

《软件测试技术》课程

实 验 指 导 书

2022 年 10 月

实验 1 完成一个简单的测试过程

（共 2 学时）

1. 实验目的

- （1）对软件测试有一个直观的感受。
- （2）对软件测试过程有一个完整的感受。
- （3）加强对基本概念的理解。

2. 实验内容

- （1）基于软件工程或其他课程开发的软件系统，选定 1~2 个功能模块。如果没有，就针对 <https://www.saucedemo.com/>或 <http://automationpractice.com/index.php> 进行分析。
- （2）先做一些初步的功能测试分析，如了解功能操作的路径，有哪些输入数据，有哪些特殊、异常的数据或操作。
- （3）基于上述的分析，像用户使用产品操作软件一样，进行手工测试，发现缺陷并记录。
- （4）完成一个非规范的测试报告。

3. 实验过程

- （1）选定一个被测系统或被测模块（统称为“被测对象”）；
- （2）针对被测对象进行简单的分析，明确要测试的子功能及其功能点（更细的划分），并记录下来每个功能点（测试项）；
- （3）针对每个功能点，了解其应用场景、操作和输入/输出数据，并做记录；
- （4）根据上述记录，尽可能覆盖其应用场景、完成各种操作和数据的输入，并检查输出的结果；
- （5）在测试过程中尽可能做些异常操作（如连续单击）、输入一些特殊的数据，如输入特别长的字符串、输数字的地方输字母、输入空值、输入边界值等；
- （6）当检查结果不对时，记录异常情况或缺陷；
- （7）最后整理上述记录，思考和总结整个过程，形成一个完整的报告。

4. 实验结果

交付最后整理的测试报告，包括功能点、测试场景、输入数据、异常/缺陷等记录列表，以及自己的感受和总结。

实验 2 单元测试实验

（共 2 学时）

使用 JUnit 工具，针对 Spring Unit Testing 控制器代码中 ItemController 类进行测试，编写对应的测试类以完成单元测试，最终提交测试代码。

```
package com.sprint.unittesting.unittesting.controller;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import
com.sprint.unittesting.unittesting.business.ItemBusinessService;
import com.sprint.unittesting.unittesting.model.Item;

@RestController
public class ItemController {

    @Autowired
    private ItemBusinessService businessService;

    @GetMapping("/dummy-item")
    public Item dummyItem() {
        return new Item(1, "Ball", 10, 100);
    }

    @GetMapping("/item-from-business-service")
    public Item itemFromBusinessService() {
        Item item = businessService.retreiveHardcodedItem();

        return item;
    }

    @GetMapping("/all-items-from-database")
    public List<Item> retrieveAllItems() {
        return businessService.retrieveAllItems();
    }

}
```

实验 3 系统功能测试

(共 3 学时)

1. 实验目的

- (1) 巩固所学到的测试方法;
- (2) 提高实际的测试能力。

2. 实验前提

- (1) 理解 6.1 节所描述的功能测试分析思路;
- (2) 熟悉测试用例设计测试方法 (第 3 章内容);
- (3) 选择一个被测试的 Web 应用系统 (SUT)。

3. 实验内容

针对被测试的 Web 应用系统进行功能测试, 发现其存在的缺陷。

4. 实验环境

- (1) 每 3~5 个学生组成一个测试小组。
- (2) SUT 可以是已经上线的外部系统, 最好是最近上线的商业系统; 也可以是重新部署的系统, 如在“软件工程”课程完成的应用系统。
- (3) 每人或每两人有一台 PC, 安装了 3 种浏览器, 如 IE、Firefox、Chrome。
- (4) 网络连接, 能够访问被测系统。

5. 实验过程

- (1) 小组讨论, 分析 SUT, 确定测试的范围, 列出测试项 (功能点)。
- (2) 按功能点分工, 如有 12 功能点, 则每组四人, 每个学生分到 3 个功能点。
- (3) 基于本章所学的测试方法, 每个学生设计 20 个或更多的测试用例, 这些用例要求相对关键, 对功能点的验证有效, 或发现 Bug 的可能性更高。
- (4) 每个学生向本组其他学生讲解自己是如何设计这些测试用例的, 其他学生提出意见, 然后大家讨论, 最后修改、完善之前写的测试用例。
- (5) 先选择 Firefox 执行全部测试用例, 然后根据自己判断, 决定在 IE、Chrome 上执行哪些测试用例。学生甲可以执行自己设计的测试用例, 也可以交叉执行, 即学生甲执行学生乙设计的测试用例、学生乙执行学生甲设计的测试用例。
- (6) 记录所发现的缺陷。
- (7) 基于 Selenium+Webdriver, 将之前设计的测试用例转化为测试脚本, 如 Java 格式的脚本, 在 Eclipse 的 JUnit 框架上执行和调试这些脚本。
- (8) 最后写出一个完整的测试报告, 包括分析思路、功能点清单、如何设计测试用例、脚本开发遇到哪些问题、测试环境、测试结果、Bug 列表、其它的体会感想等。

6. 实验结果

交付测试用例、测试脚本, 以及完整的测试报告。

实验 4 性能测试

(共 3 学时)

1. 实验目的

- (1) 巩固所学到的系统性能测试方法；
- (2) 提高使用系统性能测试工具的能力。

2. 实验前提

- (1) 掌握系统性能测试方法；
- (2) 熟悉系统性能测试过程和工具使用的基本知识；
- (3) 选择一个被测试的移动应用系统（SUT）。

3. 实验内容

- (1) 针对被测试的移动应用系统的后端（Web 服务器、应用服务器、数据库服务器等）进行性能测试。
- (2) 针对被测试的移动应用系统的前端（App）进行性能测试。

4. 实验环境

- (1) 每 3~5 个学生组成一个测试小组。
- (2) SUT 安装在一台或多台独立的服务器上，也可以是外部系统。
- (3) 每人或每两人有一台安装了 Java 运行环境的 PC。
- (4) 准备一台基于 Android 的手机或基于 iOS 的 iPhone 手机。
- (5) 网络连接，能够访问 SUT。

5. 后端性能实验过程

- (1) 小组讨论性能测试方案和小组成员分工。
- (2) 下载后端性能测试工具 JMeter 或 nGrinder。
- (3) 部署 JMeter 或 nGrinder 分布式测试环境，有控制器、多个测试机。
- (4) 选择 SUT 多个关键性的页面，录制或开发脚本。
- (5) 脚本参数化：测试数据文件配置、用户自定义变量等。
- (6) 采样器：覆盖两种协议（如 HTTP、JDBC 或 JMS）。
- (7) 针对 HTTP 协议，需要设置断言、Cookie 管理、默认值等。
- (8) 测试多组负载，如并发用户数为 100、500、1000 等。
- (9) 根据聚合报告、图形结果等，进行结果分析。

6. 前端性能实验过程

- (1) 登录 PerfDog 官网，下载正确的桌面应用程序。
- (2) USB 连接手机，自动检测添加手机到应用列表中。
- (3) 设定测试模式（USB 模式测试或 WIFI 模式测试）。
- (4) 选择被测试的 App 应用开始测试。
- (5) 测试过程中采集性能数据。
- (6) 保存测试结果。
- (7) 进行测试数据统计与分析。
- (8) 编写测试报告。

7. 实验结果

- (1) 记录测试完整过程（工具安装、环境设置、负载及其模式设置、脚本录制和开发、

监听器、结果分析)，包括脚本文件。

（2）提交前、后端性能测试报告（word 格式），描述所做的测试、遇到的问题、负载模式、结果分析等，包括主要工具执行截图等。

实验 5 安全性测试

（共 2 学时）

1. 实验目的

- （1）巩固所学到的安全性测试方法；
- （2）提高使用安全性测试工具的能力。

2. 实验前提

- （1）了解常见的 web 安全性漏洞，如 OWASP Top 10 web 安全性漏洞。
- （2）掌握基本的安全性测试方法。
- （3）熟悉常见的安全性测试工具。

3. 实验内容

针对被测试的 Web 应用系统进行安全性测试。

4. 实验环境

- （1）每 3~5 个学生组成一个测试小组。
- （2）每人或每两人有一台安装了 Java 运行环境的 PC。
- （3）网络连接，能够访问互联网。

5. 实验过程

- (1) 了解 Metasploit、Backtrack5、W3af、ZAP 这 4 个安全性工具 / 框架, 适当做些对比分析, 探讨哪个更适合自己使用。
- (2) 选定 1~2 个安全性测试工具, 如 ZAP、Metasploit, 然后进行人员的工作分工。
- (3) 下载、安装和调试工具, 如 ZAP、Metasploit。
- (4) 可以分别使用 ZAP、Metasploit 针对 www.testfire.net 进行安全性测试, 发现安全性漏洞, 再对每个漏洞进行手工验证、分析。
- (5) 可以分别使用 ZAP、Metasploit 针对 cwe.mitre.org 进行安全性测试, 发现安全性漏洞, 再对每个漏洞进行手工验证、分析。
- (6) 如果可能, 也可以针对自己之前开发的系统进行安全性测试。
- (7) 对两个工具的功能和使用体验, 进行适当的讨论和总结。
- (8) 对所有发现的安全性漏洞, 进行讨论和总结。

6. 实验结果

提交安全性测试报告 (word 格式), 描述测试工具选择的利用、工具使用心得、发现的安全性漏洞、漏洞分析等, 包括主要工具执行截图等。

实验 6 部署自动化测试框架

(共 3 学时)

1. 实验目的

- (1) 巩固所学到的自动化测试框架相关知识;
- (2) 提高自动化测试的动手能力。

2. 实验前提

- (1) 了解自动化测试框架的构成和常见的自动化测试框架;
- (2) 访问 Robot Framework 官方网站: <https://robotframework.org/>;
- (3) 熟悉 Linux 系统或其他适合部署自动化测试框架的环境。

3. 实验内容

- (1) 安装自动化测试框架。
- (2) 设置和集成相应的测试工具。
- (3) 简单使用自动化测试框架

4. 实验环境

- (1) 每 3~5 个学生组成一个测试小组;
- (2) 准备 2-3 台 PC 和一台服务器;
- (3) 网络连接, 能够访问服务器和外部互联网资源。

5. 实验过程

- (1) 小组讨论部署方案和小组成员分工;
- (2) 如果没有 Python, 安装 Python;
- (3) 下载安装 Robot framework, 如 `pip install robotframework`;
- (4) 安装 Robot framework 的 GUI 界面, 如 `wxPython+ robotframework-ride`;
- (5) 安装第三方库(Robot framework 插件), 如 SeleniumLibrary、HTTP RequestsLibrary、AppiumLibrary、RESTinstance 等;
- (6) 启动 Robot framework (RF), 使用 RF 内建库, 完成一些基本的操作 (如时间日期、过程、操作系统等)、RF 自身的关键字驱动脚本。
- (7) 结合实验 3, 实现 RF 的集成, 生成测试报告;
- (8) 基于 RF 和第三方库完成接口测试的脚本等, 生成测试报告。

6. 实验结果

交付完整的实验报告, 包括记录安装和使用完整过程, 遇到的问题以及如何解决问题的, 使用的效果, 包括关键使用的界面截图等。

实验 7 制定测试计划

(共 3 学时)

1. 实验目的

- (1) 巩固所学到的测试需求分析方法;
- (2) 运用所学到的估算方法;
- (3) 提高实际的测试计划能力。

2. 实验前提

- (1) 理解测试目标和测试计划相关知识;
- (2) 良好地理解质量模型;
- (3) 良好地完成了之前的功能测试和专项测试的实验;
- (4) 选择之前功能测试和专项测试的应用系统 (SUT)。

3. 实验内容

- (1) 针对 SUT 进行测试需求分析, 界定测试范围;
- (2) 分析并列出测试项;
- (3) 估算测试工作量;
- (4) 了解测试风险, 并制定测试策略;
- (5) 测试计划的编写。

4. 实验过程

- (1) 小组讨论, 分析 SUT, 重新明确测试目标, 不限于功能要求, 还包括非功能性要

求。

(2) 根据测试目标和代码的改动情况, 分析哪些要测试、哪些可以不错, 即确定测试的范围。

(3) 根据所确定的测试范围, 进行分解, 列出测试项(包括功能测试项和专项测试项)。

(4) 用 WBS 或敏捷中扑克牌估算法等来估算每一个测试项的工作量, 并进行汇总。

(5) 针对这些测试内容, 识别出其中的测试风险, 并分析和列出主要的测试风险。

(6) 一起讨论, 看看能否针对这些测试风险, 找到相应的测试对策(策略)。

(7) 依据 10.5.1 节所学的主要内容, 将前面的内容汇总起来, 形成一个简要的测试计划。

5. 实验结果

提交一份简要的测试计划。

实验 8 Jenkins+Docker 实现 Java 应用的持续构建

(共 3 学时)

1. 实验目的

- (1) 巩固所学到的部署测试基础设施的方法;
- (2) 提高实际的测试环境搭建能力;
- (3) 理解基础设施即代码的概念及方法。

2. 实验前提

- (1) 理解 5.8 节软件应用构建过程;
- (2) 理解 12.3 节和 12.4 节所描述的测试环境部署方法;
- (3) 学习参考文档: 使用 Maven 在 Jenkins 流水线中构建 Java 程序 (<https://www.jenkins.io/zh/doc/tutorials/build-a-java-app-with-maven/>), 理解如何使用 Maven、Jenkins、Docker 持续构建 Java 应用程序。

3. 实验内容

采用 Jenkins Pipeline 和 Docker 搭建一个 Java 应用的 Jenkins 流水线, 完成一个 Java 应用从编译、静态代码分析到单元测试、打包的过程。

