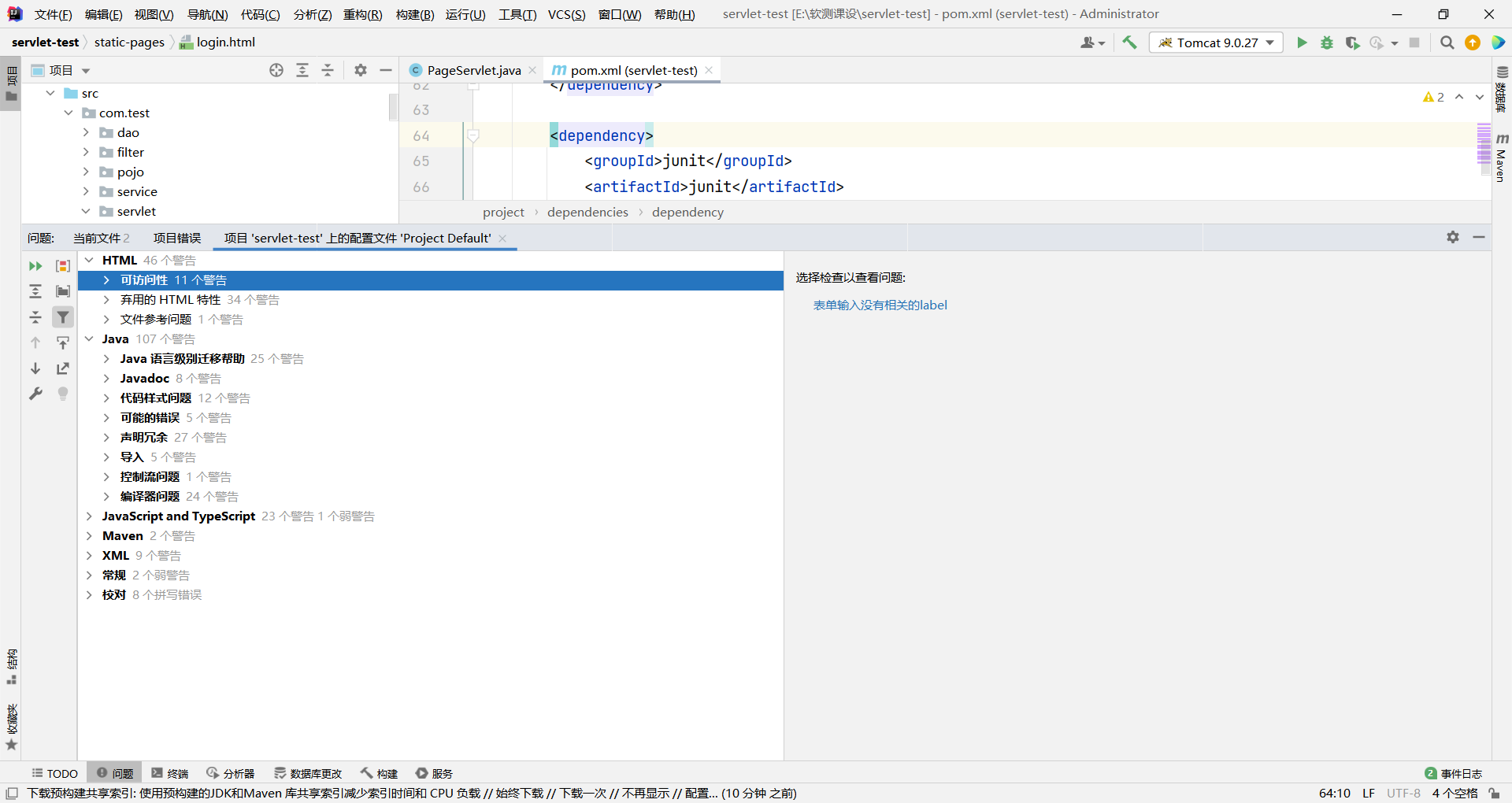
# 测试报告

本网上银行系统包括用户登录、余额查询、用户转账、查询交易记录、修改用户密码等功能。这是一个基于Java Web技术栈开发的Web应用项目，主要采用Servlet + JSP的开发模式。该测试报告包含对系统进行的以下5项测试：代码覆盖测试、单元测试、集成测试、系统测试、性能测试。

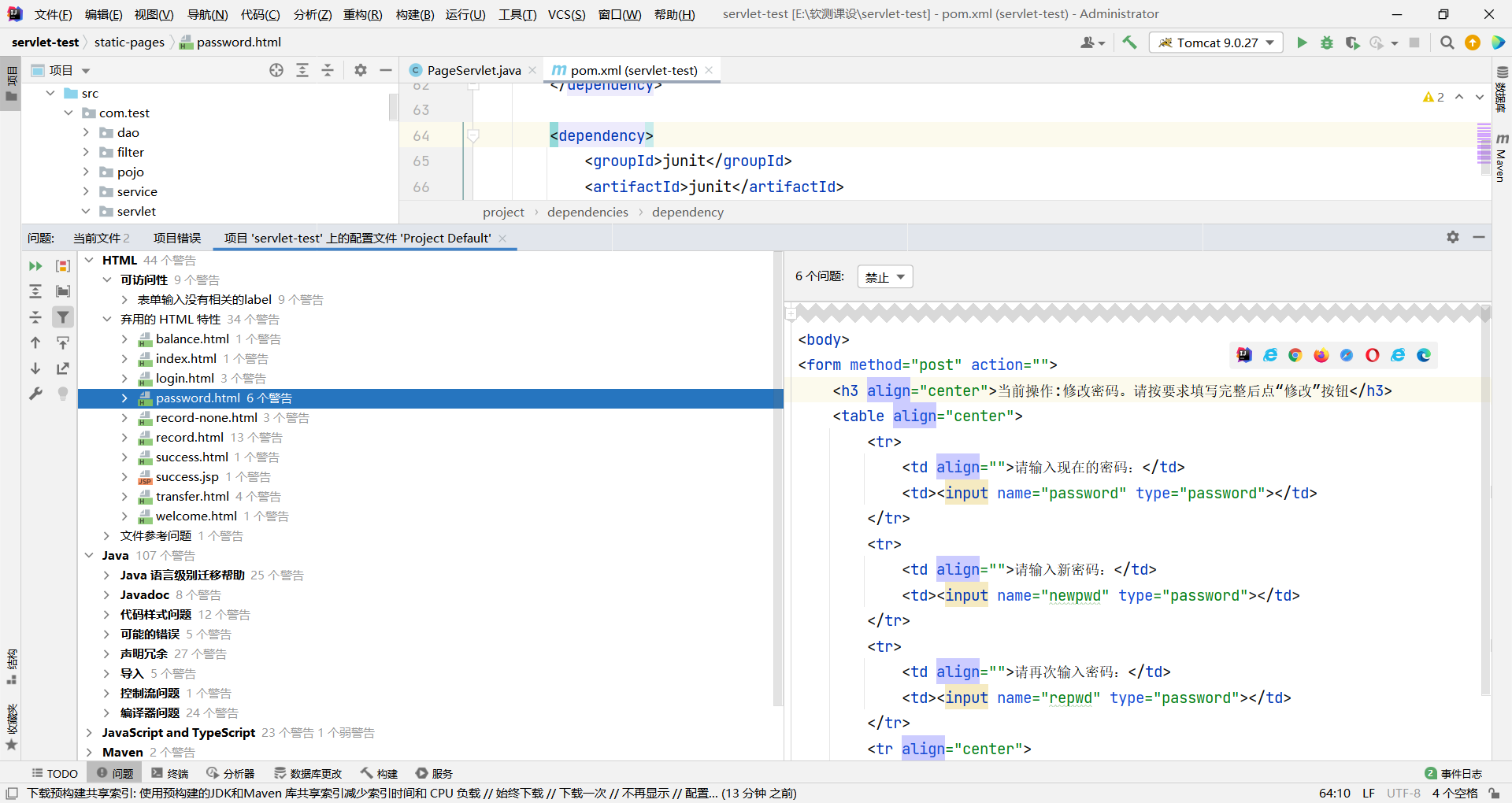
一、代码覆盖测试：

在代码检查方面，我们使用idea自带的检查代码功能。



这里挑选几个重要的问题：

1.在 HTML 4.01 标准中，align属性已经被标记为弃用。例如<h3 align="center">这种写法，在更标准的代码实践中是不推荐的。原因是这种表现性的属性（align用于控制元素的对齐方式）会将内容（结构）和表现（样式）混合在一起。



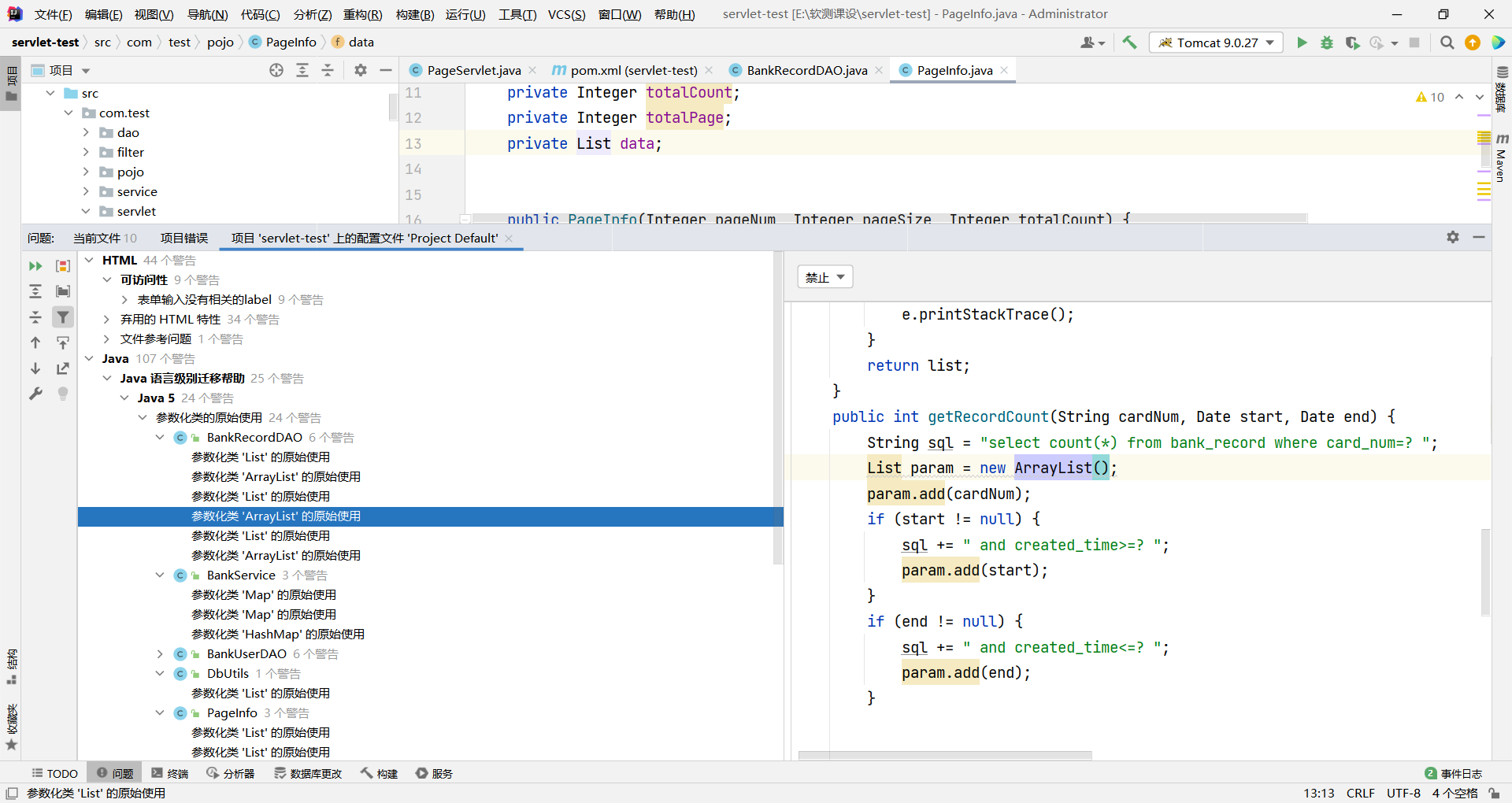
解决办法：用css样式来实现元素对齐的效果，使得代码更加模块化和易于维护。当需要修改样式时，只需要在 CSS 文件中进行调整，而不用在每个 HTML 标签中寻找相关的属性。如

h3 {

text - align: center;

}

2.没有参数化List、Map、HashMap等类的使用。在没有参数化类（泛型）之前，从集合（如List）中取出元素时，需要进行类型转换，并且容易出现ClassCastException。

