# SDYERP系统测试作品介绍文档

## 项目概要介绍及覆盖测试方向

本项目为 ERP 系统，主要面向 大型商超，旨在为商超的多业务场景提供全方位的数字化支持。系统功能覆盖从基础信息管理、商品中心、到零售出库等核心业务模块。ERP 系统的目标是实现对复杂业务流程的高效管理，提高商超的运行效率，优化资源配置。

* 测试覆盖以下主要方向：

功能测试：

通过人工检测方法，分别从功能分解法、正向测试、反向测试、端到端测试、边界测试、用户操作模拟来验证 ERP 系统功能的完整性和准确性，包括登录、基础信息管理、商品中心、零售出库管理、采购订单管理等模块的操作流畅性与逻辑正确性。

接口自动化测试：

使用 pytest+allure+requests+cov 进行接口测试，主要覆盖以下关键接口：

1. 登录接口。
2. 新增商品接口。
3. 新增零售单接口。
4. 新增采购单接口。
5. 新增角色接口。
6. 新增用户接口。
7. 新增仓库接口。

性能测试：

使用 JMeter 测试系统在不同负载场景下的表现，包括：

稳定场景： 商品新增功能。

高负载场景： 会员新增功能。

异常场景： 零售出库操作。

测试方向细化：

测试请求和响应的时效性。

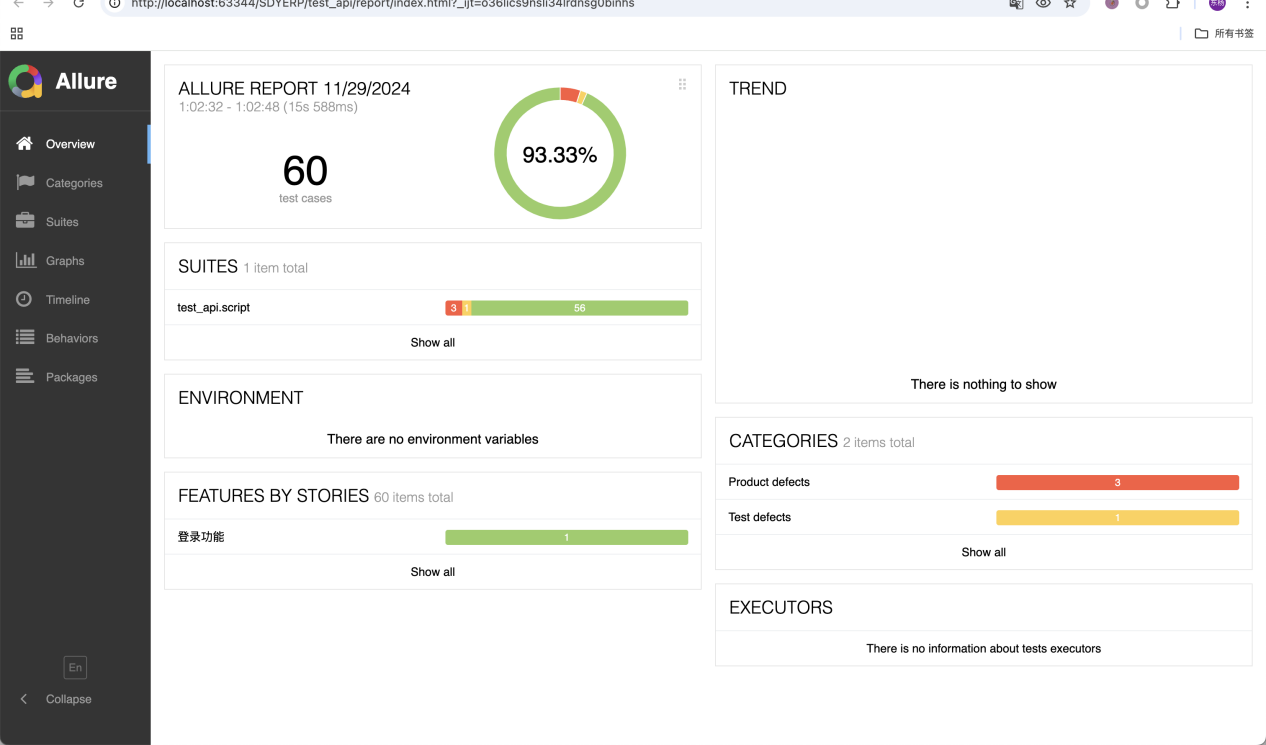
系统在高并发、极限负载下的响应与稳定性。

异常情况下的错误处理能力及降级策略。

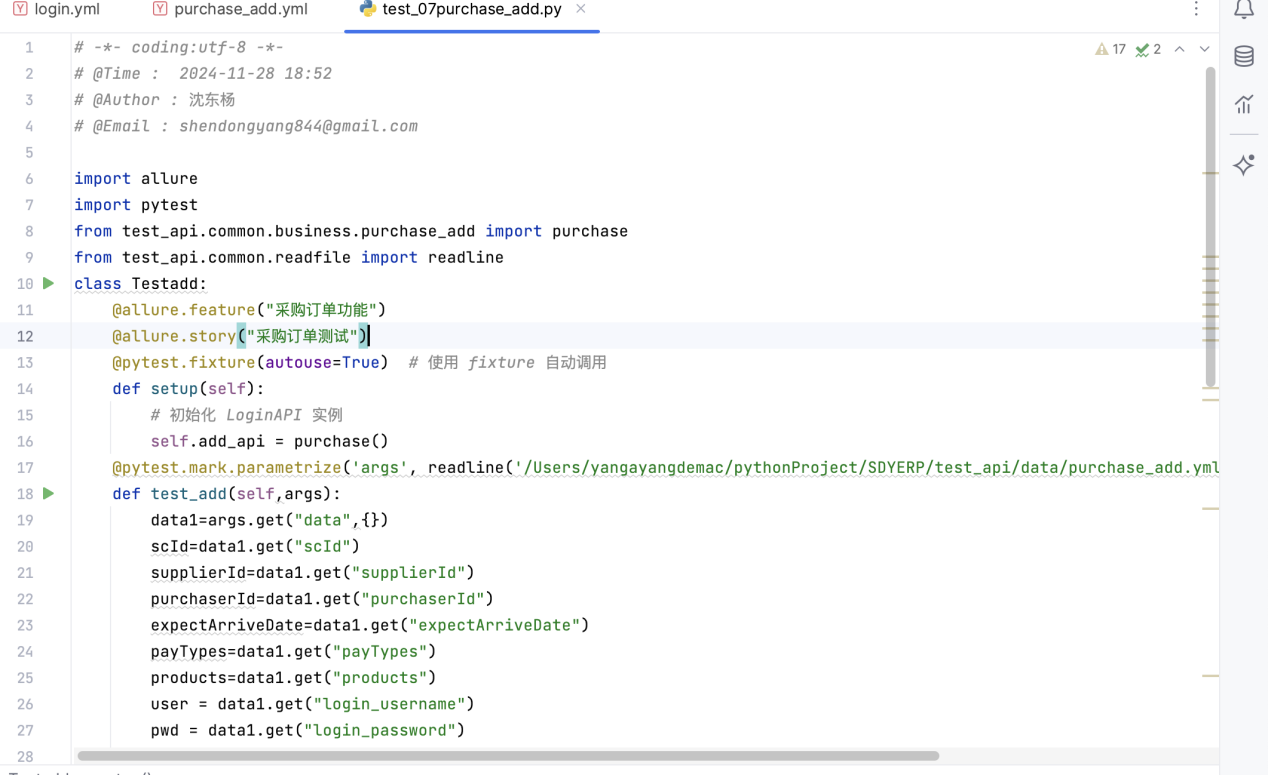
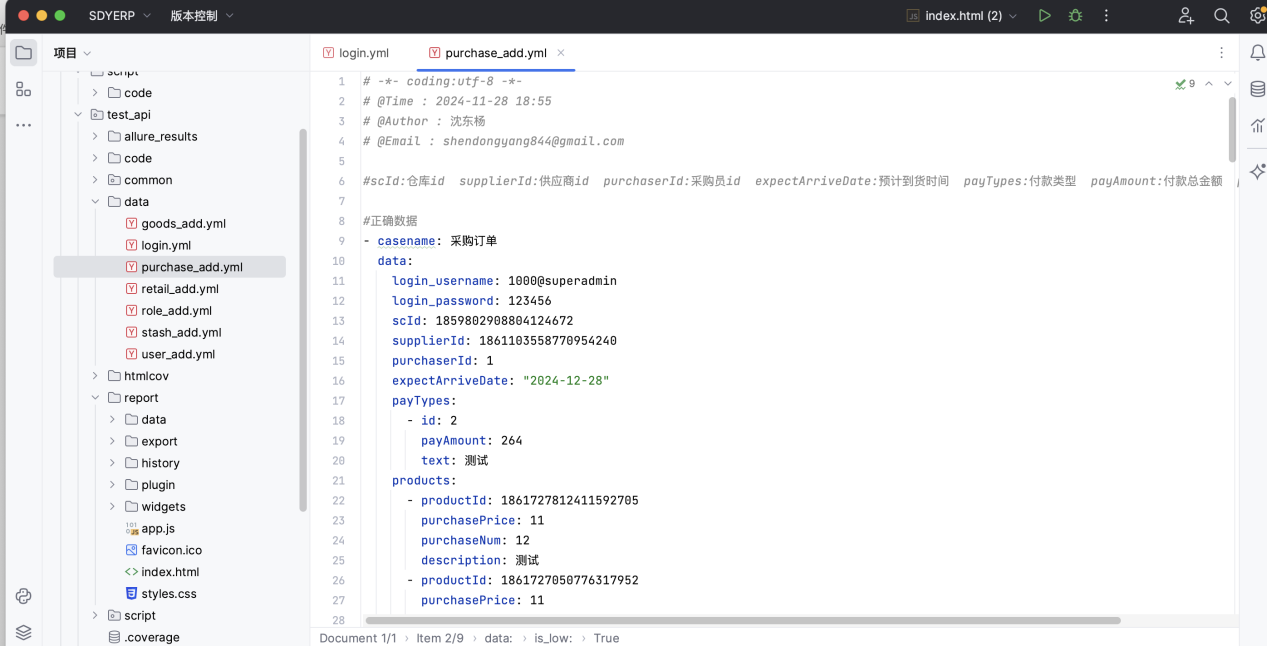
## **项目实施测试亮点**