4. 实验环境

- (1) 每 3~5 个学生组成一个实验小组;
- (2) SUT 选择 Github 或 Gitee 上一个开源的自带单元测试用例的 Springboot 项目。
- (3) 每个人有一台 PC 并连接到互联网。

5. 实验过程

- (1) 小组讨论, 列出需要下载并安装的软件工具及安装方式。
- (2) 以小组为单位, 在 Github 上选择一个带有单元测试用例的开源 Springboot 项目, 将该项目复制到本地, 确认可以在本地顺利通过构建(包括单元测试)。
- (3) 每个学生向本组其它学生讲解自己对软件应用构建过程的理解。

- (4) 完成下载并安装需要的软件工具
- (5) 将 Jenkins 作为 Docker 容器并从 jenkinsci/blueocean Docker 镜像中运行。
- (6) 在 Jenkins 中创建一个流水线，“构建触发器”中勾选“定时构建”，定义每 15min 构建一次。
- (7) 创建一个 Jenkinsfile，该文件完成的任务包括在 Docker 中运行 Maven 容器，构建 Java 应用、运行单元测试。将 Jenkinsfile 文件提交到本地的 Java 项目 Git 仓库。
- (8) 在 Jenkins 的 Blue Ocean 界面查看构建报告，检查是否每 15min 自动构建，并查看每个步骤的详细日志。
- (9) 在 Jenkins 中修改流水线配置，“构建触发器”中勾选“轮询 SCM”，定义每 15min 轮询一次是否提交了代码。
- (10) 修改项目中的代码，使用 git 命令向本地代码仓库提交代码更改。
- (11) 在 Jenkins 的 Blue Ocean 界面查看构建报告，检查是否在提交了代码变更后触发了一次构建，并查看每个步骤的详细日志。

6. 实验结果

- (1) 提交 Java 应用的自动化构建环境。
- (2) 提交 Jenkins Blue Ocean 界面中流水线运行成功的界面截图和日志文件。

实验 9 安装和使用缺陷跟踪系统 MantisBT

(共 3 学时)

1. 实验目的

- (1) 巩固所学到的有关缺陷的知识；
- (2) 提高缺陷报告和跟踪的能力。

2. 实验前提

- (1) 理解 13.2 节和 13.3 节所描述的内容；
- (2) 了解 MantisBT 及其运行的环境。

3. 实验内容

- (1) 安装缺陷跟踪系统 MantisBT；
- (2) 使用 MantisBT 报告、查询、跟踪和统计缺陷。

4. 实验环境

- (1) 每 3~5 个学生组成一个测试小组；
- (2) 最好有一两台 PC 和一台服务器，如果没有条件，可以将 PC 做服务器；
- (3) 网络连接，能够访问互联网资源。

5. 实验过程

- (1) 安装 PHP 环境，如 XAMPP、phpstudy 安装包。
- (2) 安装和设置数据库，如 MySQL。
- (3) 启动 PHP 和数据库环境，并创建 MySQL 用户和数据库；

- (4) 下载 MantisBT 压缩包，将 MantisBT 压缩包解压在 WWW 文件夹下；
- (5) 进入 MantisBT，完成安装、初步配置，包括创建所需的用户；
- (6) 配置文件（含图片）上传、预览、邮箱等功能，开始使用 MantisBT；
- (7) 创建项目，在项目中增加功能模块、功能需求项等；
- (8) 为某些需求项增加测试用例；
- (9) 报告若干缺陷，并关联所添加的测试用例；
- (10) 进行缺陷查询、统计等；
- (11) 根据前面的过程，编写安装和使用报告。

6. 交付成果

提交安装和使用报告，包括安装步骤，碰到的问题以及如何解决的，关键界面截图等。

实验 10 基于 MeterSphere 的综合实验

（共 3 学时）

1. 实验目的

- (1) 巩固所学到的接口测试、性能测试、测试管理和自动化测试等知识；
- (2) 提升接口测试和性能测试的实际动手能力；
- (3) 掌握测试项目和用例管理、测试计划、缺陷管理的方法与实践。

2. 实验前提

- (1) 学完第 5~7 章和第 9~13 章的内容，即基本掌握了接口测试、性能测试、测试用例设计和管理、缺陷跟踪管理、测试计划和测试报告等内容；
- (2) 了解 Linux 操作、Docker 基本命令、MeterSphere 及其运行的环境。

3. 实验内容

- (1) 在线或离线安装 MeterSphere；
- (2) 创建项目、建立项目成员和基础测试配置环境；
- (3) 完成一个完整的接口测试，包括接口调试与定义、接口测试用例的设计、基于真实场景的自动化测试等；
- (4) 完成一个完整的性能测试，包括场景配置、压力配置、高级配置、结果分析等；
- (5) 完成一个完整的功能测试流程，包括设计用例、用例评审、制订一个测试计划、缺陷管理和生成报告。

4. 实验环境

- (1) 每 3~5 个学生组成一个测试小组；
- (2) 一台安装了 CentOS/RHEL 7.x 服务器或 PC，其配置为 4 核 8GB 100GB 磁盘；
- (3) 网络连接，能够访问互联网资源。

5. 实验过程

1) 安装 MeterSphere 及其实验环境

- (1) 在线或离线部署 MeterSphere 平台;
- (2) 安装一个被测的应用系统, 并提供了开放的接口访问, 如任务调度系统 <https://m3.fit2cloud.com>、支持 Swagger API 的宠物商店 <https://petstore.swagger.io/>、电商系统 API <https://gz.fit2cloud.com/swagger-ui.html#/>、JumpServer API <https://jms-qa.fit2cloud.com>。

2) 接口测试实验

- (1) 阅读接口文档, 了解各个接口及其参数的含义;
- (2) 快速调试接口, 输入基础信息(如请求方式、URL)、请求参数(如 QUERY 参数、REST 参数), 调试成功存为接口测试用例;
- (3) 再进行更多的接口调试, 如覆盖不同的请求方式、改变请求参数的值, 生成更多的接口测试用例;
- (4) 设置一个端到端的业务场景, 复制或者引用之前的测试用例, 并进行正确的排序和组合, 形成一个有效、合理的接口自动化过程, 并调试、执行生成自动化场景报告;
- (5) 查看测试报告, 并进行分析;
- (6) 编写接口测试报告。

3) 性能测试实验

- (1) 确定被测试对象, 也可以用上述被测的接口测试系统;
- (2) 创建一个性能测试, 其场景可以引用上面接口测试场景, 也可以通过接口场景或者接口测试用例直接转化为性能测试, 还可以导入本地电脑上的存量 JMeter 的 JMX 文件;
- (3) 进行压力配置, 包括选择本地资源池, 设置并发用户数、压测时长、Ramp-Up 时间和分配策略, 选择执行方式以及是否开启 RPS 等;
- (4) 进行高级配置, 如 CSVDataSet、自定义变量;
- (5) 保存、执行并查看性能测试结果, 进行结果分析;
- (6) 重复步骤(2)~(5), 如调整并发用户数、压测时长、Ramp-Up 时间和分配策略等, 优化测试效果, 可以通过多次报告比对查看性能测试数据;
- (7) 进行性能指标分析, 看看能否发现一些性能瓶颈问题。
- (8) 编写性能测试报告。

4) 功能测试流程

- (1) 进入测试跟踪;
- (2) 进入功能用例, 创建左边模块目录结构;
- (3) 先以脑图方式创建测试用例;
- (4) 再以列表方式补充相关信息, 加入到相应的模块;
- (5) 创建相应的测试评审, 可以对测试用例的内容进行评论或者上传附件等;
- (6) 创建并执行测试计划, 将相应的功能用例、接口用例、接口场景、性能测试等加入相应的测试用例;
- (7) 缺陷管理, 包括创建缺陷、修改缺陷(改变缺陷状态)等;

- (8) 运行测试计划，查看测试报告；
- (9) 编写功能测试报告。

5) 整理实验报告

将前面 3 份测试报告和平台 MeterSphere 安装等合并，整理出一份完整的实验报告，包括 MeterSphere 安装步骤、碰到的问题以及如何解决的，API 测试和性能测试的过程（含关键界面截图）、结果分析及其思考、总结等。

6. 实验结果

提交实验报告、测试脚本和其他资源文件。

【实验 10】基于 MeterSphere 的综合实验报告



扫码看视频

1. 安装 MeterSphere 及其实验环境

1.1. 部署服务器要求

- 操作系统：CentOS 7.x / Redhat 7.x
- CPU/内存：最低要求 4C8G，推荐 8C16G
- 磁盘空间：50G

检查操作系统：cat /etc/redhat-release

```
[root@node-123 ~]# cat /etc/redhat-release
CentOS Linux release 7.7.1908 (Core)
[root@node-123 ~]#
```

检查内存和 CPU、磁盘信息

```
[root@node-123 ~]# free -g
              total        used        free      shared  buff/cache   available
Mem:           7          3          0           0           3           4
Swap:          1           0           1           0           0           0

[root@node-123 ~]# cat /proc/cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 79
model name     : Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz
stepping       : 1
microcode      : 0xb000021
cpu MHz        : 1698.033
cache size     : 15360 KB
physical id    : 0
siblings       : 1
core id        : 0
cpu cores      : 1
apicid         : 0
initial apicid : 0
fpu            : yes
fpu_exception  : yes
cpuid level    : 13
wp             : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
erfpu pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_de
dtherm arat pln pts
bogomips       : 3396.06
```

检查存储空间：df -h

```
[root@node-123 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        3.8G   0 3.8G   0% /dev
tmpfs           3.9G   0 3.9G   0% /dev/shm
tmpfs           3.9G  13M 3.8G   1% /run
tmpfs           3.9G   0 3.9G   0% /sys/fs/cgroup
/dev/mapper/centos-root 97G  15G  83G  16% /
/dev/sdal       1014M  149M  866M  15% /boot
tmpfs           781M   0 781M   0% /run/user/0
```

1.2. 下载安装包

打开并注册网址：

<https://community.fit2cloud.com/#/products/metersphere/downloads>

获取 MeterSphere 最新的安装包地址, 下载完成后上传至服务器的/opt 目录

```
[root@node-123 ~]# cd /opt/
[root@node-123 opt]# ls
metersphere metersphere-offline-installer-v2.2.0 metersphere-offline-installer-v2.2.0.tar.gz
[root@node-123 opt]# ll
total 2850636
drwxr-xr-x. 6 root root      4096 Oct 25 16:39 metersphere
drwxr-xr-x. 6 root root      235 Oct 25 16:39 metersphere-offline-installer-v2.2.0
-rw-r--r--. 1 root root 2919043490 Oct 25 15:05 metersphere-offline-installer-v2.2.0.tar.gz
[root@node-123 opt]#
```

社区版下载

v2.2.0 (Latest)

其他版本 ▾

v2.2.0



2022-09-21 23:40:56

新增功能

UI测试支持自定义组合指令(X-Pack增强包内)
接口测试支持自定义模板(X-Pack增强包内)
文件管理支持对接第三方仓库(X-Pack增强包内)

功能优化

测试跟踪: 测试计划/用例评审页用例标题展示优化;
接口测试: 接口CASE高级搜索增加路径搜索;
接口测试: TCP协议支持一键将xml文本格式转换成表格格式;
接口测试: 场景变量支持查询以及导入、导出;
UI测试(X-Pack): 列表批量执行与测试计划执行时增加失败重试机制;
UI测试(X-Pack): 测试报告支持一键分享;
UI测试(X-Pack): UI场景支持查看被引用列表;
UI测试(X-Pack): 支持在配置文件中配置浏览器的默认分辨率;
系统设置(X-Pack): 消息通知模板新增测试计划报告相关字段;
通用功能(X-Pack): 版本筛选支持手动录入版本号。

1.3. 解压安装包

以 root 用户 ssh 登录到目标机器，并执行如下命令。

```
cd /opt
```

```
# 解压安装包
```

```
tar zxvf metersphere-release-v1.0.3-offline.tar.gz
```

```
[root@node-123 opt]# tar xzvf metersphere-offline-installer-v2.2.0.tar.gz
metersphere-offline-installer-v2.2.0/
metersphere-offline-installer-v2.2.0/quick_start.sh
metersphere-offline-installer-v2.2.0/.github/
metersphere-offline-installer-v2.2.0/.github/workflows/
metersphere-offline-installer-v2.2.0/.github/workflows/auto-pr.yml
metersphere-offline-installer-v2.2.0/msctl
metersphere-offline-installer-v2.2.0/install.sh
metersphere-offline-installer-v2.2.0/metersphere/
metersphere-offline-installer-v2.2.0/metersphere/docker-compose-kafka.yml
metersphere-offline-installer-v2.2.0/metersphere/docker-compose-redis.yml
metersphere-offline-installer-v2.2.0/metersphere/docker-compose-prometheus.yml
metersphere-offline-installer-v2.2.0/metersphere/docker-compose-node-controller.yml
metersphere-offline-installer-v2.2.0/metersphere/docker-compose-mysql.yml
metersphere-offline-installer-v2.2.0/metersphere/conf/
metersphere-offline-installer-v2.2.0/metersphere/conf/my.cnf
metersphere-offline-installer-v2.2.0/metersphere/conf/version
metersphere-offline-installer-v2.2.0/metersphere/conf/redis.conf
metersphere-offline-installer-v2.2.0/metersphere/conf/prometheus/
metersphere-offline-installer-v2.2.0/metersphere/conf/prometheus/prometheus.yml
metersphere-offline-installer-v2.2.0/metersphere/conf/metersphere.properties
metersphere-offline-installer-v2.2.0/metersphere/version
metersphere-offline-installer-v2.2.0/metersphere/docker-compose-seleniarm.yml
metersphere-offline-installer-v2.2.0/metersphere/docker-compose-base.yml
metersphere-offline-installer-v2.2.0/metersphere/docker-compose-server.yml
metersphere-offline-installer-v2.2.0/metersphere/logs/
metersphere-offline-installer-v2.2.0/metersphere/logs/README.md
metersphere-offline-installer-v2.2.0/install.conf
metersphere-offline-installer-v2.2.0/Jenkinsfile
metersphere-offline-installer-v2.2.0/README.md
metersphere-offline-installer-v2.2.0/docker/
metersphere-offline-installer-v2.2.0/docker/bin/
metersphere-offline-installer-v2.2.0/docker/bin/docker-runc
metersphere-offline-installer-v2.2.0/docker/bin/docker-containerd
metersphere-offline-installer-v2.2.0/docker/bin/docker-containerd-shim
metersphere-offline-installer-v2.2.0/docker/bin/dockerd
```

