2017-3-23

Leftovers

单元测试文档

（迭代2）

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 变更原因 | 版本号 |
| 吴志成 | 2017-3-23 | 初稿 | V1.0 |
| 全体人员 | 2017-4-18 | 进行测试，完善文档 | V2.0 |

**文档更新记录表：**

目录

[1.单元测试计划概述 3](#_Toc476609063)

[2.单元测试的范围 3](#_Toc476609064)

[3.单元测试的策略 3](#_Toc476609065)

## 1.单元测试计划概述

本文档为在Quantour System进行单元测试时提供的有关任务安排、方法、资源和进度方面的指导，目标是从代码中发现Bug，以备后续移除Bug参考，提高软件质量，最终达到用户需求。

本文档的读者主要是开发工程师。执行人员有4人，将执行2个工作日。

参考文献：《软件需求规格说明书》、《软件架构说明书》、《详细设计说明书》

## 2.单元测试的范围

单元测试必须能够展示出Quantour System中类的每个方法都能正常工作。单元测试的概念是将一个单元和该系统的其余部分隔离开来独立测试，因此可能需要创建测试驱动（test driver）代码。要求测试驱动代码用JUnit编写，当被测单元被提交准备完成项目构建时，相关的测试驱动代码必须已经被提交到团队的代码库中。

## 3.单元测试的策略

1. 对方法(method)的单元测试

检查对所有调用对象的使用

验证对所有数据结构的处理

验证对所有文件的处理

验证控制流的不变性

检查所有循环的正常终止

检查所有循环的异常终止

验证所有错误条件的处理

检查定时和同步

(2)对类(class)的单元测试

1)结合方法的执行，推荐对类的测试

选择最常见的执行序列

需要包括可能导致缺陷的序列

手工计算预期的属性值

1. 集中对每个属性进行单元测试

初始化，然后执行会影响属性的方法序列

1. 验证每个类的不变性

验证默认初始值时的不变性

执行方法序列

验证不变性仍然正确

## 4.测试用例列表：

|  |  |
| --- | --- |
| 测试用例ID | 测试用例描述 |
| AccountTest | Account中方法的测试 |
| AnalysisTest | Analysis中方法的测试 |
| PortfolioValueTest | PortfolioValue中方法的测试 |
| UniverseTest | Universe中方法的测试 |
| MarketBLServiceImplTest | MarketBLServiceImpl中方法的测试 |
| StockTest | Stock中方法的测试 |
| BackTestBLServiceTest | BackTestBLService中方法的测试 |

### 4.1 AccountTest

class AccountTest {  
 */\*\*  
 \* Created by Hiki on 2017/4/16.  
 \*/* private Account account;  
  
 private LocalDate date = LocalDate.*parse*("2014-01-21");  
  
 @BeforeEach  
 void setUp() {  
  
 double capitalBase = 100000.0;  
 BacktestDataService bs = new BackTestDao();  
 List<String> universe = bs.getStockCodesByBoard(Board.*Main\_Board*);  
  
 try {  
 account = new Account(capitalBase, universe);  
 } catch (StockNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 @Test  
 void isLimitUp() {  
 boolean limit = account.isLimitUp("1", date);  
 System.*out*.println(limit);  
 }  
  
 @Test  
 void isSuspended() {  
 boolean suspended = account.isSuspended("1", date);  
 System.*out*.println(suspended);  
 }  
  
 @Test  
 void isLimitDown() {  
 boolean limit = account.isLimitDown("1", date);  
 System.*out*.println(limit);  
 }  
  
 @Test  
 void isSTStock() {  
 boolean st = account.isSTStock("30");  
 *assertTrue*(st);  
 }  
  
 @Test  
 void isInStockHoldings() {  
 System.*out*.println(account.isInStockHoldings("1"));  
 *assertFalse*(account.isInStockHoldings("1"));  
 *assertFalse*(account.isInStockHoldings("300091"));  
 *assertFalse*(account.isInStockHoldings("300198"));  
 }

}

### 4.2 AnalysisTest

class AnalysisTest {  
  
 Analysis tested;  
  
 @BeforeEach  
 void setUp() throws FileNotFoundException {  
 File file = new File("C:\\Users\\kevin\\OneDrive - smail.nju.edu.cn\\Documents\\Study\\2017\_Spring\\SE3\\Homework\\Project\\Quantour\\Src\\Quantour\\src\\main\\resources\\data\\Test\\Book2.csv");  
 BufferedReader reader = new BufferedReader(new FileReader(file));  
 String[] lines = reader.lines().toArray(String[]::new);  
 PortfolioValue strategy = new PortfolioValue(new DataFrame(lines, ","));  
  
