QQTang软件覆盖率分析报告

* 1. 1 覆盖率分析环境

本次测试针对QQTang项目服务器端的大厅系统，开发语言为Java。

本次测试中，采用JUnit工具进行单元测试，并使用Eclipse的EclEmma插件进行代码覆盖率分析。测试主要针对HallInfo类与Room类共同组成的大厅模块，接口函数全在HallInfo类中。

* 1. 2.1 软件项

分析中用到的软件项如下表所示。

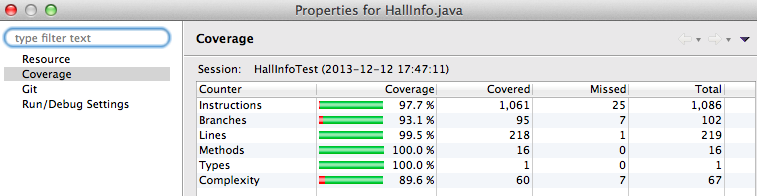
表1 软件项

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **用途** | **配置** | **备注** |
| JUnit | 单元测试 | 默认 |  |
| EclEmma | 覆盖率统计 | 默认 |  |

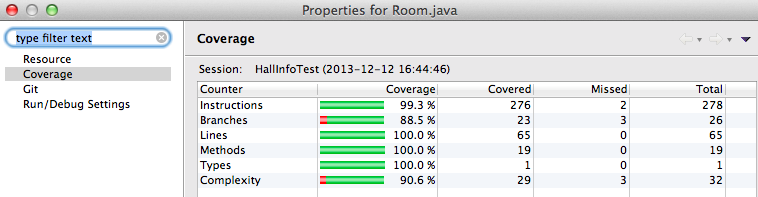
3 覆盖率分析结果

整体覆盖率如下图所示：

1． HallInfo类的覆盖率：



2．Room类的覆盖率：



综合统计，代码行数覆盖率为**99.6%**，分支覆盖率为**92.1%**。详见导出的统计结果。

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 方法名 | 结构覆盖率(%) | 备注 |
| 1 | start(Room) | 96% | HallInfo类 |
| 2 | setGameFinishedInfo(int, JSONArray) | 98% | HallInfo类 |
| 3 | getGameStartInfo(String) | 96% | HallInfo类 |
| 4 | setGameStartInfo(int JSONArray) | 96% | HallInfo类 |
| 5 | logout() | 95% | HallInfo类 |
| 6 | getEvents(int) | 94% | HallInfo类 |
| 7 | getMessages(String) | 93% | HallInfo类 |
| 8 | getStatus | 91% | HallInfo类 |
| 9 | execute(JSONObject) | 100% | HallInfo类 |
| 10 | getDetails(String) | 100% | HallInfo类 |
| 11 | leave(int) | 95% | Room类 |
| 12 | addMessage(String) | 100% | Room类 |
| 13 | isReady() | 100% | Room类 |
| 14 | getEmpty() | 100% | Room类 |
| 15 | getMessages() | 100% | Room类 |
| 16 | enter(int, String) | 100% | Room类 |
| 17 | setAddr(int, String, int) | 100% | Room类 |
| 18 | ready(int) | 100% | Room类 |
| 19 | unready(int) | 100% | Room类 |
| 20 | setKey(String) | 100% | Room类 |
| 21 | getStatus(int) | 100% | Room类 |
| 22 | getUser(int) | 100% | Room类 |
| 23 | getKey(int) | 100% | Room类 |
| 24 | getPort(int) | 100% | Room类 |
| 25 | getIp(int) | 100% | Room类 |
| 26 | getStage() | 100% | Room类 |
| 27 | setStage(int) | 100% | Room类 |
| 28 | getNum() | 100% | Room类 |

* 1. 4.覆盖率差异分析

对于代码覆盖，测试中有1条语句未覆盖到，将原因总结如下：

|  |  |
| --- | --- |
| **未覆盖语句** | **原因分析** |
| gameTalk.send(data.toString()) | 这句话需要调用GameTalk类的方法，在针对HallInfo的单元测试中构造一个GameTalk的实例代价很大。  在集成测试时将针对这类接口函数进行测试。 |

对于分支覆盖，未覆盖的原因有以下两种：

1．（7个）该分支的作用是assert。若代码写错，则进行一种比较保守的异常处理。在编码正确的情况下，程序不会运行到该分支。

3．（3个）该分支是针对某种异常的处理，然而构造这种异常的代价过大，已经远大于编码的代价。所以选择在集成测试时测试这种情况。

* 1. 5.讨论与总结

通过单元测试，发现了一些隐性的Bug。这些Bug在正常情况下不会影响系统运行，但在输入非法的情况下，则会导致系统崩溃。

单元测试基本覆盖到了所有的语句。但是仍然有接近10%的分支没有被覆盖到。

其中，有8个分支是为了防止代码写错而加上的异常处理，在代码正确的情况下自然不会进入这些分支。在确保代码正确的情况下，可以将它们去掉来提高分支覆盖率，但这样做似乎比较危险。另一种方法是将他们替换为assert，我将在今后的工程中注意这点。

另外2个分支是由于测试代价过大而没有覆盖。拿其中一个距离，为了测试这个分支，我需要虚拟4个用户进入大厅、进入房间、准备，游戏开始后，再让4个玩家同时断线，从而检查对断线情况的处理，这个过程至少30行吧。然而，这个异常处理的代码只有3行。对于这类得不偿失的Case，我认为应该在集成测试中着重测它们，若集成测试出错再肉眼检查这部分代码的正确性。

总之，通过覆盖率来补充测例，非常有助于完善测试样例。