**软件测试实验报告**

姓名：叶小凡 学号：221220043

**1 C++ 单元测试**

**1.1 实验环境**

操作系统：Windows 10

系统类型：x64-based PC

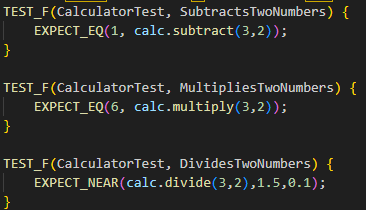
编译器：Visual Studio 2022

**1.2 实验内容**

**1.2.1 创建第一个单元测试**

▪ 实验过程

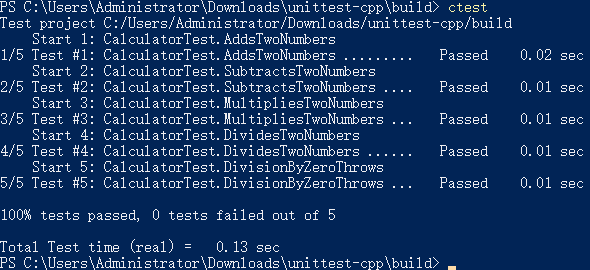
①完成 SubtractsTwoNumbers, MultipliesTwoNumbers, DividesTwoNumbers 的测试用例。（ DividesTwoNumbers 由于精度问题使用 EXPECT\_NEAR断言）。



②构建DivisionByZeroThrows测试用例，验证除以0是能否触发异常。



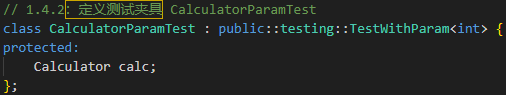
▪ 运行结果：测试通过√



**1.2.2 值参数化测试**

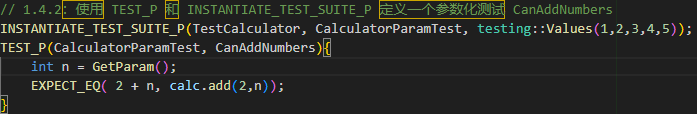
▪ 实验过程

①定义新测试夹具 CalculatorParamTest。

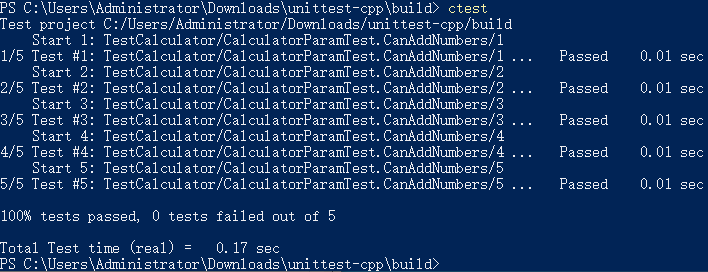


②使用 INSTANTIATE\_TEST\_SUITE\_P实例化该测试套件，使用五个参数[1,2,3,4,5]。

使用TEST\_P定义CanAddNumbers用例，使用GetParam()调用一个参数，用EXPECT\_EQ测试add方法。



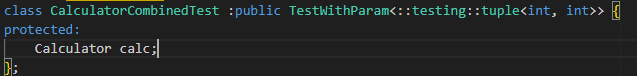
▪ 运行结果：测试通过√



**1.2.3 在值参数化测试中考虑组合**

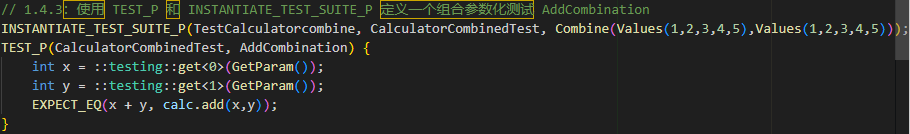
▪ 实验过程

①定义一个新的测试夹具 CalculatorCombinedTest，参数用两个 int 组成的 tuple。

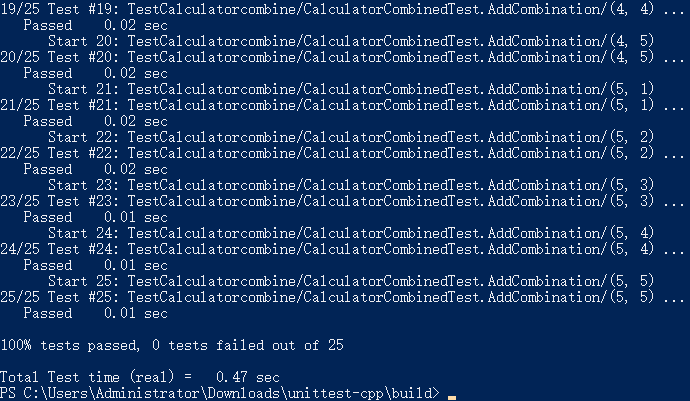


②使用 INSTANTIATE\_TEST\_SUITE\_P实例化该测试套件，使用 Combine 组合两个 Values 参数集[1,2,3,4,5]。

使用TEST\_P 定义 AddCombination 用例，使用 GetParam() 调用两个参数，测试 add 方法，并使用 EXPECT\_EQ 作为断言。



