NBA 数据分析系统 NAS(NBA Analysis System)

**迭代二总测试报告文档**

南京大学TBD工作组

2015-04-13

目录

[1．引言 2](#_Toc406533546)

[1.1编写目的 2](#_Toc406533547)

[1.2测试范围 2](#_Toc406533548)

[1.3参考文献 3](#_Toc406533549)

[1.4术语和缩略语 3](#_Toc406533550)

[2.详细情况 4](#_Toc406533551)

[2.1总测试结果概述 4](#_Toc406533552)

[2.1.1测试用例设计 4](#_Toc406533553)

[2.1.2测试环境与配置 5](#_Toc406533554)

[2.1.3 测试方法和工具 7](#_Toc406533555)

[2.2详细测试结果 7](#_Toc406533556)

[2.2.1.单元测试用例执行结果 7](#_Toc406533557)

[2.2.2. 集成测试用例执行结果 27](#_Toc406533558)

[2.2.3.集成测试 32](#_Toc406533559)

[2.2.4.单元测试 35](#_Toc406533560)

[2.2.5评估产品质量 38](#_Toc406533561)

[2.2.6度量数据总结 39](#_Toc406533562)

[2.3决策理由 39](#_Toc406533563)

[2.4总结与建议 40](#_Toc406533564)

[2.4.1总体性评价 40](#_Toc406533565)

[2.4.2安装环境 40](#_Toc406533566)

[2.4.3经验总结 40](#_Toc406533567)

[3.其他 42](#_Toc406533568)

[3.1文档修订历史 42](#_Toc406533569)

# 1．引言

## 1.1编写目的

本测试报告为NBA 数据分析系统迭代二的测试报告，目的在于总结迭代二测试阶段的测试情况以及分析测试结果，描述系统是否符合用户需求，是否已达到用户预期的功能目标，并对测试质量进行分析。

测试报告参考文档提供给用户、测试人员、开发人员、项目管理者、其他管理人员和需要阅读本报告的高层经理阅读。

## 1.2测试范围

测试主要根据用户需求说明书和软件需求规格说明书以及相应的文档进行系统测试，包括功能测试、性能测试、安全性和访问控制测试、用户界面测试等，而单元测试和集成测试由开发人员来执行。

主要测试功能包括：球员管理，比赛管理以及球队管理

## 1.3参考文献

1.[IEEE-829-2008]

2. NBA 数据分析系统用例文档

3 NBA 数据分析系统需求规格说明书

## 1.4术语和缩略语

|  |  |
| --- | --- |
| **术语或缩略语** | **全意** |
| NAS | NBA 数据分析系统 |
| HCI | 持续集成 |
| 严重缺陷 | 系统发生死机，异常，终止，返回错误信息的情况。 |
| LOC | 代码行数 |
| NULL Pointer Exception | 空指针异常，一般产生于没有正确初始化对象或者是对象赋值有误 |

# 2.详细情况

## 2.1总测试结果概述

NBA 数据分析系统的测试从2015-4-10开始持续到2015-4-13，共持续4天，按时完成测试计划。

### 2.1.1测试用例设计

1.集成测试

※概述：使用持续集成，体系结构设计阶段为每一层编写了Stub和Driver,用自动化集成工具完成，在构造阶段完成。

※测试用例选择：

随机选择+边界值分析

2.单元测试

※概述:

1.测试了复杂的业务逻辑，包括数据排序，脏数据处理，系统的状态控制，在构造阶段对业务逻辑层的逻辑进行了单元测试。

2.数据文件的读写进行了单元测试，确保数据层的操作正确。

※测试用例选择：

随机选择+边界值分析

### 2.1.2测试环境与配置

#### 2.1.2.1软硬件环境

表2.1.2.1 软硬件环境

|  |  |  |
| --- | --- | --- |
| **硬件环境** | **应用服务器** | **客户端** |
| 硬件配置 | CPU:Intel 酷睿i5 2520M  CPU2.5GHZ stepping 01  Memory:4GB  HD:Intel GMA HD 3000 | CPU Intel 酷睿i5 2520M  CPU2.5GHZ stepping 01  Memory:4GB  HD: Intel GMA HD 3000 |
| 软件配置 | OS:windows8.1专业版  JDK: 1.8  Tomcat :7.0 | OS:windows8.1专业版  OS:Ubuntu12.1.3  JDK: 1.8  JRE: 1.8 |
| 网络环境 | 10M LAN | 10M LAN |