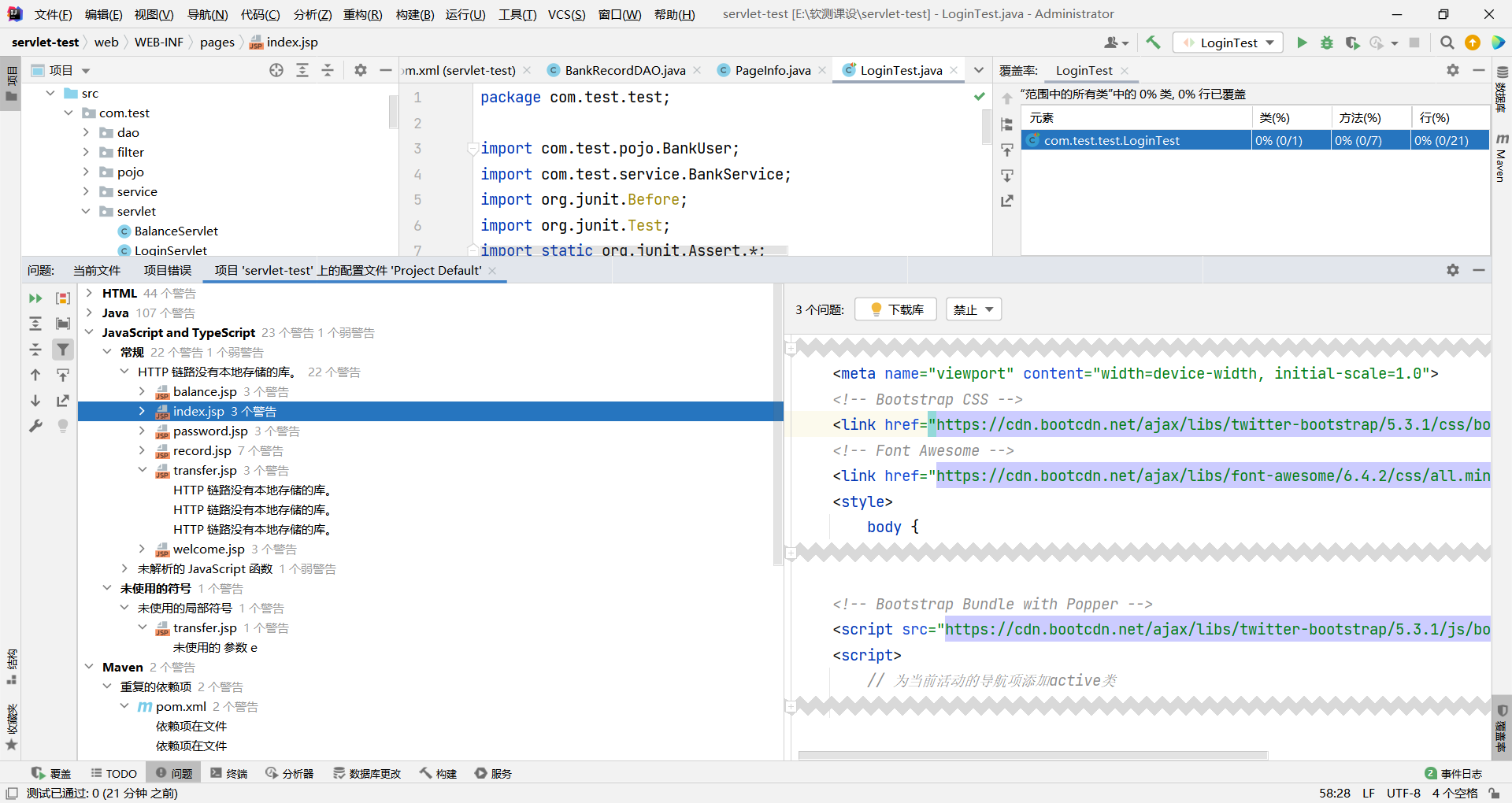


解决办法：在一开始使用这些类的时候就指定参数类型，比如

List param = new List<String>();

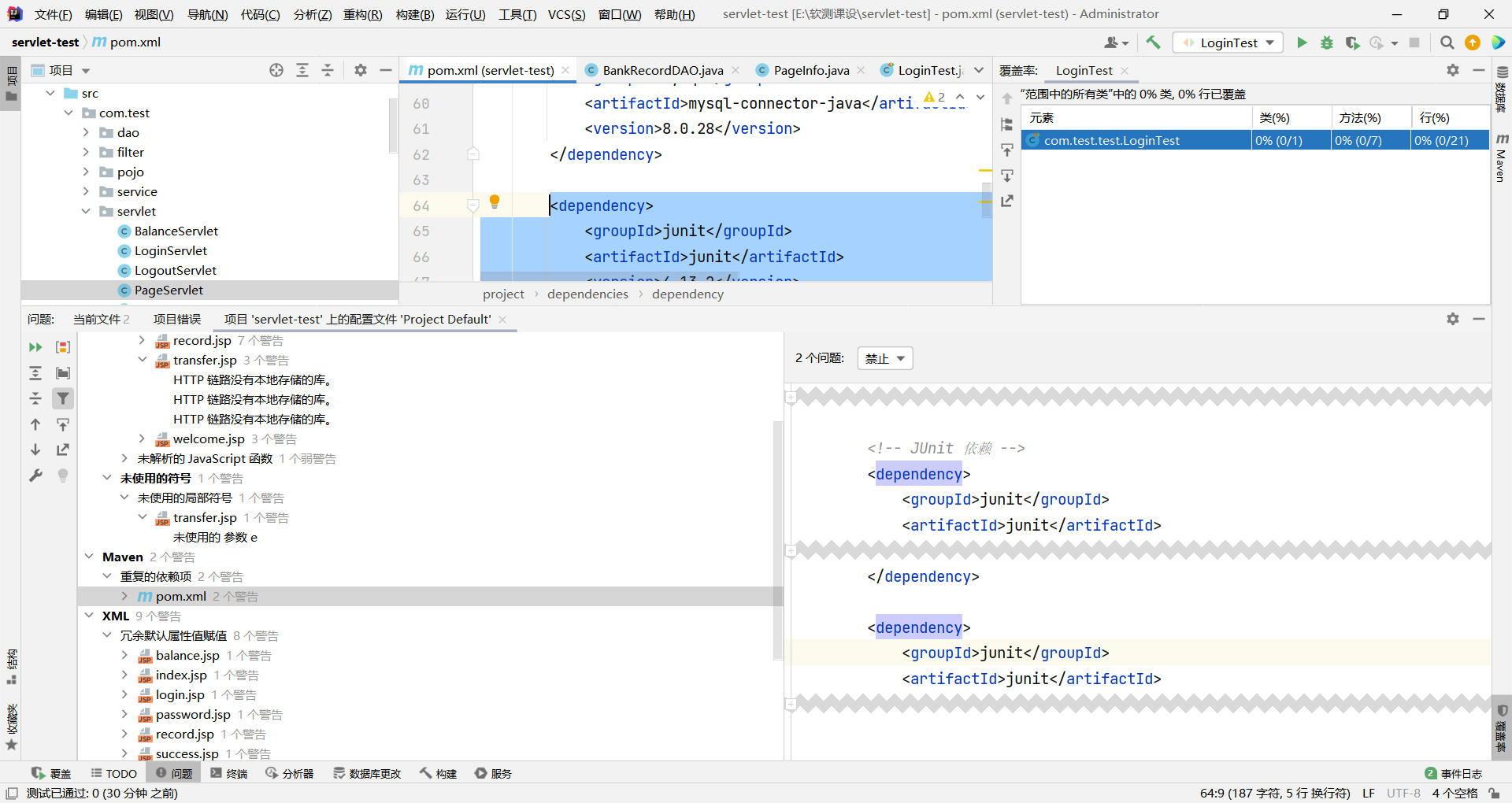
编译器会在编译阶段检查添加到List中的元素类型是否符合指定的类型参数，这样可以在编译时就发现类型不匹配的问题，提高了代码的安全性。

1. HTTP链路没有本地存储的库。在引入Bootstrap CSS、Font Awesome等前端框架时，从外部 CDN引入库文件，会受到网络状况的影响。在用户访问网页时，如果网络延迟较高，可能会使页面呈现明显的空白或样式错乱，直到 CSS 文件加载完成，严重影响用户体验。



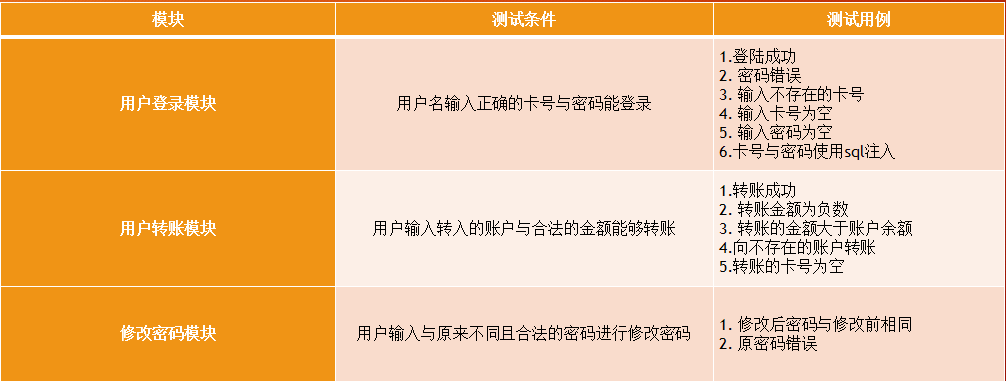
解决办法：去下载需要的库文件到本地，从本地引入这些库文件。

1. Maven存在重复的依赖项。配置文件里出现了两次的junit的配置，虽然不完全相同。当<scope>为test时，JUnit库仅用于测试阶段，不会包含在最终的产品中。而当<scope>为compile时，JUnit库会被包含在主项目的编译和运行过程中。这可能会导致一些问题，比如增加了最终产品的体积，并且在生产环境中引入了测试框架，这通常是不必要的。



解决方法：删除<scope>为compile的配置部分，因为junit只需要用在测试部分。

二、单元测试：



用户登录模块：

LoginTest 单元测试分析

一、测试概述

本次单元测试针对BankService的login方法进行了全面的功能性测试，涵盖了正常登录、错误密码登录、不存在用户登录、空用户名登录、空密码登录以及防范 SQL 注入攻击等多种情况，旨在确保login方法的正确性和安全性。

二、测试用例执行情况

（一）测试成功登录的情况（testLoginSuccess）

测试目的：验证当用户输入正确的用户名（卡号）和密码时，login方法能够成功返回对应的用户对象，且用户对象的卡号与输入的卡号一致。

测试代码：



测试分析：通过模拟BankService的login方法返回模拟的正确用户对象，然后调用login方法获取实际返回的用户对象，使用assertNotNull断言确保返回值不为空，即登录成功，并使用assertEquals断言验证返回用户对象的卡号与预期一致。

（二）测试密码错误的情况（testLoginWithWrongPassword）

测试目的：检查当用户输入正确的用户名但错误的密码时，login方法应返回null，表示登录失败。

测试代码：



测试分析：模拟BankService的login方法在传入正确卡号但错误密码时返回null，然后调用login方法获取实际返回值，使用assertNull断言验证返回值为null，符合预期的登录失败情况。

（三）测试用户不存在的情况（testLoginWithNonExistentUser）

测试目的：验证当用户输入不存在的用户名（卡号）时，login方法返回null，以确保系统能够正确处理不存在用户的登录请求。

测试代码：



测试分析：模拟BankService的login方法在传入不存在的卡号时返回null，实际调用login方法后，通过assertNull断言确认返回值为null，表明系统对不存在用户的登录处理正确。

（四）测试空用户名登录的情况（testLoginWithNullUsername）

测试目的：确认当用户名为空时，login方法返回null，防止因空用户名导致的异常或错误登录情况。

测试代码：

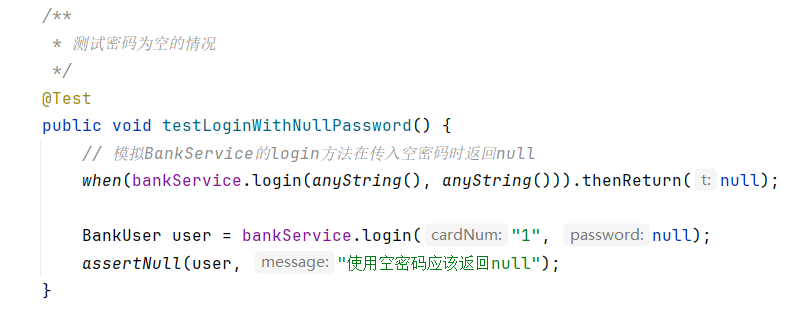


测试分析：模拟BankService的login方法在传入空卡号时返回null，实际测试中调用login方法并使用assertNull断言验证返回值为null，满足空用户名登录应返回null的预期。

