白盒测试评估报告

版本 1.0

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 15 /04/2022 | 1.0 | 白盒测试报告 | 康艺潇，林舒怀，林祺龙，徐惠东 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 范围 4

1.3 定义、首字母缩写词和缩略语 4

1.4 参考资料 4

1.5 概述 4

2. 测试过程明细 5

2.1 基于DD路径的测试方法说明 5

2.2 登录、二维码相关功能模块的测试 5

2.3 用户管理功能模块的测试 9

2.4 Unity 的白盒测试 12

3. 测试结果概述 13

4. 测试结果展示图 13

5. 小组成员分工及贡献 15

测试评估报告

# 简介

本测试报告是针对AR Firework项目的测试文档。AR Firework 使用 AR 技术来完成烟花实验。本次测试针对 Spring Boot后端的 2 个层次3个模块共 34 个函数及 Unity 前端的 1个层次2个模块共 3 个函数进行测试。

本次测试为白盒测试，采用了基于路径的测试方法和基于数据流的测试方法，所有测试用例按功能名存储在四个文件中，本次测试采用junit5作为测试框架，针对相应服务中存在的依赖，使用mock的方式进行处理，默认在测试该服务时，其他服务的正确性，使用IntelliJ IDEA编写和运行测试用例。

## 目的

本测试报告覆盖了针对Spring Boot 后端Controller、Service两个层次的白盒测试及Unity 前端 Dto 层次的白盒测试，在测试方法上我们实践了基于DD路径、和基于数据流的测试方法，说明了这些测试用例设计的过程、原因和得出的结论。

## 范围

本测试报告适用于“AR Firework”项目的产品，其后端基于 Spring Boot 框架开发，前端基于Unity开发，最终产品能在云服务器和本地运行。

测试范围为Spring Boot后端中 QrCodeController 和 QrCodeServiceImpl 两个层次中的登录、二维码相关功能和用户管理功能以及Unity前端中Dto层次中烟火相关功能，设计模块之间函数的调用以及函数内部的处理。

SpringBoot后端选择这两个模块的理由在于这两个模块基本涵盖了后端的所有功能。其他的模块，QrCodeRepositoryImpl 与 QrCodeServiceImpl 本质上是等同关系，因此仅挑选 QrServiceImpl 的函数即可。QrCodeDaoImpl 中各函数较为分散且代码过于简单，在本次课程实验中不具备复杂性的要求，因此在本次测试中不予以测试。

而针对上述几个模块，我们对测试的 37 个函数都采用了基于DD路径的测试方法和基于数据流的测试方法。

## 定义、首字母缩写词和缩略语

|  |  |
| --- | --- |
| 缩写词 | 含义 |
| AR Firework | 本次测试软件的名称 |
| QrCodeController, QrCodeServiceImpl | 本次测试Spring Boot后端所针对的接口类 |
| FireworkGroupDto | 本次测试Unity后端所针对的接口类 |
| 白盒测试 | 基于代码及其内在逻辑的测试方法 |
| Junit5 | 本次测试使用java语言的测试框架 |

## 参考资料

[1] Jorgensen P C. Software testing: a craftsman's approach[M]. Auerbach Publications, 2013.

## 概述

本次测试报告包含针对典型测试用例设计的说明，测试结果和覆盖率展示，以及成员分工与贡献度。

# 测试过程明细

## 2.1 基于DD路径的测试方法说明

因为基于DD路径的测试方法本身存在多种覆盖评价标准，因此我们对它们尽可能采用不同的覆盖方法进行测试，包括语句覆盖、条件覆盖、多条件覆盖、分支+循环覆盖、包含最多2次循环的所有路径、路径测试，力求实践尽可能多的测试方法和指标。

关于基于DD路径的测试方法的覆盖指标的描述如下：

* 语句覆盖，即测试用例覆盖所有执行语句；
* 条件覆盖，即测试用例覆盖每一个条件；
* 多条件覆盖，即测试用例覆盖每种条件组合；
* 循环覆盖，即测试用例覆盖不循环、1次循环、2次循环、多次循环等情况；
* 包含最多2次循环的所有路径，意义显然；
* 路径测试，即测试用例要覆盖所有可能执行的路径。

## 2.2 登录、二维码相关功能模块的测试

## 针对登录、二维码相关功能模块，我们分别采用基于DD路径的测试方法和基于数据流的测试方法。登录功能模块共有9个**主要待测函数（实际多于9个）**，分别为QrCodeController 层次中 CheckSession, Register, Login, GenerateRandomTags, GetTagOrders 和 OrCodeServiceImpl 层次中 Register 和 Login, GenerateRandomTags, GetTagOrders。

## 2.2.1 基于DD路径的测试

下面我们将以QrCodeController和QrCodeService层次中 用户登录 模块中的Login函数和条件覆盖指标为例，介绍基于DD路径的测试用例生成的过程，其它函数和用例生成的细节可见附件。