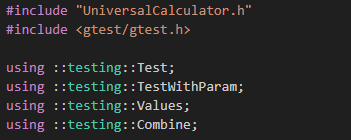
▪ 运行结果：测试通过√



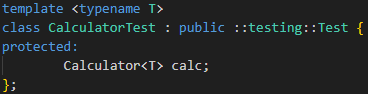
**1.2.4 类型参数化测试**

▪ 实验过程——创建测试文件 universalCalculator\_test.cc

①头文件等



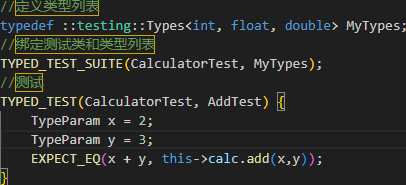
②定义测试套件 CalculatorTest



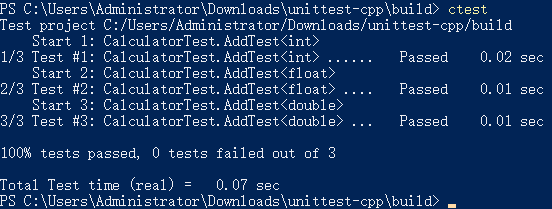
③定义类型列表 MyTypes，包含要测试的类型 int, float, double。

使用 TYPED\_TEST\_SUITE 绑定测试类和类型列表。

使用TYPED\_TEST编写测试逻辑。使用 TypeParam 引用类型参数，使用 EXPECT\_EQ 测试。



▪ 运行结果：测试通过√



**1.3 思考与体会**

▪ **问题1**：上述内容使用的均为 EXPECT 断言。那么，在什么场景下适合使用 ASSERT 断言？

答：如果ASSERT断言失败，测试将会停止执行，有利于快速定位错误，但会影响到测试的覆盖范围；如果EXPECT 断言失败，测试仍将继续执行，有利于获得更大测试覆盖范围，但可能导致测试过于宽松，无法确保测试正确性。

因此，选择哪种类型的断言取决于具体需求。想要快速定位并停止测试的场景下适合用ASSERT断言。

▪ **问题2**：如果输入参数的全排列面临组合爆炸问题，你有什么策略减少测试用例数量？

答： 可以采用等价类划分的策略。将输入参数划分为等价类，选择一个代表性参数进行测试，减少测试用例数量。

▪ **问题3**：在类型参数化测试中，如何处理类型之间行为的差异？

答：可以使用基类和多态的方法。所有特定类型的实现都继承自这个基类，而在具体的子类中实现各类型的特殊逻辑，从而在运行时可以根据具体类型执行相应方法，良好地处理类型之间行为的差异。

**2 Java 单元测试**

**2.1 实验环境**

操作系统：Windows 10

系统类型：x64-based PC

编译器：Visual Studio 2022

JDK版本： OpenJDK 21

**2.2 实验内容**

**2.2.1 等价类划分**

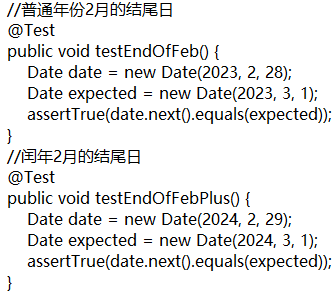
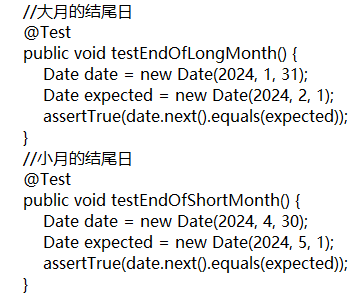
▪ 实验过程