### 接口自动化测试亮点：

1. 框架与工具

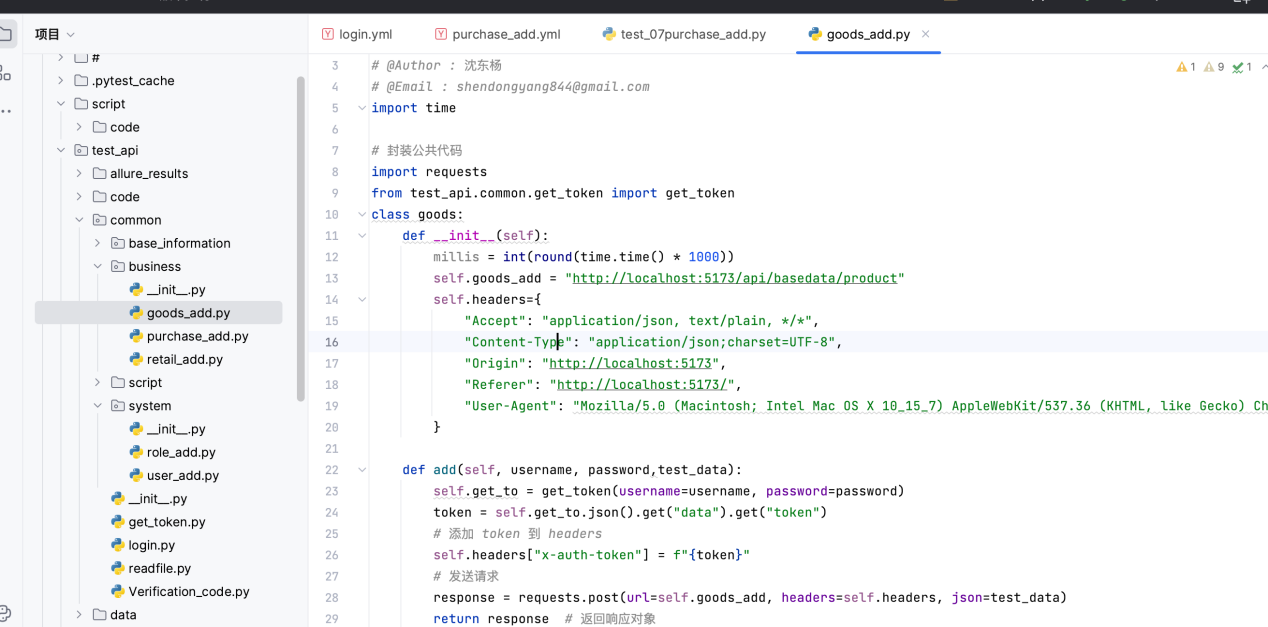
* 使用 pytest 框架，结合 allure 生成清晰、可视化的测试报告，支持快速定位问题，提升问题排查效率。

1. 参数化测试

* 测试用例采用 YAML 参数化封装，通过自定义方法读取并解析 YAML 文件的数据结构，结合 @pytest.mark.parametrize 动态加载参数，极大提升了多场景测试的覆盖率与测试效率。
* 灵活支持正向用例、反向用例和边界用例的设计，实现测试数据与测试逻辑分离，便于后续维护和扩展。

1. 接口请求封装

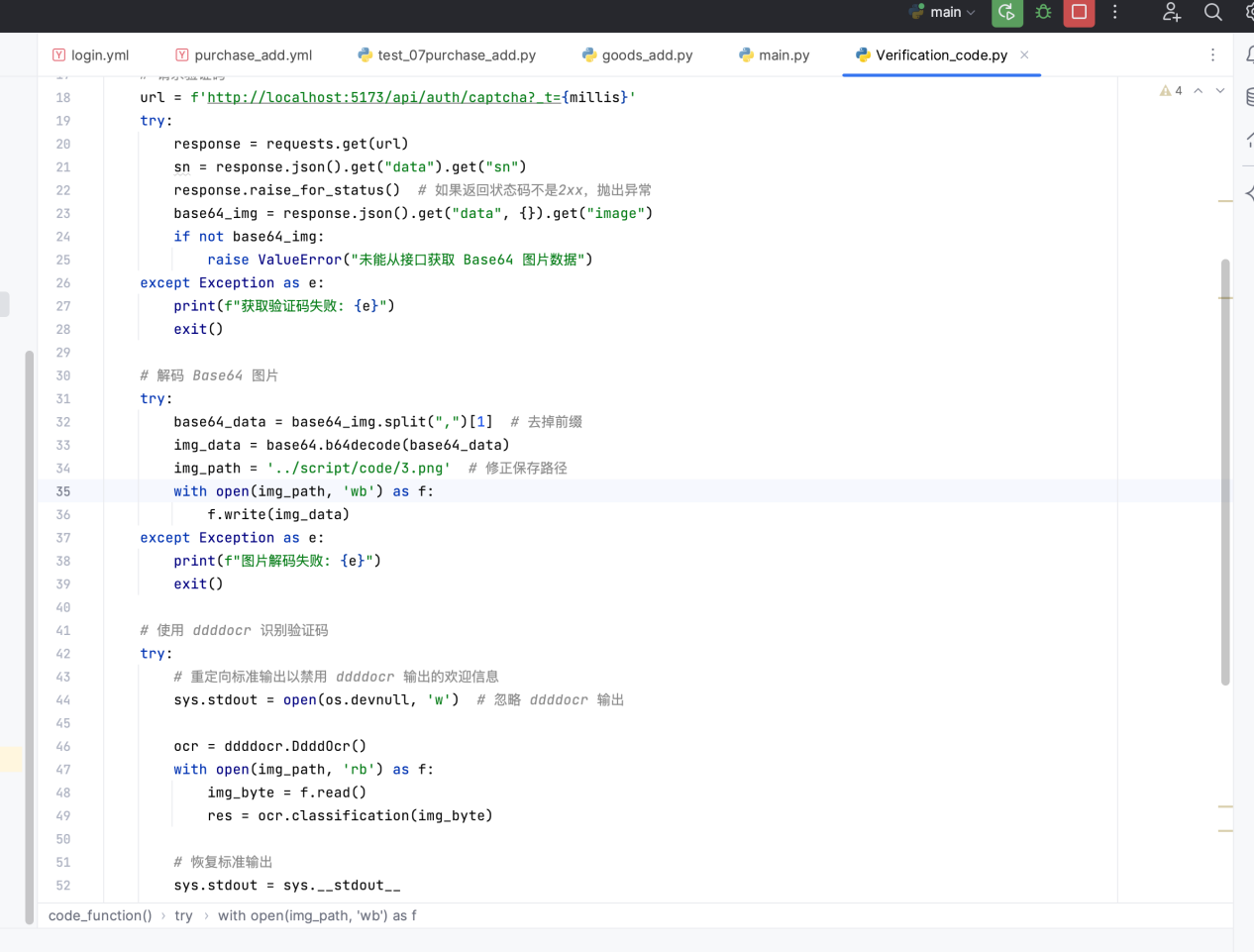
* 基于 requests 库封装了接口请求方法，支持常用的 GET、POST 请求，统一了参数处理、响应结果验证等逻辑。
* 所有接口请求封装成独立的模块，支持动态添加请求头和多种认证方式，适应复杂业务场景的需求。



* 响应数据的格式校验和断言逻辑封装为通用方法，提升测试的可维护性和扩展性。

1. 验证码处理自动化

* 登录流程中，专门封装了验证码处理逻辑，通过读取验证码图片的 Base64 编码并解码后，使用 ddddocr 进行高效识别，模拟真实场景下的验证码校验操作。
* 通过自动化处理减少了手工干预，提升测试的稳定性和连续性。



1. 全面的实例设计

* 实例覆盖登录、创建、查询、更新、删除等核心接口，兼顾不同业务场景的正反向测试逻辑。
* 针对高频接口和复杂交互接口，细化测试用例设计，确保接口的稳定性和正确性。