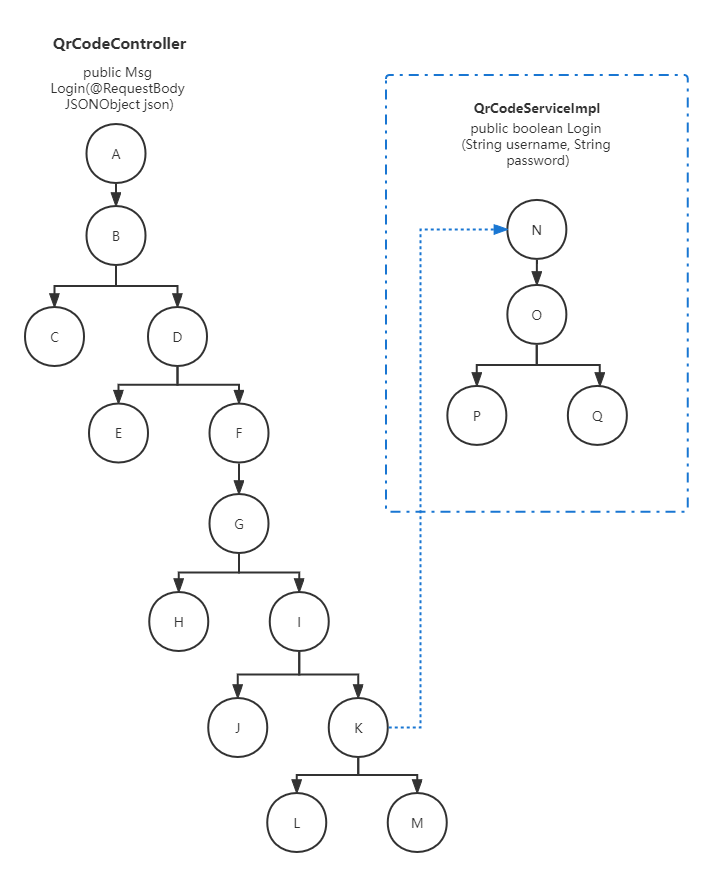
以下为QrCodeController.Login的设计过程：

1. 节点B检验Session是否为空，因此有2个条件分支。
2. 如果USERNAME为空则进入节点C返回Msg(2)，否则进入节点D。
3. 节点D检验Session中USERNAME是否为空，因此有2个条件分支。
4. 如果USERNAME不为空则直接使用Session中的USERNAME和PASSWORD登录即可，否则进入节点F通过函数参数JSON格式中获取USERNAME和Base64格式的PASSWORD。
5. 节点G将Base64格式PASSWORD解析成合法形式，如果解析失败则进入节点H返回Msg(3)，解析成功则进入节点I，因此有2个条件分支。
6. 节点I对解析后的USERNAME和PASSWORD进行合法检查，如果不合法则进入节点J返回Msg(1)，合法则进入节点K调用QrCodeService.Login检查二者匹配性。
7. 如果QrCodeService.Login返回false则进入节点L返回Msg(1)，返回true则进入节点M存入Session并返回Msg(0)。

以下为QrCodeService.Login的设计过程：

1. 节点N/O通过QrCodeController.Login传来的USERNAME和PASSWORD参数判断该用户状态是否合法，如果非法则进入节点P并返回False，否则进入节点Q并返回True。

总计11个测试用例覆盖了所有条件分支，其DD路径图如下图所示，而具体的测试用例可见附件。

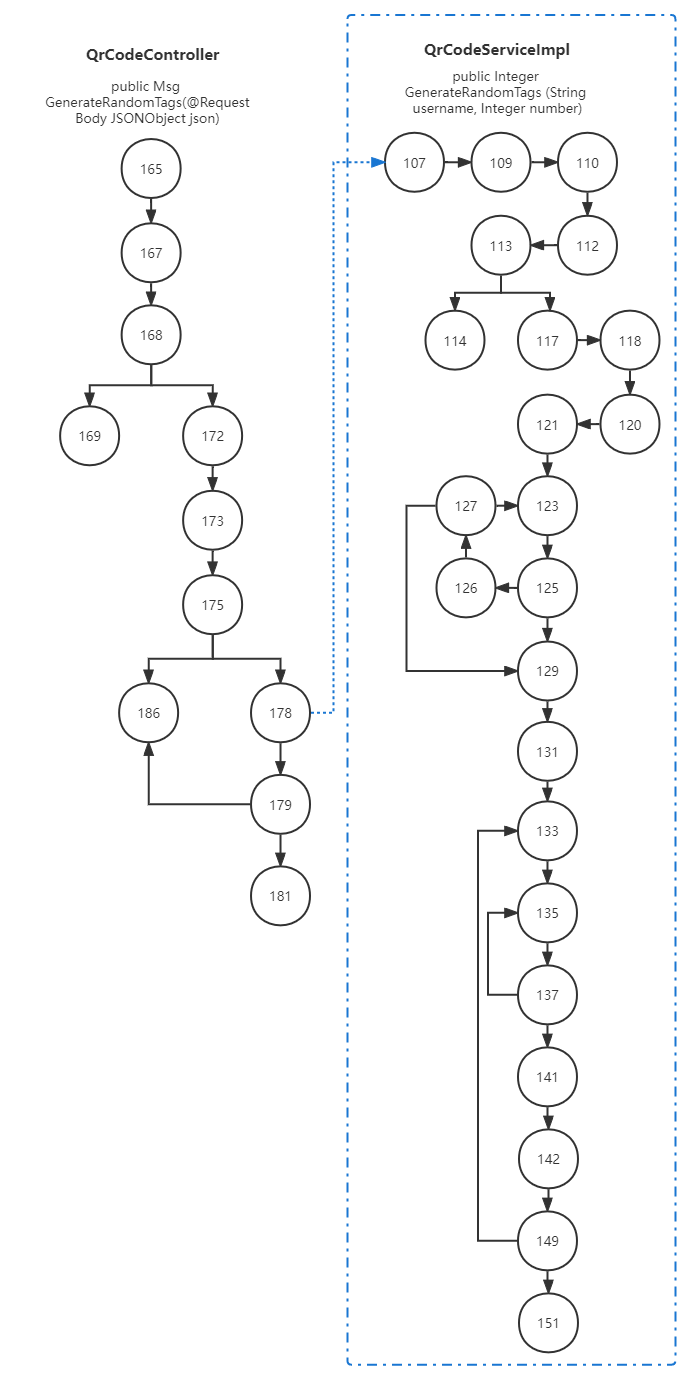


**图2.2.1 QrCodeController.Login和QrCodeService.Login的DD路径图**

## 2.2.2 基于数据流的测试

## 我们选取QrCodeController和QrCodeService层次中 用户登录 模块中的GenerateRandomTags函数为例，详细介绍基于数据流的测试用例的生成过程，而其他函数的测试用例生成过程类似则略去，具体材料可以参考附件。