为测试Data类中的Next()方法合理划分等价类

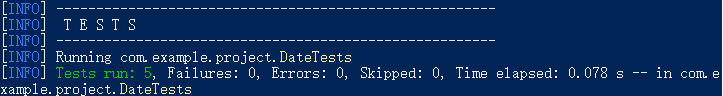
*思路*：首先，将需要测试的日子分为中间日和结尾日（1号也算作中间日）。

对于中间日的测试，不分月份。

对于结尾日的测试，分为大月、小月和二月，其中二月又分为闰年和平年。



▪ 运行结果：测试通过√

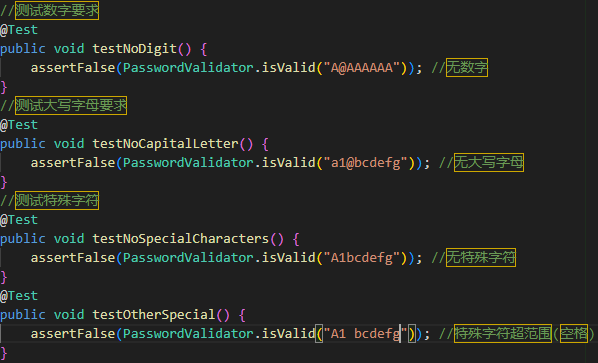


**2.2.2 边界值分析**

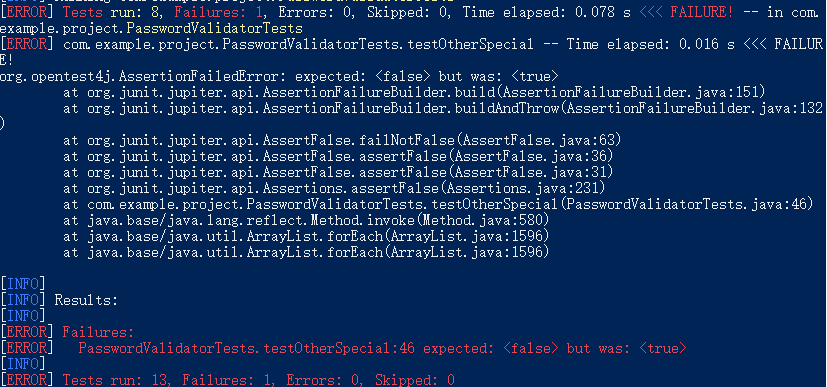
▪ 实验过程

编写测试逻辑。分别对长度、数字、特殊字符的边界情况进行测试。





▪ 运行结果：测试不通过×



测试结果表明，原函数对于特殊字符的检查存在漏洞，使用!Character.isLetterOrDigit()会导致空格等不符合要求的字符被错误的当作特殊字符。

▪ 修改原函数：可以将符合要求的特殊字符定义为一个字符集来检查。



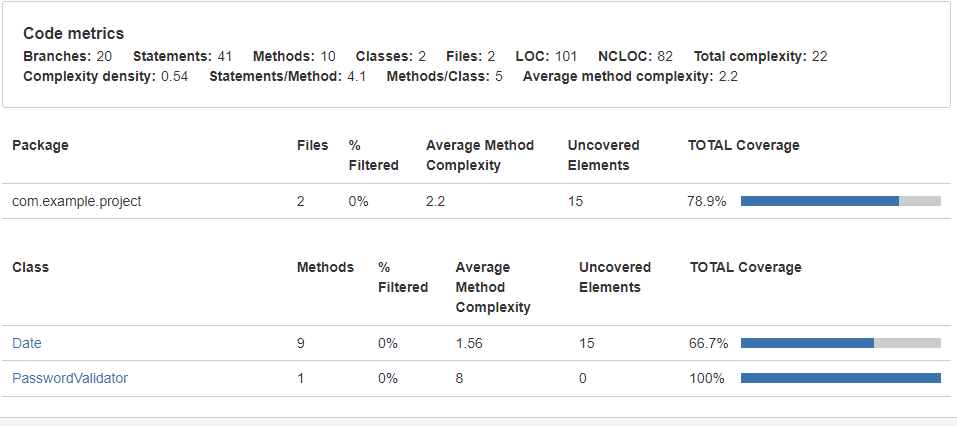


▪ 运行结果：测试通过√



**2.2.3 覆盖率度量**

mvn clover:instrument clover:clover 生成报告



▪ 各指标含义：

①Methods

该指标表示代码中所有方法的总数。它可以是测试的直接目标，例如是否测试了每个业务逻辑方法。

该指标有利于确保代码中每个重要的业务逻辑都有相应的测试用例覆盖。如果方法数量较多而测试覆盖率较低，则可能需要增加测试用例以提高代码质量。

②% Filtered

该指标表示在计算覆盖率时被过滤掉的代码部分，如第三方库、工具类或已知的低风险代码等不需要测试的代码。

该指标帮助测试者了解过滤的代码量，如果过滤率过高，可能要考虑重新评估过滤标准，确保关键功能未被忽视。

③ Average Method Complexity

该指标衡量方法的复杂性，常用的度量标准是圈复杂度（Cyclomatic Complexity），它计算了代码的独立路径数量。

较高的复杂度意味着方法可能更难以理解和测试，容易引入错误。它可以作为重构的一个指标，鼓励开发者简化复杂的逻辑，也有利于测试者集中精力对复杂度高的部分进行更全面的测试。

④ Uncovered Elements

该指标代表未被测试覆盖的代码元素，可能是方法、分支或行。

揭示了哪些部分的代码没有被测试，帮助团队识别测试的盲点。

⑤Total Coverage

该指标代表了代码中被测试用例覆盖的总比例。通常是被覆盖的行数或分支数除以总的行数或分支数。

总覆盖率是评估测试质量的一个直接指标。较高的覆盖率通常意味着代码质量较高，错误风险较低。

▪ 本报告中，各数据显示，对于Data的测试，测试方法复杂度不高，但覆盖率较低，这表面我需要完善Data的测试逻辑，优化测试用例；对于Password的测试，方法复杂度稍高，但覆盖率较高，错误风险较低，暂时可以不用修改。