（五）测试空密码登录的情况（testLoginWithNullPassword）

测试目的：检验当密码为空时，login方法返回null，保障系统对空密码登录请求的正确处理。

测试代码：



测试分析：模拟BankService的login方法在传入空密码时返回null，实际测试中调用login方法并通过assertNull断言确认返回值为null，符合空密码登录返回null的设计要求。

（六）测试 SQL 注入攻击的情况（testLoginWithSQLInjection）

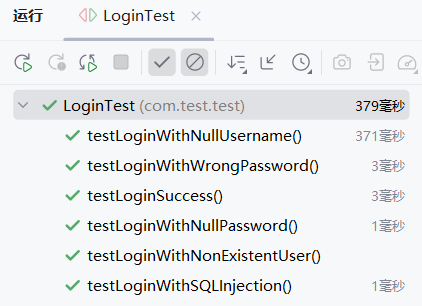
测试目的：测试login方法对 SQL 注入攻击的防范能力，确保即使传入具有 SQL 注入风险的参数，系统也能正确处理，不返回敏感信息且返回null。

测试代码：



测试分析：模拟BankService的login方法在传入 SQL 注入攻击的参数时返回null，实际调用login方法后，assertNull断言验证返回值为null，表明系统对 SQL 注入攻击具有一定的防范能力，未受到攻击影响而泄露数据或执行恶意 SQL 语句。

测试结果：



用户转账模块：

TransferTest 单元测试分析

一、测试概述

本次单元测试主要针对BankService的transfer方法进行了功能验证和异常情况测试，旨在确保该方法在各种正常和异常的转账场景下能够正确执行并返回预期的结果。测试用例涵盖了转账成功、转账金额为负数、转账金额大于账户余额、向不存在的账户转账以及转账卡号为空等多种情况，基本覆盖了transfer方法可能遇到的主要业务逻辑分支和异常条件。

二、测试用例执行情况

（一）测试转账成功的情况（testTransferSuccess）

测试目的：验证当提供合理的转账参数（即付款人卡号、收款人卡号和正数转账金额）时，transfer方法能够成功执行转账操作，并返回包含success=true的结果Map。

测试代码：



测试分析：通过模拟BankService的transfer方法返回模拟的成功转账结果Map，然后调用transfer方法并获取实际返回结果，使用assertTrue断言验证success字段为true，符合预期的转账成功情况。

（二）测试转账金额为负数的情况（testTransferWithNegativeAmount）

测试目的：检查当转账金额为负数时，transfer方法应拒绝转账并返回包含success=false和特定错误消息（“转账金额不能为负数”）的结果Map。

测试代码：



测试分析：模拟BankService的transfer方法在传入负数金额时返回相应的失败结果Map，实际调用transfer方法后，通过assertFalse断言验证success字段为false，并使用assertEquals断言验证错误消息与预期一致，表明系统对负数转账金额的处理正确。

（三）测试转账金额大于账户余额的情况（testTransferWithAmountGreaterThanBalance）

测试目的：确保当转账金额超过付款人账户余额时，transfer方法能够正确识别并返回包含success=false和 “您的账户余额不足” 错误消息的结果Map。

测试代码：



测试分析：模拟BankService的transfer方法在传入大于余额的金额时返回相应的失败结果，实际测试中调用transfer方法后，assertFalse和assertEquals断言验证了返回结果的正确性，系统对余额不足情况的处理符合预期。

（四）测试向不存在的账户转账的情况（testTransferToNonExistentAccount）

测试目的：验证当尝试向不存在的账户转账时，transfer方法返回包含success=false和 “该用户不存在” 错误消息的结果Map，以保证系统对不存在收款账户的情况处理得当。

测试代码：



测试分析：模拟BankService的transfer方法在传入不存在的收款卡号时返回相应的失败结果，实际调用transfer方法后，断言验证了返回结果中success字段为false且错误消息正确，说明系统能够正确处理向不存在账户转账的情况。

（五）测试转账的卡号为空的情况（testTransferWithNullCardNum）

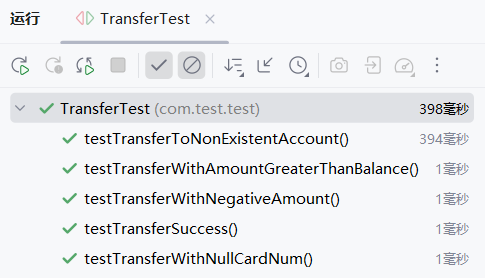
测试目的：检查当转账的目标卡号为空时，transfer方法能够正确处理并返回包含success=false和 “转账目标不能为空” 错误消息的结果Map，同时验证是否可能抛出NullPointerException异常（如果有相关风险），并确保异常信息符合预期（如果需要）。

测试代码：



测试分析：在处理过程中捕获到了NullPointerException异常，这表明代码在处理空指针情况时可能存在潜在的风险，需要进一步检查和优化代码，以确保在生产环境中能够更健壮地处理此类情况。从断言验证的结果来看，返回的错误消息与预期一致，说明在正常业务逻辑处理方面，系统对空卡号情况的处理基本符合要求，但需要解决异常处理的潜在问题。

测试结果：



修改密码模块：

PasswordModifyTest 单元测试分析

一、测试概述

本次单元测试旨在对BankService的updatePassword方法进行功能性验证，重点关注了修改密码过程中两种常见的失败情况：一是新密码与旧密码相同，二是原密码错误。通过模拟这些场景，使用Mockito框架来模拟BankService的行为，并运用断言来验证方法的返回值是否符合预期的业务逻辑。

二、测试用例执行情况

（一）测试修改后密码与修改前相同的情况（testModifyPasswordSameAsBefore）

测试目的：验证当用户尝试将密码修改为与原密码相同时，updatePassword方法能够正确识别并返回表示失败的标识（假设为 0），同时通过断言确保返回值不等于表示成功的标识（假设为 1），以此来保证密码修改逻辑的正确性，避免不必要的密码更新操作。

测试代码：



测试分析：通过模拟BankService的updatePassword方法在接收到相同的新旧密码时返回 0，实际调用该方法后，使用assertEquals断言验证返回值确实为 0，并且使用assertFalse断言进一步确认返回值不等于 1，与预期的业务逻辑一致，即系统正确处理了密码未发生实际变化的情况，未执行修改操作并返回了正确的失败标识。

（二）测试原密码错误的情况（testModifyPasswordWithWrongOldPassword）

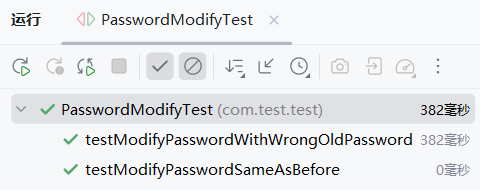
测试目的：检查当用户提供错误的原密码进行密码修改时，updatePassword方法能够准确判断并返回表示原密码错误的特定标识（假设为 -1），同时通过断言确保返回值不等于成功标识 1，以此确保系统对原密码验证机制的有效性，防止未经授权的密码修改操作。

测试代码：



测试分析：模拟BankService的updatePassword方法在传入错误的原密码时返回 -1，实际调用该方法后，assertEquals断言验证返回值为 -1，assertFalse断言确认返回值不为 1，表明系统能够正确识别原密码错误的情况并返回相应的错误标识，符合预期的安全和业务逻辑要求。