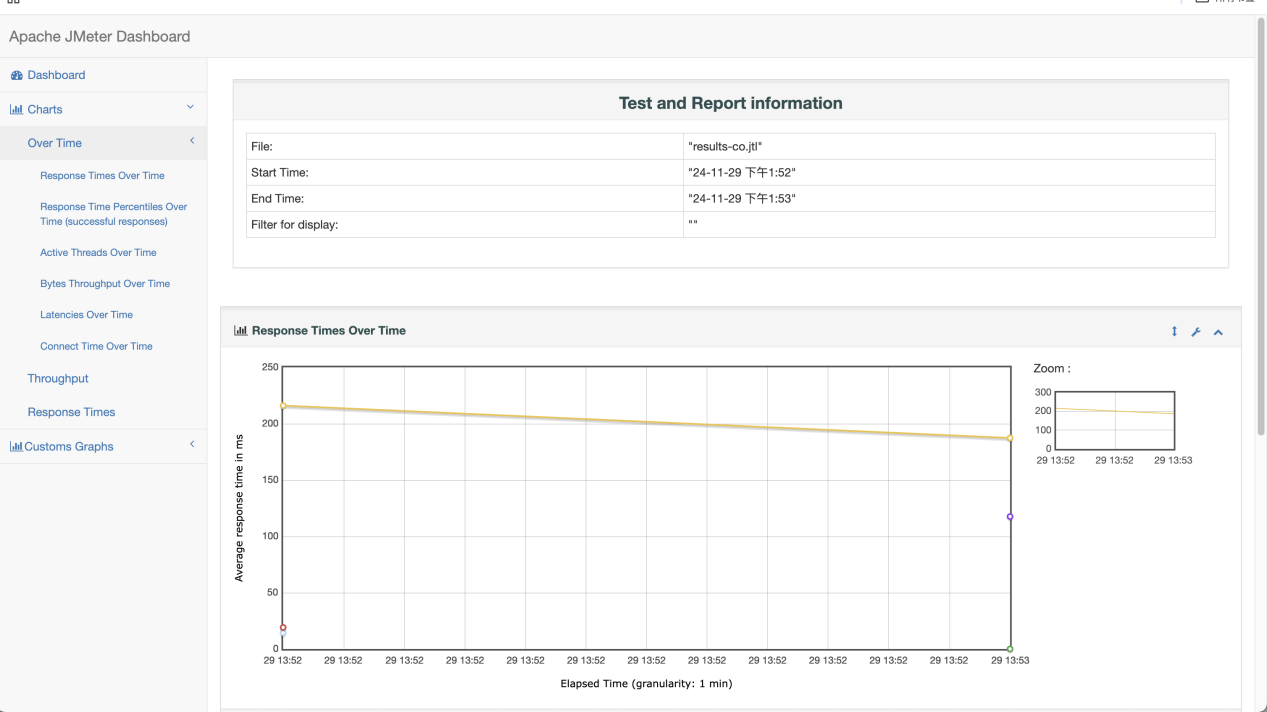
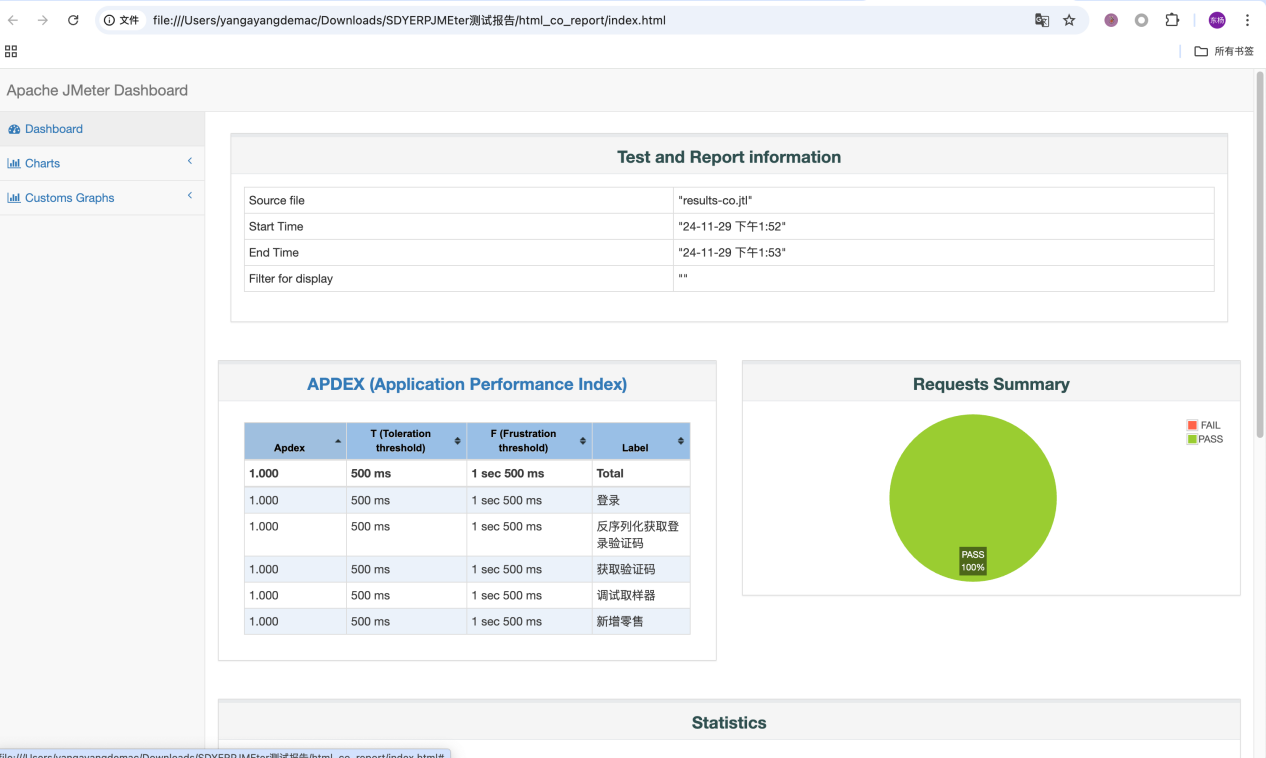
1. 优势与成果

* 模块化与高复用性：将测试逻辑、参数配置和接口请求完全解耦，方便未来扩展新功能或修改接口逻辑。
* 智能化处理：验证码自动识别的实现，模拟真实用户操作场景，降低人为干预成本，提升测试逼真度。

### 性能测试亮点：

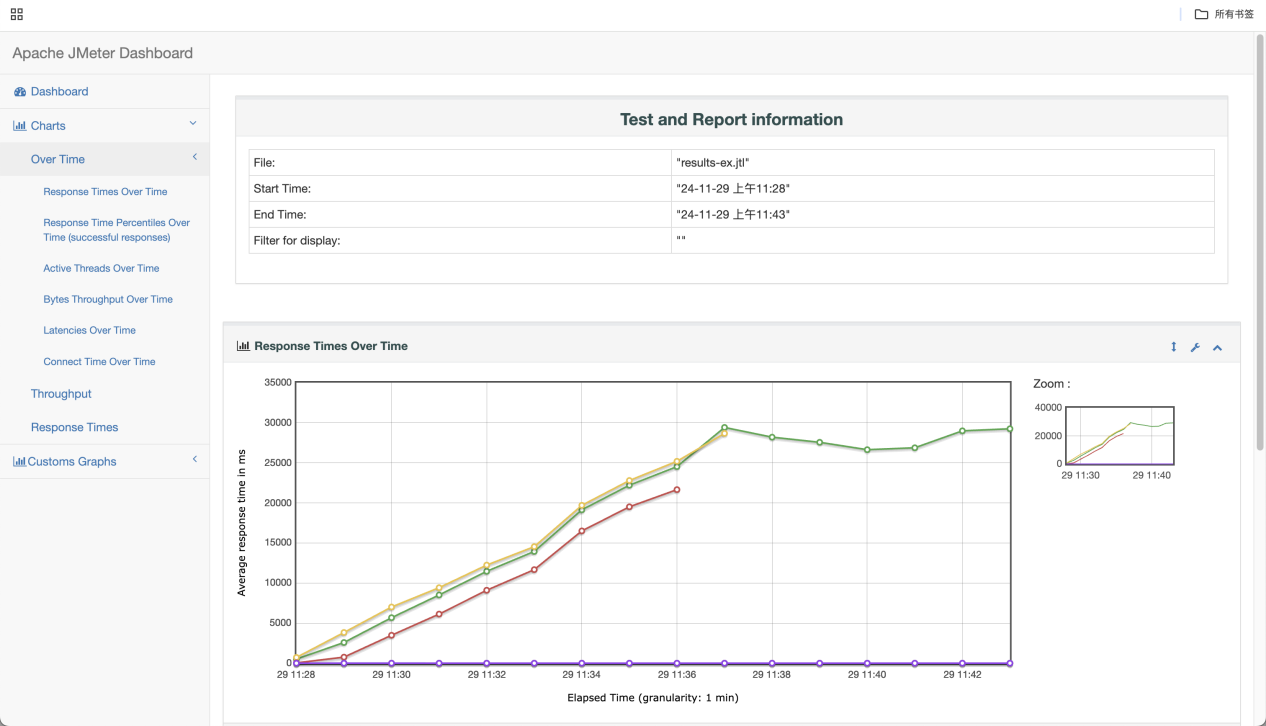
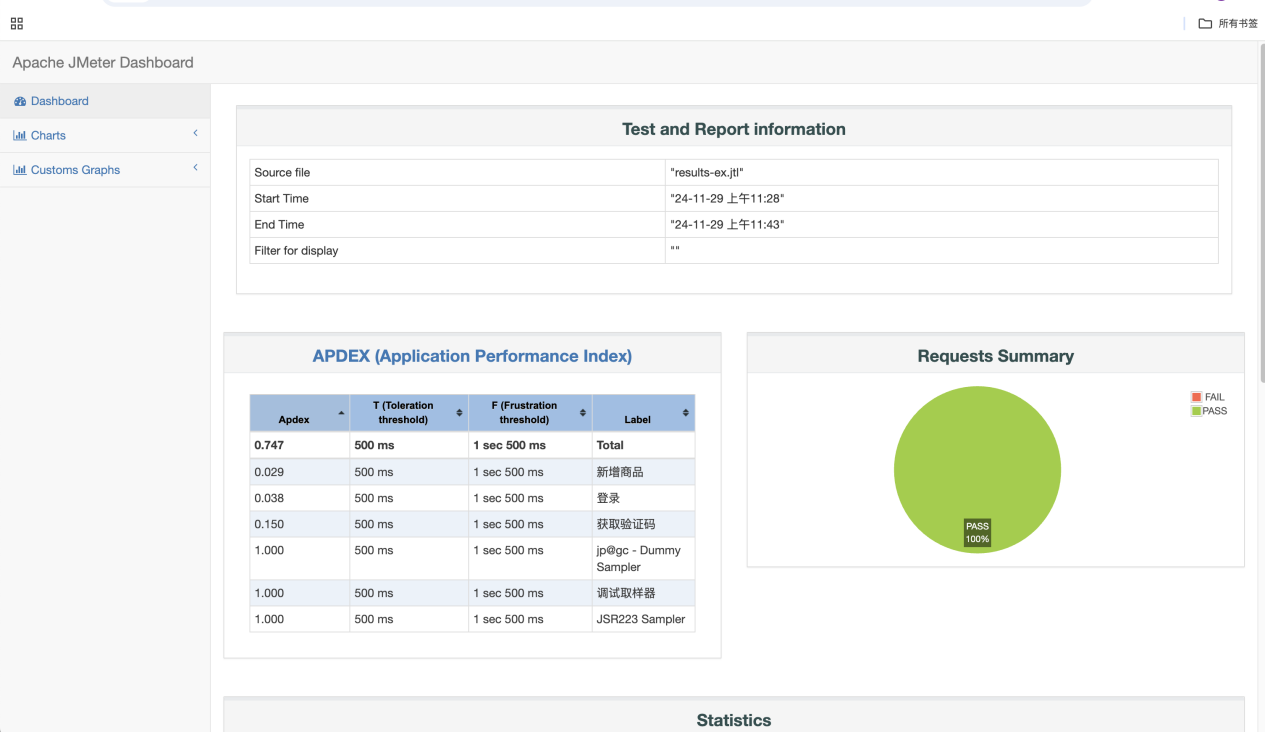
1. 稳定性场景

* 目标：模拟日常负载场景，验证系统在正常业务操作下的稳定性。
* 场景描述：通过商品新增操作，设定合适的并发用户数和操作频率，评估系统在典型业务负载下的响应时间、资源使用率及错误率，确保其能长期稳定运行。



