解析方式，我们直接把JSON数据的字符串返回给逻辑层，让逻辑层自行处理。

UI与逻辑层之间的数据传递使用VO对象封装。VO仅包含用户所需要的信息。

# 3.组合视角

## 3.1开发包示意

AnyQuant系统的最终开发包设计如表1所示

|  |  |
| --- | --- |
| 开发（物理）包 | 依赖的其他开发包 |
| chartui | Blfactoryservice, stockviewblservice, graphblservice, statisticblservice, vo,tool |
| benchui | blfactoryservice ,stockviewblservice, vo,tool |
| chartsmanageui | blfactoryservice ,statisticblservice，stockviewblservice，graphblservice, vo,tool |
| dialogui | tool |
| industryanalysisui | blfactoryservice, formatcheck, vo,tool |
| navigationui | / |
| statisticui | blfactoryservice ,statisticblservice, formatcheck, vo,tool |
| stockmanageui | blfactoryservice ,stockmanageblservice, formatcheck, vo,tool |
| stockviewui | blfactoryservice ,stockviewblservice, formatcheck, vo,tool |
| blfactoryservice | / |
| formatcheck | po |
| graphblservice | po |
| statisticblservice | po |
| stockmanageblservice | po |
| stockviewblservice | po |
| blfactory | graphblservice, statisticblservice, stockmanageblservice, stockviewblservice |
| graphbl | stockviewblservice,tool,po |
| statisticbl | stockviewblservice,tool,po |
| stockmanagebl | dataservice,tool,po |
| stockviewbl | dataservice,tool,po |
| dataservice | AnyQuant提供的API,等 |
| data | po |
| po | logindataservice |
| vo | stockdataservice |
| tool |  |

**表1 AnyQuant系统的最终开发包设计**

## 3.2物理部署

AnyQuant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AnyQuant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**

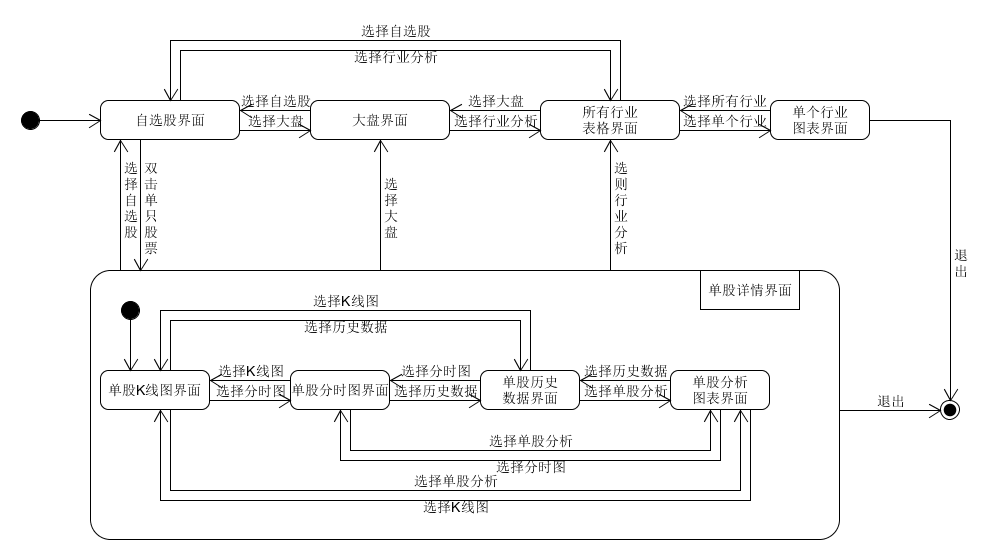
|  |  |
| --- | --- |
| 层 | 职责 |
| 启动模块 | 负责初始化网络通信机制，启动用户界面 |
| 用户界面层 | 基于窗口的进销存管理系统客户端用户界面 |
| 业务逻辑层 | 对于用户界面的输入进行响应并进行业务处理逻辑 |
| 数据层 | 从AnyQuant网站访问数据，持久化存入本地，提供指定数据给逻辑层使用 |