1.4. 执行安装脚本

进入安装包目录

```
cd metersphere-offline-installer-v2.2.0/
```

运行安装脚本

```
/bin/bash install.sh
```

```
[root@node-123 metersphere-offline-installer-v2.2.0]# bash install.sh
存在已安装的 MeterSphere，安装目录为 /opt/metersphere，执行升级流程
[MeterSphere Log]: 拷贝安装文件到目标目录
'/opt/metersphere/docker-compose-kafka.yml' -> '/opt/metersphere/docker-compose-kafka.yml' (backup: '/opt/metersphere/docker-compose-kafka.yml.20221025-1639')
'/opt/metersphere/docker-compose-redis.yml' -> '/opt/metersphere/docker-compose-redis.yml' (backup: '/opt/metersphere/docker-compose-redis.yml.20221025-1639')
'/opt/metersphere/docker-compose-prometheus.yml' -> '/opt/metersphere/docker-compose-prometheus.yml' (backup: '/opt/metersphere/docker-compose-prometheus.yml.20221025-1639')
'/opt/metersphere/docker-compose-node-controller.yml' -> '/opt/metersphere/docker-compose-node-controller.yml' (backup: '/opt/metersphere/docker-compose-node-controller.yml.20221025-1639')
'/opt/metersphere/docker-compose-mysql.yml' -> '/opt/metersphere/docker-compose-mysql.yml' (backup: '/opt/metersphere/docker-compose-mysql.yml.20221025-1639')
'/opt/metersphere/conf/my.cnf' -> '/opt/metersphere/conf/my.cnf' (backup: '/opt/metersphere/conf/my.cnf.20221025-1639')
'/opt/metersphere/conf/version' -> '/opt/metersphere/conf/version' (backup: '/opt/metersphere/conf/version.20221025-1639')
'/opt/metersphere/conf/redis.conf' -> '/opt/metersphere/conf/redis.conf' (backup: '/opt/metersphere/conf/redis.conf.20221025-1639')
'/opt/metersphere/conf/prometheus/prometheus.yml' -> '/opt/metersphere/conf/prometheus/prometheus.yml' (backup: '/opt/metersphere/conf/prometheus/prometheus.yml.20221025-1639')
'/opt/metersphere/conf/metersphere.properties' -> '/opt/metersphere/conf/metersphere.properties' (backup: '/opt/metersphere/conf/metersphere.properties.20221025-1639')
'/opt/metersphere/version' -> '/opt/metersphere/version' (backup: '/opt/metersphere/version.20221025-1639')
'/opt/metersphere/docker-compose-seleniam.yml' -> '/opt/metersphere/docker-compose-seleniam.yml' (backup: '/opt/metersphere/docker-compose-seleniam.yml.20221025-1639')
'/opt/metersphere/docker-compose-base.yml' -> '/opt/metersphere/docker-compose-base.yml' (backup: '/opt/metersphere/docker-compose-base.yml.20221025-1639')
'/opt/metersphere/docker-compose-server.yml' -> '/opt/metersphere/docker-compose-server.yml' (backup: '/opt/metersphere/docker-compose-server.yml.20221025-1639')
'/opt/metersphere/logs/README.md' -> '/opt/metersphere/logs/README.md' (backup: '/opt/metersphere/logs/README.md.20221025-1639')
[MeterSphere Log]: ===== 开始安装 =====
[MeterSphere Log]: 检测到 Docker 已安装，跳过安装步骤
[MeterSphere Log]: 启动 Docker
Redirecting to /bin/systemctl start docker.service
[MeterSphere Log]: 检测到 Docker Compose 已安装，跳过安装步骤
[MeterSphere Log]: 加载镜像
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/jmeter-master:5.5-ms2-jdk11
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/kafka:3.2.0
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/metersphere:v2.2.0
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/ms-datastreaming:v2.2.0
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/ms-node-controller:v2.2.0
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/mysql:8.0.30
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/node-exporter:latest
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/prometheus:latest
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/redis:6.2.0
Loaded image: registry.cn-qingdao.aliyuncs.com/metersphere/seleniam-grid-all:4.1.4-20220519
[MeterSphere Log]: 启动服务
Stopping ms-server ... done
Stopping ms-data-streaming ... done
Stopping nodeexporter ... done
Stopping ms-node-controller ... done
Stopping mysql ... done
Stopping kafka ... done
Stopping ms-prometheus ... done
Stopping redis ... done
Removing ms-server ... done
Removing ms-data-streaming ... done
Removing nodeexporter ... done
Removing ms-node-controller ... done
Removing mysql ... done
```

等待安装脚本执行完成后，查看 MeterSphere 状态

msctl status

```
[root@node-123 metersphere-offline-installer-v2.2.0]# msctl status
```

Name	Command	State	Ports
kafka	/opt/bitnami/scripts/kafka/...	Up (healthy)	0.0.0.0:9092->9092/tcp
ms-data-streaming	/deployments/run-java.sh	Up (healthy)	
ms-node-controller	/deployments/run-java.sh	Up (healthy)	0.0.0.0:8082->8082/tcp, 0.0.0.0:9190->9190/tcp
ms-prometheus	/bin/prometheus --config.f...	Up (healthy)	0.0.0.0:9090->9090/tcp
ms-server	/deployments/run-java.sh	Up (healthy)	0.0.0.0:10000->10000/tcp, 0.0.0.0:10001->10001/tcp, 0.0.0.0:10002->10002/tcp, 0.0.0.0:10003->10003/tcp, 0.0.0.0:10004->10004/tcp, 0.0.0.0:10005->10005/tcp, 0.0.0.0:10006->10006/tcp, 0.0.0.0:10007->10007/tcp, 0.0.0.0:10008->10008/tcp, 0.0.0.0:10009->10009/tcp, 0.0.0.0:10010->10010/tcp, 0.0.0.0:8081->8081/tcp
mysql	docker-entrypoint.sh mysqld	Up (healthy)	0.0.0.0:3307->3306/tcp, 33060/tcp
nodeexporter	/bin/node_exporter --path. ...	Up (healthy)	
redis	docker-entrypoint.sh redis ...	Up (healthy)	0.0.0.0:6379->6379/tcp

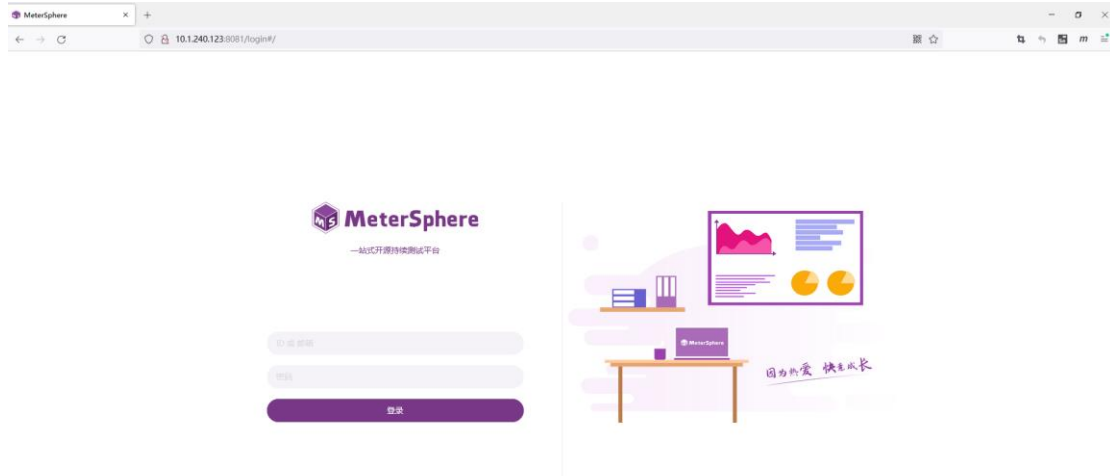
1.5. 登陆平台

安装成功后，通过浏览器访问如下页面登录 MeterSphere。

地址：http://目标服务器 IP 地址:8081

用户名：admin

密码：metersphere



1.6. 实验中遇到的问题与解决

1. 如查看日志 `docker -f -tail 20 ms-server` 遇到无法连接 Redis 服务，报错如下：

Caused by: org.redisson.client.RedisConnectionException: Unable to connect to Redis server: 10.1.240.123/10.1.240.123:6379

解决：尝试关闭服务器防火墙，重启 docker 和 MeterSphere 服务

```
systemctl stop firewalld
```

```
systemctl restart firewalld
```

```
msctl restart
```

2. 如遇到 `msctl start` 启动报错：No containers to start

```
[root@node-123 metersphere-offline-installer-v2.2.0]# msctl start
Starting ms-node-controller ... done
Starting mysql                ... done
Starting redis                 ... done
Starting kafka                 ... done
Starting ms-data-streaming    ... done
Starting ms-server             ... done
Starting prometheus           ... done
Starting nodeexporter         ... done
ERROR: No containers to start
```

解决：采用 `msctl reload` 命令，重新加载服务

3. 如采用一键安装脚本方式部署 MeterSphere 提示一直连接不了 github 网络，如下：

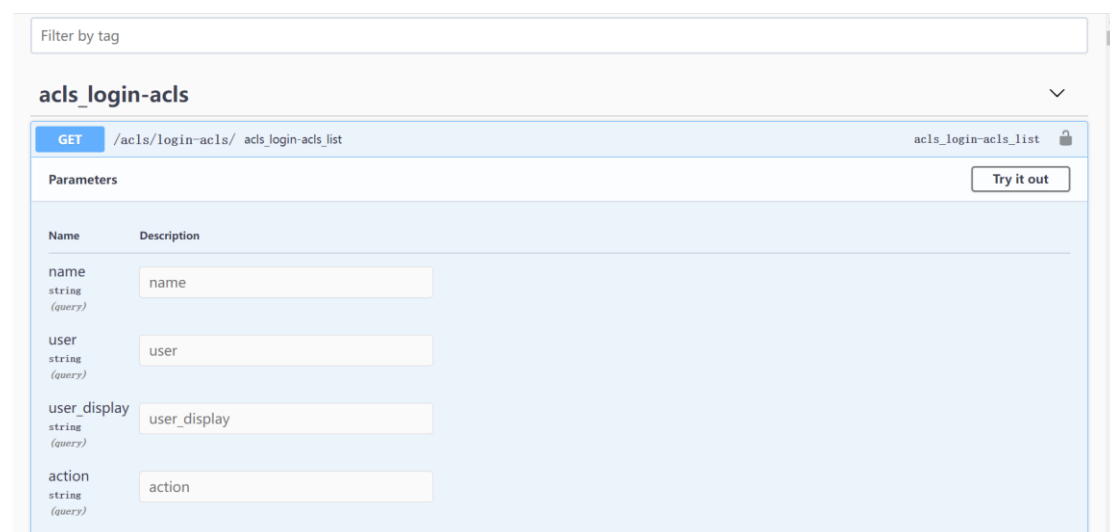
```
[root@node-122 ~]# curl -sSL https://github.com/metersphere/metersphere/releases/latest/download/quick_start.sh | bash
检测 github.com ... 1 ok
检测 github.com ... 2 failed
检测 hub.fastgit.org ... 1 failed
检测 ghproxy.com/https://github.com ... 1 ok
检测 ghproxy.com/https://github.com ... 2 ok
检测 ghproxy.com/https://github.com ... 3 ok
使用下载服务器 ghproxy.com/https://github.com
```