1. 绘制目标函数的程序流图。



## **图2.2.2.1 QrCodeController.GenerateRandomTags和QrCodeService.GenerateRandomTags的程序流图**

## 2. 统计变量的定义/使用节点并列出变量的定义-使用路径。值得注意的是，在这个例子中定义-使用路径也是定义-清除路径，中间应该不存在新的定义。

|  |  |  |  |
| --- | --- | --- | --- |
| 变量 | 定义节点 | 使用节点 | 定义-使用路径 |
| sessionJson | 167 | 168,172 | 167-168 |
| json | 165 | 173 | 165-173 |
| username | 172 | 178 | 172-178 |
| requestNumber | 173 | 175,178 | 173-178 |
| orderId | 178 | 179,181 | 178-181 |

**表2.2.2 QrCodeController.GenerateRandomTags变量的定义/使用节点和定义-使用路径**

|  |  |  |  |
| --- | --- | --- | --- |
| 变量 | 定义节点 | 使用节点 | 定义-使用路径 |
| username | 107 | 112 | 107-112 |
| number | 107 | 123 | 107-123 |
| qrCodeUser | 110,112 | 113,120 | 110-113 |
| tagOrder | 117 | 120,121,126,129,141 | 117-141 |
| tagList | 118 | 127,129,135 | 118-135 |
| i | 123 | 123 | 123 |
| tag | 125 | 126,127 | 125-127 |
| tag | 135 | 137 | 135-137 |
| flag | 131 | 142,149 | 131-149 |
| secureRandom | 133 | 137 | 133-137 |
| res | 109 | 114,141,151 | 109-151 |

## 表2.2.2.2 QrCodeService.GenerateRandomTags变量的定义/使用节点和定义-使用路径

## 3. 选取测试覆盖指标，生成测试用例。

## 在这一步，我们使用 全使用准则 作为测试覆盖指标，即选择每一个定义节点到所有使用节点以及其后续节点的定义-清除路径。而对于所有的定义-使用路径，我们可以用尽可能少的测试用例去覆盖尽可能多的目标路径。而具体生成的测试用例，我们放在附件中予以展示，此处略去。

## 4. 编写代码，进行测试。

## 步骤 1 ~ 3 就是基于数据流的测试，接下来我们可以编写代码，进行测试。值得一提的是，因为对于数据流测试的测试覆盖指标本身就是我们测试用例生成的依据，因此按照全使用准则，我们的覆盖率必然是100%，但是在其他覆盖率（比如条件覆盖率、分支覆盖率等）语境下，覆盖率可能会有所下降。

结束了对数据流测试的用例生成过程的介绍，接下来是实践过程中，我们对于数据流测试的感想：

相比于基于路径的测试方法，数据流测试方法的实践过程更加繁琐，一旦代码量增加，或者是出现循环、变量多个定义/使用，我们就很难用手工的方式来实现。而书上也提及，目前针对数据流测试的商业软件很少，因此这种白盒测试方法似乎显得比较无用。我们认为，基于数据流的测试方法应该是作为辅助的角色，帮助基于路径的测试方法，提供更多样化的测试用例，提高测试覆盖率的效果。

## 2.3 用户管理功能模块的测试

## 针对用户管理功能模块，我们分别采用基于DD路径的测试方法和基于数据流的测试方法。用户管理功能模块共有10个**主要待测函数（实际多于10个）**，分别为QrCodeController 层次中GetUserList, DeleteUser, AuthUser, BlockUser, UnBlockUser和 QrCodeServiceImpl 层次中GetUserList, DeleteUsers, AuthUsers, BlockUsers, UnBlockUsers。

## 2.3.1 基于DD路径的测试

下面我们将以QrCodeController和QrCodeService层次中 用户管理 模块中的AuthUser函数和路径覆盖指标为例，介绍基于DD路径的测试用例生成的过程，其他函数和用例生成的细节可见附件。