**1.3 思考与体会**

▪ **问题1**：在设计等价类测试用例时，如何确保测试用例的代表性？

答：覆盖所有等价类：确保每个有效等价类和无效等价类至少有一个代表性的测试用例。

边界值测试：特别关注边界值，因为边界值往往是引发错误的关键点。对于每个边界值，设计专门的测试用例。

特殊情况测试：除了边界值外，还应考虑特殊情况，如空值、特殊字符、异常数据等，并设计相应的测试用例。

▪ **问题2**：为什么边界值容易出错？边界值分析和等价类划分如何结合使用？

答：不同数据类型在边界值处理上可能存在不同限制和行为；软件开发人员在处理边界情况时也可能遗漏条件或设定错误。

进行测试时可以先运用等价类划分的策略大致划定测试范围，然后选取不同等价类的边界值进行测试，简化测试流程。

**3 Web 测试**

**3.1 实验环境**

操作系统：Windows 10

系统类型：x64-based PC

编译器：Visual Studio 2022

JDK版本： OpenJDK 21

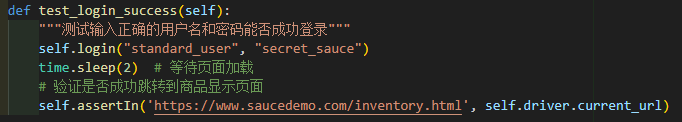
Python版本：python 3.12.7

**3.2 实验内容**

**3.2.1 基于unittest编写测试用例**

**▪ 登录功能**

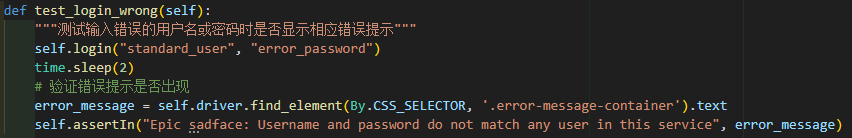
（示例）①测试输入正确的用户名和密码能否成功登录



②测试输入错误用户名或密码时是否显示相应错误提示

思路：以错误的密码登录，等待页面加载，通过assert验证错误信息是否出现。

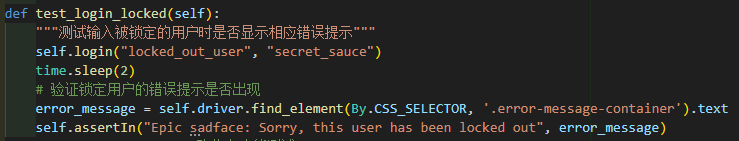
代码：



③测试输入被锁定的用户是否显示相应错误提示