**层之间调用的接口**

|  |  |  |
| --- | --- | --- |
| 接口 | 服务调用方 | 服务提供方 |
| loginblservice  stockmanageblservice  stockviewblservice  statisticblservice  graphblesrvice  formatcheck  factory | 客户端展示层 | 客户端业务逻辑层 |
| logindataservice  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8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

|  |  |
| --- | --- |
| 模块 | 职责 |
| Mainui | 主要用于负责界面的显示和界面之间的跳转工作。 |
| analyseui | 负责各种股票信息图的展示功能 |
| linearchart | 负责读取股票分时交易信息，并以折线图的形式展现在界面上。 |
| klinear | 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。 |
| barchart | 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。 |
| chartsmanageui | 负责管理几种股票展示的图表。 |
| stockmanageui | 负责展示用户管理个人收藏界面。 |
| stockviewui | 负责以表格的形式展示股票的信息。 |

## 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 |
| stockviewbl | 负责实现查看股票和大盘信息的功能 |
| factory | 负责以工厂的形式向界面提供逻辑层的服务 |
| formatcheck | 负责检查用户输入的格式 |
| graphbl | 负责提供图表展示界面需要的服务 |
| statisticbl | 负责提供单只股票与行业分析数据 |

### 4.3.2 逻辑层接口定义的思考过程

#### 4.3.2.1 loginblservice包接口定义思考

最初考虑到迭代一的管理用户收藏股票的功能点，才决定要实现用户登录与帐号信息管理的功能。在迭代一后期报告时被告知此功能无需实现，现已废弃不用。

#### 4.3.2.2 stockviewblservice包接口定义思考

主要根据数据层提供的数据来定义接口，到现在的变化不大。

#### 4.3.2.3 factory包接口定义思考

在迭代一实现逻辑层后，发现界面在股票详细信息查看的界面会卡顿，意识到由于每次界面跳转后都会new一个新的逻辑层实现，导致逻辑层每次都要重新加载大量信息到界面。故添加逻辑层工厂，在一个逻辑层的实现被创造后无需再次重新新建，减少界面的卡顿现象。

#### 4.3.2.4 formatcheck包接口定义思考

处于防御式编程与系统健壮性的考虑，无论界面层的参数是怎样的获得的，在调用逻辑层的接口时，逻辑层需要对界面的输入进行格式与有效性检查。

#### 4.3.2.5 graphblservice包接口定义思考

##### （1）LinearChartBLService定义过程：

这个逻辑层负责实现展示最近一天一只股票的分时价格图，输入参数就是一直股票的编号，返回的是PriceInfoVO，包含了这一天内所有交易的时间与价格的变化情况。

但是后期意识到两个问题：

一是折线图功能过于单一，故又增加了选择日期查看价格折线图，和显示对应股票当天交易量柱状图的功能，丰富了接口入口参数。

二是希望折线图可以复用，考虑到折线图数据的种类之间的差异在于数据有几种，也就是折线有几个，故将折线图数据包VO封装成SeriesLinearChartVO，代表有几列数据的折线图VO。这里的股价折线图有两列数据，分别是即时价格与累计均价，故接口的返回参数定义为TwoSeriesLinearChartVO。

**（2）BarChartBLService定义过程：**

前期只考虑了成交量这一种柱状图，所以接口定义的不是很好，可扩展性也不是很强。某天开会后结合不同用例的需要，重新设计了接口，坐到了很好的代码复用和扩展。

在整理代码的时候，发现逻辑不是很清晰，于是做成了策略模式。

**（3）PieChartBLService定义过程：**

饼图唯出现在个股分析中，所以，接口约束条件比较足，定义的完善，不再需要修改。

**（4）KLinearChartBLService接口定义思考**

这个类是展示某一时间段某只股票对应K线图的逻辑实现，输入参数为股票编号和时间范围，返回一个KLinearChart。考虑到在数据获取上可能出现时间的错误输入，因此在检查到不符合规范的时间时会抛出对应的异常