解决：采用下载离线安装包方式进行安装

2. MeterSphere 接口测试实验

2.1. 阅读理解接口文档

先阅读接口文档，了解各个接口及其参数的含义；打开 swagger 文档，了解各接口含义和参数含义

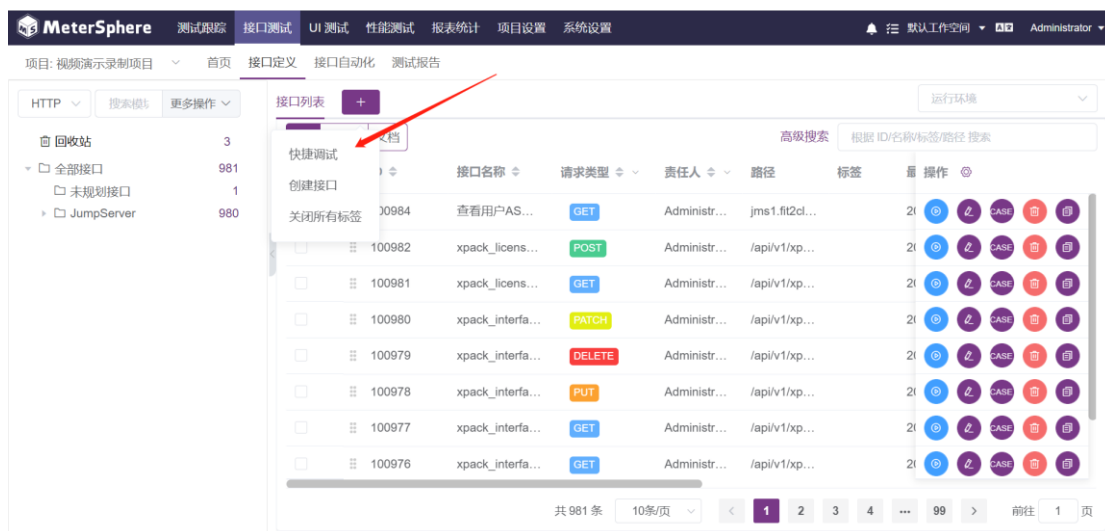


需要注意的是，针对特定接口的参数，必填参数在调用接口的时候需补充完成，其余补充参数根据实际场景进行填写。

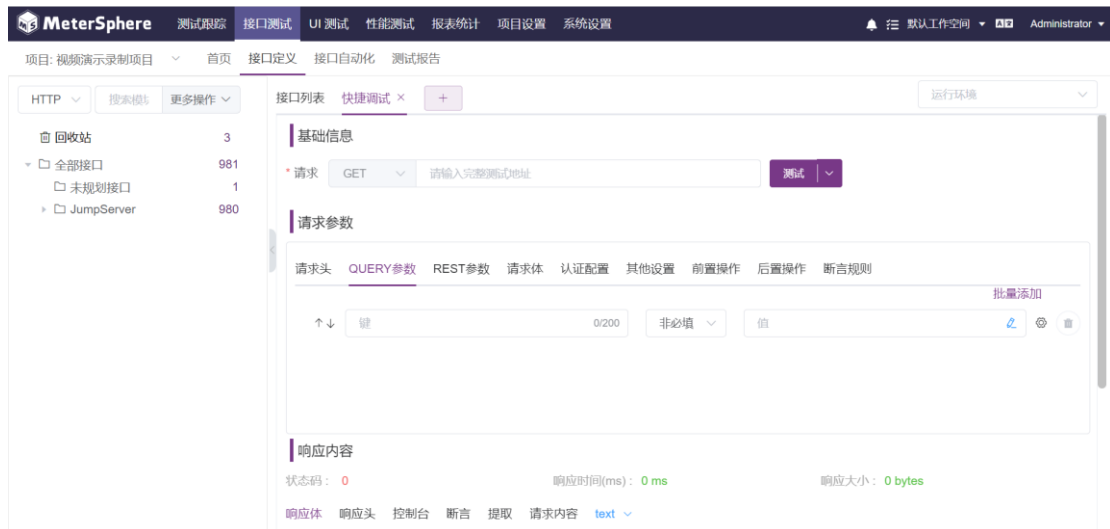


2.2. 接口的快速调试

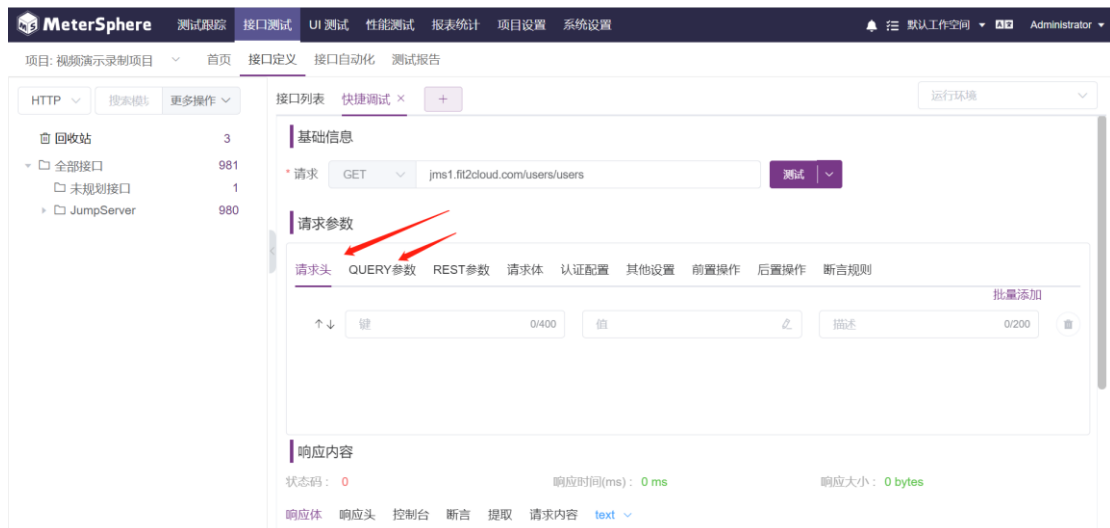
快速调试接口：输入基础信息（如请求方式、URL）、请求参数（如 QUERY 参数、REST 参数），调试成功存为接口测试用例；进入到 metersphere 平台，点击快捷调试



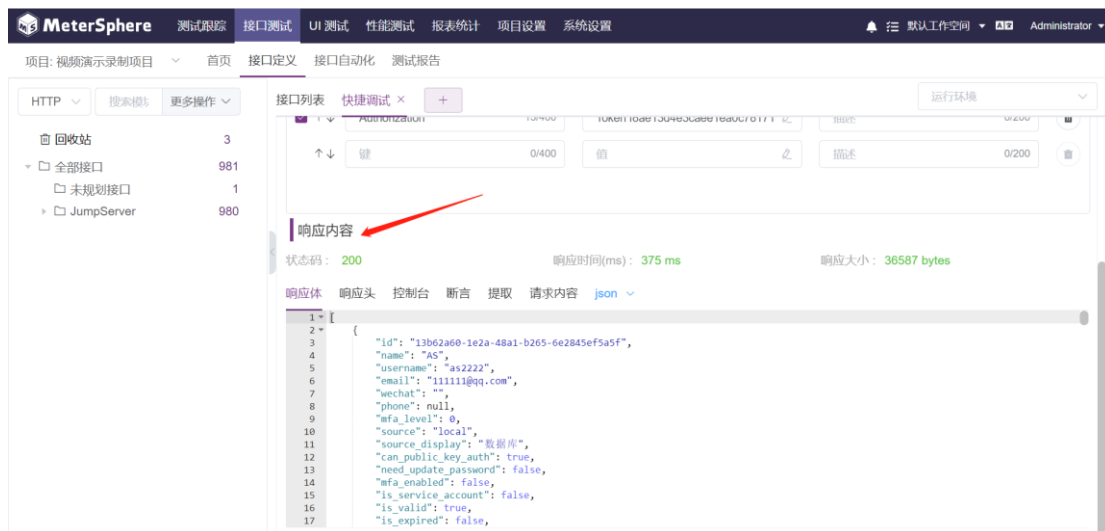
填写对应的接口信息，进行测试



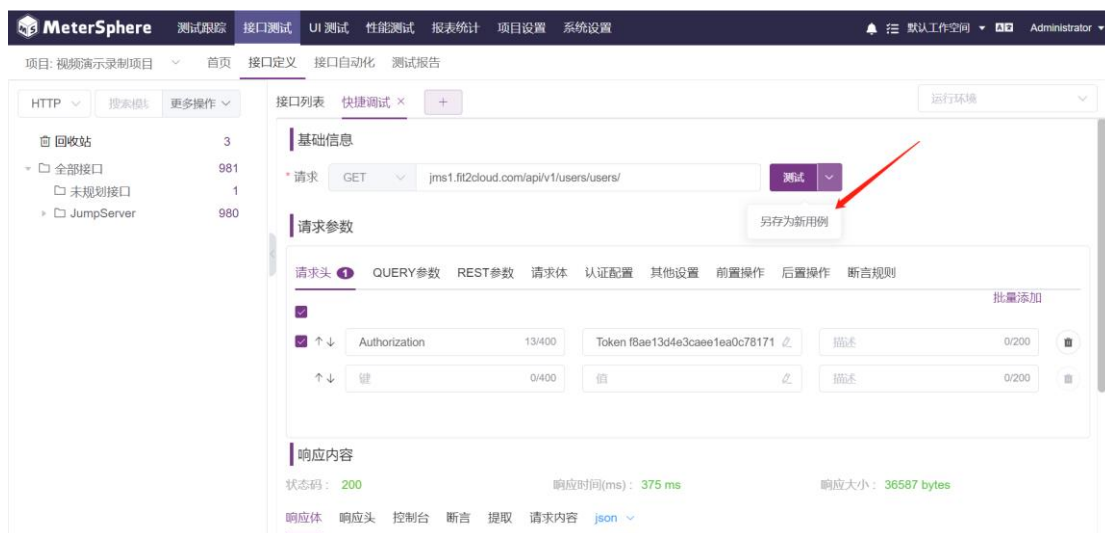
快捷调试需要输入完整的测试地址，根据接口文档，将请求头，对应需要提供的参数补充完成。



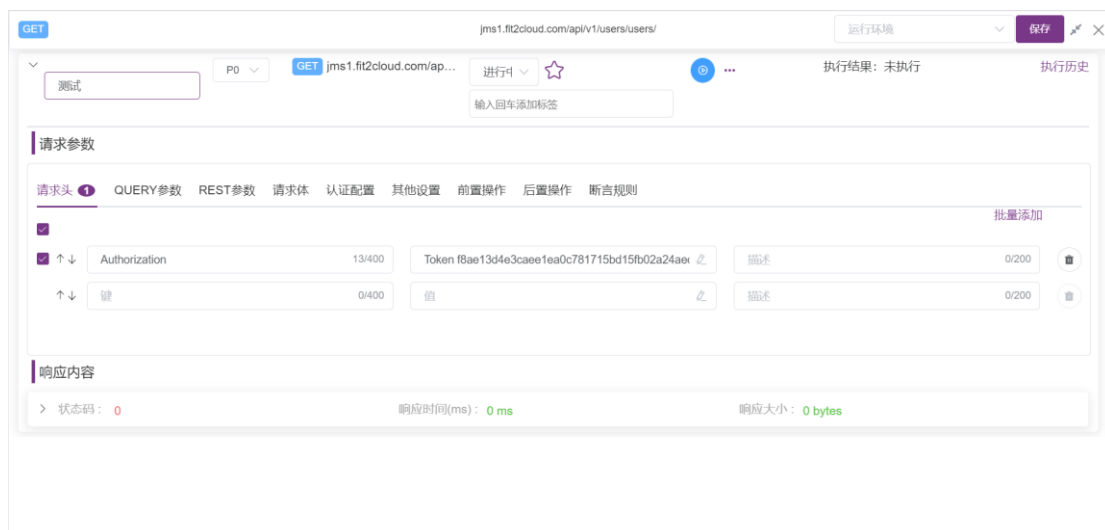
测试完后后，检查对应的响应内容是否与预期一致。



如与预期一致，可将其存为新用例。

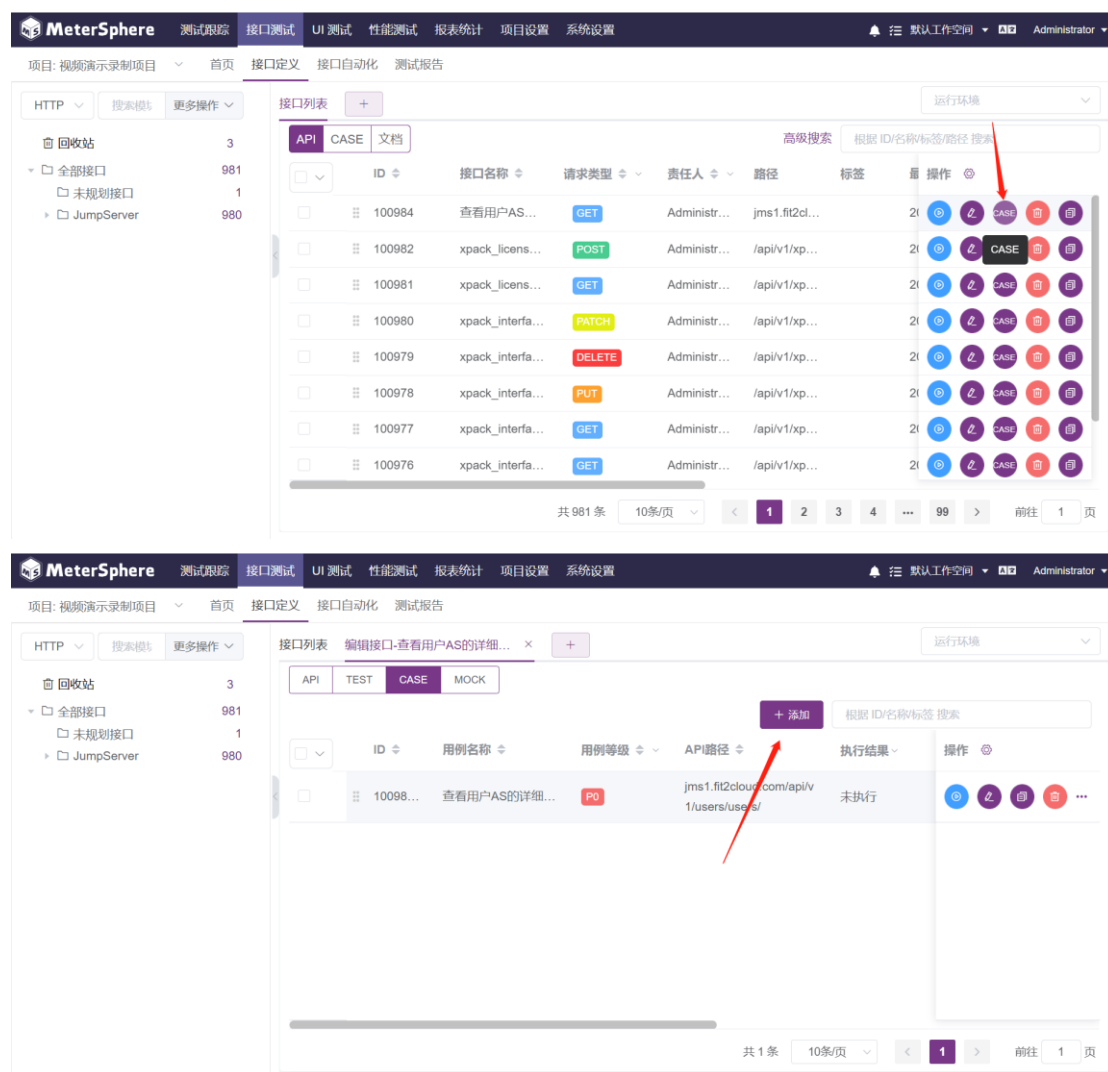


将对应信息填写完成，保存即。



2.3. 接口用例创建

再进行更多的接口调试，如覆盖不同的请求方式、改变请求参数的值，生成更多的接口测试用例；根据上述中调试方式，添加更多的用例信息，进行保存，尽量覆盖更多场景。一个 api 可以引伸出多个 case，供后续不同的场景引用。