思路：将登陆用户名改为“locked\_out\_user”，捕捉页面出现的错误提示，利用assert检查错误信息是否符合要求。

代码：



**▪ 购物车功能**

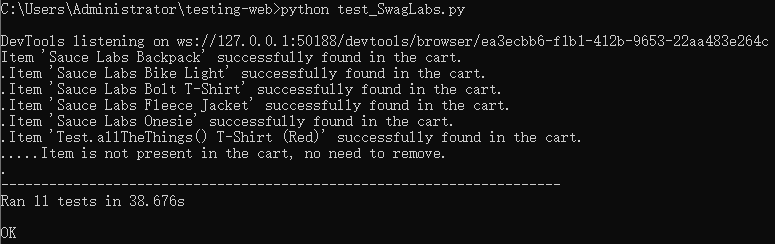
①测试添加商品到购物车功能

思路：登陆页面，重置测试状态，打开产品列表页，通过button\_id找到商品，并点击添加按钮，然后通过item\_name查找元素以验证商品是否添加到购物车。

代码：



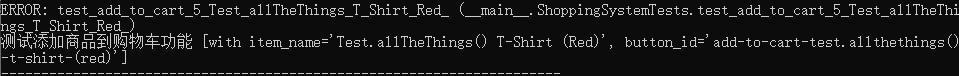
▪ 实验结果：测试通过√

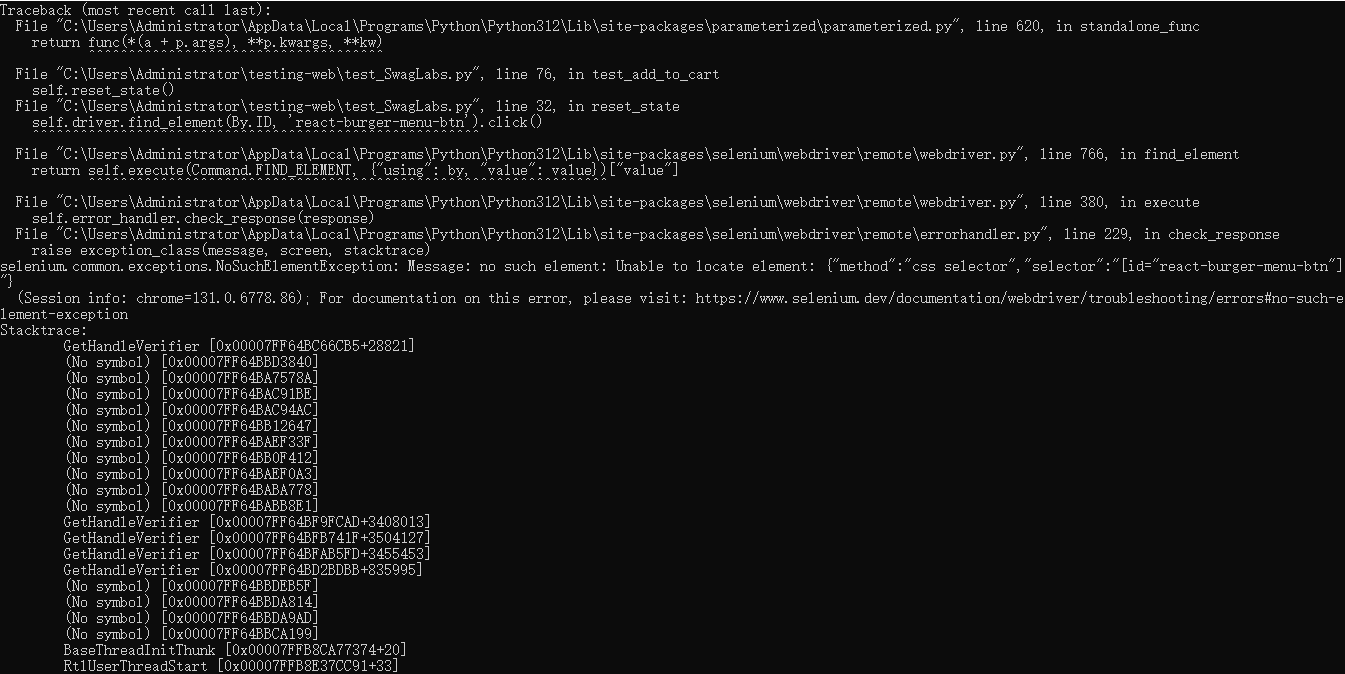


②修改test\_add\_to\_cart用户名为error\_user

实验结果：

（六个商品添加购物车测试均失败。以其中一条信息为例）

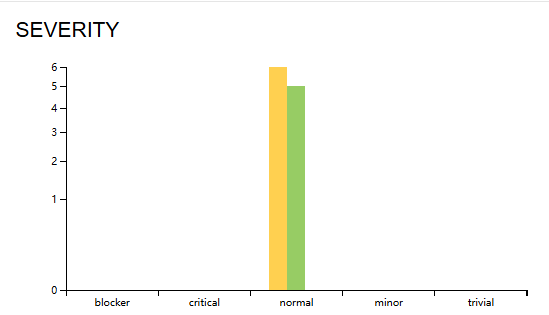
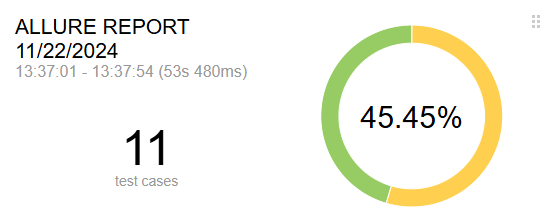


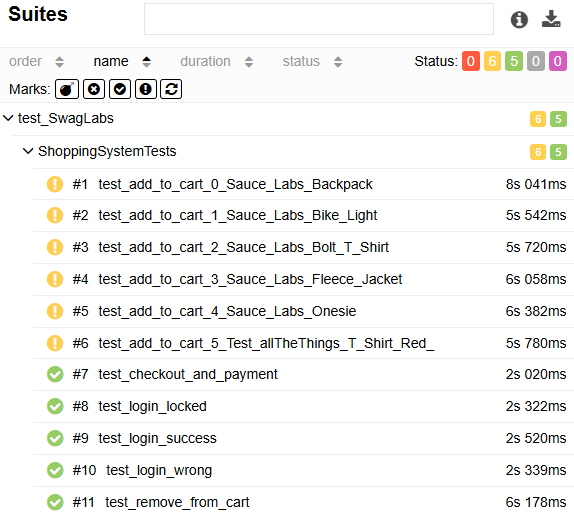


错误分析：

用户名违规，导致登陆失败,页面无法加载，购物车功能无法正常使用。登录、页面加载、按钮组件均出现故障。

**3.2.2基于Allure生成测试报告**





▪ **问题1**：请说明哪些测试用例报错了？体现了什么功能性缺陷？

答：Test\_add\_to\_cart的测试用例均报错，体现出该测试的功能性缺陷之一是未能考虑用户登录失败的情况，如果登陆失败，会影响后面的网页操作全部无法进行。可以添加更多断言检查使测试结果更清晰。