测试结果：



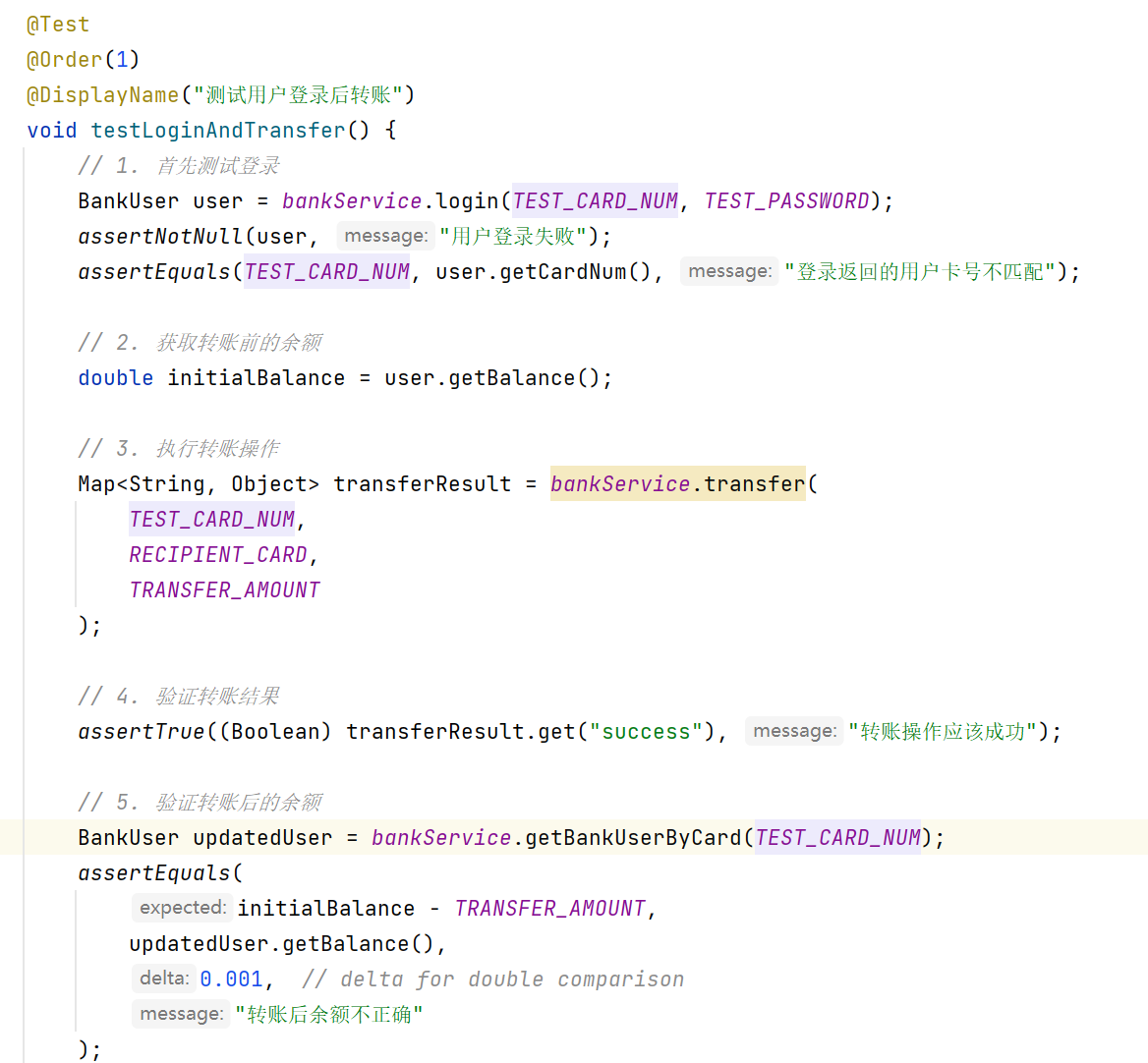
三、集成测试：



（一）登录后转账

测试目的：验证完整的转账业务流程，包括余额变动的正确性和交易记录的生成。

测试代码：

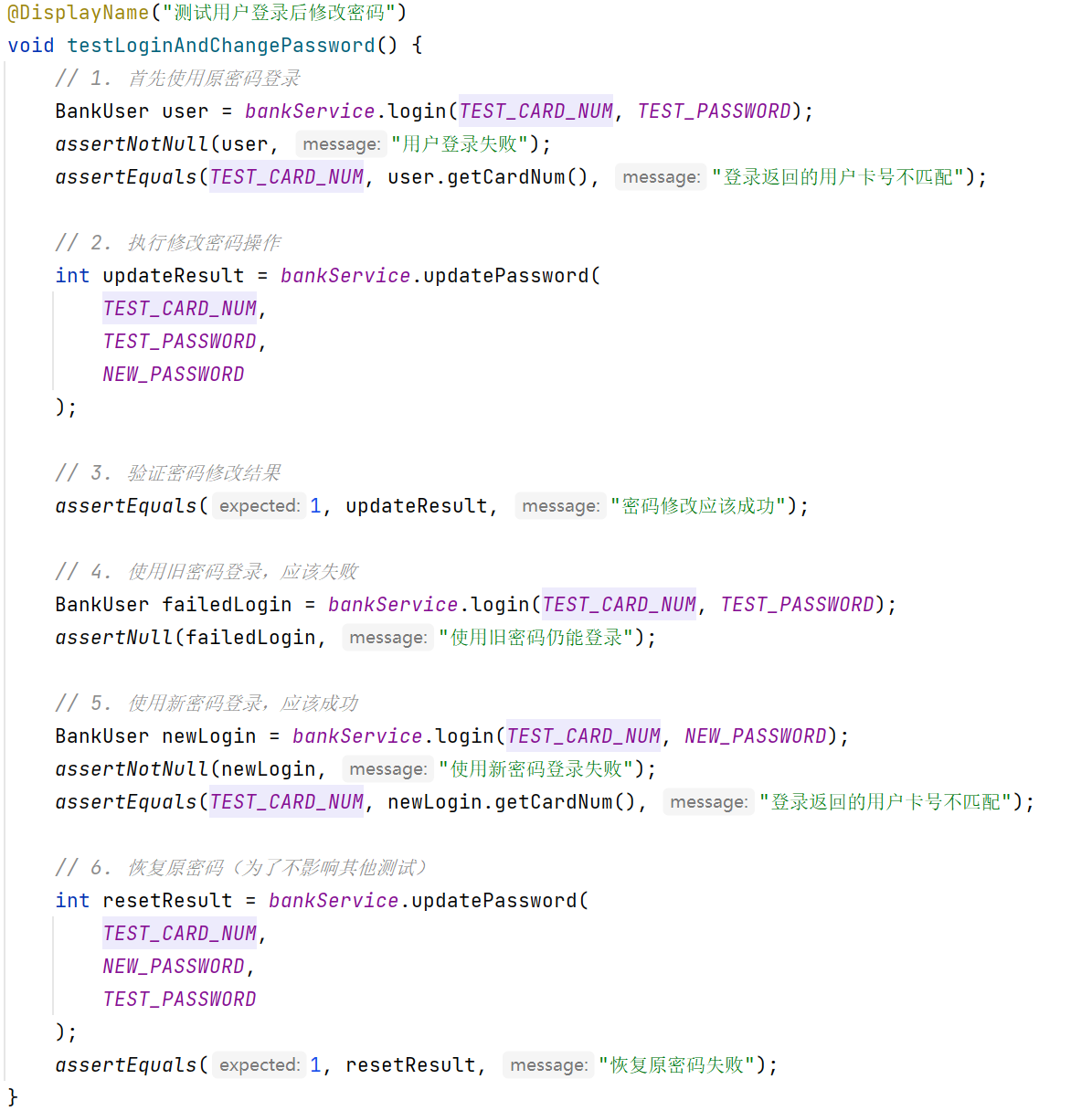


测试分析：通过验证转账前后的账户余额变化和交易记录的生成，确保转账功能在正常情况下能够正确执行，且数据一致性得到保障。

（二）登录后修改密码

测试目的：验证登录模块与修改密码模块之间配合能否正常运作。

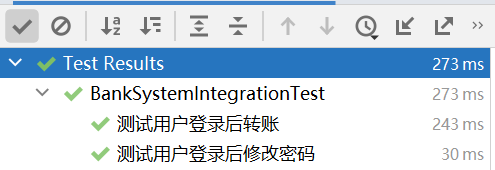
测试代码：



测试分析：

用户登录后修改密码的行为后可以用新密码登录，表明登录模块与修改密码模块之间没有问题。

测试结果：



四、系统测试：

一、测试概述

本次系统测试使用Selenium WebDriver模拟真实用户操作，对整个网上银行系统进行端到端的功能测试，包括完整用户操作流程、系统安全性、性能测试以及错误处理等多个方面，旨在验证系统的整体功能完整性、安全性和可用性。

二、测试用例执行情况



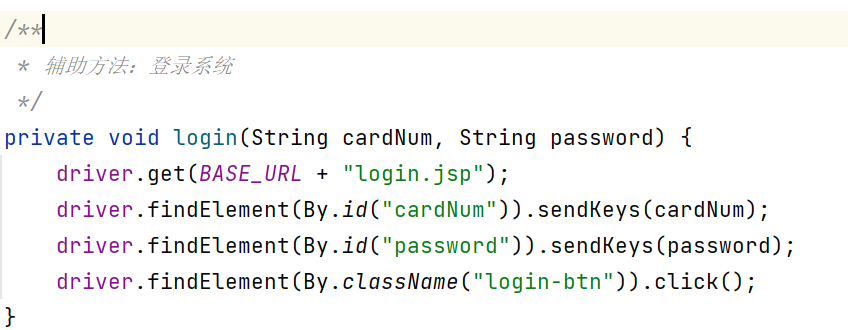
代码：











测试分析：通过模拟用户的完整操作流程，验证系统各个功能模块之间的衔接和交互是否正常，确保用户可以顺利完成所有核心业务操作。每个步骤都包含了必要的等待和验证，确保操作的可靠性。



测试代码：



测试分析：通过模拟未授权访问和恶意输入，验证系统的安全防护措施是否能够有效阻止非法访问和攻击，确保系统的安全性。



代码：



测试分析：通过记录和测量系统响应时间，验证系统性能是否满足用户体验要求，确保各项操作能在可接受的时间内完成。

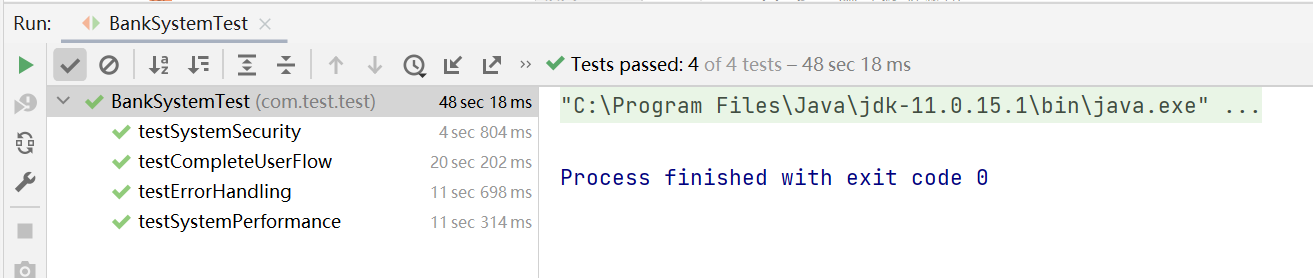


代码：



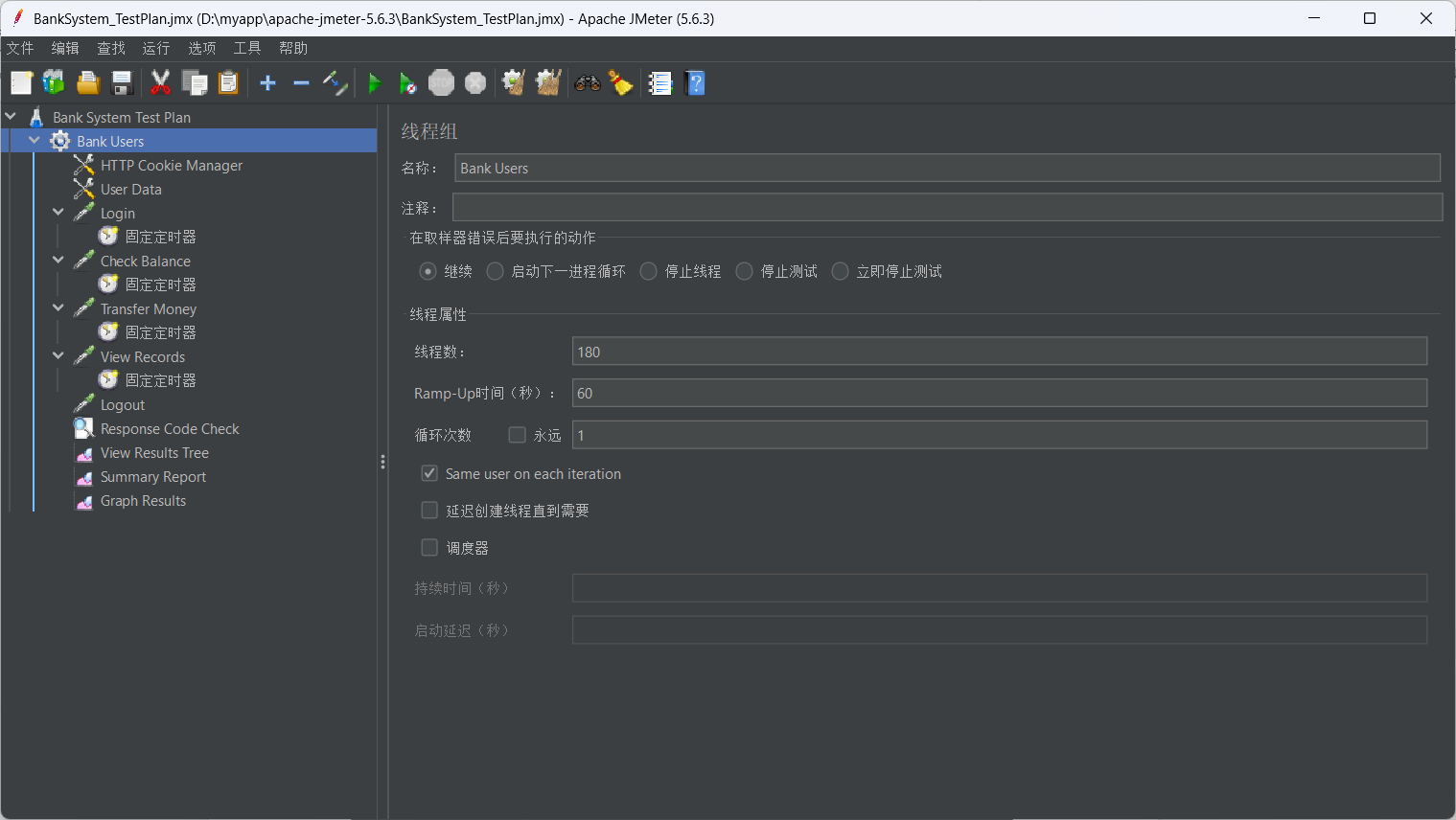
测试分析：通过模拟各种错误输入和异常情况，验证系统是否能够正确识别这些情况，并给出适当的错误提示，确保系统的健壮性。