#### 2.1.2.2网络拓扑



图2.1.2.2 应用服务器端和客户端（分布式系统）

### 2.1.3 测试方法和工具

表2.1.3 测试方法和工具

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **分类标准** | **类别** | **测试方法** | **测试工具** | **备注** |
| 测试对象 | 单元测试 | 黑盒与白盒测试相结合。  部分使用TestDriver进行开发。对较为复杂的逻辑使用了语句覆盖的白盒测试，其他大多使用随机测试。 | Junit4 |  |
| 集成测试 | CI | jenkins |  |
| 系统测试 | 参照软件需求规格说明，手工测试 | 随机测试 |  |
| 测试目标 | 功能 | 黑盒、手工、回归 | 随机测试 |  |
| 性能 | 黑盒、手工、回归 | 随机测试 | 测试处理业务时间 |
| 安全性 | 黑盒、手工、回归 | 无 |  |

## 2.2详细测试结果

### 2.2.1.单元测试用例执行结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | 名称 | 条件 | 输入 | 期望结果 | 实际结果 | 备注 |
| Testcase001 | 读取比赛数据 | 存在对应的数据文件 | 文件路径（"Data/matches"） | 读取所有的文件并加载在ArrayList中 | 读取所有的文件并加载在ArrayList中 |  |
| Testcase002 | 得到所有比赛数据 | 数据已经读取至ArrayList中 | 无 | 返回所有比赛数据的ArrayList | 返回所有比赛数据的ArrayList |  |
| Testcase003 | 读取球员数据 | 存在对应的数据文件 | 文件路径（"Data/players/ portrait"） | 读取所有的文件并加载在ArrayList中 | 读取所有的文件并加载在ArrayList中 |  |
| Testcase004 | 得到所有球员数据 | 数据已经读取至ArrayList中 | 无 | 返回所有比赛数据的ArrayList | 返回所有比赛数据的ArrayList |  |
| Testcase005 | 读取球员头像 | 存在对应的图像文件 | 文件路径（"Data/players/ info "） | 读取所有的文件并加载在ArrayList中 | 读取所有的文件并加载在ArrayList中 |  |
| Testcase006 | 读取球队数据 | 存在对应的数据文件 | 文件路径（"Data/teams"） | 读取所有的文件并加载在ArrayList中 | 读取所有的文件并加载在ArrayList中 |  |
| Testcase007 | 得到所有球队数据 | 数据已经读取至ArrayList中 | 无 | 返回所有比赛数据的ArrayList | NULLPOINT | 解决问题后，返回所有比赛数据的ArrayList |
| Testcase008 | 读取球队队徽 | 存在对应的图像文件 | 文件路径（"Data/teams"） | 读取所有的文件并加载在ArrayList中 | 读取所有的文件并加载在ArrayList中 |  |
| Testcase009 | 根据球队名字得到球队对象 | 已经得到了所有球队的ArrayList | “” | 得到null | 得到null |  |
| Testcase010 | 根据球队名字得到球队对象 | 已经得到了所有球队的ArrayList | “LAL“ | 得到该team的对象 | 得到该team的对象 |  |