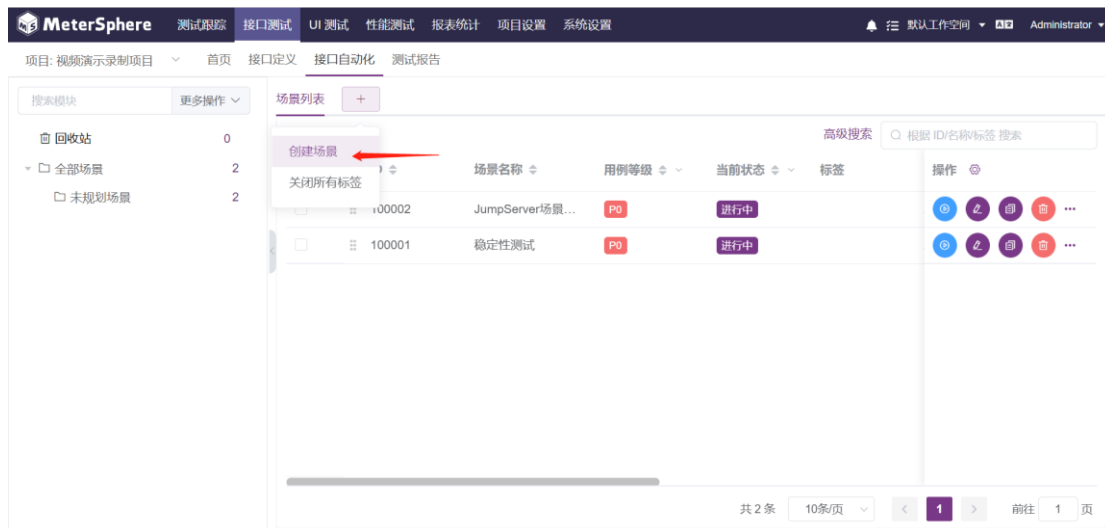


将对应信息填写好，保存即可

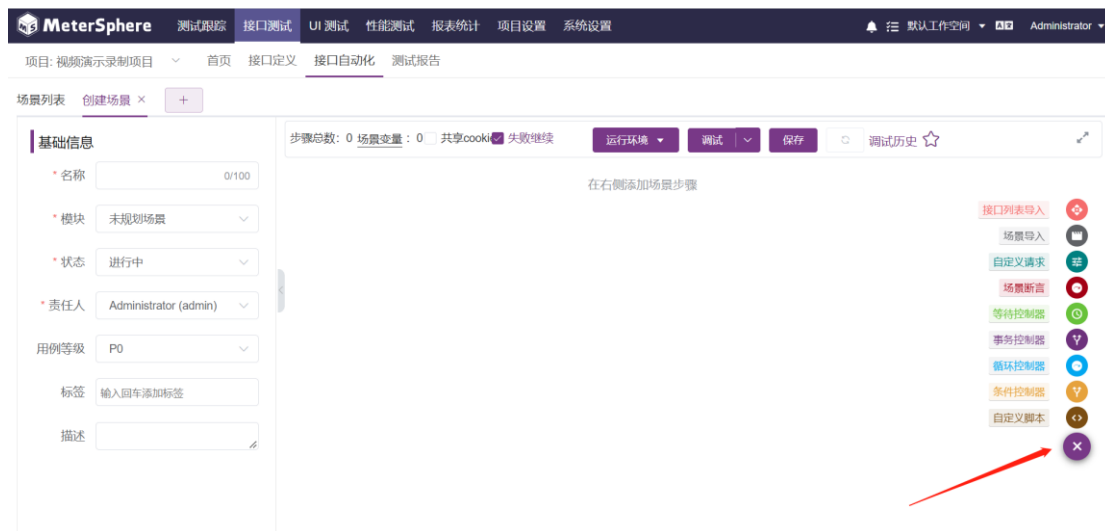


2.4. 接口自动化场景创建

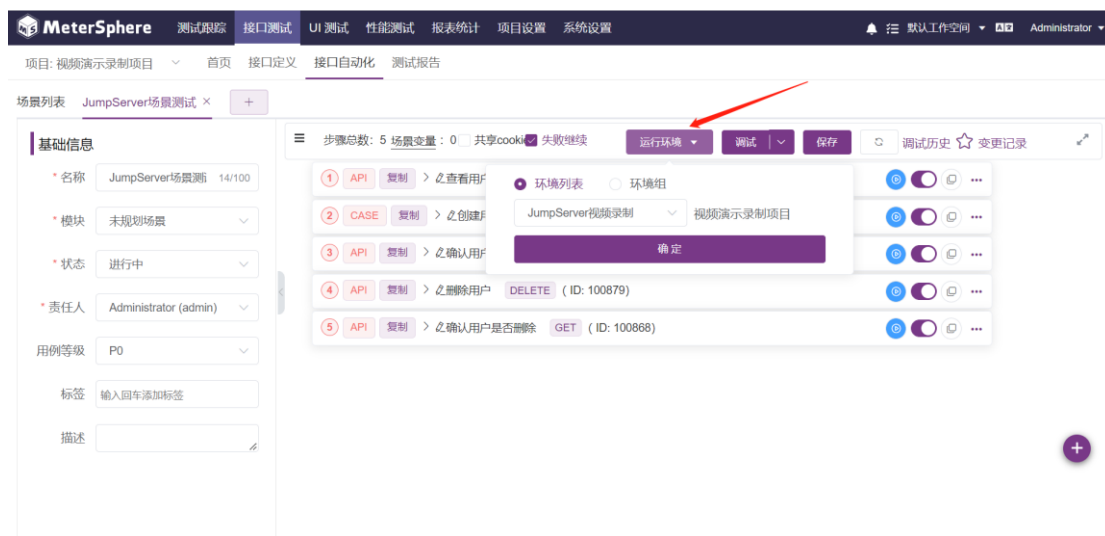
设置一个端到端的业务场景，复制或者引用之前的测试用例，并进行正确的排序和组合，形成一个有效、合理的接口自动化过程，并调试、执行生成自动化场景报告。添加一个接口自动化场景