#### 4.3.2.6 statisticsblservice包接口定义思考

#### **（1）IndustryViewService定义过程：**

对股票按照行业进行统计分析，参考了一些商业股票应用网站，主要思考出两个层次：

一是对于各个行业进行行业间对比分析，计算行业的统计信息，展现各个行业的基本情况。

二是在一个行业内部进行统计分析，统计行业内价格、交易量、与大盘对比涨跌幅等信息，给出这个行业的统计分析结果。

故接口分为这两种定义。

**（2）SingleViewBLService定义过程：**

最开始考虑，因为是做单股分析的BL，所以从界面的元素考虑，接口的定义重点在于考虑全面。接口设计的问题不大，这个类的修改主要是来自于功能点的增加。

**（3）SingleStatisticBLService定义过程：**

作为计算单股指标的逻辑层接口，设计的时候在于，防御式编程，怎么和界面做好交互，处理各种潜在异常。

### 4.3.3逻辑层的接口规范

#### 4.3.3.1 stockviewbl包的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| stockviewbl.getAllStock(String year, String exchange) | 语法 | public List<StockVO> getAllStock(String year, String exchange) |
| 前置条件 | 已输入格式正确的年份与交易所代码 |
| 后置条件 | 系统返回指定年份和交易所的全部股票信息列表 |
| stockviewbl.getStock(String name, String start, String end, List<String> fields) | 语法 | public StockVO getStock(String name, String start, String end, String fields) |
| 前置条件 | 已输入格式正确的股票代码，起止时间和指定的数据字段 |
| 后置条件 | 系统返回指定股票代码的相应股票 |
| Stockviewbl.getAllBenchmark() | 语法 | public List<BenchmarkVO> getAllBenchmark() |
| 前置条件 | 无 |
| 后置条件 | 系统返回获取所有可用的大盘指数 |
| stockviewbl.getBenchmark(String bench, String start, String end, String fields) | 语法 | public BenchmarkVO getBenchmark(String bench, String start, String end, String fields) |
| 前置条件 | 已输入格式正确的大盘代码，起止时间和指定的数据字段 |
| 后置条件 | 系统返回指定时间内的大盘指数 |
| stockviewbl.getAllFields() | 语法 | public List<String> getAllFields() |
| 前置条件 | 无 |
| 后置条件 | 系统返回股票的可用交易数据字段 |
| stockviewbl . getNamesAndNumbers() | 语法 | public HashMap<String,HashMap<String,String>> getNamesAndNumbers() |
| 前置条件 | 无 |
| 后置条件 | 系统返回股票的可用交易数据字段 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| stockdataservice.getAllStock(String year, String exchange) | 返回指定年份和交易所的所有股票列表 | |
| stockdataservice.getStock(String name, String start, String end, String fields) | 返回指定股票代码的相应股票交易数据 | |
| stockdataservice.getAllBenchmark() | 系统返回获取所有可用的大盘指数名称 | |
| stockdataservice.getBenchmark(String bench, String start, String end, String fields) | 系统返回指定的大盘指数 | |
| stockdataservice.getAllFields() | 系统返回股票的可用交易数据字段 | |

#### 4.3.3.2 factory包的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| BLFactoryService.getStockViewService | 语法 | StockViewService getStockViewService() |
| 前置条件 | 无 |
| 后置条件 | 返回 StockViewService的实现 |
| BLFactoryService.getLinearChartBLService | 语法 | LinearChartBLService getLinearChartBLService() |
| 前置条件 | 无 |
| 后置条件 | 返回 LinearChartBLService的实现 |
| BLFactoryService.getBarChartBLService | 语法 | BarChartBLService getBarChartBLService() |
| 前置条件 | 无 |
| 后置条件 | 返回 BarChartBLService的实现 |
| BLFactoryService. getSingleViewService | 语法 | public SingleViewService getSingleViewService() |
| 前置条件 | 无 |
| 后置条件 | 返回 BarChartBLService的实现 |
| BLFactoryService. getBarChartBLService | 语法 | public BarChartBLService getBarChartBLService() |
| 前置条件 | 无 |
| 后置条件 | 返回 BarChartBLService的实现 |
| BLFactoryService. getSingleStatisticBLService | 语法 | SingleStatisticBLService getSingleStatisticBLService() |
| 前置条件 | 无 |
| 后置条件 | 返回 BarChartBLService的实现 |
| BLFactoryService. getPieChartBLService | 语法 | PieChartBLService getPieChartBLService() |
| 前置条件 | 无 |
| 后置条件 | 返回 BarChartBLService的实现 |