 File file1 = new File("C:\\Users\\kevin\\OneDrive - smail.nju.edu.cn\\Documents\\Study\\2017\_Spring\\SE3\\Homework\\Project\\Quantour\\Src\\Quantour\\src\\main\\resources\\data\\Test\\Book3.csv");  
 BufferedReader reader1 = new BufferedReader(new FileReader(file1));  
 String[] lines1 = reader1.lines().toArray(String[]::new);  
 PortfolioValue benchmark = new PortfolioValue(new DataFrame(lines1, "\t"));  
 tested = new Analysis(strategy, benchmark);  
 }  
  
 @Test  
 void getOverview() {  
 BacktestResultOverviewVO backtestResultOverviewVO = tested.getOverview();  
 }  
  
 @Test  
 void getWinRate() {  
 WinRateVO winRateVO = tested.getWinRate(5);  
 }  
  
 @Test  
 void getReturnDistribution() {  
 ReturnDistributionVO returnDistributionVO = tested.getReturnDistribution(5);  
 }  
  
}

### 4.3 PortfolioValueTest

class PortfolioValueTest {  
  
 */\*\*  
 \* 被测试的对象  
 \*/* PortfolioValue tested;  
  
 @BeforeEach  
 void setUp() throws FileNotFoundException {  
 File file = new File("C:\\Users\\kevin\\OneDrive - smail.nju.edu.cn\\Documents\\Study\\2017\_Spring\\SE3\\Homework\\Project\\Quantour\\Src\\Quantour\\src\\main\\resources\\data\\Test\\Book2.csv");  
 BufferedReader reader = new BufferedReader(new FileReader(file));  
 String[] lines = reader.lines().toArray(String[]::new);  
 tested = new PortfolioValue(new DataFrame(lines, ","));  
 }  
  
 */\*\*  
 \* 测试获取夏普比率  
 \*/* @Test  
 void getSharpRatio() {  
 System.*out*.println("SharpRatio: " + tested.getSharpRatio());  
 }  
  
 */\*\*  
 \* 测试获取收益波动率  
 \*/* @Test  
 void getVolatility() {  
 System.*out*.println("Volatility: " + tested.getVolatility());  
 }  
  
 */\*\*  
 \* 测试获取年化收益率  
 \*/* @Test  
 void getAnnualisedReturn() {  
 System.*out*.println("Annualised Return: " + tested.getAnnualisedReturn());  
 }  
  
 */\*\*  
 \* 测试获取最大回撤  
 \*/* @Test  
 void getMaxDrawdown() {  
 System.*out*.println("Max Drawdown: " + tested.getMaxDrawdown());  
 }  
  
 */\*\*  
 \* 测试获取每日的累计收益率  
 \*/* @Test  
 void getDailyCumReturn() {  
 Series dailyCumReturn = tested.getDailyCumReturn();  
 }  
  
 */\*\*  
 \* 测试获取每日的收益率  
 \*/* @Test  
 void getDailyReturn() {  
 Series dailyReturn = tested.getDailyReturn();  
 }  
  
 */\*\*  
 \* 测试获取总收益  
 \*/* @Test  
 void getTotalReturn() {  
 System.*out*.println("Total Return: " + tested.getTotalReturn());  
 }  
  
}

### 4.4 UniverseTest

class UniverseTest {  
  
 UniverseBLService tested;  
  
 @BeforeEach  
 void setUp() {  
 tested = new Universe();  
 }  
  
 @Test  
 void createNewUniverse() throws DuplicateName {  
 tested.createNewUniverse("hghg", Arrays.*asList*("1", "2", "3", "4"));  
 }  
  
 @Test  
 void deleteUniverse() {  
 tested.deleteUniverse("wodegupiaochi");  
 }  
  
 @Test  
 void getAllUniverses() {  
 List<UniverseVO> universeVOS = tested.getAllUniverses();  
 }  
  
 @Test  
 void getStockAttributes() {  
 try {  
 StockAttributesVO stockAttributesVO = tested.getStockAttributes("万科A");  
 } catch (StockNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
}

### 4.5 MarketBLServiceImplTest

class MarketBLServiceImplTest {  
  
 MarketBLService marketBLService;  
  
 @BeforeEach  
 void setUp() {  
 marketBLService = new MarketBLServiceImpl();  
 }  
  
 */\*\*  
 \* 测试获取某天的市场行情  
 \*  
 \** ***@throws*** *IOException IO异常  
 \*/* @Test  
 void getMarketInfo() throws IOException {  
 MarketVO marketVO = marketBLService.getMarketInfo(LocalDate.*of*(2008, 3, 13));  
 }  
  