▪ **问题2**：如果在该在线购物网站中加入了优惠券系统（限定使用时间内满x元减y元），将会需要哪些额外的功能性测试？请列出具体测试用例。

答：①测试有效时间使用优惠券是否减免（包括满x元的情况和不满x元的情况）

②测试不在有效时间内优惠券是否减免

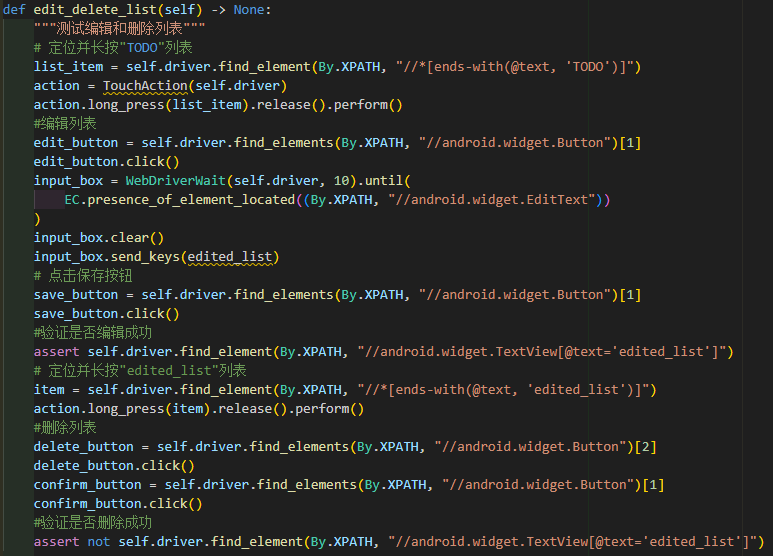
**4 移动应用测试**

**4.1 实验内容**

**4.1.1 功能性测试**

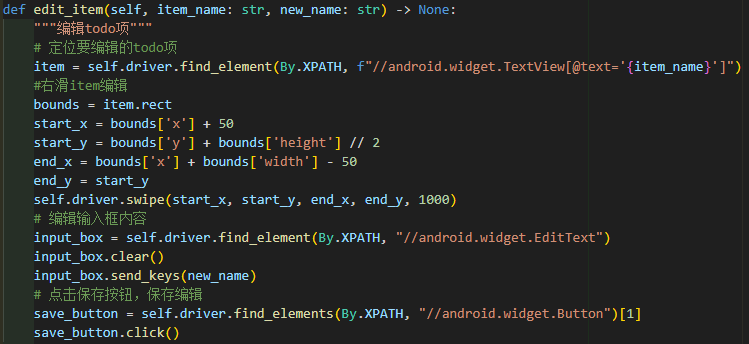
**▪ 编辑/删除列表**

思路：长按列表，并通过按钮顺位定位编辑和删除键，验证。

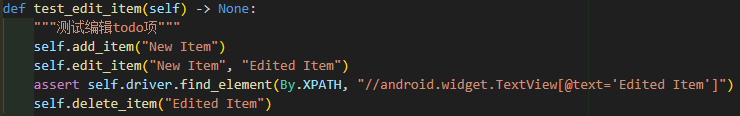


**▪ 编辑项目测试用例**

思路：模仿delete\_item思路，右滑编辑项目，完成edit\_item方法