|  |
| --- |
| 需要的服务（需接口） 暂无 |

#### 4.3.3.4 formatcheck包的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| formatcheck. isDate | 语法 | public static void isDate(String str) throws BadDateException |
| 前置条件 | 无 |
| 后置条件 | 返回检查输入是否是合法的日期的结果 |
| formatcheck. isNumber | 语法 | public static void isNumber(String from) throws BadNumberException |
| 前置条件 | 无 |
| 后置条件 | 返回检查界面输入的字符串是数字的结果 |
| formatcheck. isDateBefore | 语法 | public static void isDateBefore(String startDate, String endDate) throws BadDateException |
| 前置条件 | 无 |
| 后置条件 | 返回检查检查起点日期是否在终点日期之前的结果 |
| 需要的服务（需接口）暂无 | | |

#### 4.3.3.5 graphbl包的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| LinearChartBLService. getTradeInfoVO | 语法 | TwoSeriesLinearChartVO getLineChartVO(String number, String date) throws NotFoundException; |
| 前置条件 | 输入格式正确的股票代码与存在数据的日期 |
| 后置条件 | 返回这只股票最近一个交易日的分时股价VO |
| PieChartBLService. getPieVolumnVO | 语法 | PieChartVO getPieVolumnVO(String name, String number, String start, String end) throws NotFoundException; |
| 前置条件 | 输入格式正确的股票代码与存在数据的起止日期 |
| 后置条件 | 返回获得某时间段内单只股票交易量对比数据 |
| PieChartBLService. getPieAmountVO | 语法 | PieChartVO getPieAmountVO(String name, String number, String start, String end) throws NotFoundException; |
| 前置条件 | 输入格式正确的股票代码与存在数据的起止日期 |
| 后置条件 | 返回获得某时间段内单只股票交易金额对比数据 |
| KLinearChartBLService getKChartVO | 语法 | List<KChartVO> getKChartVO(String stockNum,PeriodEnum period,String startDate,String endDate) throws NotFoundException, BadInputException; |
| 前置条件 | 输入格式正确的股票代码与存在数据的起止日期 |
| 后置条件 | 返回获得某时间段内单只股票的K线图数据 |
| BarChartBLService. getSingleVolumeVO | 语法 | SingleVolumeVO getSingleVolumeVO(String name, String number, PeriodEnum period, String start, String end) throws NotFoundException, BadInputException; |
| 前置条件 | 输入格式正确的股票代码与存在数据的起止日期 |
| 后置条件 | 返回获得某时间段内单只股票的成交量数据 |
| BarChartBLService. getMixSingleVolumeVO | 语法 | MixSingleVolumeVO getMixSingleVolumeVO(String name, String number, PeriodEnum period, String start, String end) throws NotFoundException, BadInputException; |
| 前置条件 | 输入格式正确的股票代码与存在数据的起止日期 |
| 后置条件 | 返回获得某时间段内单只股票的成交量与成交金额数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DataFactoryTxtImpl.getInstance().getCacheDataService() | 返回CacheDataService | |
| cacheDataService.getStockNumAndName() | 返回已经缓存的股票的名称与编号 | |
| cacheDataService.getTradeInfo(number) | 返回已经缓存的股票的分时交易信息 | |
| stockdataservice.getStock(String name, String start, String end, String fields) | 返回指定股票代码的相应股票交易数据 | |