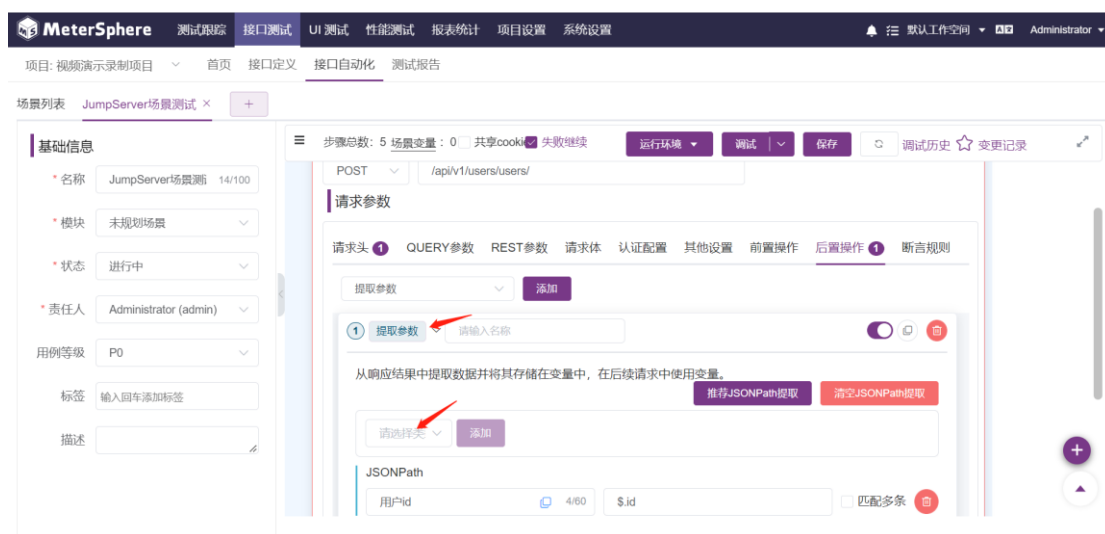
根据设定的场景，进行步骤的添加



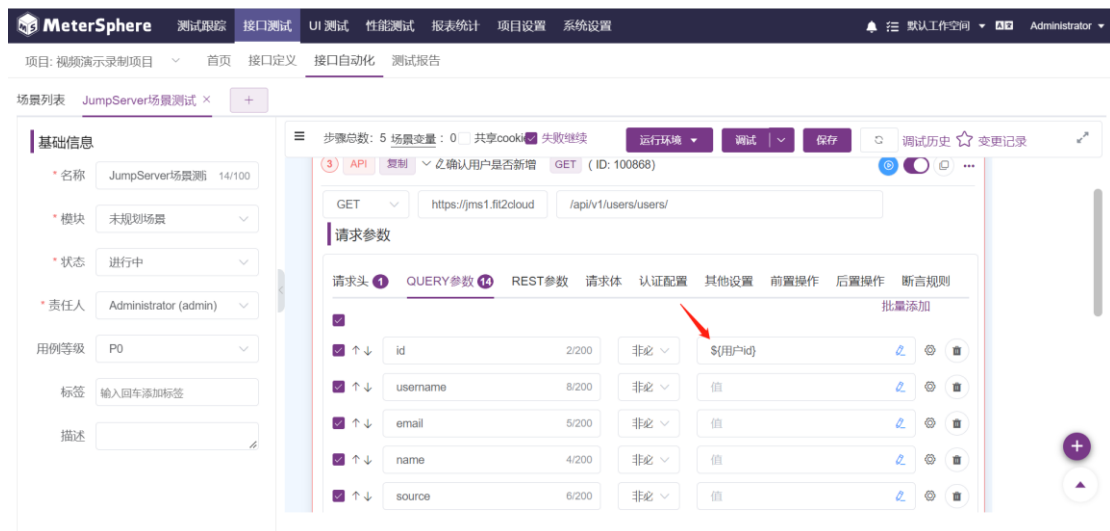
添加好对应的步骤后，选择对应的运行环境，并调试



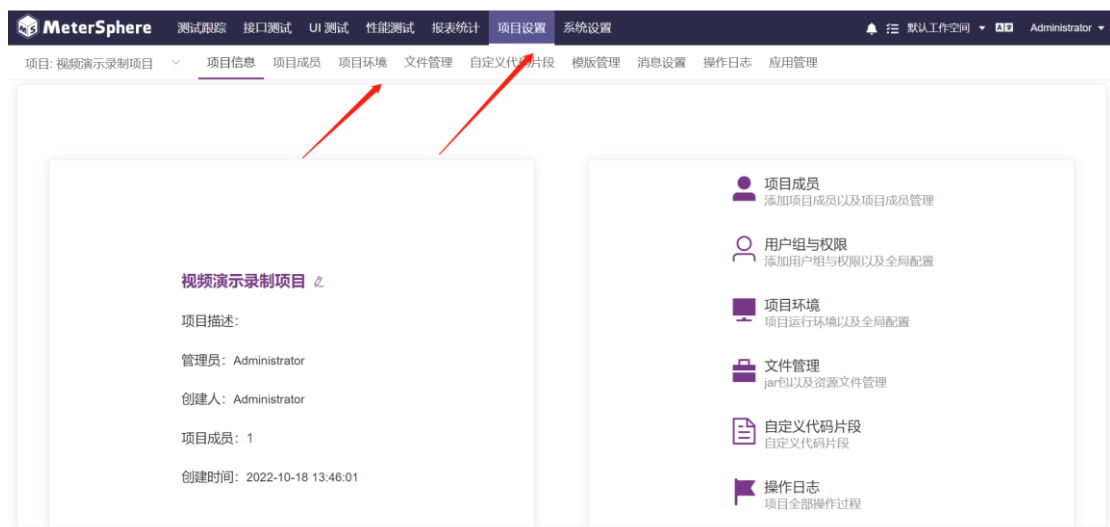
如果涉及到提取参数，可在对应接口下，添加参数提取



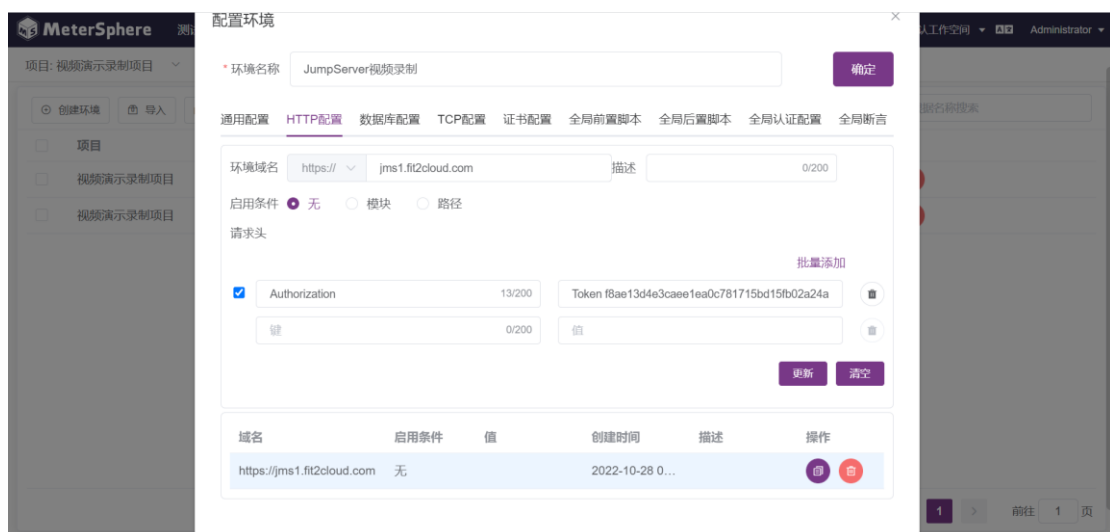
提取好之后，后续的接口可以直接进行引用



运行环境可在项目环境中添加

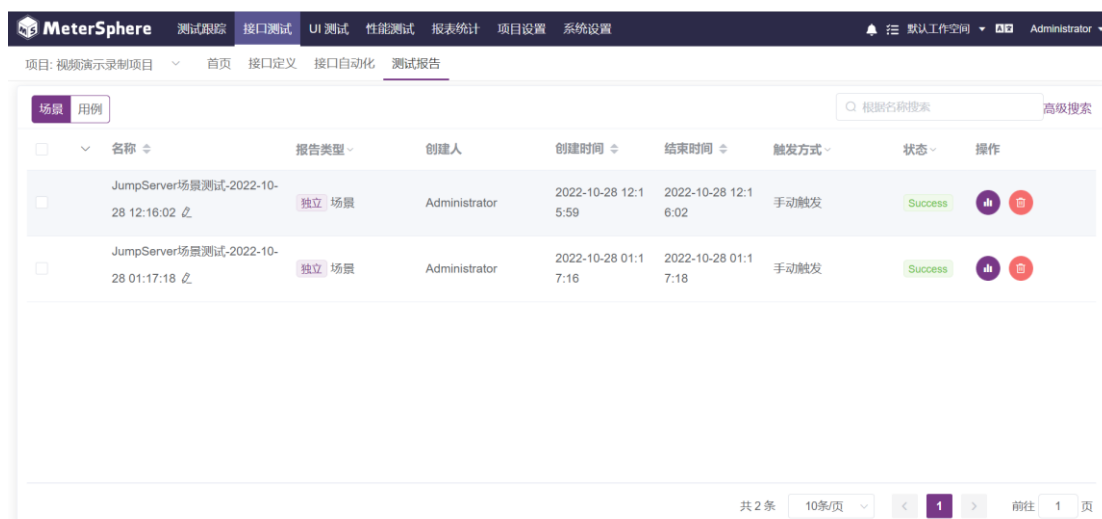


将对应信息填写好后，保存即可

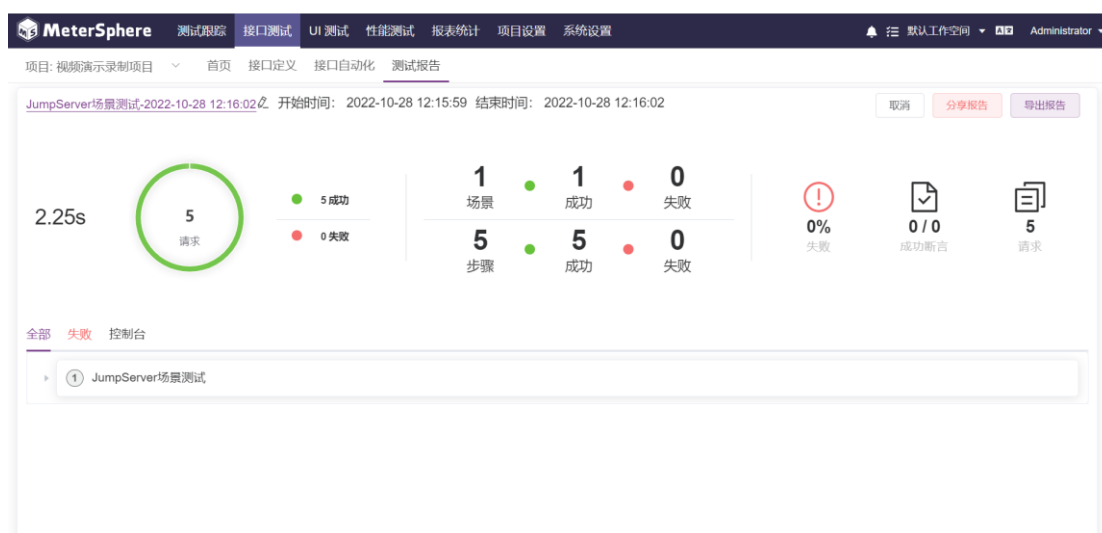


2.5. 查看测试报告，并进行分析

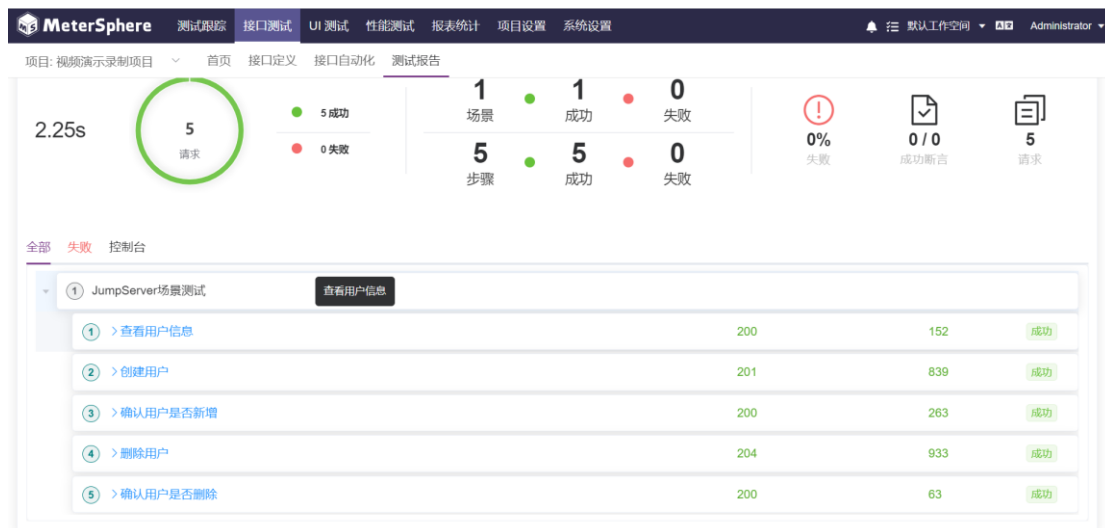
测试报告中，可看到所有的报告



点击对应的报告，即可进行查看



展开后，可进行分析，每一步都有返回信息



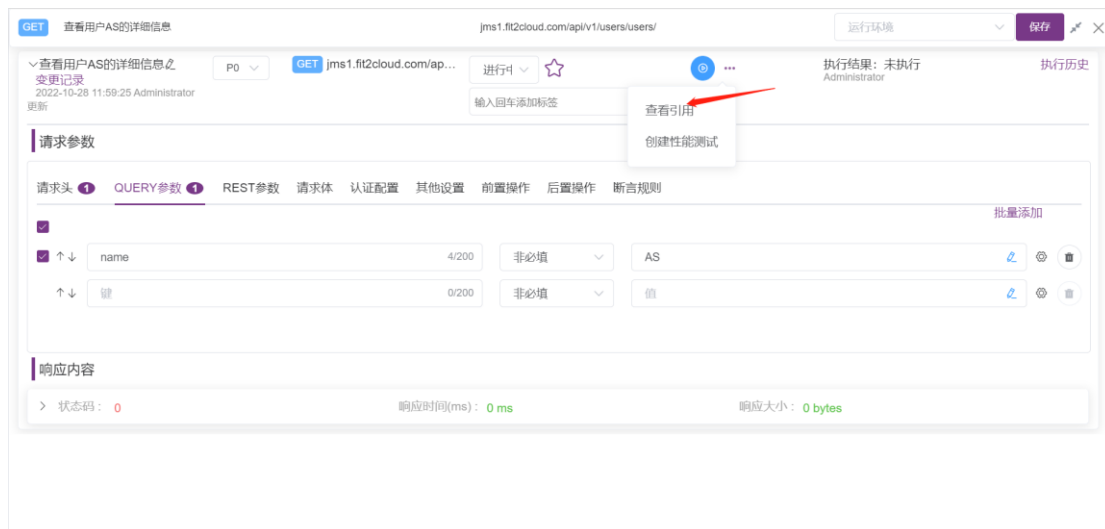
2.6. 实验中遇到的问题与解决

1. 快捷调试不成功

解决：检查快捷调试地址是否是完整的接口地址，参数是否正确

2. 如何查看 case 被哪些场景调用了

解决：点击到对应 case 里面，进行编辑，在下图所示位置可看到



3. 场景中，上个接口返回的数据，如何供下个接口调用

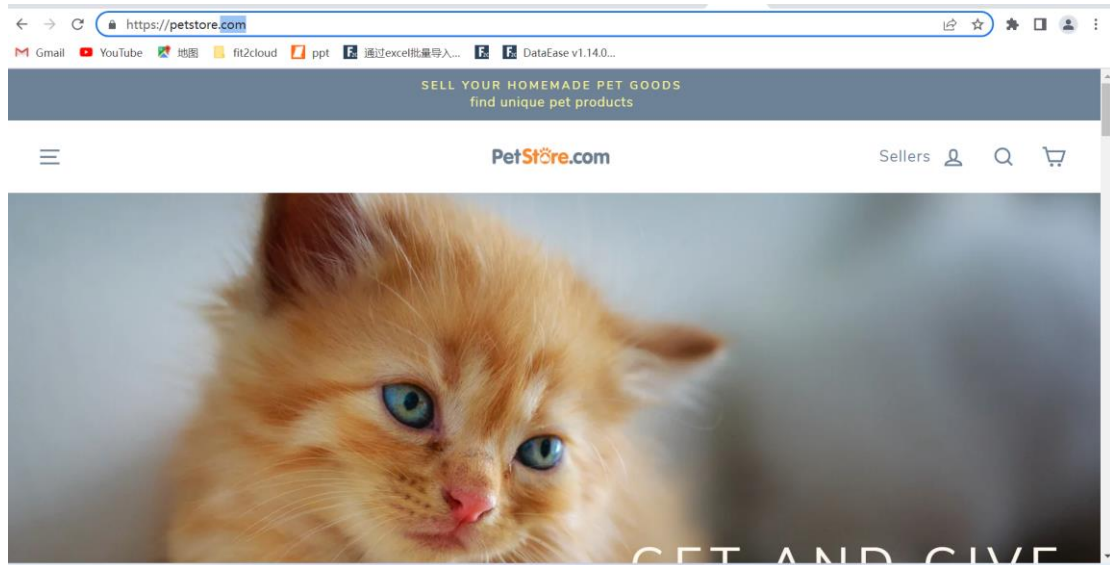
解决：在上个接口的数据返回，可以通过后置操作中提取参数的方式保存下来，需要调用的接口直接进行引用即可