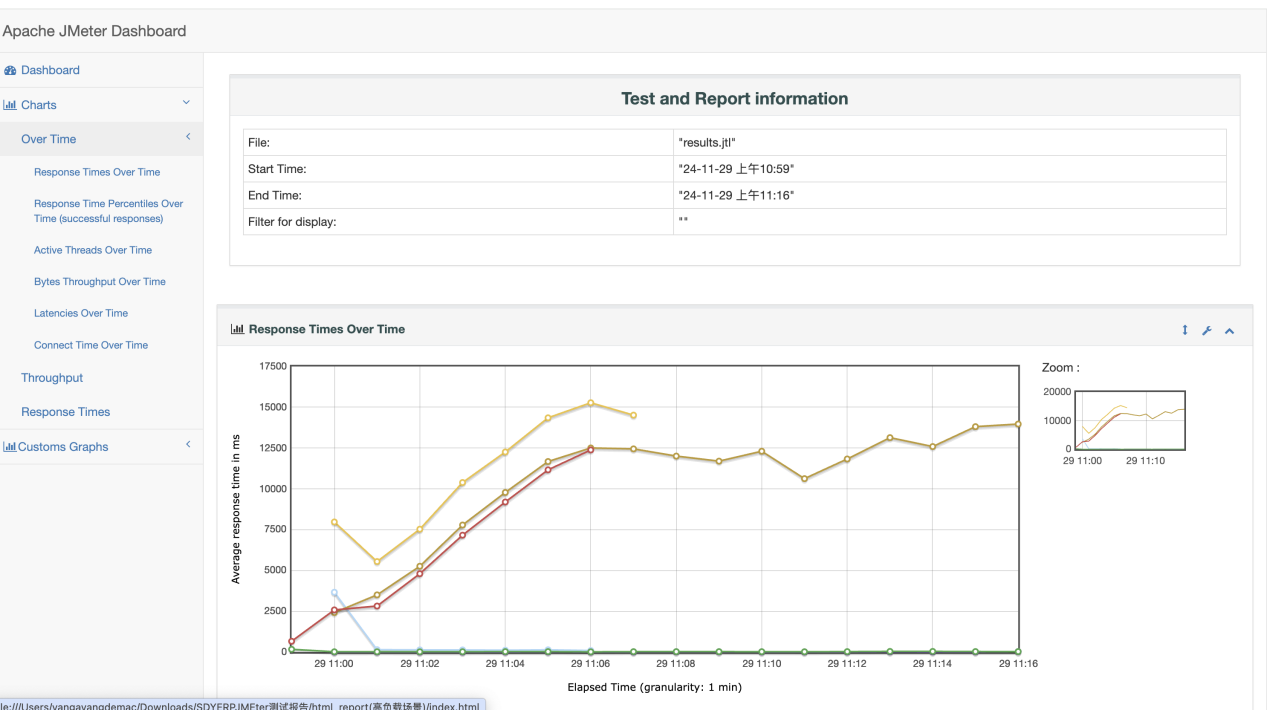
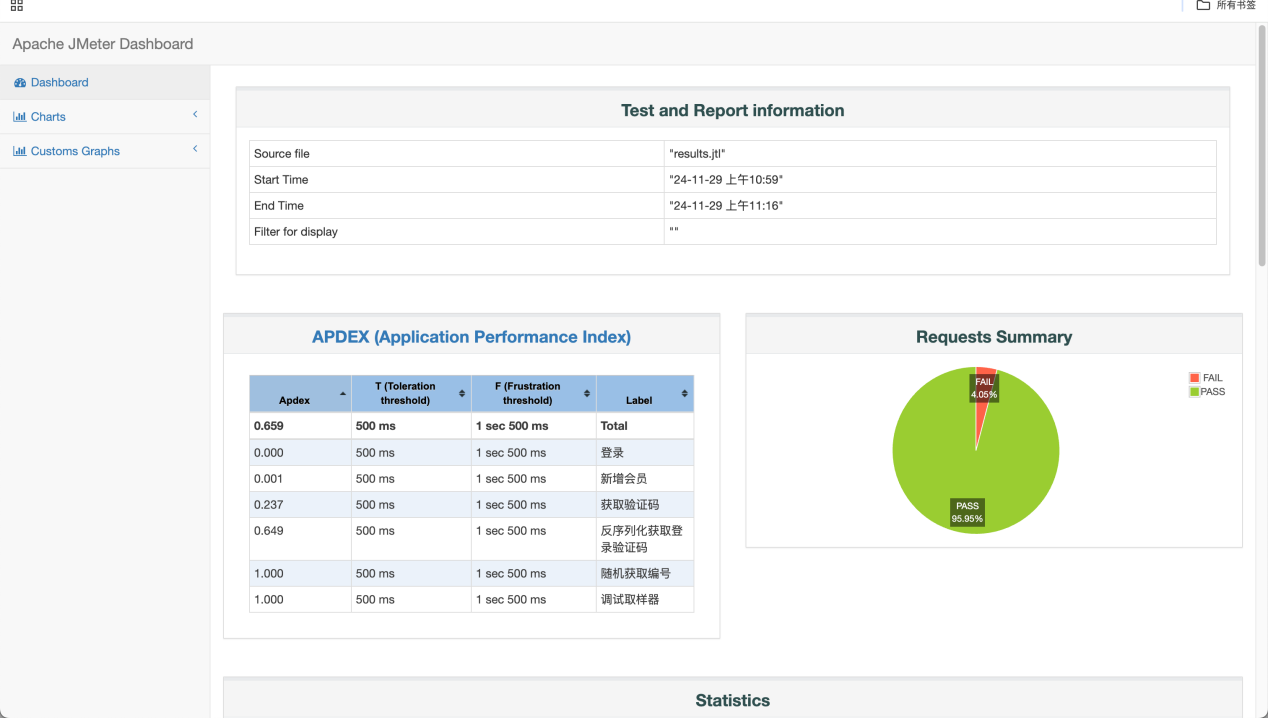
1. 高负载场景

* 目标：在高并发压力下探测系统的性能瓶颈，评估其在接近最大承载能力时的表现。
* 场景描述：通过会员新增操作模拟高并发用户行为，结合同步定时器和线程分组策略，在逐步提升负载的同时监控系统性能（如 CPU、内存、数据库连接池占用率等），帮助定位瓶颈并为后续优化提供依据。



1. 异常场景

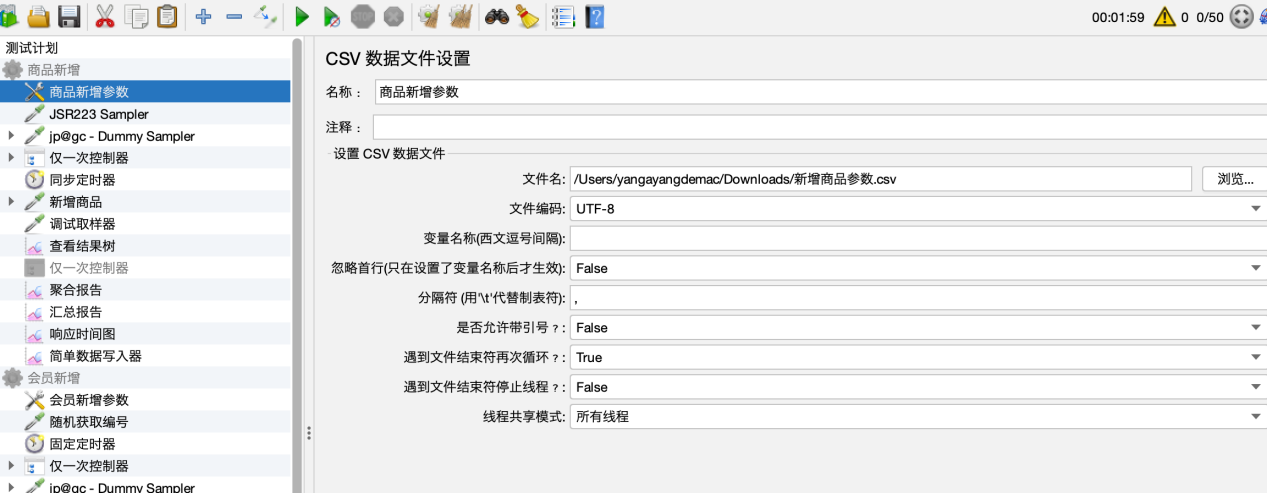
* 目标：验证系统在高负载和异常网络条件下的恢复能力及容错性。
* 场景描述：通过零售出库操作，模拟不稳定网络环境（如延迟、丢包等）与高并发的组合场景，分析系统在出现异常时的请求处理能力、失败重试策略及故障恢复效率。
* 在高负载场景中加入模拟网络延迟增加同步定时器，模拟多个用户同一时间进行操作，导致响应时间变长，出现异常

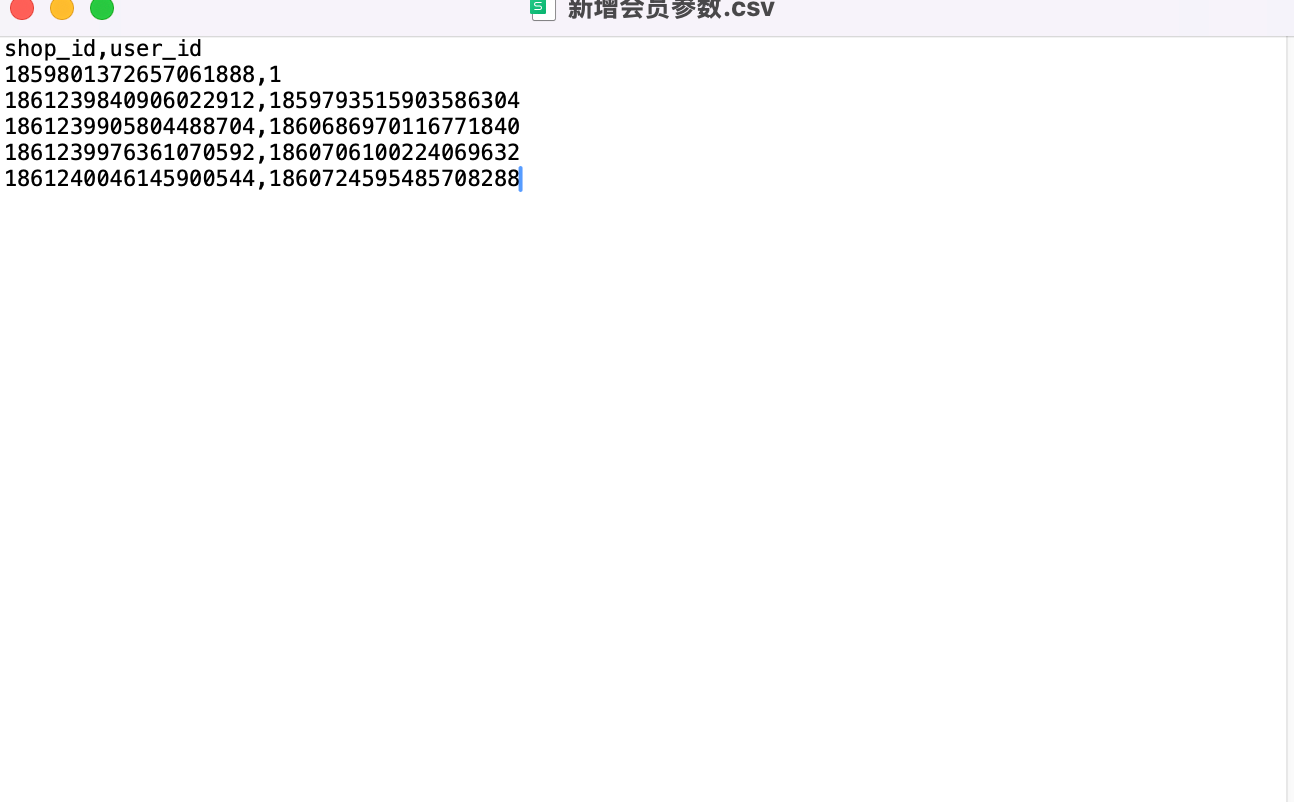


1. 自定义脚本和动态参数化

* 亮点：利用 JSR223 Sampler 编写自定义 Groovy 脚本，高效实现测试过程中参数的动态生成与变量设置。
* 优势：在复杂场景中，通过动态处理请求参数（如随机数生成、时间戳注入等）提升测试灵活性，并减少重复性工作量。