以下为QrCodeController .AuthUser的设计过程：

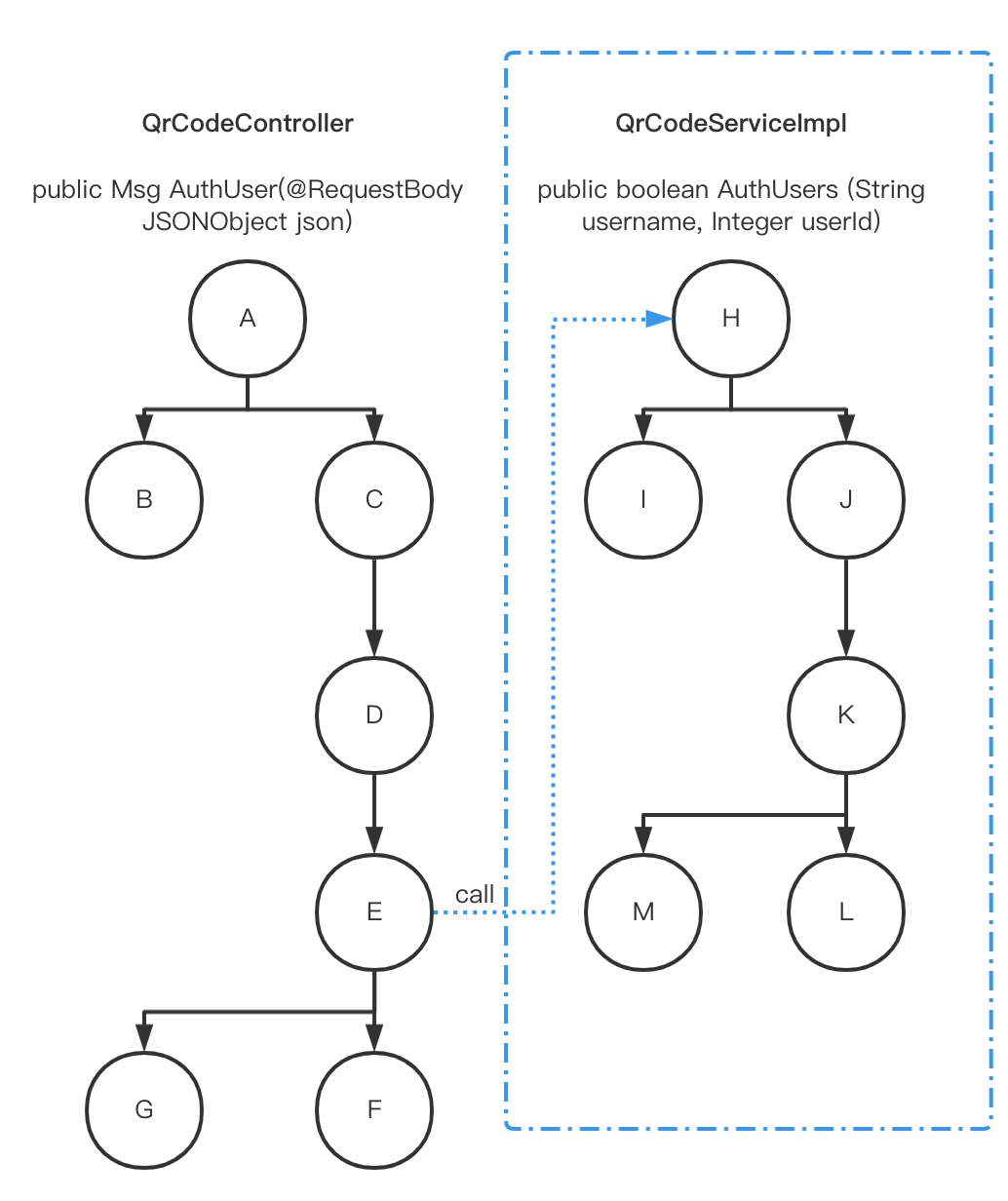
1. 节点A检验Session中USERNAME是否为空，因此需要有2种组合。
2. 如果USERNAME为空则进入节点B直接返回MSG(2)，否则进入节点C。
3. 节点D中则从Session中获取USERNAME和USERID。
4. 节点E进入QrCodeService.AuthUser检查USERNAME和USERID是否匹配，需要2种组合。
5. 如果USERNAME和USERID不匹配则进入G直接返回MSG(1)，否则进入F直接返回MSG(0)。

以下为ArCodeService.AuthUser的设计过程：

1. 节点H通过QrCodeController.AuthUser 传来的USERNAME和USERID参数判断该用户类型是否合法。如果非法则进入节点I返回False，否则进入节点J获取用户信息。