| Testcase011 | 根据球员名字的到球员对象 | 已经得到了所有球员的ArrayList | “” | 得到null | 得到null |  |
| Testcase012 | 根据球员名字的到球员对象 | 已经得到了所有球员的ArrayList | ” Kobe Bryant “ | 得到该player的对象 | 得到该player的对象 |  |
| Testcase013 | 得到今日比赛 | 存在对应的比赛数据文件 | 文件路径（"Data/matches"） | 读取并返回今日的所有比赛 | 读取并返回今日的所有比赛 |  |
| Testcase014 | 得到比赛数据王 | 存在对应的比赛数据文件 | 无 | 返回比赛数据王 | 返回比赛数据王 |  |
| Testcase015 | 增加比赛数据 | 存在新增的比赛数据文件 | 文件路径（"Data/matches "） | 读取新增的文件并加载在ArrayList中 | NULLPOINT | 解决问题后，读取新增的文件并加载在ArrayList中 |
| Testcase016 | 得到当赛季数据王 | 球员数据已经读取至ArrayList中 | -king points season | 返回当前赛季的数据王 | 返回当前赛季的数据王 |  |
| Testcase017 | 得到当日数据王 | 球员数据已经读取至ArrayList中 | -king points daily | 返回当日的数据王 | 返回当日的数据王 |  |
| Testcase018 | 得到赛季最火球员（前5） | 球员数据已经读取至ArrayList中 | -hot points -5 | 返回赛季最火球员（前5） | 返回当今赛季所有球员 | 经修改后，返回赛季最火球员（前5） |
| Testcase019 | 得到进步最快球员 | 球员数据已经读取至ArrayList中 | 无 | 返回进步最快的球员 | NULLPOINT | 经修改后，返回进步最快的球员t |
| Testcase020 | 计算球员高阶数据 | 比赛数据和球员数据已经读取至ArrayList中 | 无 | 计算球员高阶数据 | 计算球员高阶数据 |  |
| Testcase021 | 计算球员热点数据 | 比赛数据和球员数据已经读取至ArrayList中 | 无 | 计算球员热点数据 | 计算球员热点数据 |  |
| Testcase022 | 跟据命令参数返回相应的球队 | 已经得到了所有球队的ArrayList | Rebound E | 得到相应的球队 | 得到相应的球队 |  |
| Testcase023 | 得到赛季球队王 | 已经得到了所有球队的ArrayList | 无 | 得到相应的球队 | 得到相应的球队 |  |
| Testcase024 | 得到联盟平均数据 | 已经得到了所有球员，球队和比赛的ArrayList | 无 | 返回联盟平均数据 | 返回联盟平均数据 |  |
| Testcase025 | 计算球队高阶数据 | 比赛数据和球员数据已经读取至ArrayList中 | 无 | 计算球队高阶数据 | 计算球队高阶数据 |  |
| Testcase026 | 计算球队热点数据 | 比赛数据和球员数据已经读取至ArrayList中 | 无 | 计算球队热点数据 | 计算球队热点数据 |  |
| Testcase027 | 测试比赛文件增加时是否会做出相应的增加比赛动作 | 已经得到了所有比赛的数据并且有新的数据进入 | 无 | 检测到刷新并自动增加新的比赛数据 | 无反应 | 修改后，检测到刷新并自动增加新的比赛数据 |
| Testcase028 | 得到所有球队log（svg文件读取处理） | 已经得到了所有球队的ArrayList | 无 | 得到所有的svg文件并存入内存 | 得到所有的svg文件并存入内存 |  |