#### 4.3.3.6 statisticblservice包的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| IndustryViewService. getBasicIndustryInfo | 语法 | IndustryVO getBasicIndustryInfo(String industryName, IndustryPeriodEnum period) throws NotFoundException |
| 前置条件 | 输入正确的行业名称与时间间隔 |
| 后置条件 | 系统返回行业基本数据 |
| IndustryViewService. getIndustryPrice | 语法 | OneSeriesLinearChartVO getIndustryPrice(String industryName, IndustryPeriodEnum period) throws NotFoundException; |
| 前置条件 | 输入正确的行业名称与时间间隔 |
| 后置条件 | 系统返回行业价格统计信息 |
| IndustryViewService. getCompareLinearChartVO | 语法 | TwoSeriesLinearChartVO getCompareLinearChartVO(String industryName,IndustryPeriodEnum period) throws BadInputException, NotFoundException; |
| 前置条件 | 输入正确的行业名称与时间间隔 |
| 后置条件 | 系统返回行业涨跌幅与大盘涨跌幅对比数据 |
| IndustryViewService. getIndustryVolume | 语法 | IndustryVolumeVO getIndustryVolume(String industryName, IndustryPeriodEnum period) throws NotFoundException; |
| 前置条件 | 输入正确的行业名称与时间间隔 |
| 后置条件 | 系统返回行业的资金流向统计信息 |
| SingleViewService. getStockMACD | 语法 | LinearChartVO getStockMACD(String stock\_num, String startDate, String endDate)throws BadInputException, NotFoundException; |
| 前置条件 | 输入正确的股票编号和起止时间 |
| 后置条件 | 系统返回macd数值数据包 |
| SingleViewService. getStockEMA | 语法 | LinearChartVO getStockEMA(String stock\_num, String startDate, String endDate)throws BadInputException, NotFoundException; |
| 前置条件 | 输入正确的股票编号和起止时间 |
| 后置条件 | 系统返回ema指数数据包 |
| SingleViewService. getStockRSI | 语法 | LinearChartVO getStockRSI(String stock\_num, String startDate, String endDate)throws BadInputException, NotFoundException; |
| 前置条件 | 输入正确的股票编号和起止时间 |
| 后置条件 | 系统返回rsi指数数据包 |
| SingleViewService.getBasicSingleInfo | 语法 | BasicSingleVO getBasicSingleInfo(String stock\_name, String stock\_num, PeriodEnum period, String startDate, String endDate) throws BadInputException, NotFoundException; |
| 前置条件 | 输入正确的股票编号、起止时间和时间间隔 |
| 后置条件 | 系统返回对这只股票的综合分析以及预测 |
| SingleStatisticBLService. | 语法 | double getVarianceOfPrice(StockVO stock); |
| 前置条件 | 输入数据合理的股票数据包 |
| 后置条件 | 某时间段股票日价格方差 |
| SingleStatisticBLService. | 语法 | double getAvgPrice(StockVO stock); |
| 前置条件 | 输入数据合理的股票数据包 |
| 后置条件 | 某时间段股票价格平均值 |
| SingleStatisticBLService. | 语法 | String getAllVarianceOfPrice(String startDate, String endDate) throws NotFoundException; |
| 前置条件 | 输入有数据的起止日期 |
| 后置条件 | 某时间段所有股票日价格方差集合的1/4,1/2,3/4分位点 |
| SingleStatisticBLService. | 语法 | public String cal90CC(String stockname1, String stockname2) throws NotFoundException, BadInputException, IOException; |
| 前置条件 | 输入系统中存在的两只股票 |
| 后置条件 | 近90天两只股票的相关系数 |
| TechnicalAnalysisStrategy. calculateRSI | 语法 | HashMap<String,Double> calculateRSI(ComputableStock computableStock, int step) throws BadInputException; |
| 前置条件 | 输入数据合理的可计算股票数据包与时间间隔 |
| 后置条件 | 系统返回一只股票以一定时间间隔为单位的RSI指数 |
| TechnicalAnalysisStrategy. calculateMACD | 语法 | MACDResult calculateMACD(ComputableStock computableStock) throws BadInputException; |
| 前置条件 | 输入数据合理的可计算股票数据包 |
| 后置条件 | 系统返回以一定时间间隔为单位的MACD指数 |
| TechnicalAnalysisStrategy. calculateEMA | 语法 | HashMap<String, Double> calculateEMA(ComputableStock computableStock, int step) throws BadInputException; |
| 前置条件 | 输入数据合理的可计算股票数据包与时间间隔 |
| 后置条件 | 系统返回一只股票以一定时间间隔为单位的EMA指数 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| stockViewService.getBenchmark | 返回指定大盘的指定时间字段的信息 | |
| stockViewService.getStock | 返回指定股票代码的指定时间字段的信息 | |
| stockViewService.getNamesAndNumbers | 返回行业与行业中的股票名字,股票编号对 | |
| DataFactoryTxtImpl.getInstance().getCacheDataService() | 返回CacheDataService | |
| cacheDataService.getStockNumAndName() | 返回已经缓存的股票的名称与编号 | |
| cacheDataService.getTradeInfo(number) | 返回已经缓存的股票的分时交易信息 | |