 */\*\*  
 \* 测试获取市场的所有股票的默认信息  
 \*  
 \** ***@throws*** *IOException IO异常  
 \*/* @Test  
 void getAllStocks() throws IOException {  
 List<DefaultStockVO> stockVOList = marketBLService.getAllStocks();  
 }  
  
}

### 4.6 StockTest

public class StockTest {  
  
 Stock tested;  
  
 @Before  
 public void setUp(){  
 tested = new Stock();  
 }  
  
 @Test  
 public void findStockByName(){  
 LocalDate beginTime = LocalDate.*parse*("2013-12-01");  
 LocalDate endTime = LocalDate.*parse*("2014-01-01");  
 try {  
 StockVO stockVO = tested.findStockByName("坚瑞消防", beginTime, endTime);  
 System.*out*.println(stockVO.change);  
 System.*out*.println(stockVO.increase);  
 System.*out*.println(stockVO.logReturnVariance);  
 System.*out*.println(stockVO.presentPrice);  
 } catch (StockNotFoundException e) {  
 System.*out*.println(e.getMessage());  
 } catch (TimeException e) {  
 System.*out*.println(e.getMessage());  
 }  
  
 }  
  
 @Test  
 public void findStockByCode() {  
 LocalDate beginTime = LocalDate.*parse*("2013-12-01");  
 LocalDate endTime = LocalDate.*parse*("2014-04-01");  
  
 try {  
 StockVO stockVO = tested.findStockByCode("1", beginTime, endTime);  
 System.*out*.println(stockVO.getStockItemList().get(0).change);  
 System.*out*.println(stockVO.getStockItemList().get(0).increase);  
 System.*out*.println(stockVO.getStockItemList().get(30).averages.get(30));  
// System.out.println(stockVO.change);  
// System.out.println(stockVO.increase);  
// System.out.println(stockVO.logReturnVariance);  
// System.out.println(stockVO.presentPrice);  
 } catch (StockNotFoundException e) {  
 System.*out*.println(e.getMessage());  
 } catch (TimeException e) {  
 System.*out*.println(e.getMessage());  
 }  
  
 }  
  
 @Test  
 public void findSuggestions(){  
 try {  
 List<String> result = tested.findSuggestions("哇单位");  
 result.stream().forEach(s -> System.*out*.println(s));  
 } catch (StockNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 @Test  
 public void getSections(){  
  
 List<String> sections = tested.getSections();  
 sections.forEach(System.*out*::println);  
  
 }  
  
}

### 4.7 BackTestBLServiceTest

class BackTestBLServiceTest {  
  
 BackTestBLService tested;  
  
 LocalDate beginTime = LocalDate.*parse*("2006-01-01");  
  
 LocalDate endTime = LocalDate.*parse*("2008-01-01");  
  
 Benchmark benchmark = Benchmark.*HS300*;  
  
 String[] universeNames = {"全部股票"};  
  
 double capitalBase = 100000.0;  
  
 int formRate = 10;  
  
 int refreshRate = 15;  
  
  
  
  
 @BeforeEach  
 void setUp() throws StockNotFoundException {  
 tested = new BackTest();  
 }  
  
 @Test  
 void runMomentum() throws StockNotFoundException {  
  
 System.*out*.println(LocalDateTime.*now*());  
  
 tested.runMomentum(beginTime, endTime, benchmark, universeNames, capitalBase, formRate, refreshRate);  
  
 System.*out*.println(LocalDateTime.*now*());  
  
 List<PositionRecordVO> positionRecordVOS = tested.getPositionRecord();  
  
 System.*out*.println(LocalDateTime.*now*());  
  
 for (PositionRecordVO positionRecordVO : positionRecordVOS) {  
 System.*out*.println(positionRecordVO.getDate() + " " + positionRecordVO.getCapitalAll());  
 }  
 }  
  
 @Test  
 void runMeanRevision() throws StockNotFoundException {  
 tested.runMeanRevision(beginTime, endTime, benchmark, universeNames, capitalBase, formRate, refreshRate);  
  
 List<PositionRecordVO> positionRecordVOS = tested.getPositionRecord();  
  
 for (PositionRecordVO positionRecordVO : positionRecordVOS) {  
 System.*out*.println(positionRecordVO.getDate() + " " + positionRecordVO.getCapitalPresent());  
 }  
  
 }

## 5.测试总结

基本达到了100%的语句覆盖率，100%的分支覆盖率。

：