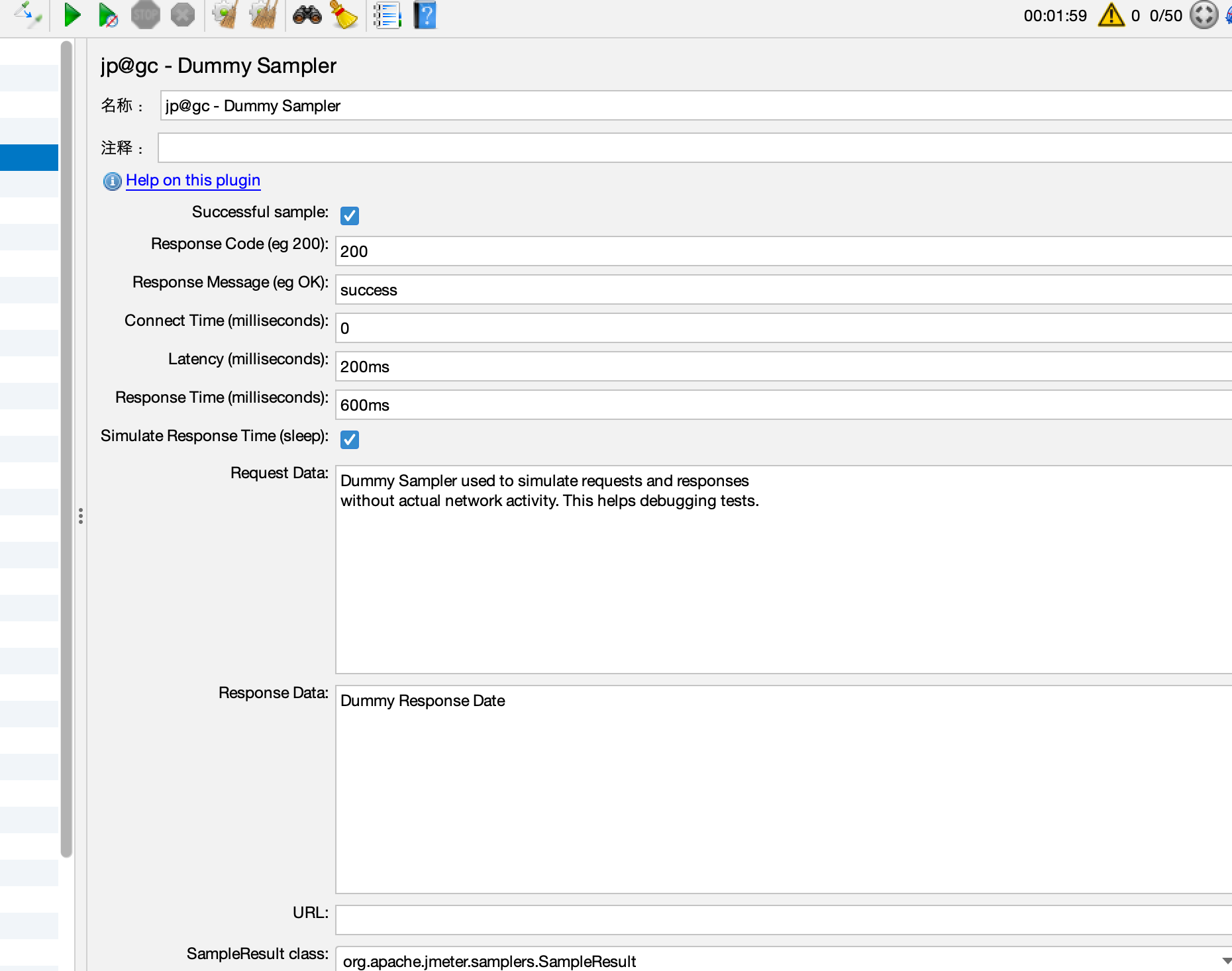
1. 批量参数化测试

* 亮点：使用 CSV 数据文件结合 JMeter 的 CSV Data Set Config 元件，实现海量测试数据的批量参数化。
* 优势：灵活定义和管理测试数据，支持多场景、多用户模拟，提高测试覆盖率和精准度，尤其在复杂业务场景中效果显著。



1. 网络延迟模拟

* 亮点：通过 Dummy Sampler 模拟真实的网络延迟环境。
* 优势：结合定时器（如 Uniform Random Timer、Constant Timer），在测试中模拟用户操作的实际间隔时间，确保测试结果更贴近真实生产环境。



1. 高效分析和可视化

* 亮点：合理使用 JMeter 监听器（如 Aggregate Report、Summary Report），实时监控和记录测试数据，并结合网络响应时间分析各项性能指标。
* 数据处理：通过使用 Simple Data Writer 保存关键测试数据，便于后续分析和生成可视化报告。
* 结果展示：借助 JMeter 内置的 HTML 报告功能，将测试数据转化为直观的图表与统计结果，为性能问题的沟通与优化提供清晰依据。

### 功能测试亮点：

1. 全面覆盖的测试用例设计

* 亮点：通过细致的需求分析，手工设计覆盖所有业务功能的测试用例，确保测试范围涵盖正常流程、异常流程及边界情况。
* 具体：
* 根据需求和业务逻辑，逐一验证核心功能的正确性。
* 针对高频使用功能、关键业务模块进行了重点测试。

1. 测试用例报告的详细记录

* 亮点：撰写了详尽的测试用例报告，包括每个测试用例的执行步骤、预期结果、实际结果和测试结论。
* 优势：清晰记录测试过程，方便团队成员复现问题或参考报告，保障测试透明性与可追溯性。

1. 异常处理与容错性验证

* 亮点：在手工测试中主动设计异常输入场景（如非法数据、操作中断等），验证系统对异常条件的提示和处理能力。
* 具体：检查系统是否能友好提示用户，避免数据丢失或逻辑错误，确保其容错性。
* 场景化测试覆盖
* 亮点：结合实际业务场景设计测试用例，从用户视角模拟真实操作流程。
* 具体：
* 完整验证了复杂的业务流程，例如多步骤交互、数据关联和结果展示。
* 对跨模块功能进行了场景联动测试，确保数据传递和流程逻辑的正确性。

1. 边界和细节测试

* 亮点：特别关注输入边界（如字段长度、特殊字符等）及系统的处理细节。
* 具体：验证表单输入限制、错误提示、页面跳转逻辑等，确保系统在细节上的精细化表现。

1. 人工灵活性与敏感度

* 亮点：手工测试能更敏锐地捕捉界面异常、逻辑错误及用户体验问题。
* 通过实际操作识别可能影响用户体验的问题，例如按钮不可用、布局错乱等。
* 灵活应对需求变更或系统调整，无需额外编写脚本。

1. 测试结果的深入分析

* 亮点：结合测试结果，总结系统存在的问题类型、分布情况及修复建议。
* 具体：通过手工记录测试发现的 bug，包括复现步骤和修复建议，为开发团队提供精准的改进方向。

1. 专注用户视角

* 亮点：手工测试从用户视角验证系统功能，确保实际操作中的易用性和直观性。
* 具体：通过模拟终端用户的操作，评估系统的友好性和使用流畅度，为功能优化提供参考依据。