3. MeterSphere 性能测试实验

3.1. 确定压测对象与脚本录制

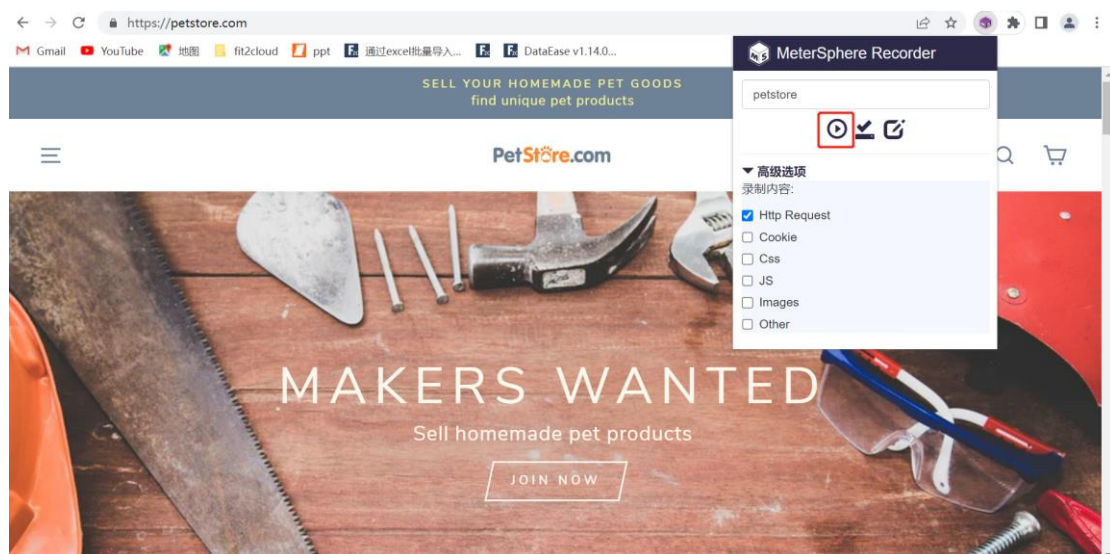
确定被测测试对象，也可以用上述被测的接口测试系统。

1) 谷歌浏览器打开宠物商店平台

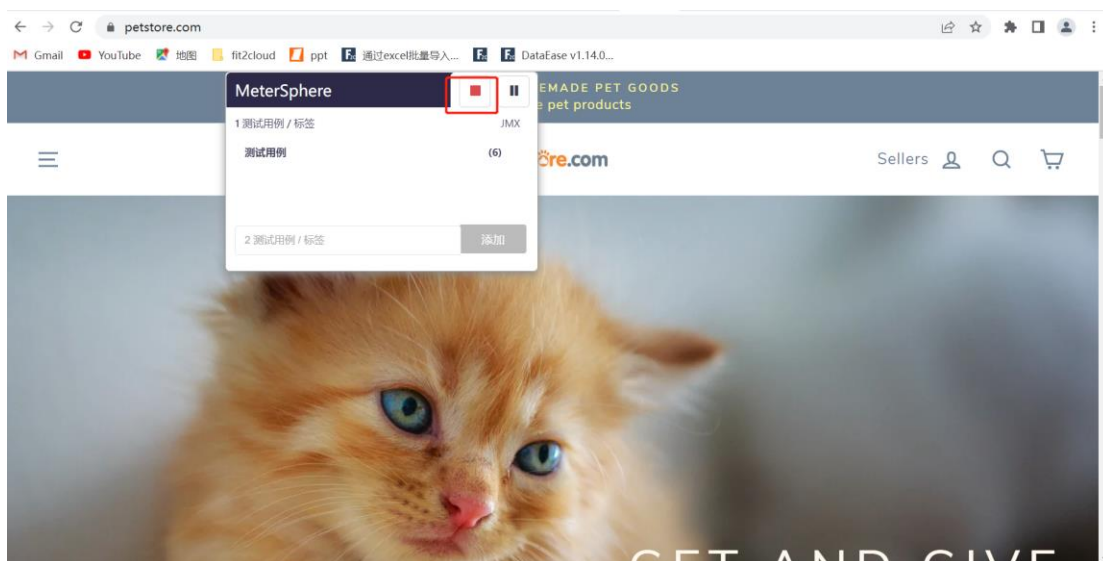


2) 加载 metersphere 插件（下载地址：

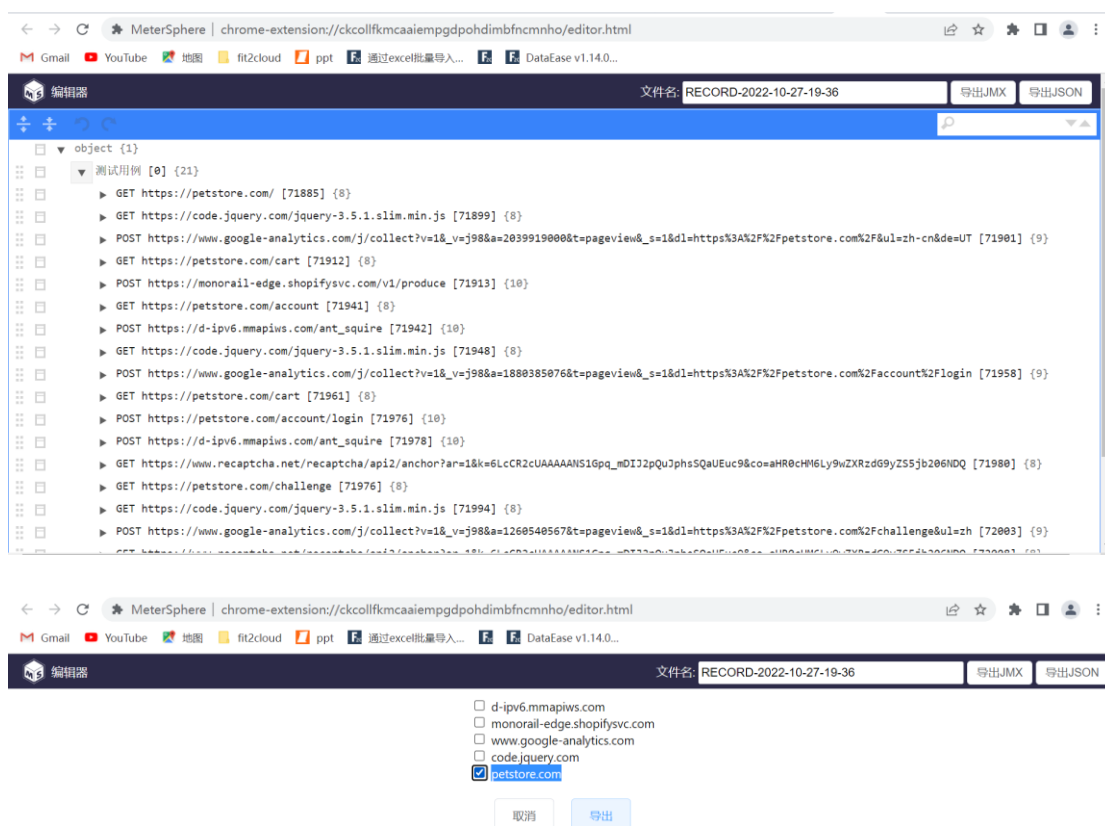
<https://github.com/metersphere/chrome-extensions>），高级选项中录制内容选择 http request，点击【开始】运行录制工具



3) 刷新宠物商店平台的首页，点击登录，完成登录宠物商店平台登录的流程，点击【停止录制】



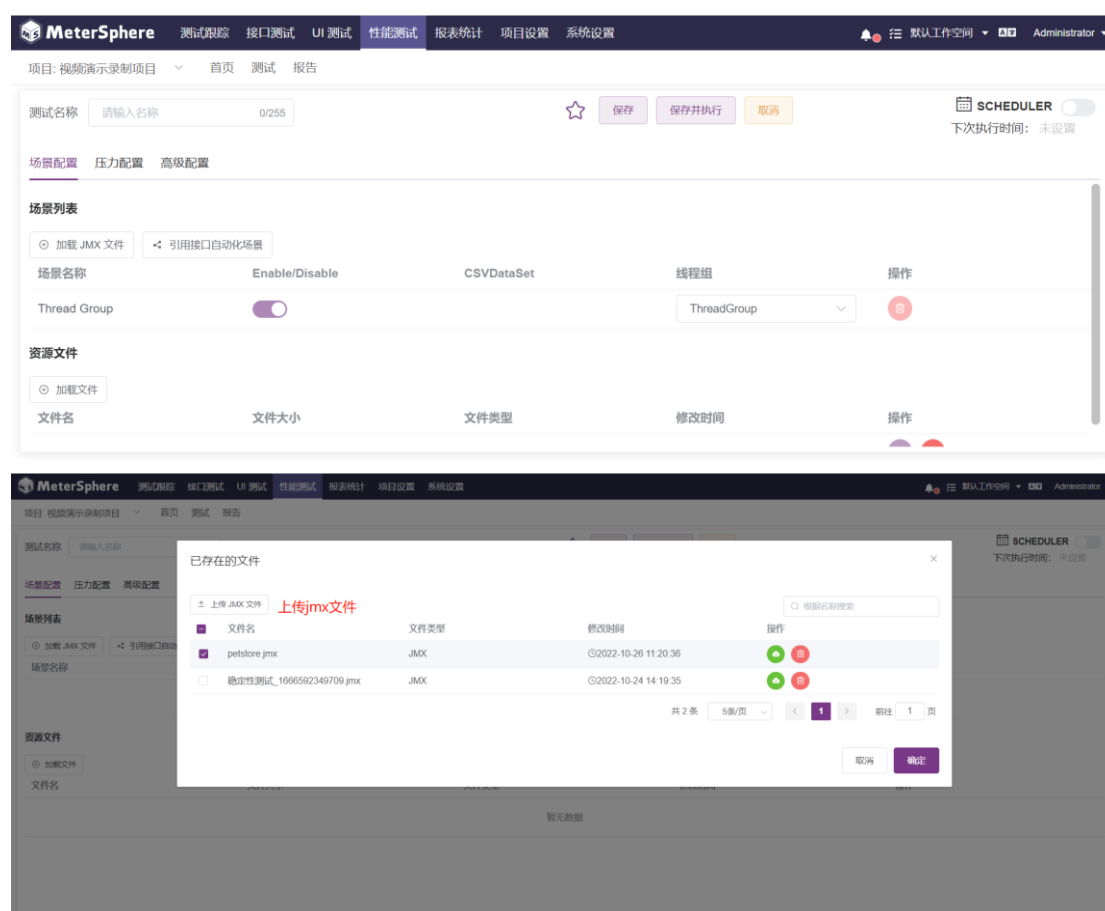
- 4) 点击【编辑】按钮进入接口页面，执行【导出 JMX】，选择宠物商店域名 petstore.com，点击【导出】，完成录制



3.2. 创建性能测试与压力参数配置

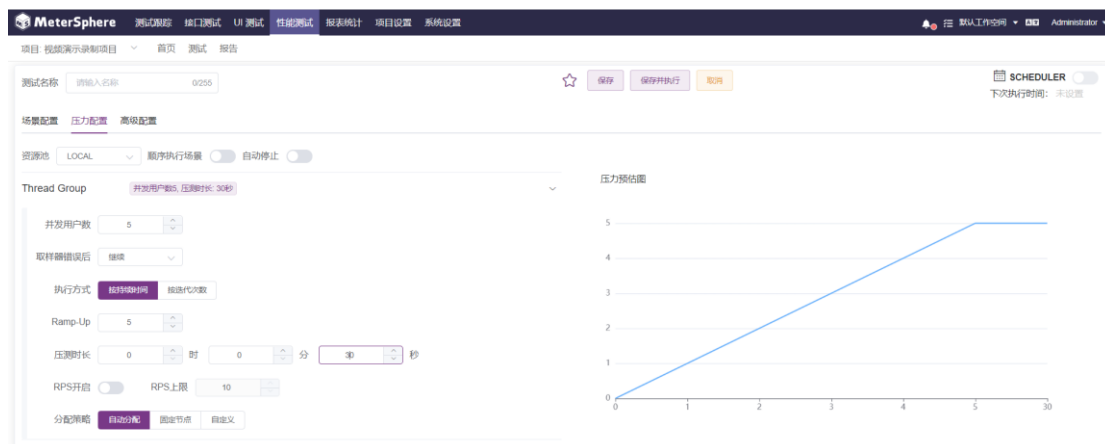
创建一个性能测试，其场景可以引用上面接口测试场景，也通过接口场景或者接口测试用例直接转化为性能测试，同时也可以导入本地电脑上的存量 JMeter 的 jmx 文件。

- 1) 点击左上角的【创建测试】，在场景配置中添加性能测试场景，点击【加载 jmx 文件】加载录制好的宠物商店的 jmx 脚本文件，也可以通过【引用接口自动化场景】加载接口测试中已调试好的自动化场景。



- 2) 进行压力配置，包括选择本地资源池，设置并发用户数、压测时长、Ramp-Up 时间和分配策略，选择执行方式以及是否开启 RPS 等；
- 压测池，这里选择默认资源池，也可以根据实际需求，切换到物理机，或者 k8s 集群资源池，

- ramp-up 参数是用来决定多长时间以内会启动所有的用户并发线程
- 用户的增长方式可以按线性增长，也可以是梯形增长方式，可以修改场景配置中的线程组来切换
- 分配策略针对集群资源池的配置相应的分配策略



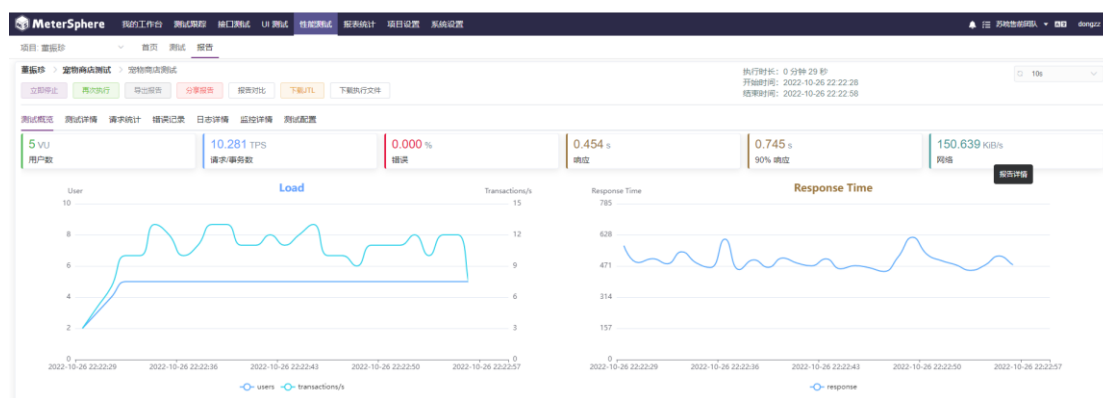
3) 进行高级配置：如 CSVDataSet、自定义变量；

- csvdataset 的话就是针对我们性能测试所需要的 csv 数据文件，如果资源池配置的是集群资源池的话，系统会根据压测策略自动来分割 CSV 数据文件里的参数
- 自定义变量功能就是添加一些变量名和变量值，这个配置项就可以直接修改我们 jmx 文件里面的相应的配置参数，这样做的好处就是我们可以使用同一个测试用例达到在不同测试场景下复用的效果