2. 节点K进一步判断该操作是否合法。如果非法则进入节点M返回False，否则进入节点L进行用户授权并返回true。

总计 6 个测试用例覆盖了所有可能执行的路径，其DD路径图如下图所示，而具体的测试用例可见附件。

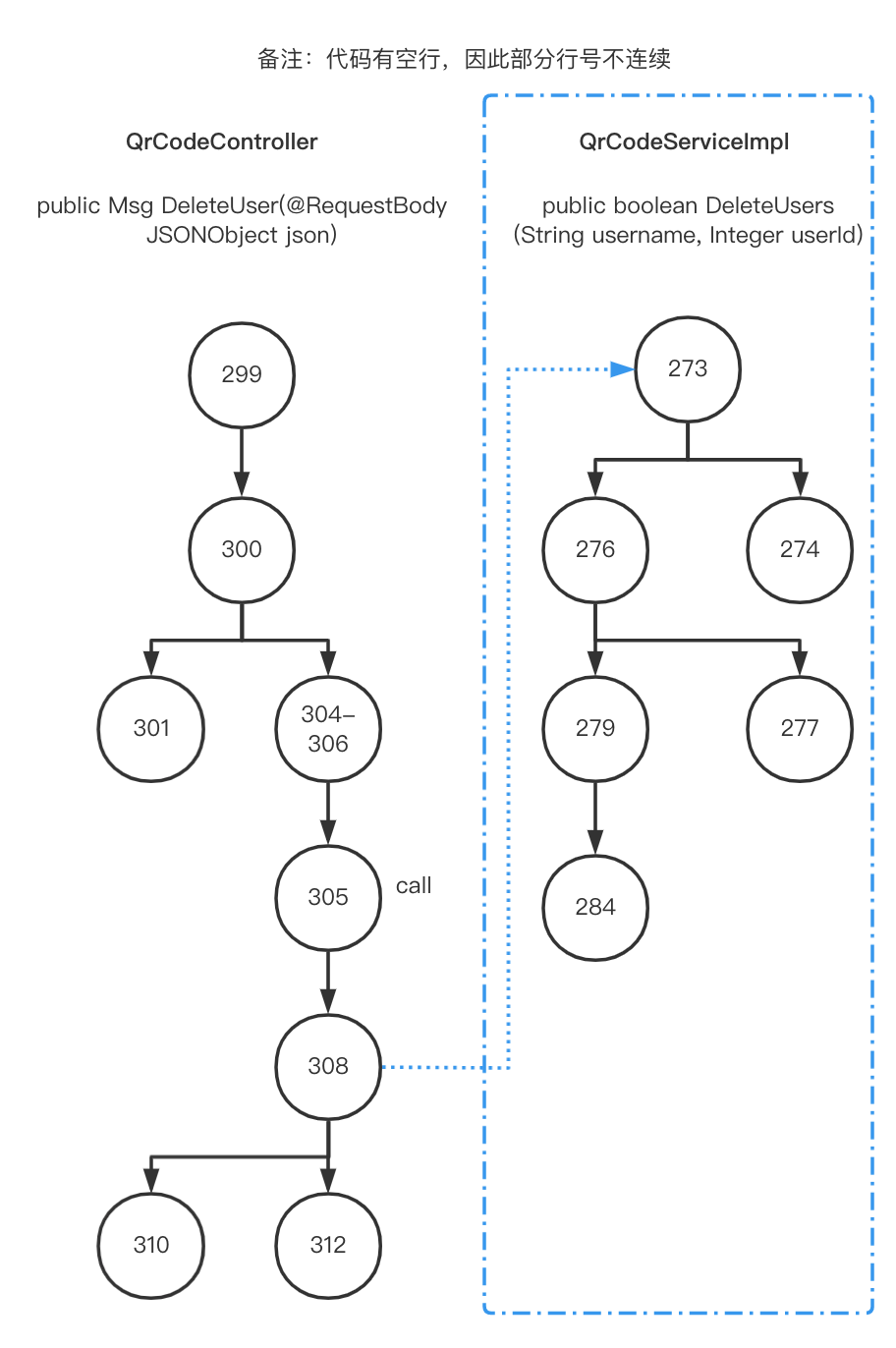


**图2.3.1 QrCodeController.AuthUser 和 QrCodeService.AuthUser的DD路径图**

## 2.3.2 基于数据流的测试

## 我们选取QrCodeController和QrCodeService层次中 用户管理 模块中的DeleteUsers函数为例，详细介绍基于数据流的测试用例的生成过程，而其他函数的测试用例生成过程类似则略去，具体材料可以参考附件。

1. 绘制目标函数的程序流图。



## **图2.3.2.1 QrCodeController.DeleteUsers和QrCodeService.DeleteUsers的程序流图**

## 2. 统计变量的定义/使用节点并列出变量的定义-使用路径。值得注意的是，在这个例子中定义-使用路径也是定义-清除路径，中间应该不存在新的定义。

|  |  |  |  |
| --- | --- | --- | --- |
| 变量 | 定义节点 | 使用节点 | 定义-使用路径 |
| json | 297 | 306 | 297-306 |
| sessionJson | 299 | 304 | 299-304 |
| username | 304 | 308 | 304-308 |
| userId | 306 | 308 | 306-308 |

**表2.3.2.1 QrCodeController.DeleteUsers变量的定义/使用节点和定义-使用路径**

|  |  |  |  |
| --- | --- | --- | --- |
| 变量 | 定义节点 | 使用节点 | 定义-使用路径 |
| username | 270 | 273 | 270-273 |
| userId | 270 | 276,279 | 270-279 |