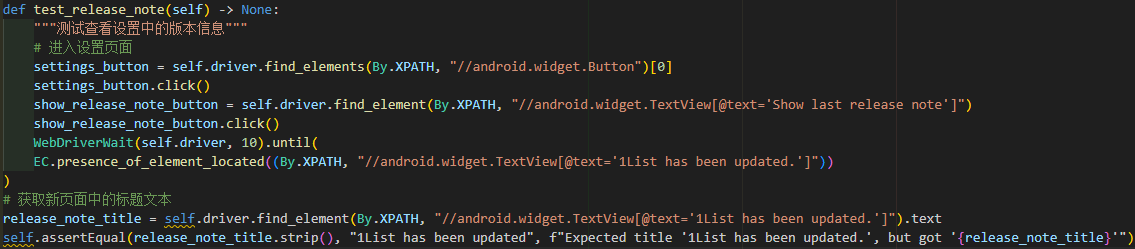


利用edit\_item测试app的编辑功能。

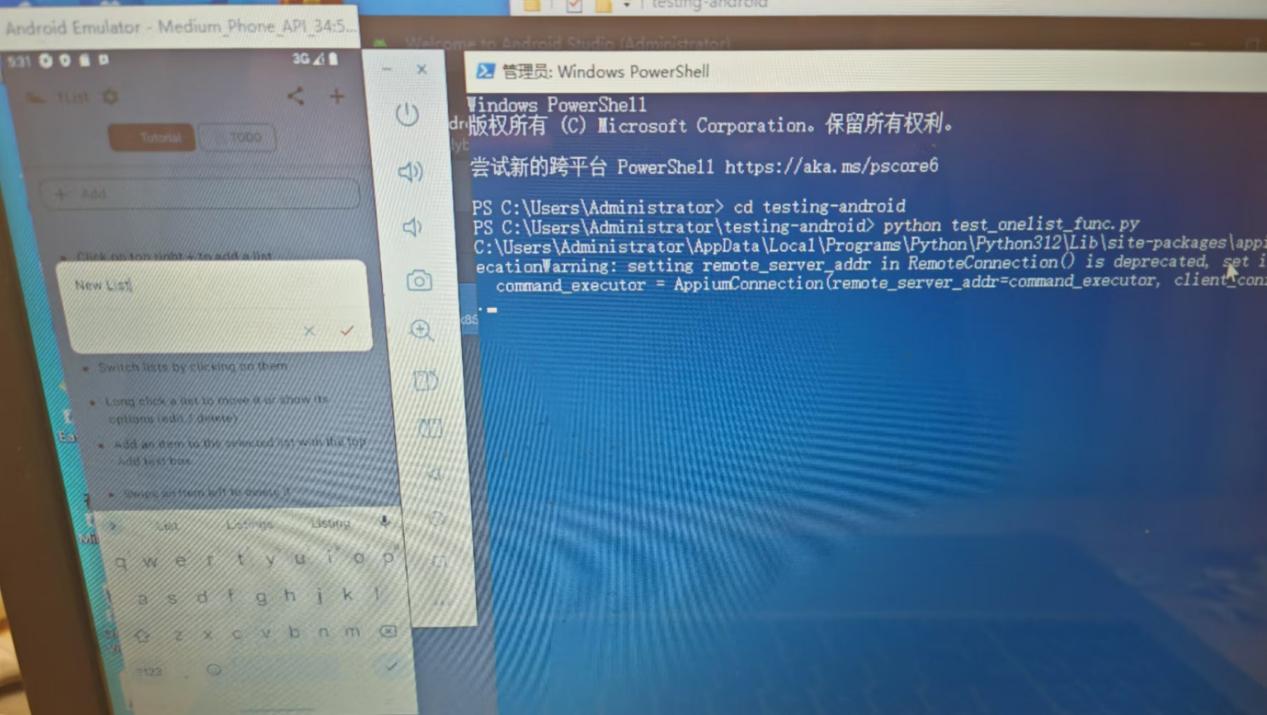


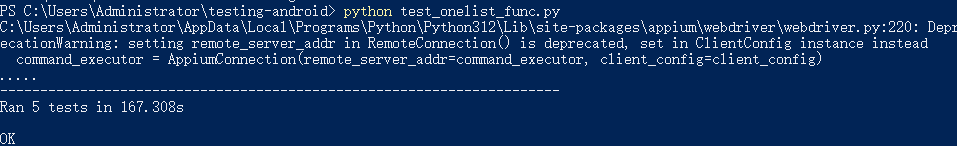
**▪ 查看版本信息测试用例**

思路：点击设置-Show last release note测试查看版本release信息的功能。



测试过程及结果：





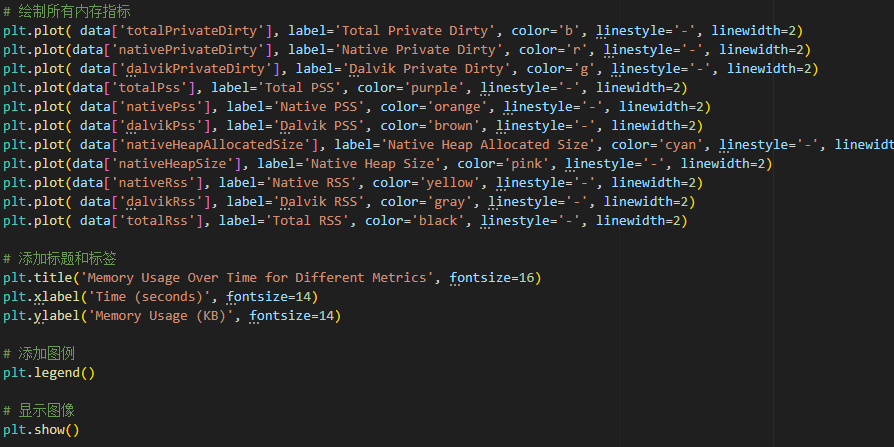
▪ **思考**：在标记项目完成测试用例 test\_mark\_item\_done 中， Appium 本身无法通过 UIAutomator2 获取文本的颜色、删除线等样式信息，此时该如何完成此测试用例？