## **测试过程中涉及的关键技术**

1. 接口测试关键技术：

使用 pytest+requests框架编写测试用例，支持测试数据的参数化。

借助 allure 生成美观的可视化报告，直观展示测试结果和执行日志。

集成代码覆盖率工具 cov，分析接口测试的覆盖程度。

1. 性能测试关键技术：

使用 JMeter 的 CSV Data Set Config 模块实现数据驱动测试。

编写 JSR223 Sampler 脚本动态获取和设置测试数据，增强测试的灵活性。

根据不同场景配置合理的线程组参数（如线程数、Ramp-Up 时间、持续时间），模拟真实的用户访问行为。

1. 功能测试关键技术：

模块化设计： 对系统功能进行分模块、分功能点测试，确保各模块之间无冲突。

边界测试： 针对关键输入字段设计边界值和异常输入测试。

场景还原： 模拟商超真实业务场景（如商新增、商品查询），验证系统逻辑的正确性。

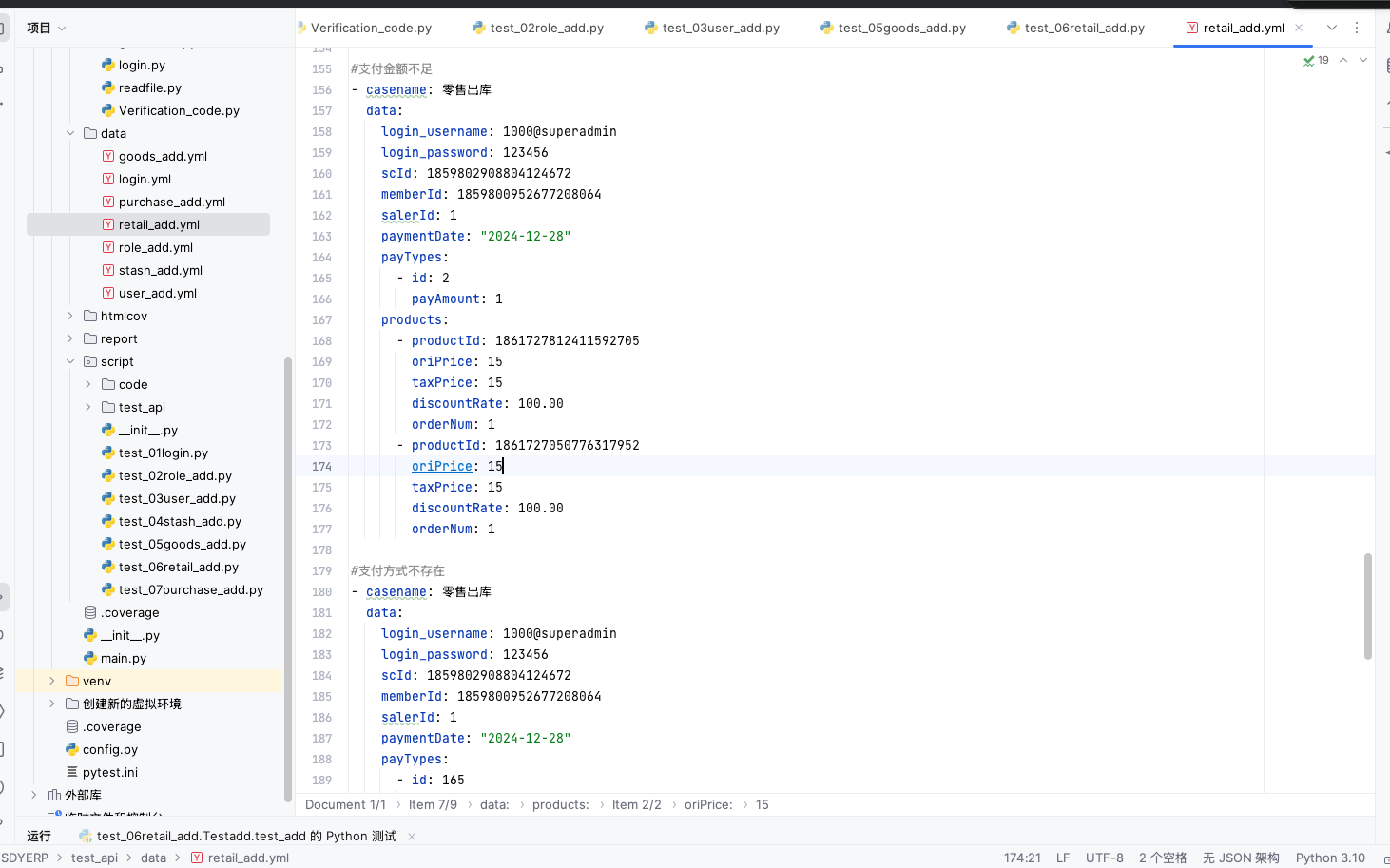
1. 测试环境与工具支持：

测试环境：pytest、JMeter、python3、wps。

测试报告工具：allure、JMEter。

## **测试中发现的问题与优化建议**

1. 接口自动化测试:在进行接口测试的时候我分别从商品中心、采购管理、零售管理、 登录等核心模块进行设计测试
2. 在进行零售出库的接口测试时发现了功能性的问题，在支付金额不足以及支付方式不存在的情况下仍可成功出库，是一个功能性的bug，正确的支付金额应该是30，我通过反面测试填写了支付金额为1，仍可通过。其次支付方式的id仅有1-6，我填写了165仍可成功。

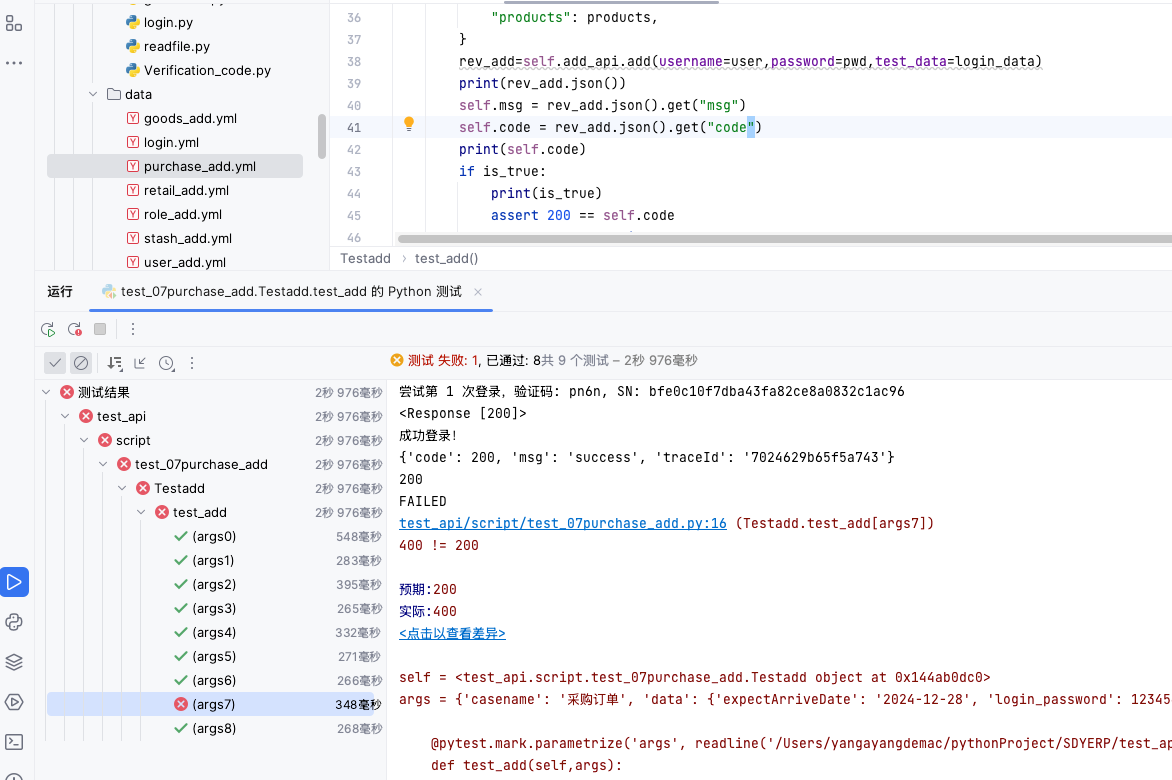






1. 在进行采购订单新增时，也存在相应的问题，我通过反面测试设计用例，在金额不足的情况下，仍可添加成功。





1. 性能测试：

为了更全面地评估系统性能，我设计了以下几个测试场景：

* 基本并发测试：从 1000 线程逐步增加到 3000 线程，监测响应时间、吞吐量及异常率。
* 高负载+同步定时器测试：在 3000 线程场景下，添加 1000 组同步定时器，模拟复杂业务场景。
* 异常点复测：针对 3000 线程 + 同步定时器的场景，进行多次测试以定位问题是否具有偶发性。
* 降级测试：从异常场景逐步降低线程数至 2000，验证系统稳定运行的安全边界。
* 线程组：
* 使用 JMeter 的线程组，模拟 1000、2000、3000 等不同并发用户。
* 同步定时器：
* 每次请求前添加同步定时器，1000 组并行请求启动，模拟真实业务压力。
* 采样器配置：
* 请求类型以 POST 为主，模拟系统的实际业务场景。
* 监听器：
* 配置聚合报告、图形结果、HTML 报告生成器，实时监控性能指标。

测试执行过程

1. 初始测试：我从 1000 线程起步，逐步增加线程数，监测系统的响应时间和吞吐量。

2. 高负载测试：

* 设置线程数为 3000，同时添加 1000 组同步定时器。
* 发现异常率上升至 4% 左右，并观察到部分接口响应超时，导致系统负载过高。

3. 异常验证：

* 降低线程数至 2000，异常率恢复到 0%。
* 通过逐步调整同步定时器的数量，确认问题与系统无法在高负载下处理过多同步请求相关。

1. 数据采集：每轮测试均保存 results.jtl 文件，并生成 HTML 报告进行分析。

问题分析

1. 高线程异常：

* 在 3000 线程场景下，系统资源（CPU、内存）使用率接近瓶颈，部分请求超时。

1. 同步定时器影响：

* 1000 组同步定时器增加了请求的处理延迟，同时加剧了线程间的竞争。

1. 资源不足：

* 测试发现，某些服务在高并发场景下存在资源分配不足的问题，如线程池配置过小或数据库连接池耗尽。

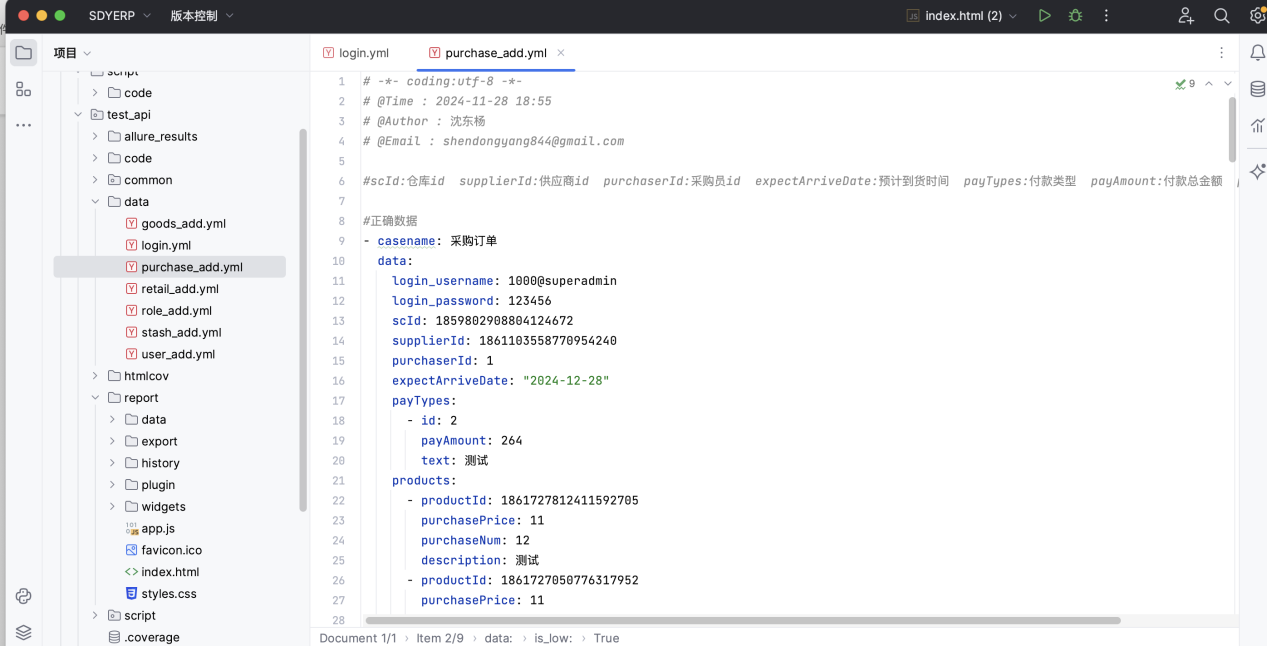
解决方案与优化建议

1. 优化线程池：根据测试结果调优线程池配置，确保线程分配合理。
2. 减少同步请求：评估业务逻辑，减少不必要的同步调用。对于确需同步的场景，可考虑分批处理或异步化。
3. 数据库优化：增加数据库连接池的容量，减少数据库查询延迟。
4. 服务限流：针对高并发场景，引入限流策略，保护核心服务稳定运行。
5. 功能测试： 在功能测试中，我选择了该系统中的核心功能点：商品新增、采购订单的新增、零售出库的新增等。我通过用户行为模拟、动态数据验证、根据经验构造错误数据结构、结合性能测试的结果来分析功能的稳定性等来切实的检测功能点是否能够正常合理的使用。