## 表2.2.2.2 QrCodeService.DeleteUsers变量的定义/使用节点和定义-使用路径

## 3. 选取测试覆盖指标，生成测试用例。

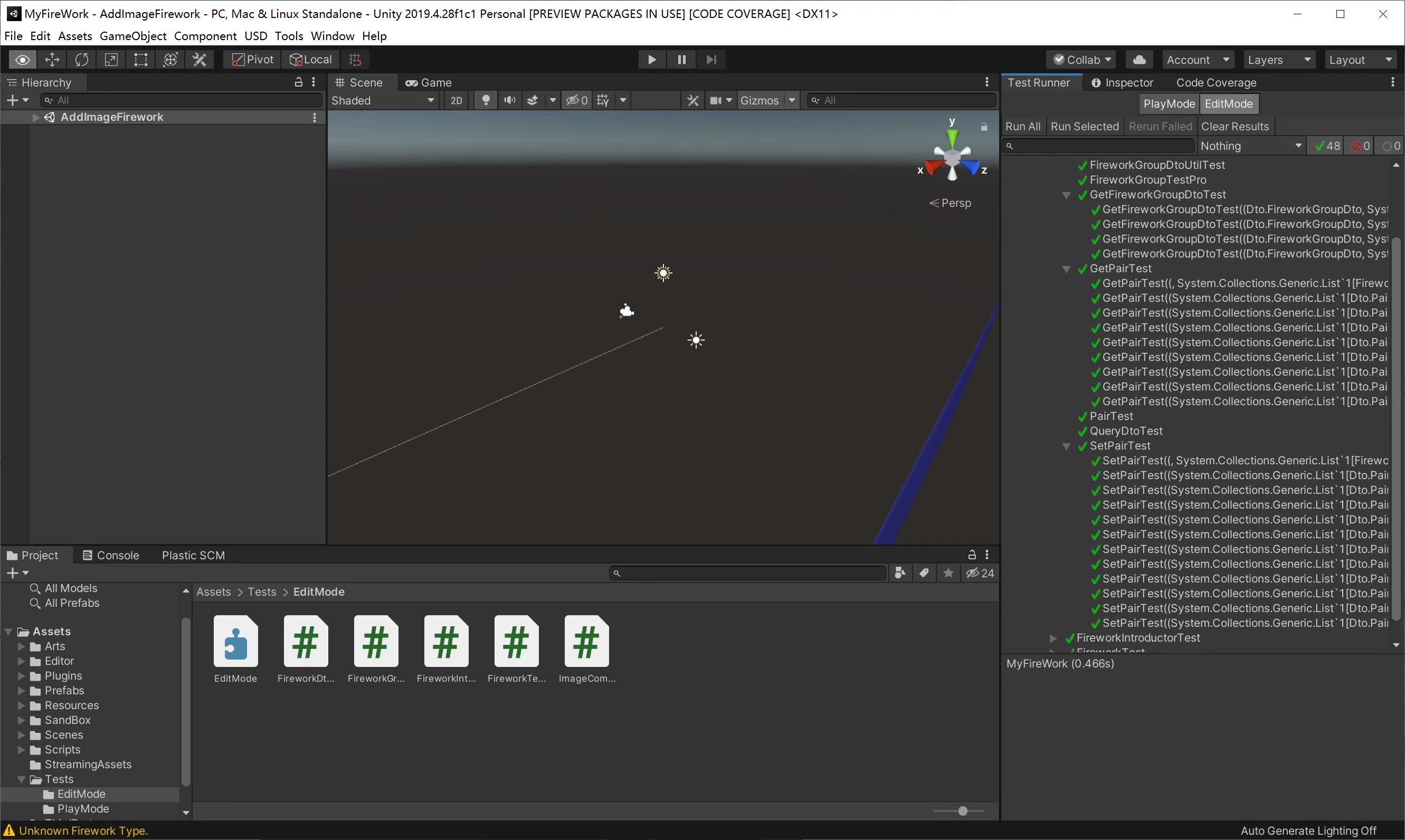
## 在这一步，我们依旧使用 全使用准则 作为测试覆盖指标，即选择每一个定义节点到所有使用节点以及其后续节点的定义-清除路径。而对于所有的定义-使用路径，我们可以用尽可能少的测试用例去覆盖尽可能多的目标路径。而具体生成的测试用例，我们放在附件中予以展示，此处略去。

## 4. 编写代码，进行测试。

## 步骤 1 ~ 3 就是基于数据流的测试，接下来我们可以编写代码，进行测试。值得一提的是，因为对于数据流测试的测试覆盖指标本身就是我们测试用例生成的依据，因此按照全使用准则，我们的覆盖率必然是100%，但是在其他覆盖率（比如条件覆盖率、分支覆盖率等）语境下，覆盖率可能会有所下降。

## 2.4 Unity的白盒测试

我们还选取了 Unity 中的 Dto.FireworkGroupDto 和 Dto.FireworkGroupDtoUtils 类中的3个函数，同样使用基于DD路径的测试方法和基于数据流的测试方法，下面是测试结果的截图，具体的测试用例和测试结果详见附件。



**图 2.4.1 Unity 白盒测试**

# 测试结果概述

本次测试的主要对象是“AR Firework”项目，主要针对项目中的用户登录和二维码相关功能、用户管理这2个方面的功能进行白盒测试，主要使用了基于DD路径的测试方法和基于数据流的测试方法，共进行了106个测试用例的测试，其中106个测试用例通过，通过率为100%。

总体来说，本次测试较好地覆盖了项目需求中能够用作白盒测试的方法，测试用例的设计符合要求测试方法的思想，测试计划得到了较好的执行，并且通过测试结果能够对项目的缺陷给出详细的方向和建议，较好地起到了白盒测试的作用。