答：如果删除线/字体颜色是由CSS样式决定的，可以尝试通过getCssValue检查样式属性。也可以考虑借助一些视觉工具。

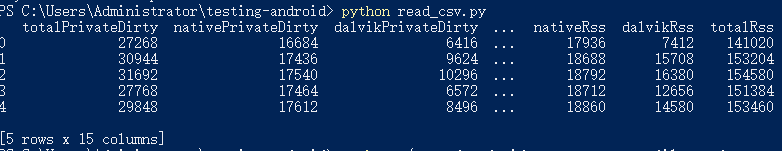
**4.1.2 非功能性测试**

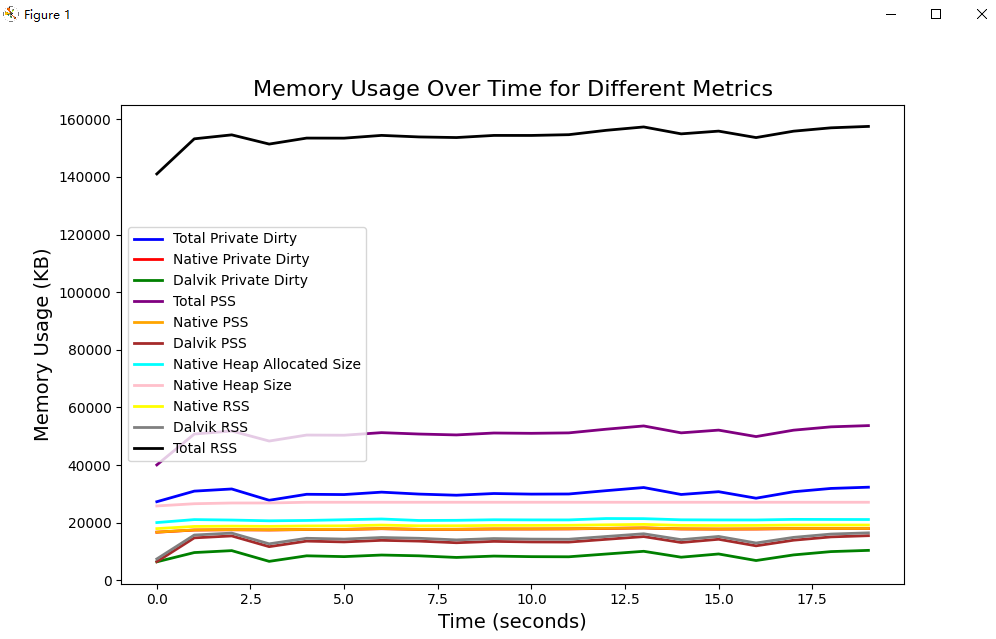
**▪ 利用Matplotlib库绘制内存指标图像**

代码：



结果：

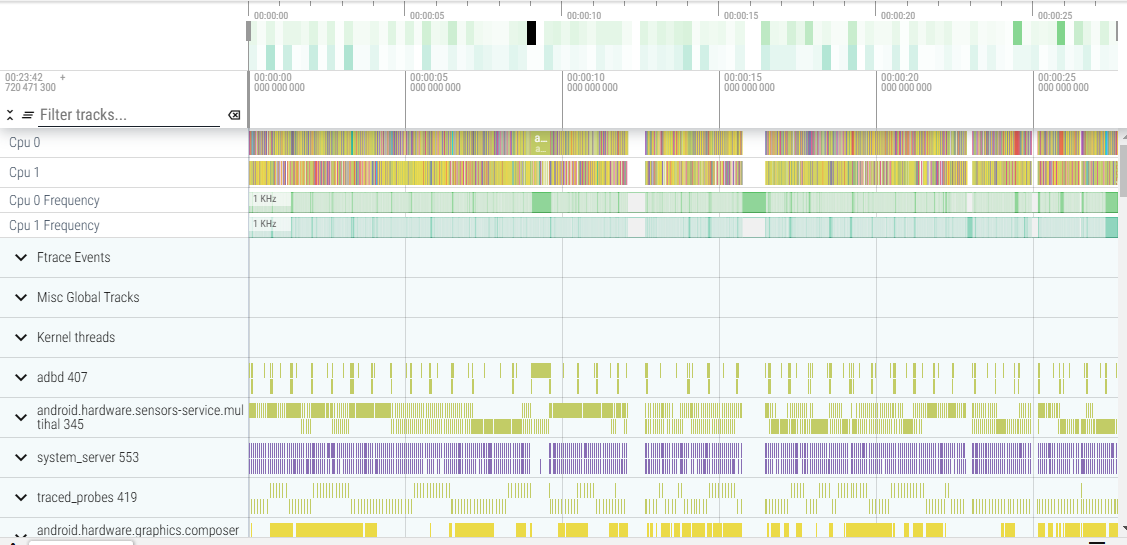


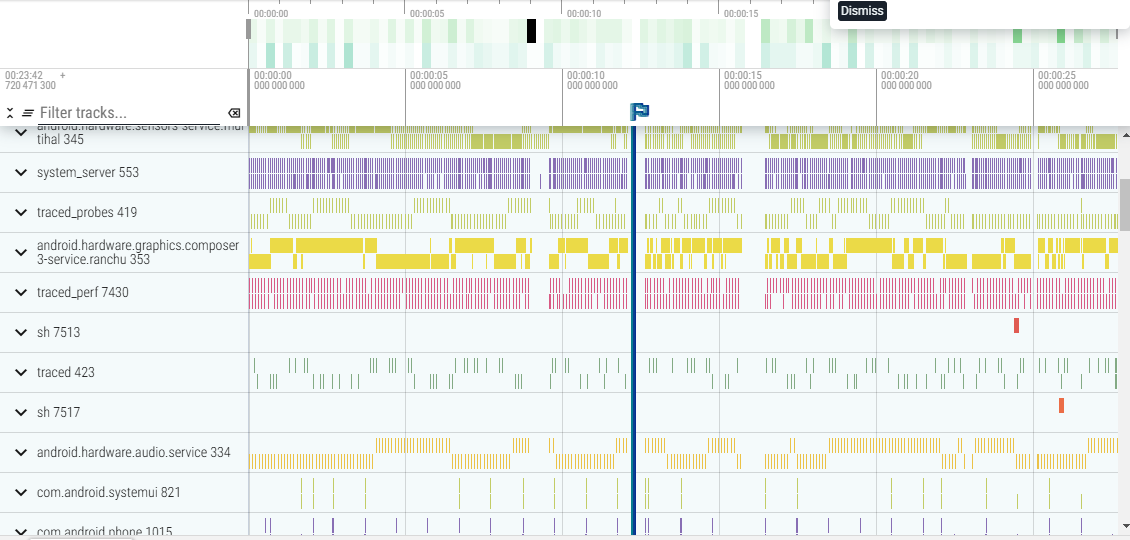


TotalRSS指标和TotalPSS指标相较之下有更为显著的上升趋势。

**▪ 利用Perfetto分析**

图像：





帧率下降的原因是多个线程争夺CPU资源。