测试结果：

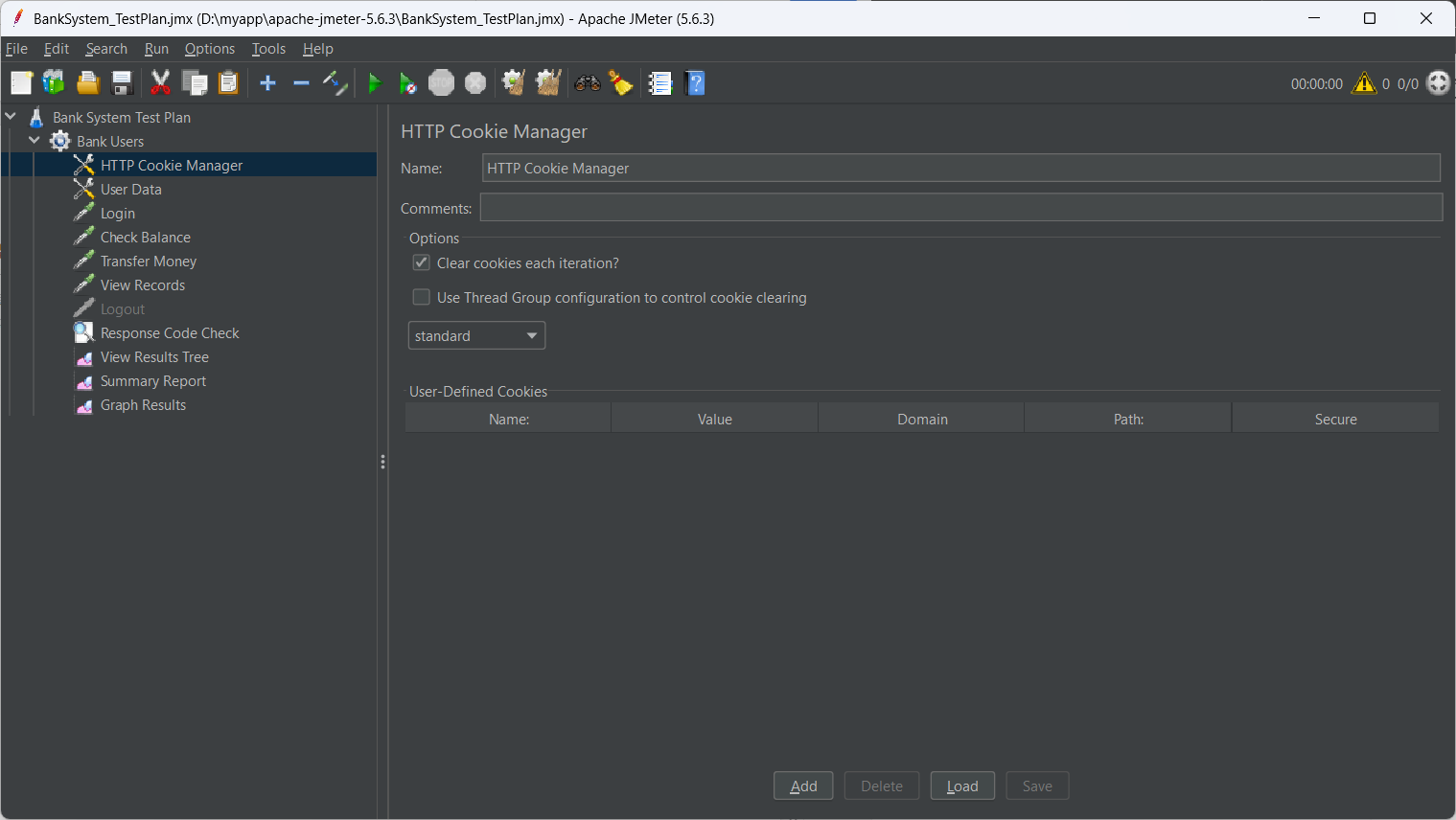


五、性能测试：

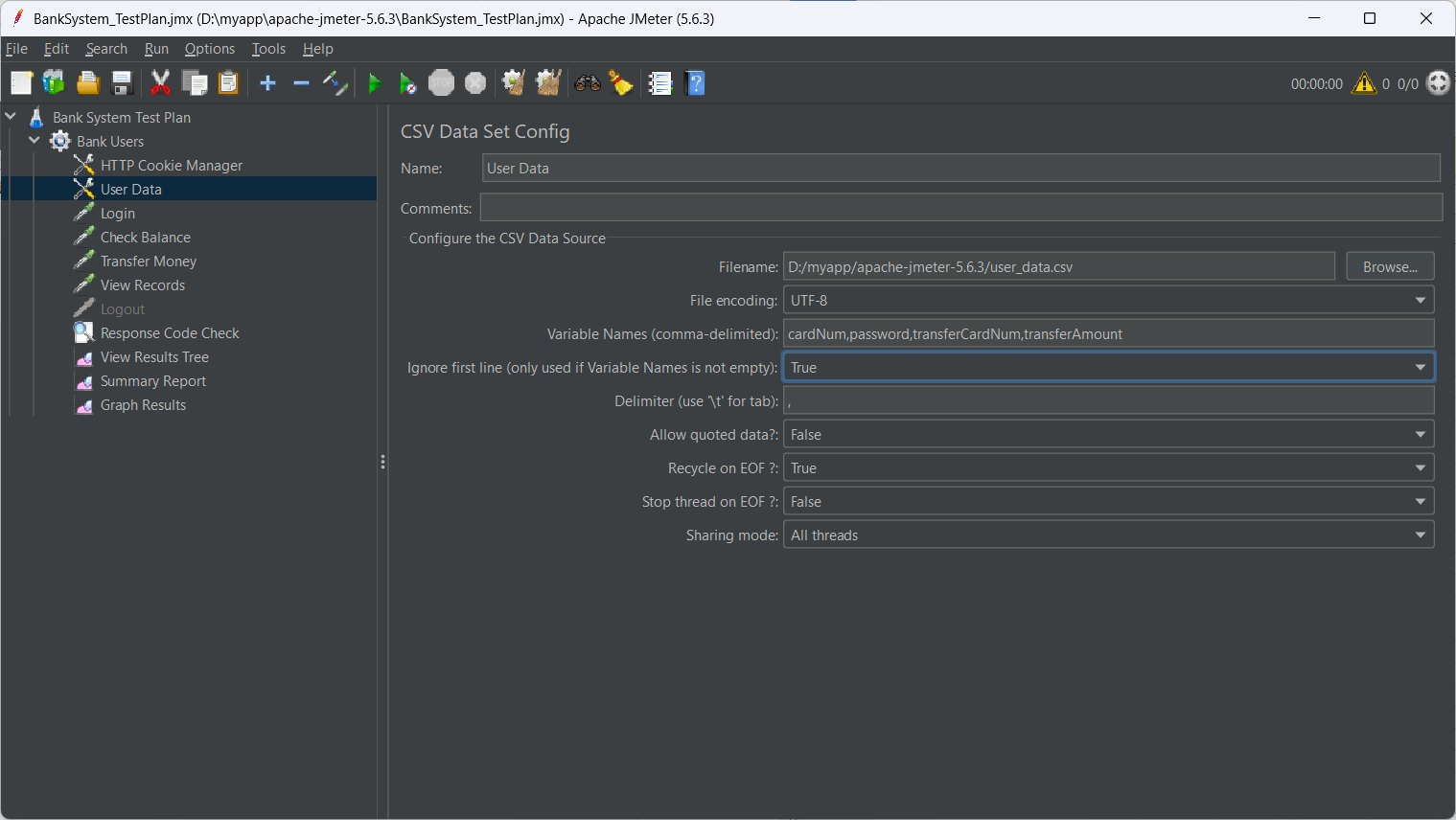
1. 设置线程组，为了模拟高峰期，我们选择让180组用户在1分钟内上线，也就是1秒上线3个用户。



1. 设定每轮测试时都清除cookie，模拟真实用户的独立会话行为，防止影响下一次测试的结果。用户在关闭浏览器后重新打开访问网站，或者在不同的设备上登录同一个账号，这些情况下通常会开始新的会话，之前的 Cookie 信息不会被保留。

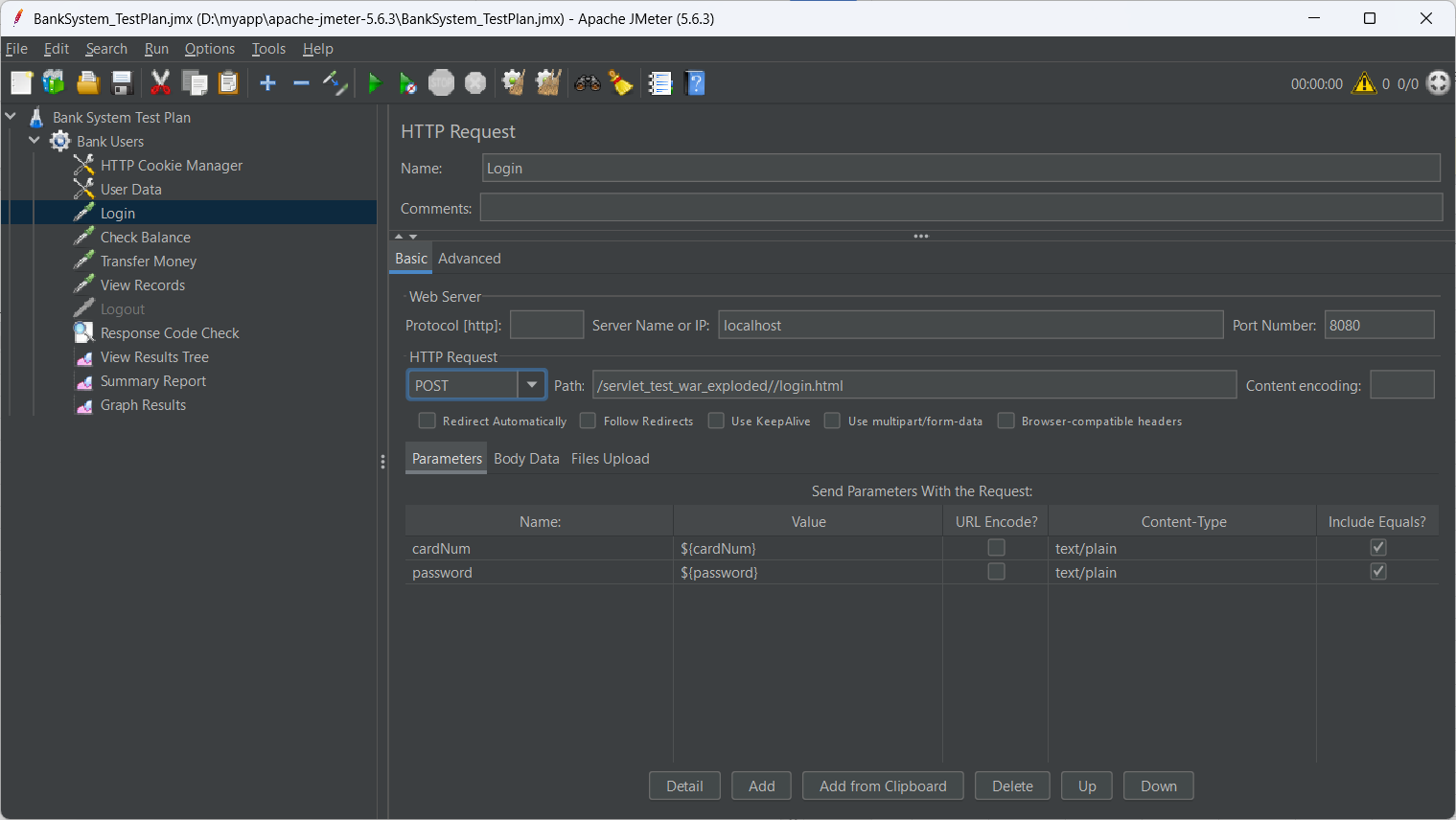


1. 为线程组设置测试用户的csv数据文件，存入用户的账号、密码、转入卡号、转账金额，以便运行测试时模拟多个用户进行测试，文件编码设置成UTF-8，用逗号来分割每一行的数据，读入数据时忽略第一行，因为第一行是变量名称。