### 2.2.2集成测试用例（详见项目中代码）执行结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | 名称 | 条件 | 输入 | 期望结果 | 实际结果 | 备注 |
| Testcase001 | 得到所有比赛数据 | 得到一个MatchDataController | 无 | 得到一个含有所有player的ArrayList | 得到一个含有所有player的ArrayList |  |
| Testcase002 | 得到所有球员数据 | 得到一个PlayerDataController | 无 | 得到一个含有所有match的ArrayList | 得到一个含有所有match的ArrayList |  |
| Testcase003 | 得到所有球队数据 | 得到一个TeamDataController | 无 | 得到一个含有所有team的ArrayList | 得到一个含有所有team的ArrayList |  |

### 2.2.3.集成测试

1.测试活动总结

在体系结构设计阶段结束的时候，为每一层开发了Stub和部分层开发了Driver。在构造阶段，除了对数据文件的读写外，其余全部采用自顶向下的开发结构，每开发一个层次，就使用下一个层次的Stub，进行maven build。逐步将下层集成至已经开发完成的代码当中。

2.测试任务的结果总结

任务1：

描述：集成Data层对数据文件的操作。

编写了Data层对数据文件读写进行封装和实现的BLController以及其实现类，用BLTest对其进行了驱动测试，以便以后集成。

结论：通过。足以完成需求。

问题：无

任务2：

描述:集成UI与BL层。

完成业务逻辑层之后将UI层中使用Stub的部分替换为实现代码将BL层集成进系统。

结论：最终通过。

任务3：

描述：集成Data层

完成Data层之后将BL层中使用Stub的部分替换为实现代码，将Data层集成进系统。

结论：最终通过。

3.缺陷和辨析

表2.2.2-1 集成测试严重缺陷分析

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **模块名称** | **缺陷编号** | **简要描述** | **分析结果** | **测试通过(pass/fail)** |
| PlayerController.init | 2.1 | 内存溢出，出现死循环 | 当时设计的多个线程出现交叉调用，结果导致冲突。 | Fail->pass |

表2.2.2-2 集成测试残留缺陷及未解决问题列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **模块名称** | **缺陷编号** | **性质** | **简要描述** | **原因分析** |
| MatchesController.ComputeData | 2.11 | 无影响缺陷 | 脏数据的处理有些问题，不够全面，对于有些脏数据只能发现，无法修改。 | 对于某些脏数据的处理，将留到迭代二或迭代三解决。 |

4.产品质量评估：

概述，在集成过程中使用hudson进行自动化构建，最终构建结果是SUCCESS。期间出现build失败的情况，查找缺陷，定位缺陷并更正。

结论：达成预期目标，测试通过。

5.度量数据

1.集成测试缺陷数据：

严重缺陷：1

一般缺陷：3

无影响缺陷：1

2.集成测试覆盖度：

被测试的模块总数：3

模块总数：3

模块覆盖度：100%

备注：模块java/main/二级目录下包的个数进行统计。

### 2.2.4.单元测试

1.测试活动总结：

大体上使用了TestDriver的思想指导开发，在构造阶段，对于有较为复杂的逻辑处理和外部内部接口的类，先编写Junit测试用例。并且编写了mock object以隔离外界环境对此类的影响，而后进行开发，开发完成之后，先进行Junit自动化测试，通过之后提交。

2.测试任务的结果总结：

保证所有进行单元测试的类，方法自动化测试通过。也即保证了该方法就有系统所期待的输入输出和不变式。

3.缺陷和辨析

表2.2.3-1 单元测试重要缺陷分析

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **模块名称** | **缺陷编号** | **简要描述** | **分析结果** | **测试通过(pass/fail)** |
| **PlayerController.** **getPlayer** | 3.1 | 某些球员找不到返回为null | 有些球员打过比赛但没有他的记录 | Fail->pass |

表2.2.3-2 单元测试残留于未解决问题列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **模块名称** | **缺陷编号** | **性质** | **简要描述** | **原因分析** |
| **无** |  |  |  |  |

4.产品质量评估：

1.概述：单元测试全部测试用例Junit自动化测试SUCCESS，达到预定的目标，对复杂逻辑基本实现代码覆盖。

2.备注：上表只显示了系统排课算法的缺陷分析，主要有两个原因。①这是系统中最复杂的逻辑，而且复杂度基本集中在构造阶段，对外接口较为简单，实现较为复杂。②在开发过程中没有记录debug的情况，没有历史记录。