# 测试结果展示图

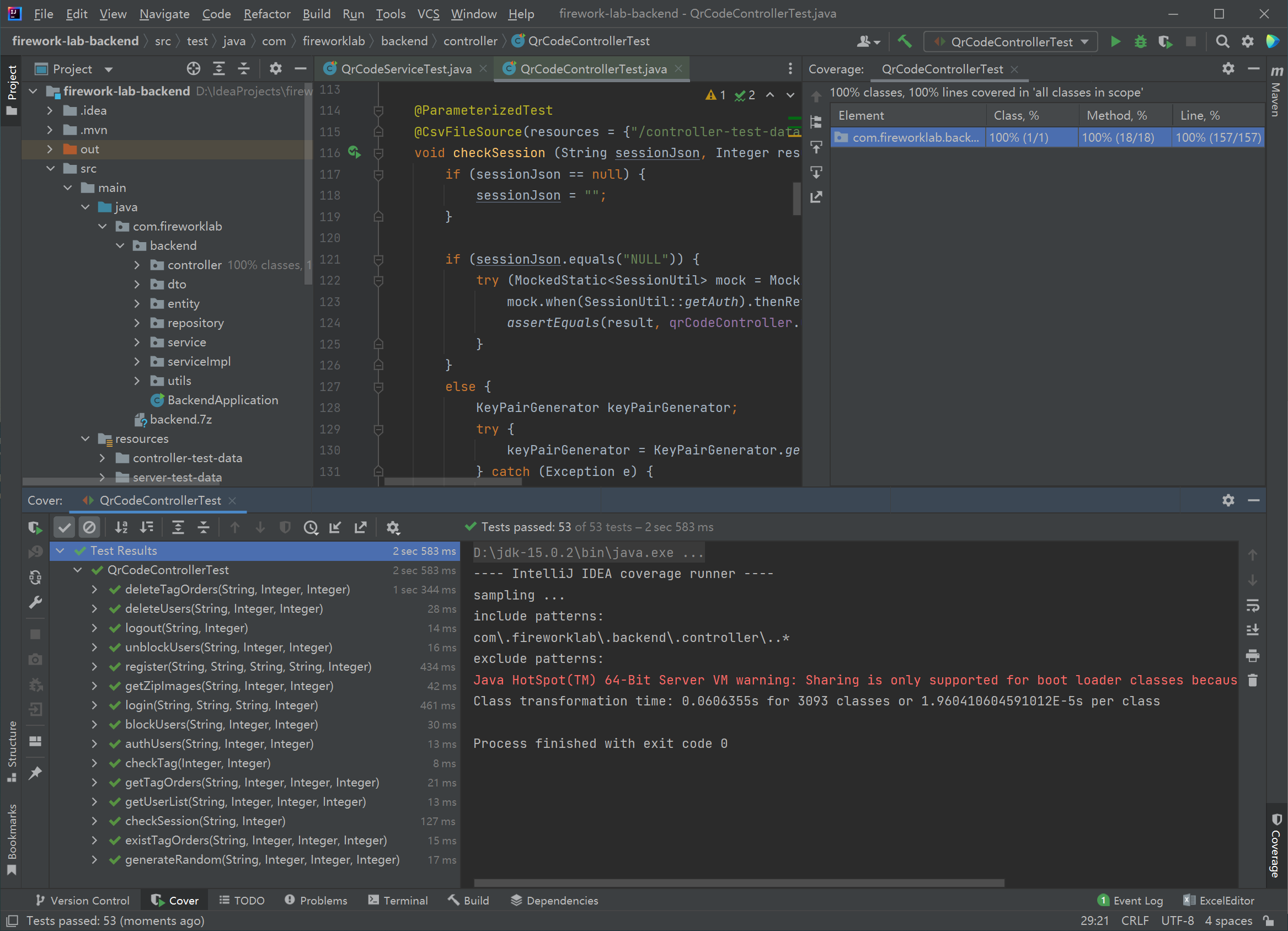


图4.1.1 QrCodeController白盒测试结果展示图

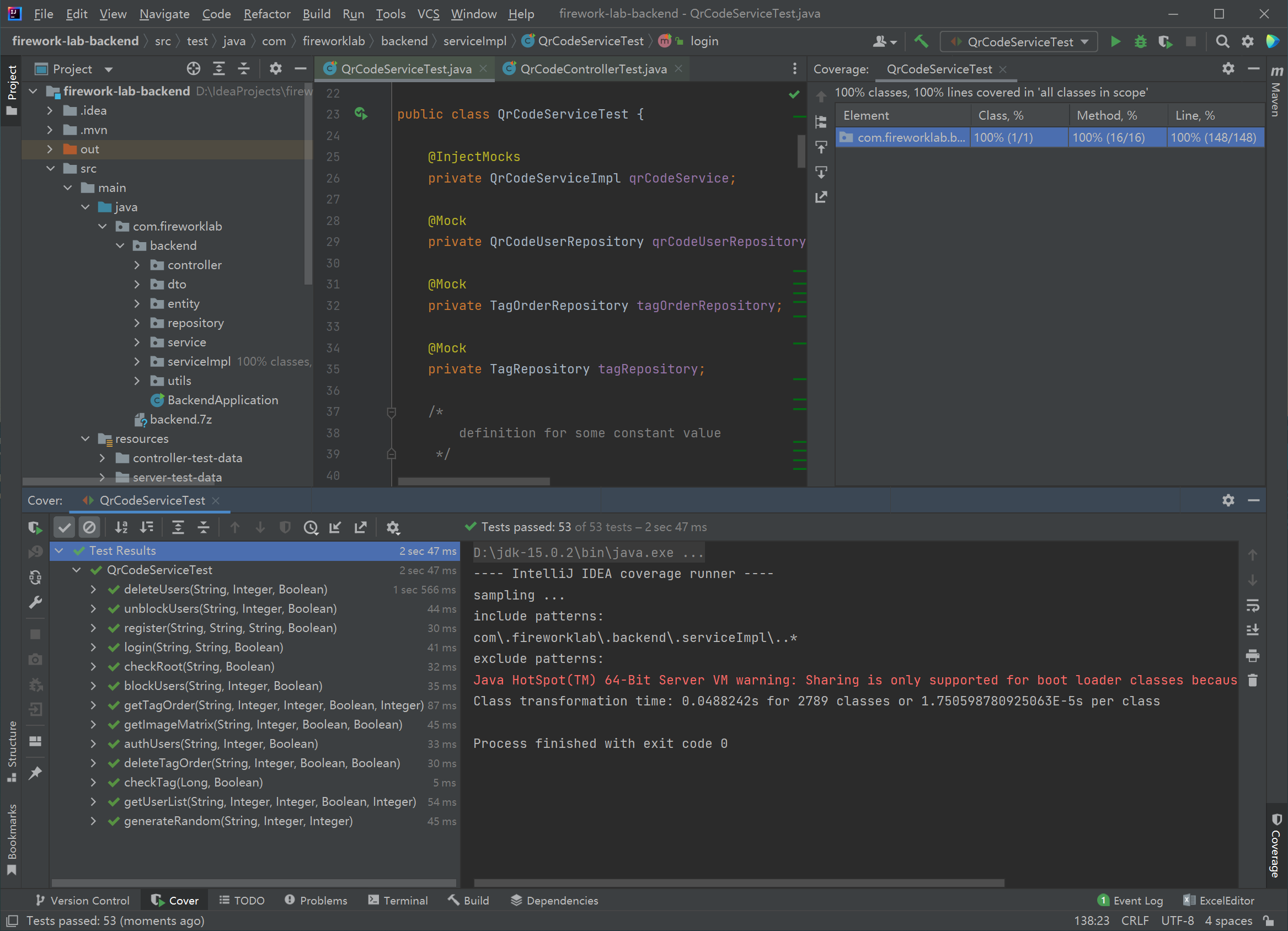


图 4.1.2 QrCodeService 白盒测试结果展示图

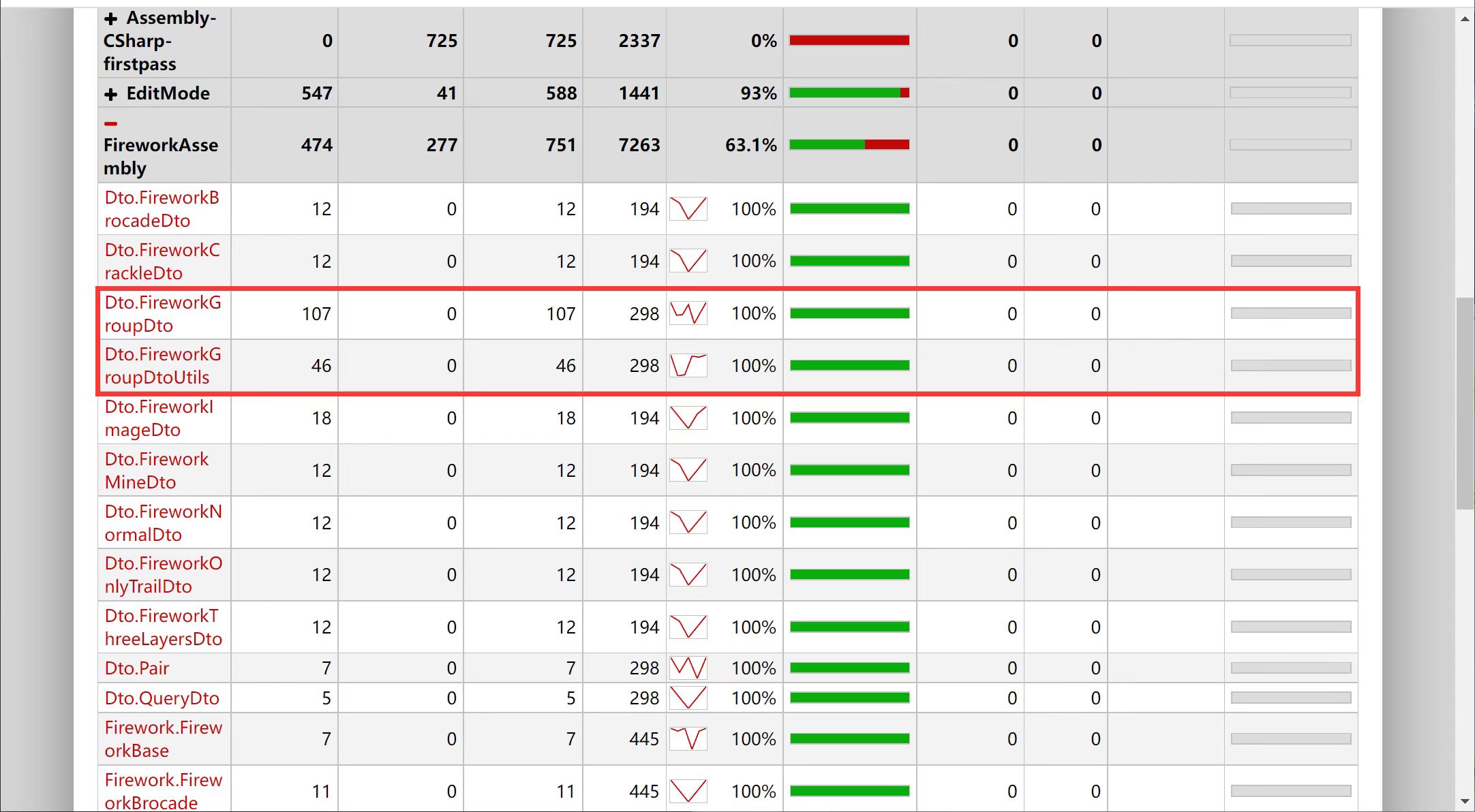


图 4.1.3 Unity 白盒测试

# 小组成员分工及贡献

本次白盒测试由林祺龙、康艺潇、林舒怀和徐惠东合作完成，康艺潇和林舒怀主要负责撰写测试计划和设计测试用例，林祺龙主要负责代码测试，徐惠东主要负责撰写测试报告，大家工作之间相互也有所交叉，最终齐心协作完成了 “AR Firework” 项目的白盒测试。

贡献度分别为 林祺龙 25%，康艺潇 25%，林舒怀 25%，徐惠东 25%。

注：第一次黑盒测试作业的贡献度是 林祺龙 25%，康艺潇 25%，林舒怀 25%，徐惠东 25%。