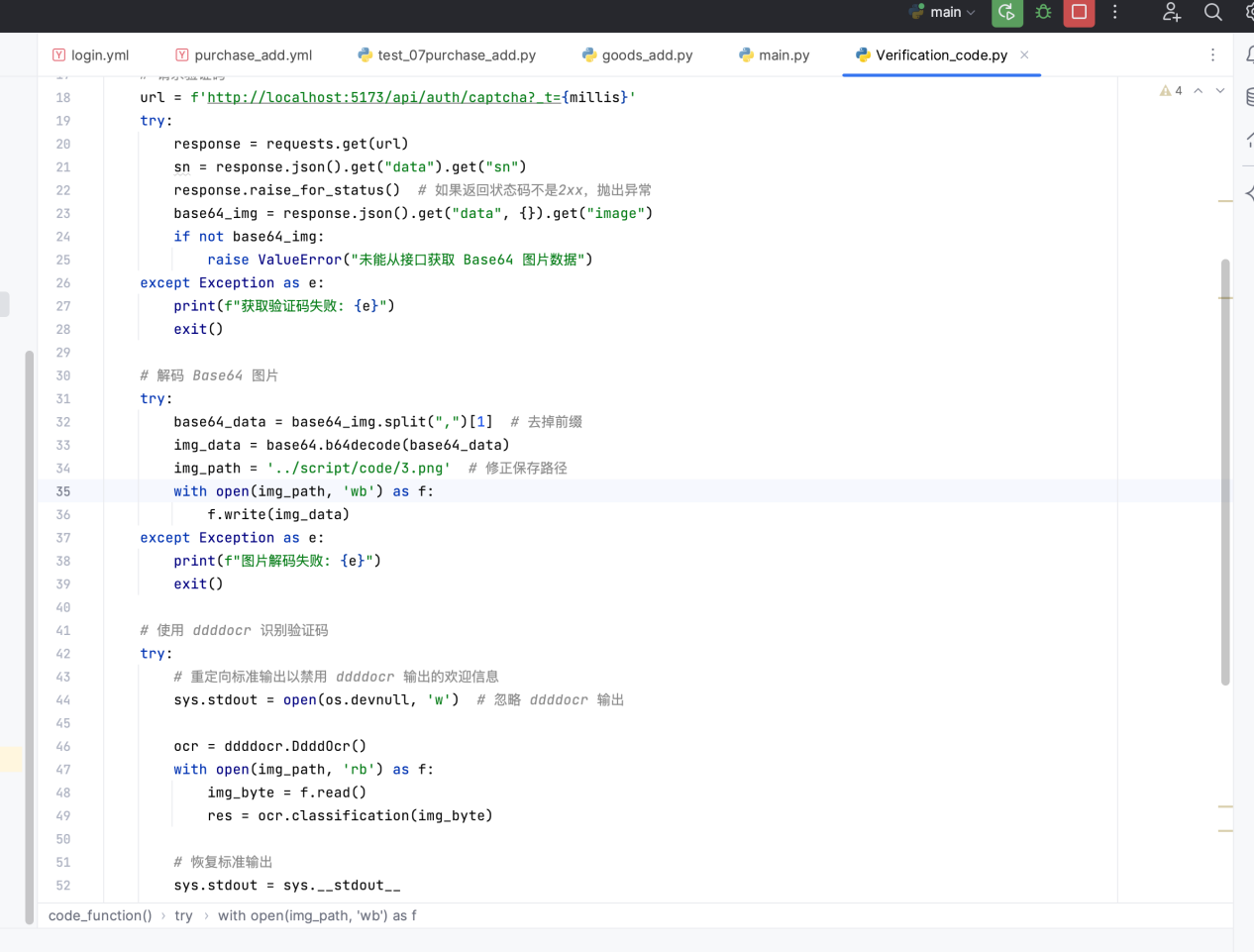
## **项目相关的截图/照片**

## 截屏2024-11-29 14.43.01

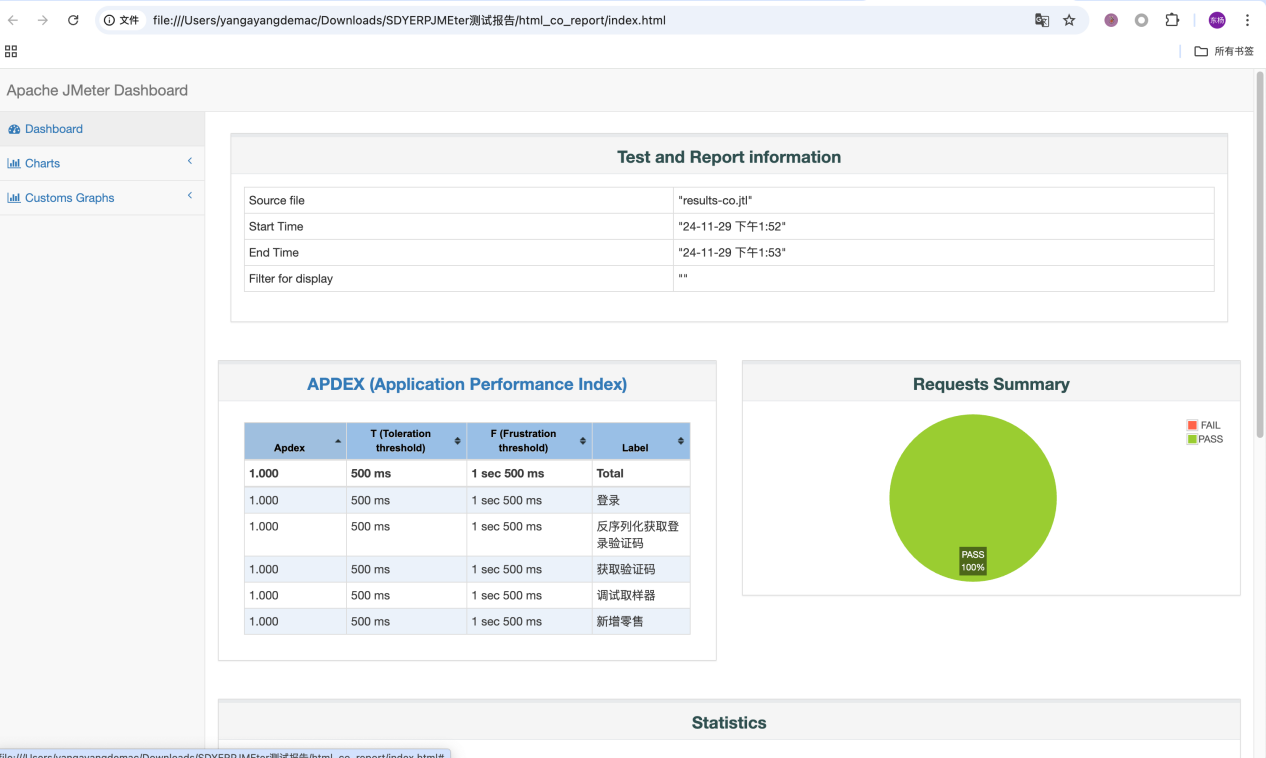
5.1接口测试报告



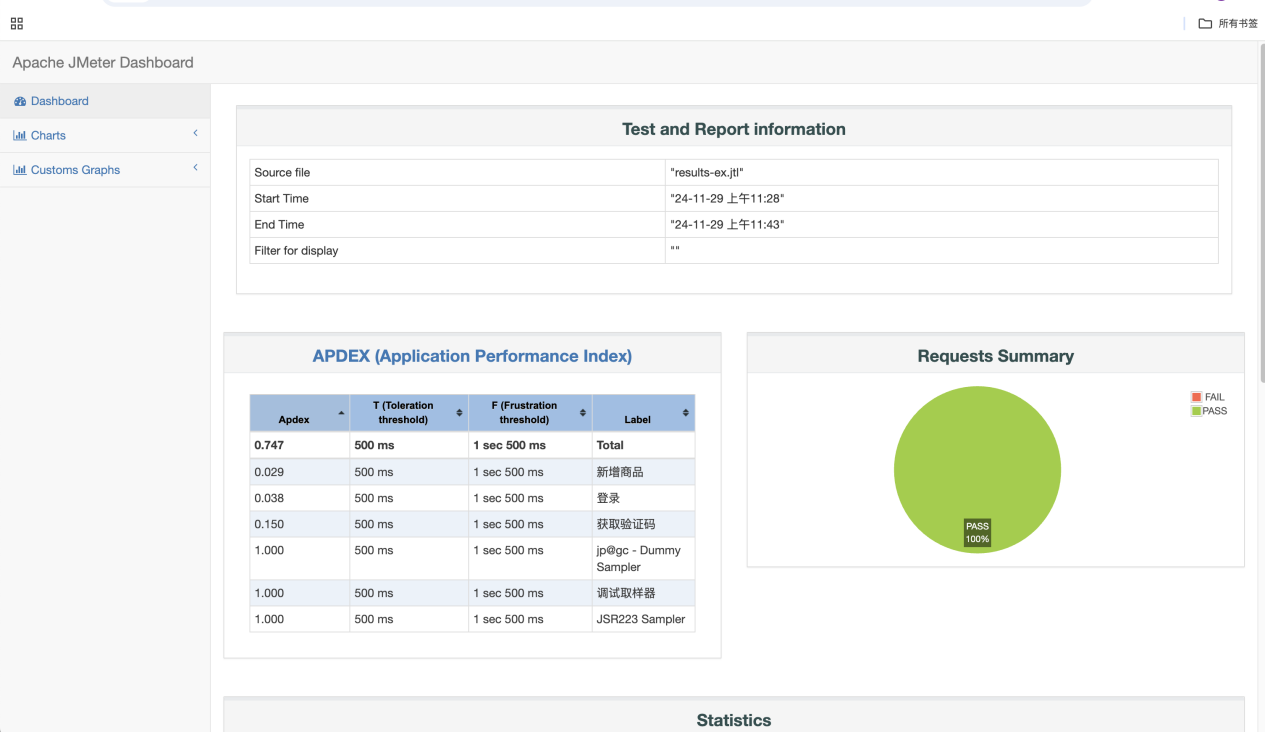
5.2参数化封装数据



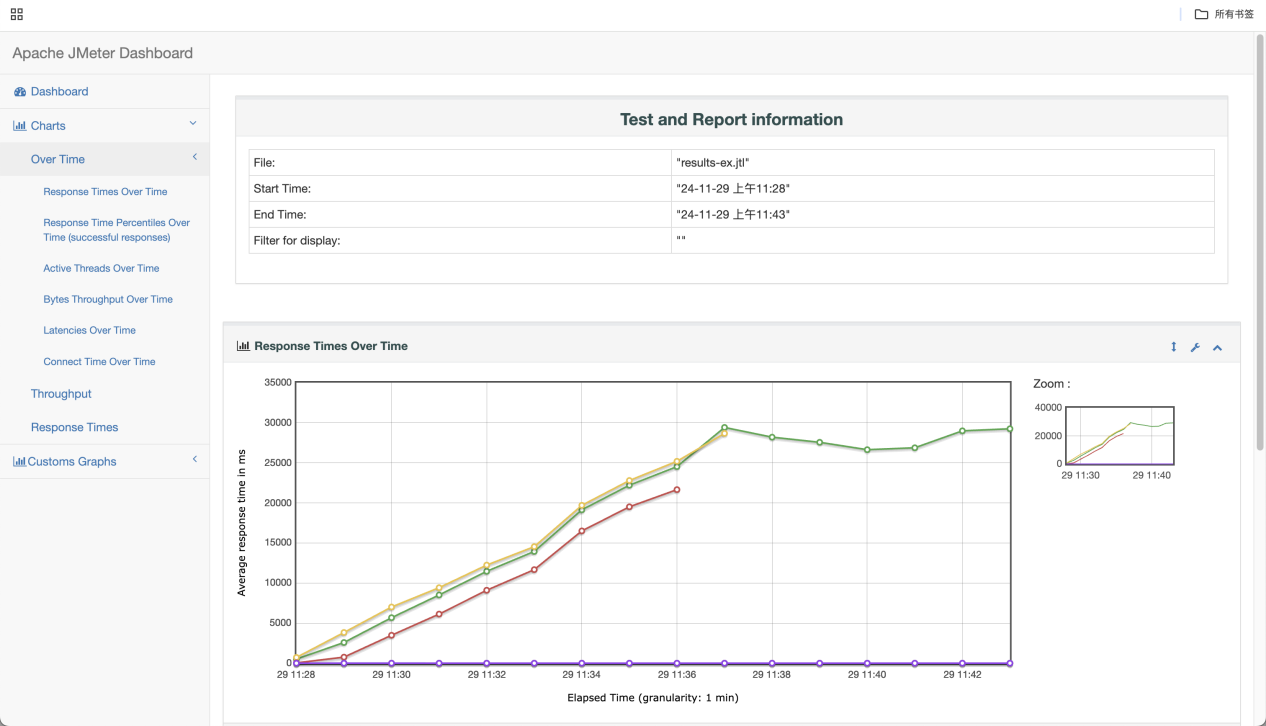
5.3识别图片验证码



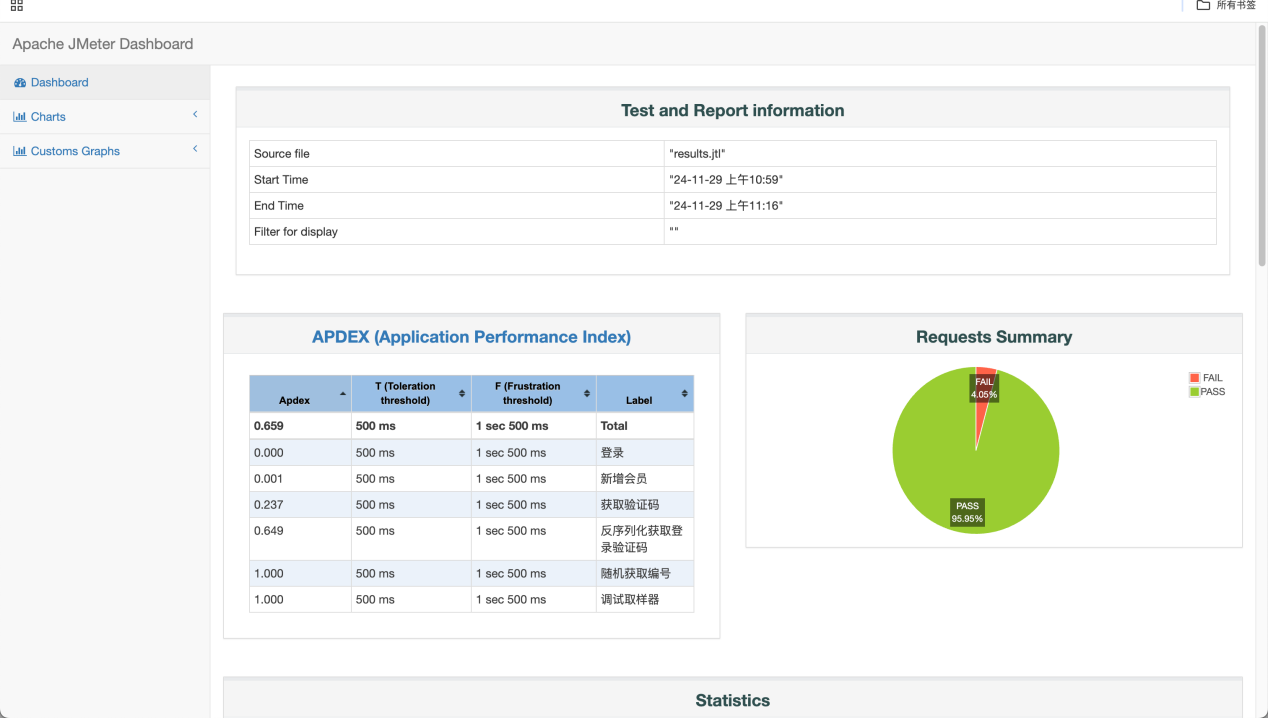