## 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 |
| stockdataservice | 提供对AnyQuant公开数据的访问功能，与对用户收藏信息的管理功能 |

### 4.4.2 数据层模块的接口规范

**stockdataservice包的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| stockdataservice.getAllStock(String year, String exchange) | 语法 | public String getAllStock(String year, String exchange) |
| 前置条件 | 已输入格式正确的年份与交易所代码 |
| 后置条件 | 系统返回指定年份和交易所的全部股票列表json数据字符串 |
| stockdataservice.getStock(String name, String start, String end, String fields) | 语法 | public String getStock(String name, String start, String end, String fields) |
| 前置条件 | 已输入格式正确的股票代码，起止时间和指定的数据字段 |
| 后置条件 | 系统返回指定股票代码的相应股票交易数据json数据字符串 |
| stockdataservice.getAllBenchmark() | 语法 | public String getAllBenchmark() |
| 前置条件 | 无 |
| 后置条件 | 系统返回获取所有可用的大盘指数json数据字符串 |
| stockdataservice.getBenchmark(String bench, String start, String end, String fields) | 语法 | public String getBenchmark(String bench, String start, String end, String fields) |
| 前置条件 | 已输入格式正确的大盘代码，起止时间和指定的数据字段 |
| 后置条件 | 系统返回指定的大盘指数json数据字符串 |
| stockdataservice.getAllFields() | 语法 | public String getAllFields() |
| 前置条件 | 无 |
| 后置条件 | 系统返回股票的可用交易数据字段 |
| UserCollectionDataService. getUserCollections(UserInfo info) | 语法 | List<String> getUserCollections(UserInfo info) |
| 前置条件 | 用户的身份已被识别认证 |
| 后置条件 | 系统返回用户收藏的股票编号列表 |
| UserCollectionDataService. addUserCollection(UserInfo info, String stockNumber) | 语法 | ResultMsg addUserCollection(UserInfo info, String stockNumber) |
| 前置条件 | 用户的身份已被识别认证，用户输入欲收藏股票编号 |
| 后置条件 | 添加收藏结果 |
| UserCollectionDataService. deleteCollcetion(UserInfo info, String stockNumber) | 语法 | ResultMsg deleteCollcetion(UserInfo info, String stockNumber) |
| 前置条件 | 用户的身份已被识别认证，用户输入欲删除股票编号 |
| 后置条件 | 删除收藏结果 |

# 5.信息视角

## 5.1 VO PO定义的选择过程

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。

由于服务器API提供的数据以JSON格式给出，数据层已经负责从服务器端读取数据与将数据缓存至本地并定时更新的责任，我们打算将JSON的解析工作交给逻辑层处理。因此，PO就是用户所需股票指定时间段内的信息JSON字符串，PO包无需定义。

VO应该包含且仅仅包含用户所需股票指定时间段内指定的信息，以达到数据耦合的效果。因此我们将VO定义为嵌套的HashMap：内层HashMap是属性关键字与属性数据的对应，外层HashMap是指定时间段内的一天与每天HashMap数据的对应。

## 5.2 VO PO定义结果

我们认为展示定义结果没有意义，VO与PO定义结果请参考代码。