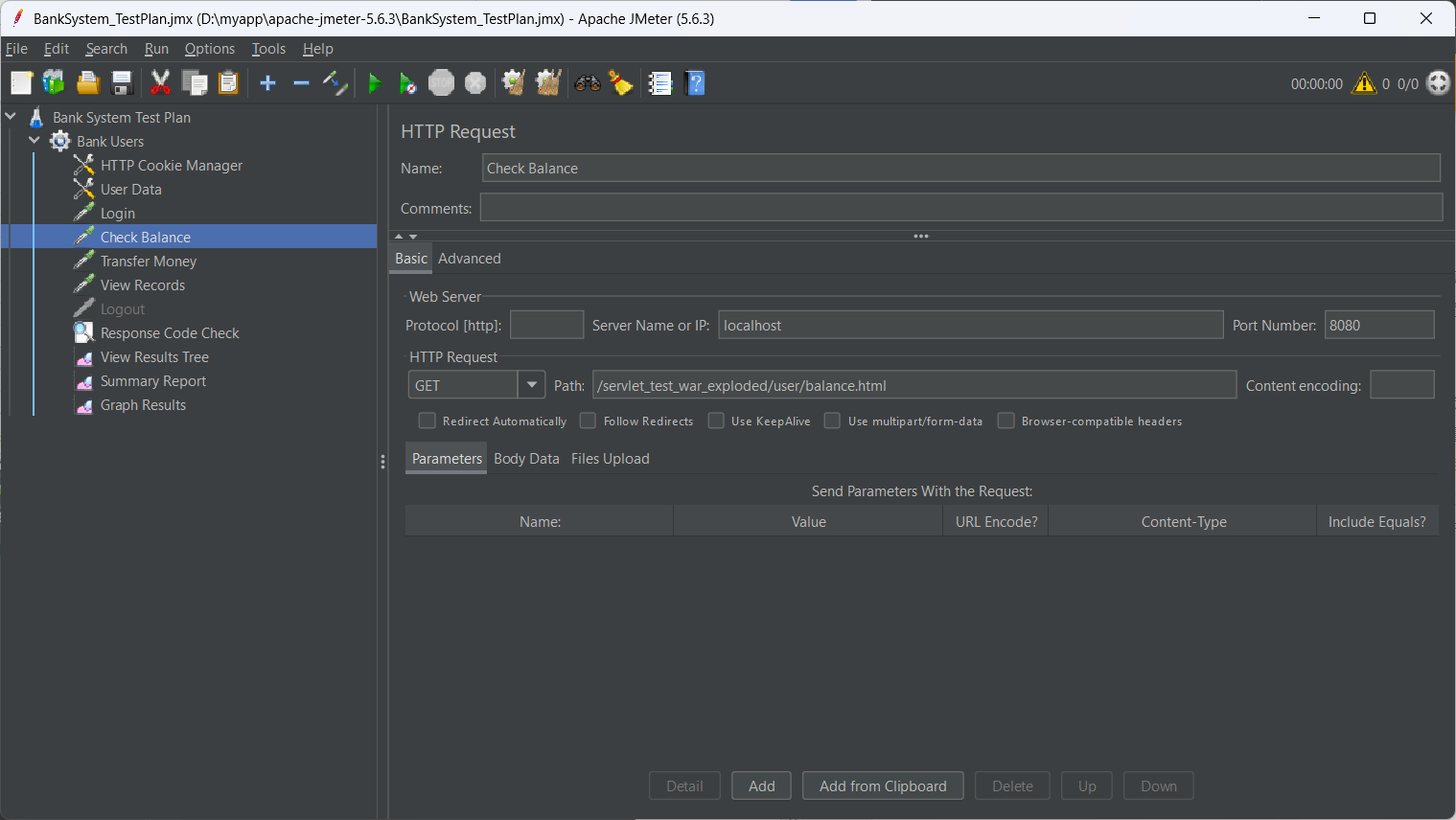




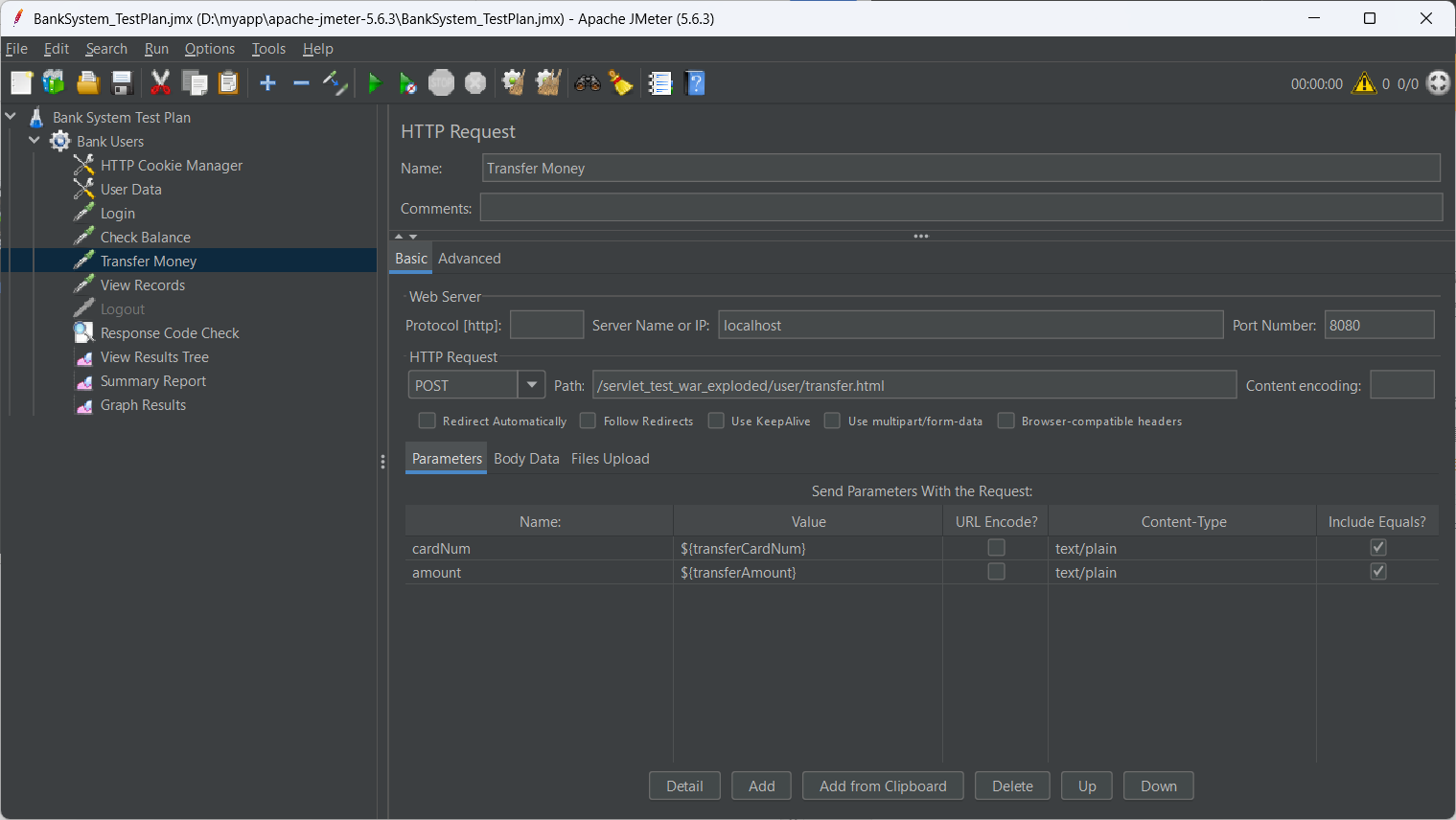
1. 设置登录，参数为cardNum和password，从csv文件中读取。



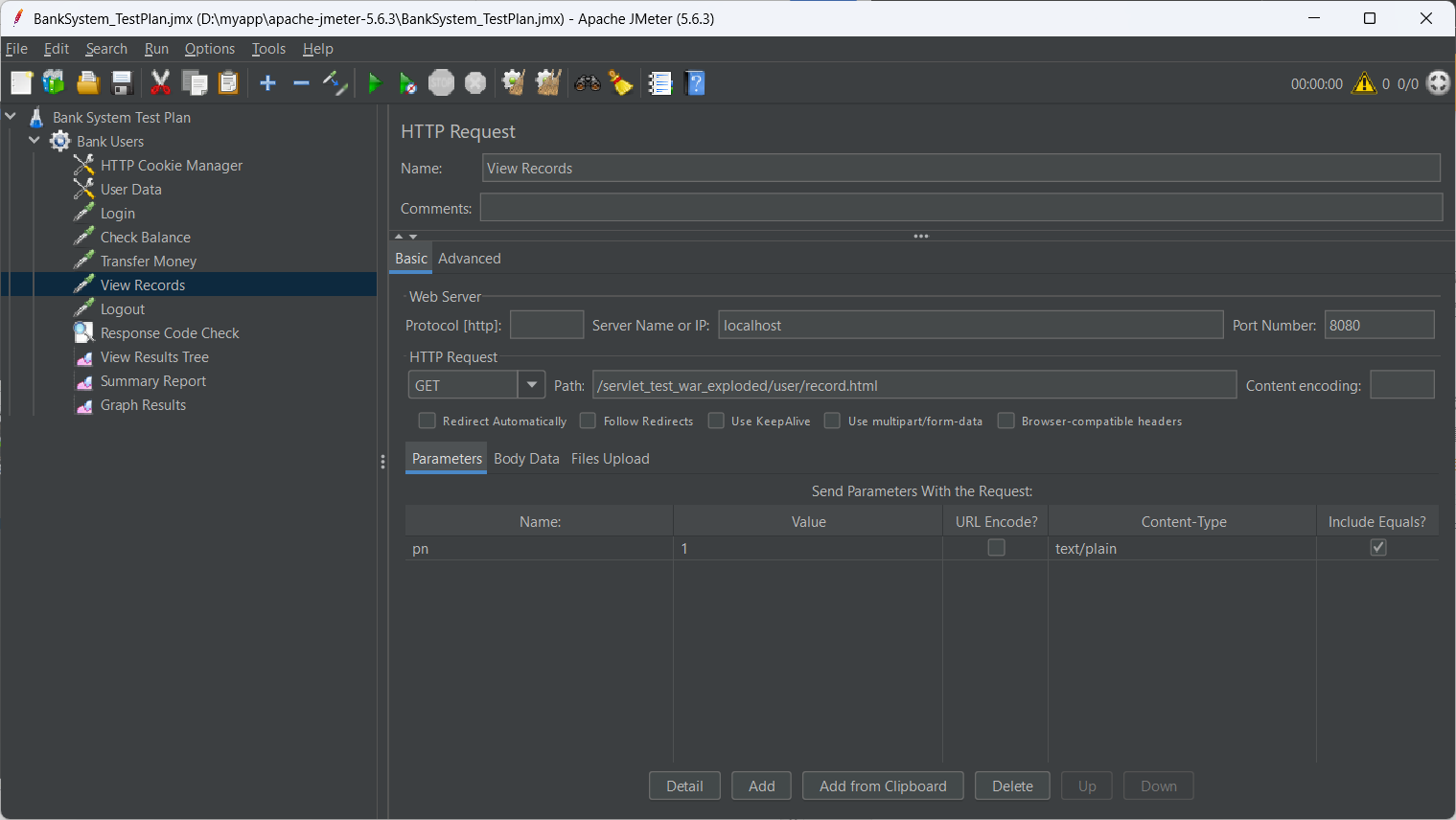
1. 设置查看余额。



1. 设置转账，参数为cardNum和amount，从csv文件中读取。



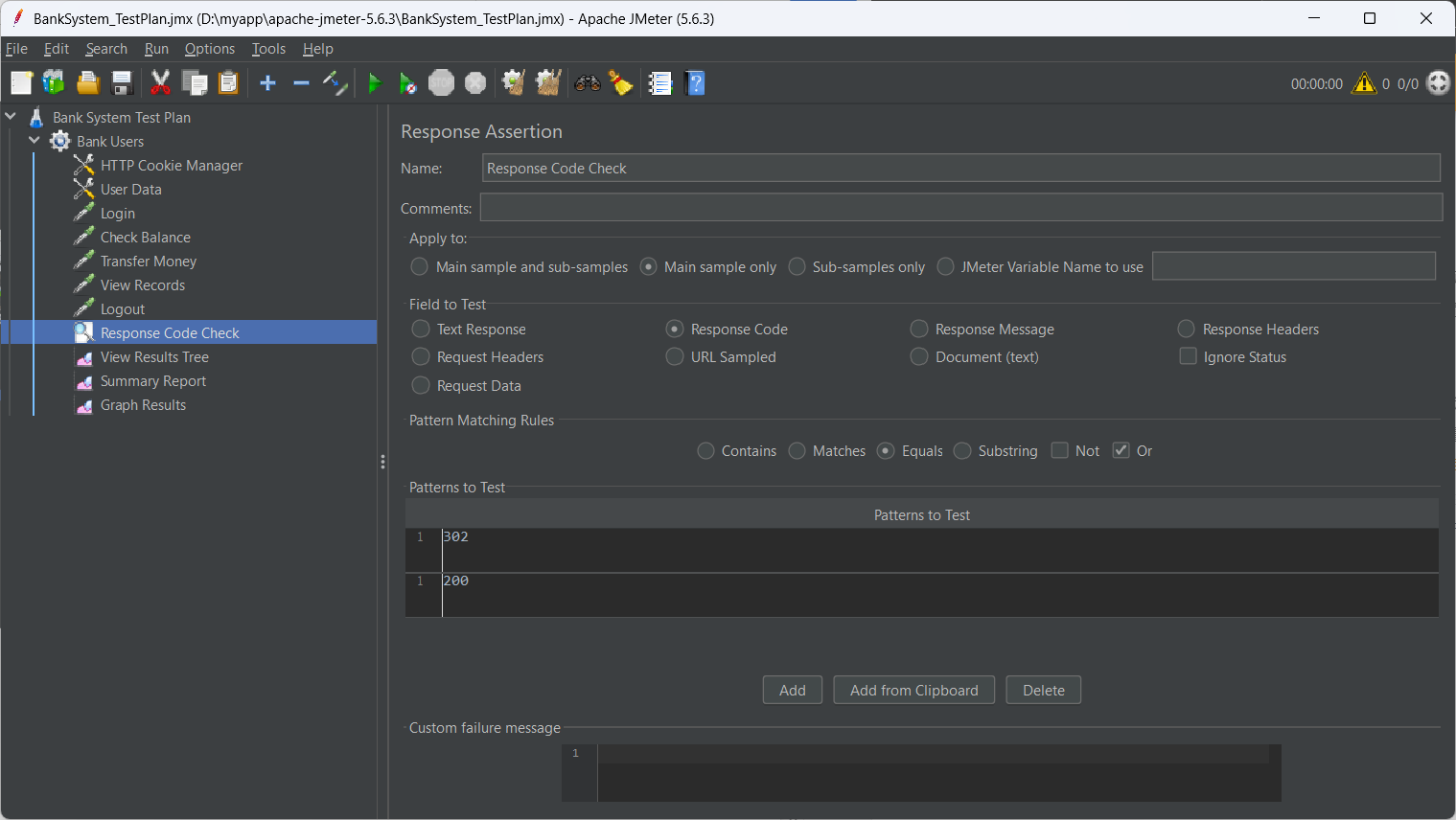
1. 设置查看转账记录，参数pg为1，即查看第一页的转账记录。



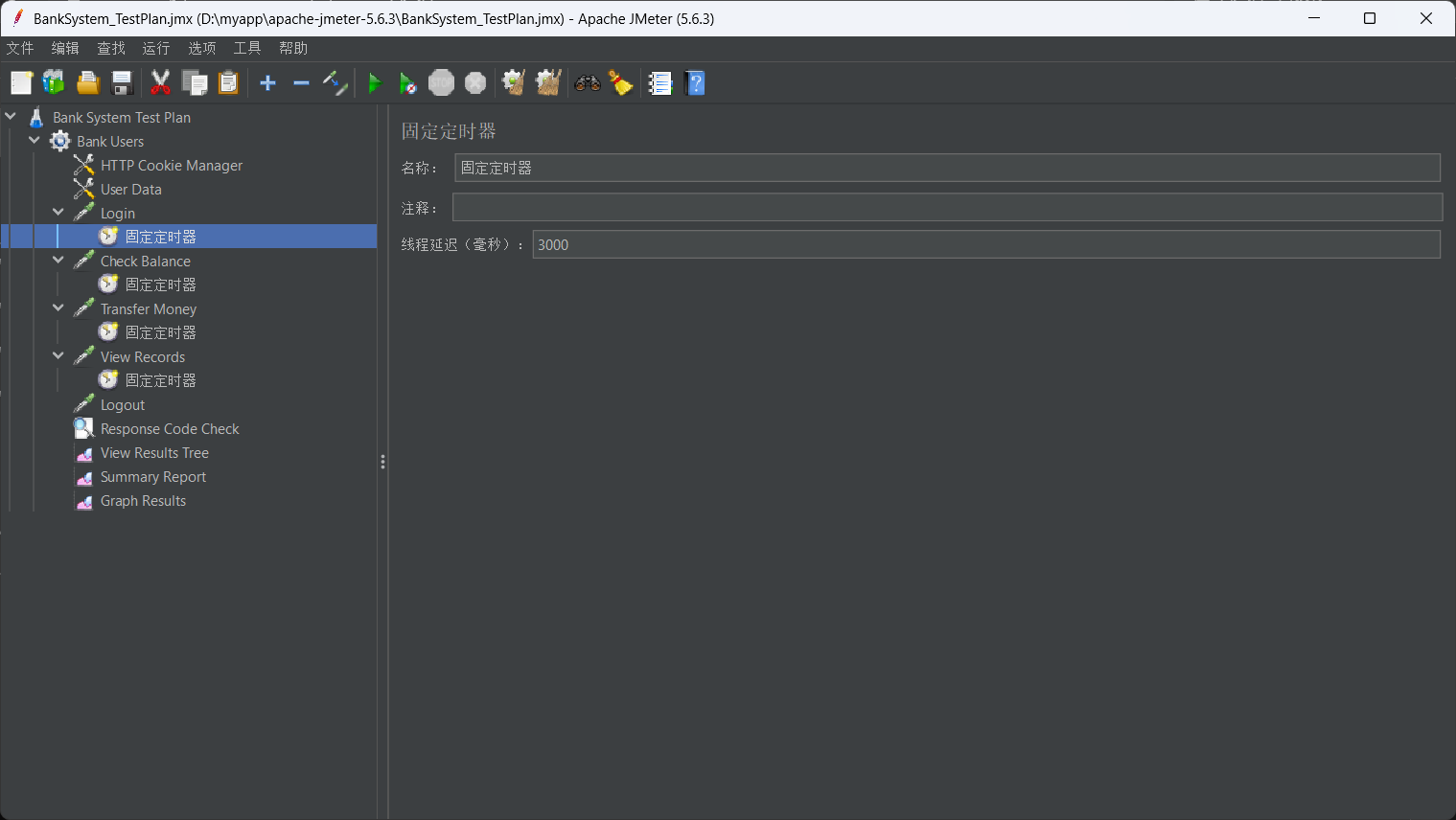
1. 设置退出登录。



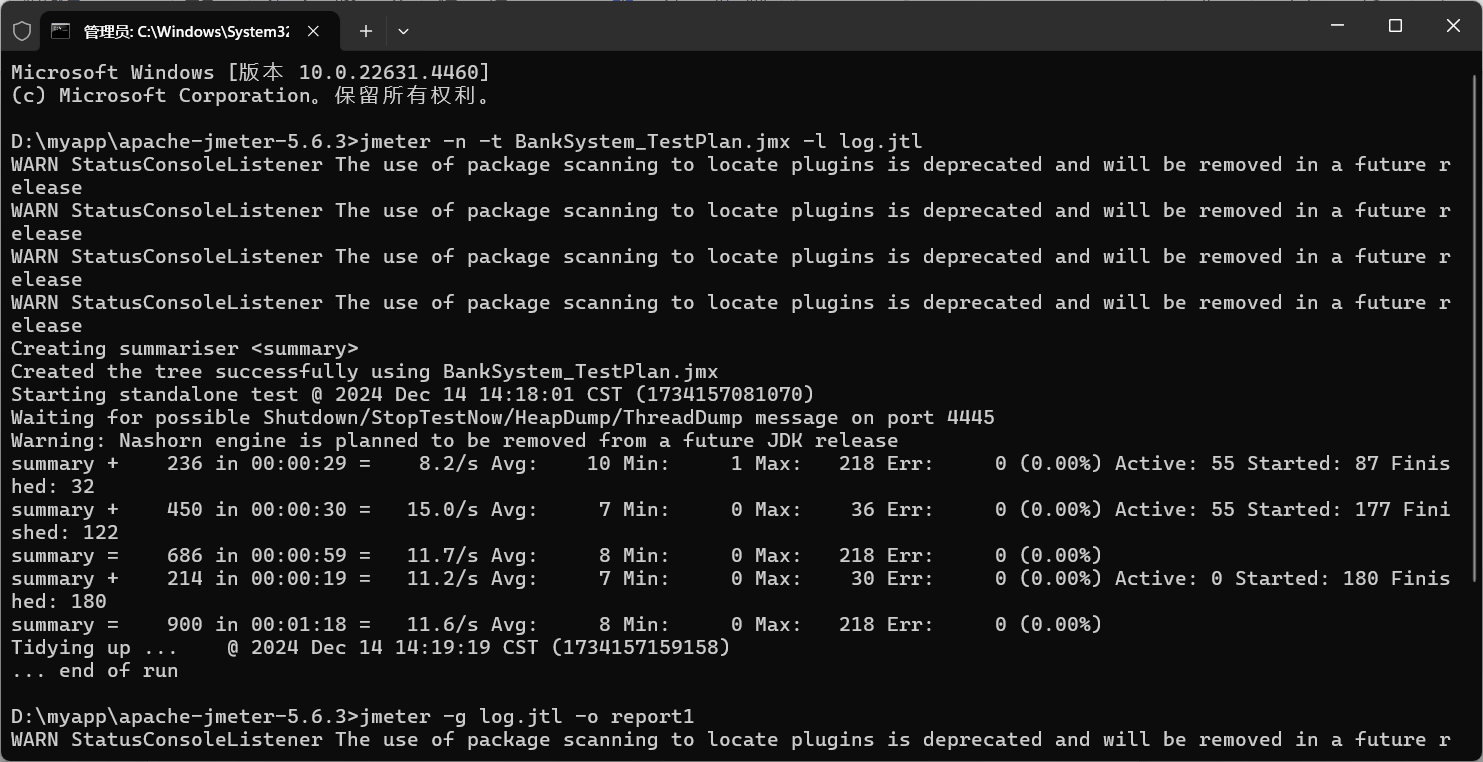