5.4性能测试（稳定场景）报告



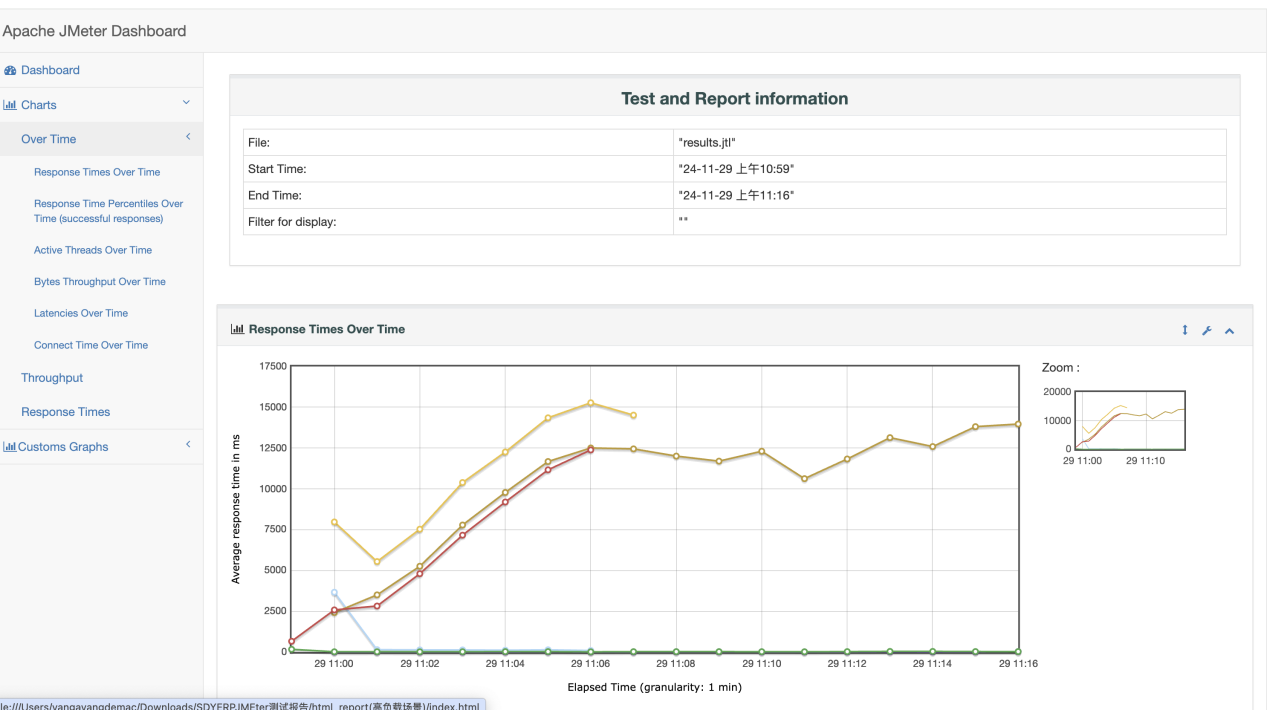
5.5性能测试（高负载场景）报告



5.6性能测试（高负载场景）时间响应报告

、

5.7性能测试（异常场景）报告



5.8性能测试（异常场景）响应时间报告



5.9功能测试用例报告

## **是否首次公开发布**

是否首次公开：是

在接口自动化测试中增加了代码覆盖率统计功能，提升了测试的全面性。

性能测试场景更贴近实际业务，新增了高负载和异常场景。

功能测试覆盖了更多边界值与异常输入的测试用例。