3.结论：质量达到预期标准，可以进行后续的打包安装工作。

5.度量数据：

1.缺陷度量：

缺陷数目难以找到真实可靠的数据。

2.覆盖度度量：

使用Maven Cobertura工具进行度量。度量结果如下图示例

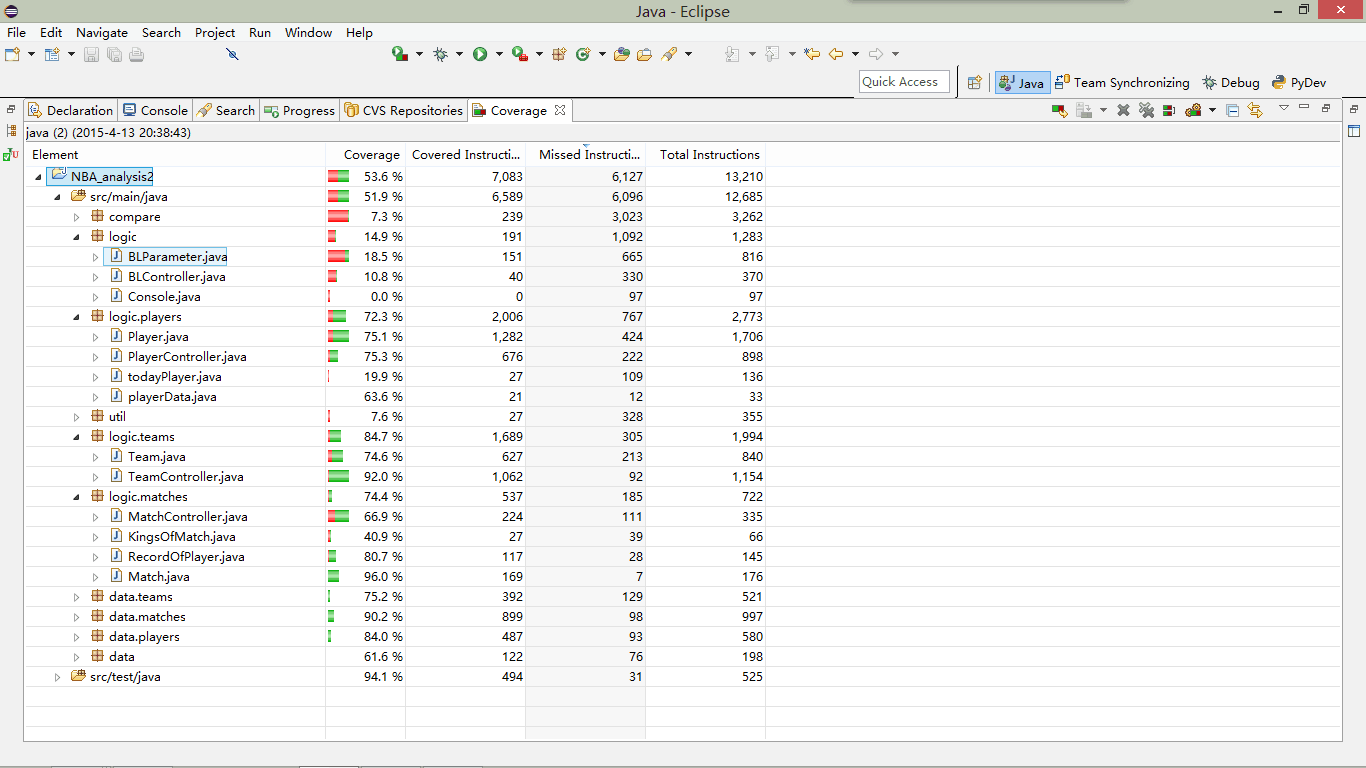


图2.2.3.5 覆盖度度量

总体来说对于有较为复杂业务逻辑（这里指不是只含有getter和setter）的Logic层和Data层基本实现了方法的全覆盖。

### 2.2.5评估产品质量

经过对系统测试，集成测试以及单元测试结果的分析总结以及对测试过程的回顾和反思。按照系统仍然存在的缺陷数目大致与最后一次测试缺陷数目持平的原则，我们认为系统的质量在功能性，部分非功能性上达到了预期。

### 2.2.6度量数据总结

单元测试的覆盖率较低（低于对10000+LOC）的要求（50%）：原因在于大量LOC分布在ui层，尤其是使用可视化界面编辑工具产生的不可修改代码当中，对这些代码，难以同时也没有必要开发测试用例。而系统的主要复杂度只要集中在Logic层，对Logic层的单元测试测试用例覆盖度更有意义。所以我们认为测试的质量是得到保证的。

## 2.3决策理由

1.集成测试：

通过：maven clean build输出结果为SUCCESS

不通过：maven clean build 输出结果为FAILTURE

有条件通过：无

2.单元测试：

通过：junit自动测试执行结果为全部通过SUCCESS

不通过：junit自动测试执行结果为全部FAILTURE

有条件通过：junit自动测试执行结果部分SUCCESS部分FAILTURE

# 3.其他

## 3.1文档修订历史

|  |  |  |  |
| --- | --- | --- | --- |
| **版本信息** | **变更人** | **主要内容** | **备注** |
| 1.0.1 | 陈元亮 | 创建 | 2015-04-13 |
|  |  |  |  |