1. 设置网站返回状态码的判断，这里设置200跟302都是合法的，因为302是重定向的返回码，我们网站的登录功能在登录成功后使用重定向定位到主页。登录功能使用重定向的原因是在用户成功登录后，若用户刷新页面，浏览器会重新发送上一次的请求（因为转发后的 URL 在浏览器地址栏中不会改变），这可能导致登录表单被重复提交。而使用重定向会改变浏览器地址栏中的 URL，当用户刷新页面时，不会重新提交登录表单，从而避免了重复提交的问题。登录成功后的页面可以独立地被加载和展示，它不需要依赖于登录时的请求和处理过程，使得代码结构更加清晰，便于维护和扩展。



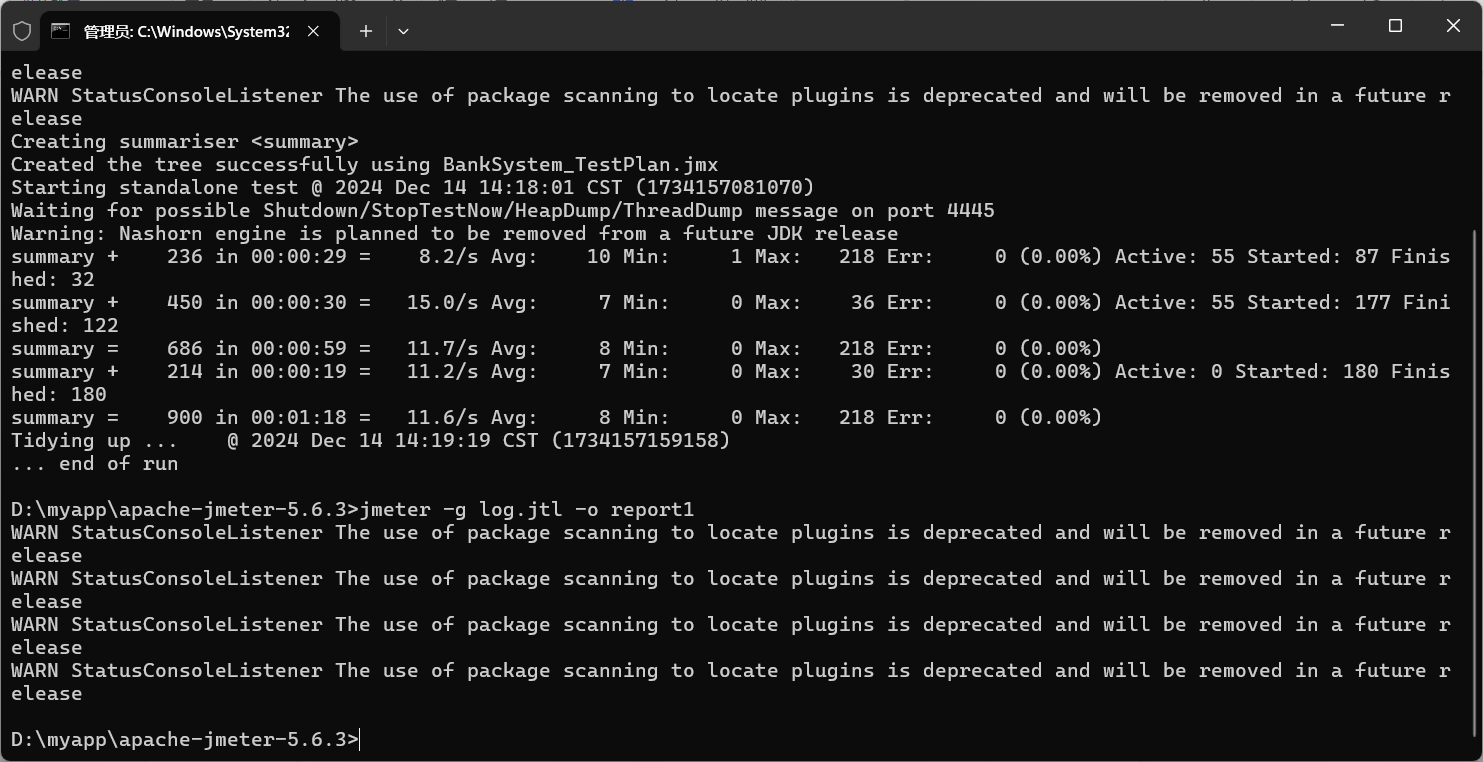
1. 在各个操作之间插入定时器，模拟更真实的操作。

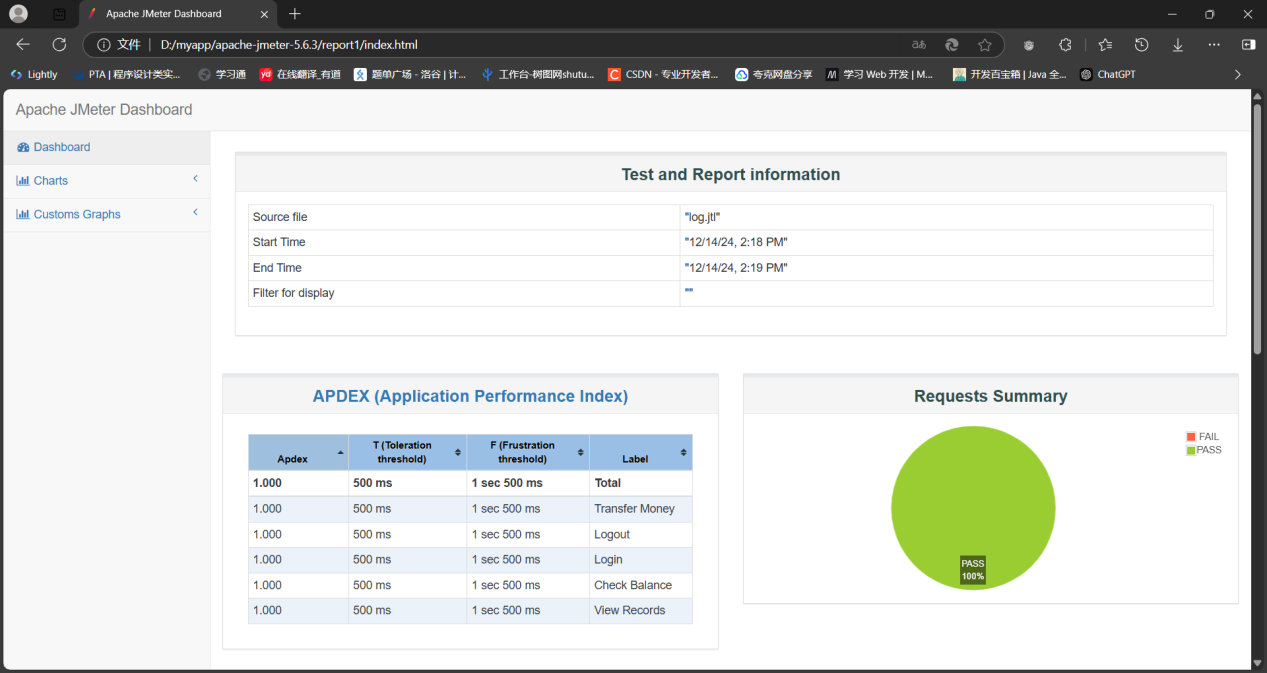


1. 在命令行输入jmeter -n -t BankSystem\_TestPlan.jmx -l log.jtl，以命令行模式运行实施性能测试（已配好环境变量，可简化命令）。



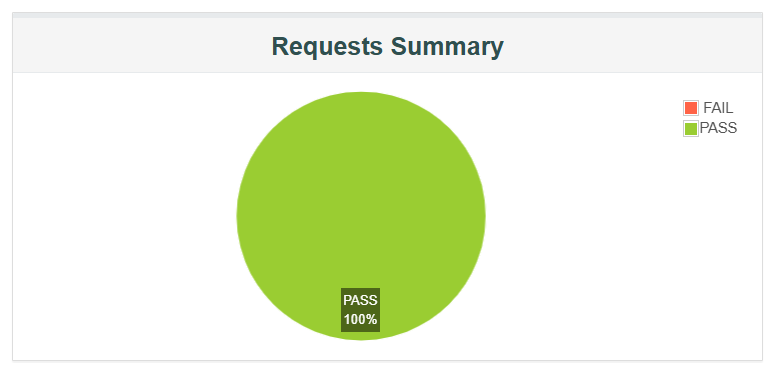
1. 在命令行输入jmeter -g log.jtl -o report1，就会产生report1目录，里面的index.html 打开就是报告。



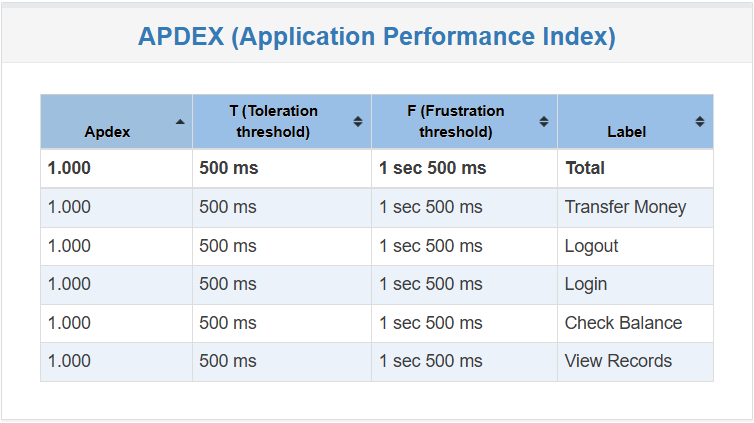


结果分析：

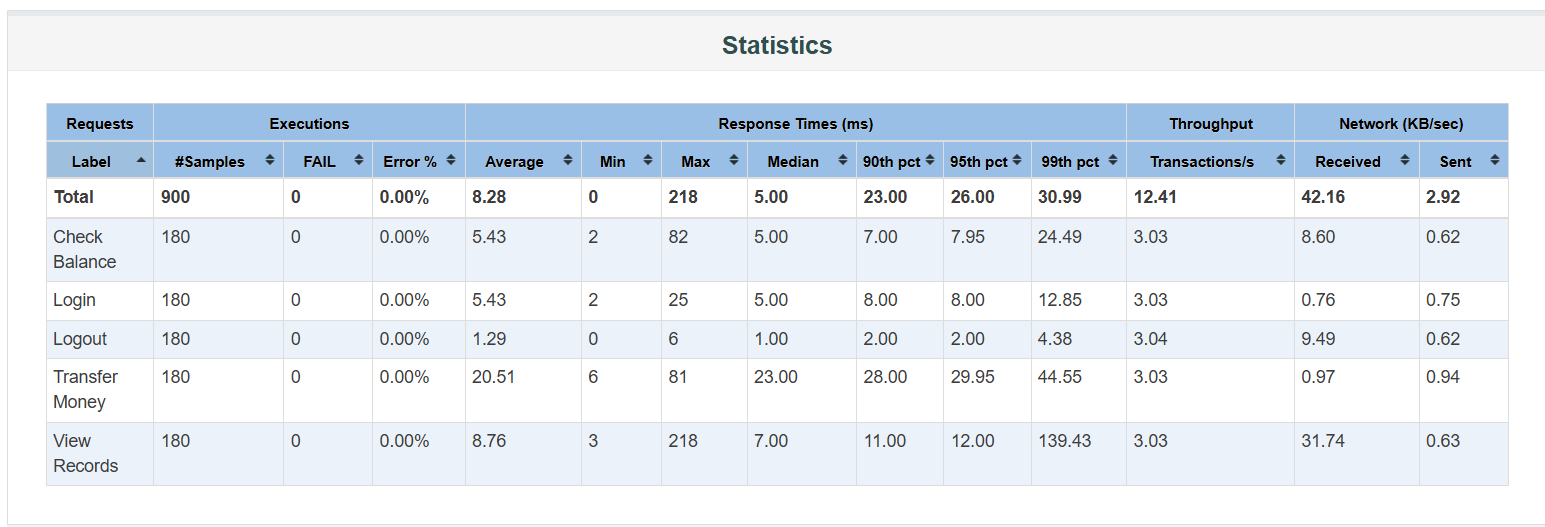
所有请求类型的错误率均为0.00%，表明系统在高并发情况下能够稳定处理请求，没有出现崩溃或错误，具有高并发处理能力。



对网上银行系统而言，特别是实时转账操作，需要系统极低的响应时间，所以我们设定总体的用户容忍响应时间为500ms，最大容忍1500ms，测试结果Apdex（满意度）值越大表现越好，Apdex为1表明用户体验很满意。



在900个sample中，平均响应时间为8.28毫秒，表明系统响应速度较快。部分请求的最大响应时间较高（如“Check Balance”和“View Records”），可能表明在这些操作上存在性能瓶颈或资源争用问题，所以这一部分的系统功能需要再优化。“View Records”请求的接收网络流量显著高于其他请求，是因为该操作涉及大量数据的返回，而其他请求的接收和发送流量相对较低，表明数据传输效率较高。



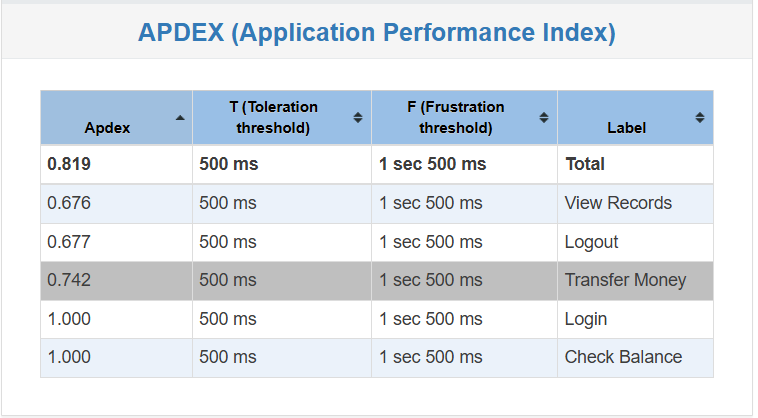
代码档：

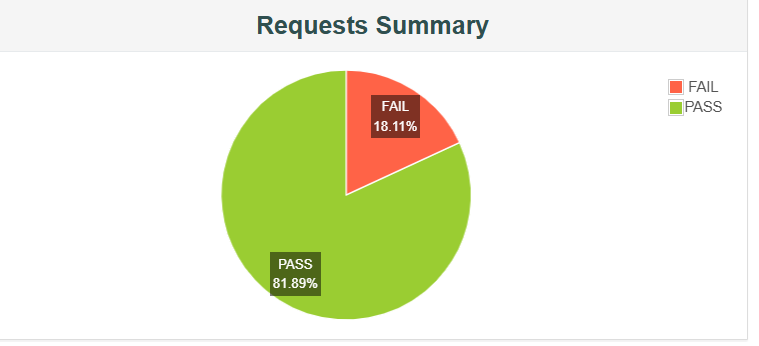


补充测试：

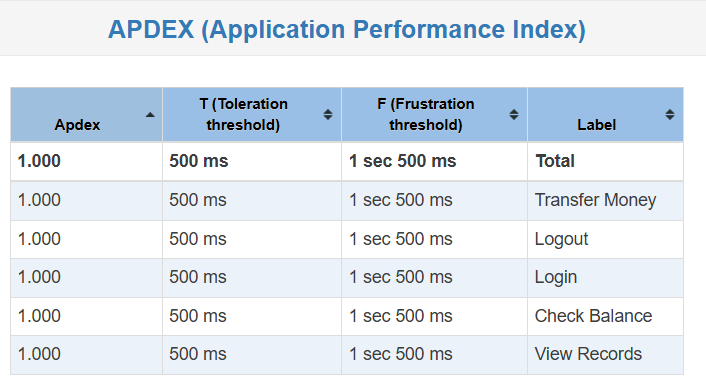
为了测试系统能承受的最大并发用户数，我后面又增加测试了不同的线程组：1分钟内上线180个用户、1分钟内上线600个用户、1分钟内上线1200个用户、30秒上线2400个用户、30秒上线1500个用户、30秒上线2100个用户。

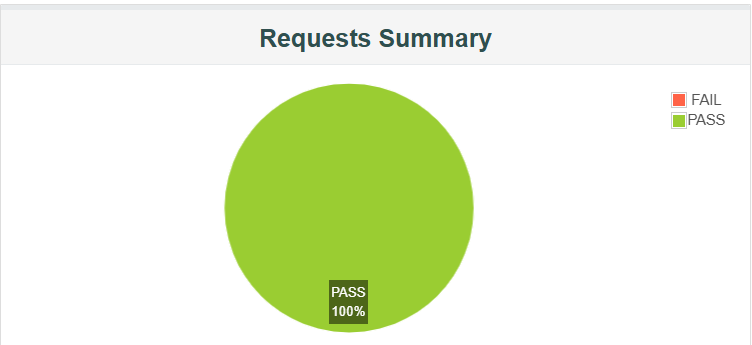
其中30秒上线2400个用户，请求失败高达18%，用户满意度大幅下降



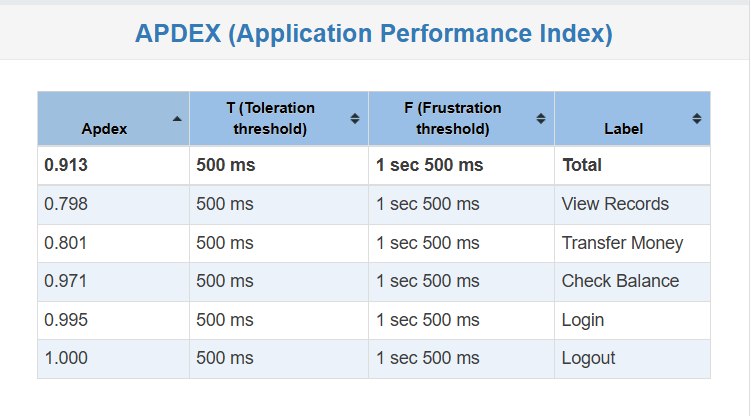


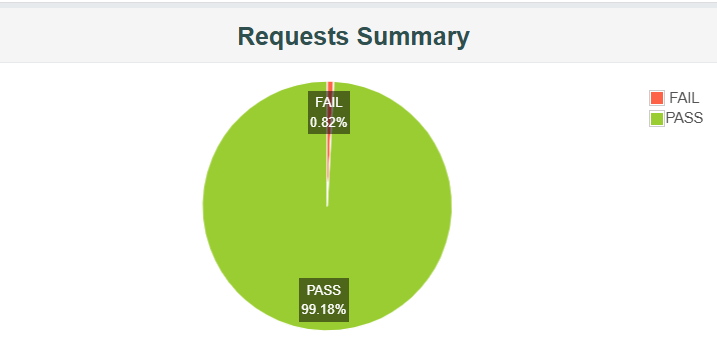
30秒上线1500个用户一切正常





30秒上线2100个用户用户满意度略微下降，请求大部分成功





所以系统每秒最大能承受的用户数大约为70个