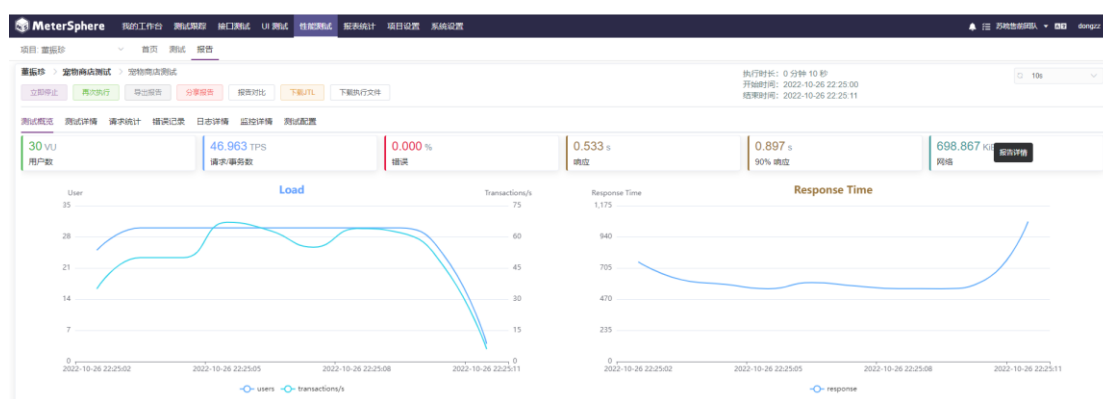


3.3. 性能报告解读分析

- 1) 保存、执行，并查看性能测试结果，进行结果分析；
 - 当用户数为 5 时. 压测时间 30 秒，Ramp-Up 调整为 5 秒，qps 达到 10，平均响应时间稳定在 0.45 秒左右，未出现任何报错。

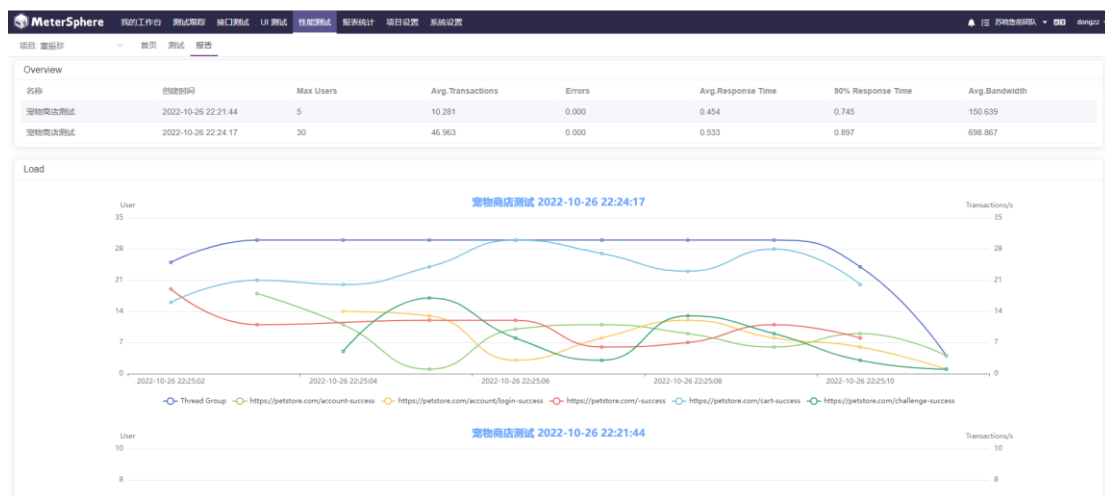


- 2) 重复 b) ~ e) 步骤，如调整并发用户数、压测时长、Ramp-Up 时间和分配策略等，优化测试效果，可以通过多次报告比对查看性能测试数据；
 - 将用户并发数调整到 30. 压测时间调整到 10 秒，Ramp-Up 调整为 3 秒，qps 达到 47，平均响应时间稳定在 0.5 秒左右，未出现任何报错。



- 3) 进行性能指标分析，看看能否发现一些性能瓶颈问题。
 - 比对两次报告发现当用户数从 5 调整到 30，他的响应时间和传输速度未受

到影响，说明此宠物商店平台至少可以支撑 30 个并发用户同时使用平台。



3. 4. 实验中遇到的问题与解决

1. 压测报告折现图形可以配置吗？

解决：在创建性能测试时的高级设置中可以修改报告数据的聚合时间。

2. MeterSphere 可以监控被测系统服务器指标吗？

解决：MeterSphere 使用 Prometheus 进行发压机以及被测系统服务器的监控，可以在性能测试-高级配置里面，添加被测系统服务器的 node_exporter 的地址。主服务会在安装 MeterSphere 系统时默认安装，而其他服务器，则需要单独安装 node_exporter，即可在执行性能测试的时候完成相关指标的监控。

3. 性能测试并发量加大的时候报错 Non HTTP response code:

java.net.SocketTimeoutException

4. 解决：在高级配置界面增加超时时间。



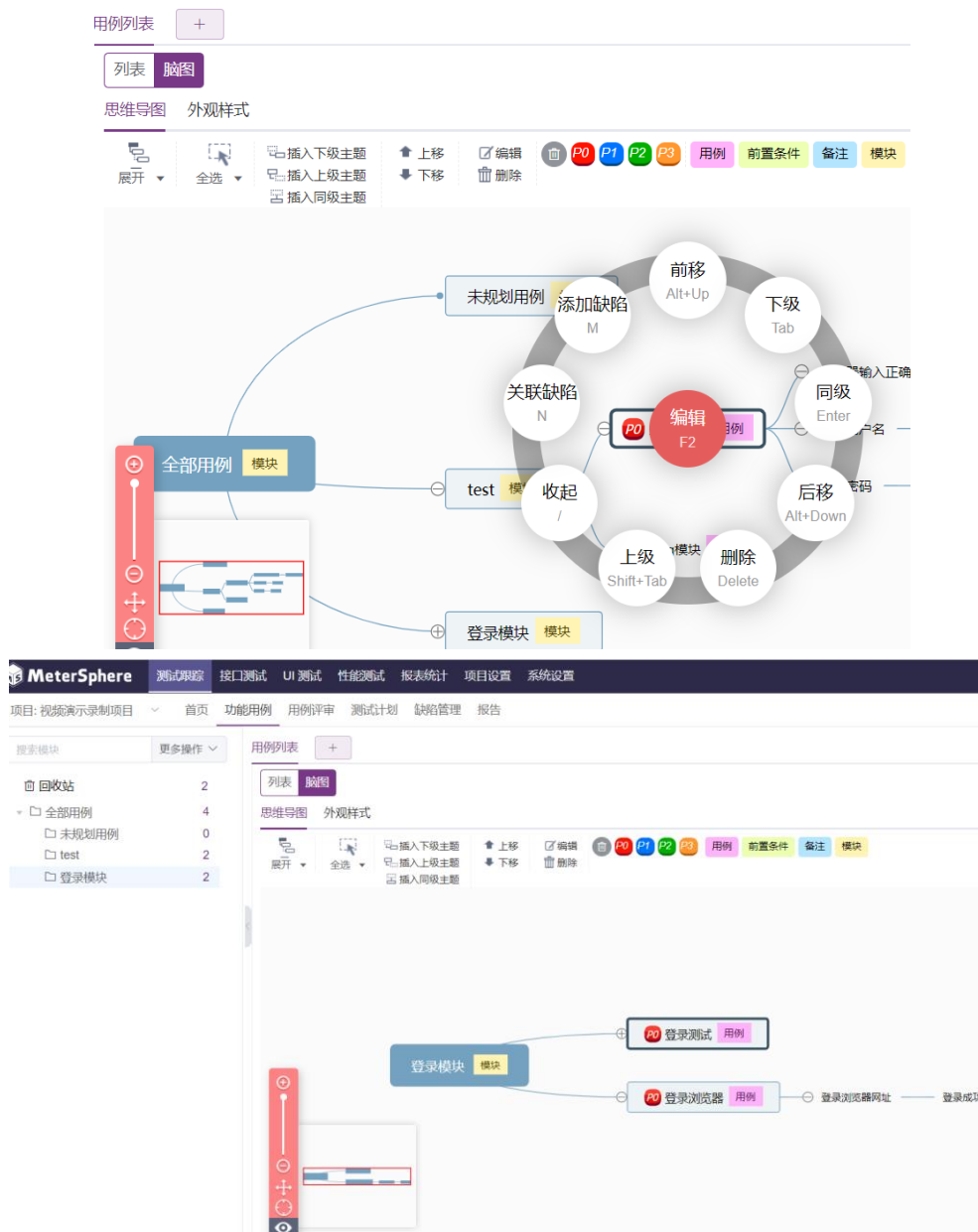
4. MeterSphere 功能测试流程实验

4.1. 创建功能测试用例

- 1) 进入测试跟踪；进入功能用例，创建左边模块目录结构；
 - 进入【功能用例】界面。左上方显示当前项目，项目下方显示此项目的全部测试用例模块。
 - 选中模块树列表【全部用例】，点击【+】号按钮在模块总目录下创建一级模块。选中任意上级模块即可在该模块下创建次级模块。

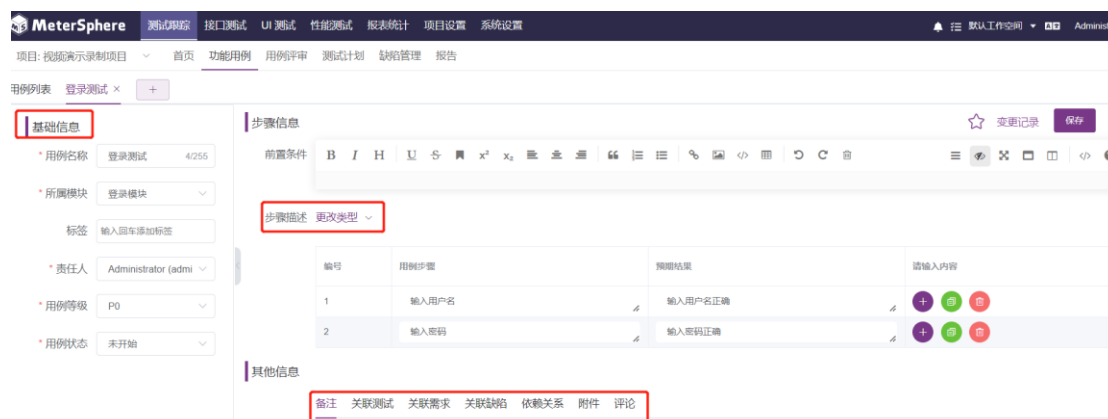


- 2) 先以脑图方式创建测试用例
 - 在用例列表页面点击【脑图】切换以脑图形式展现用例。
 - 右键主题区域触发功能键。各级主题使用标签方式区分文本内容，支持使用标签方式标记用例优先级。



3) 再以列表方式补充相关信息，加入到相应的模块；

- 选择新建用例所属模块，点击列表上方【+】下拉菜单选择【新建用例】。
- 进入用例详情页面进行编辑，填写基础信息后即可将用例保存至用例列表。
- 在新建用例详情页面中，进一步填写用例步骤信息，为用例添加备注、附件和评论，支持为用例关联测试、关联缺陷、关联缺陷。编辑完成后，点击【保存】将此条用例保存至列表。
- 新建用例将在用例列表中呈现，点击此条用例或点击此条用例后的【编辑】按钮可重新进入用例详情页面进行编辑。



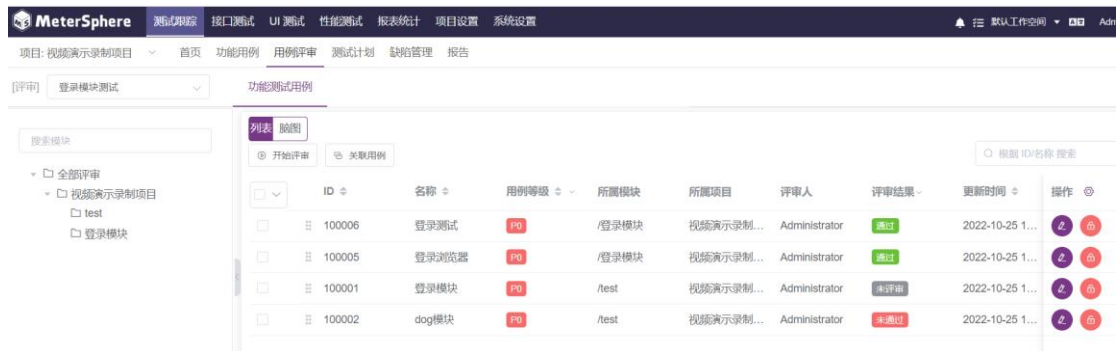
4) 新建测试用例可以和平台已有的接口测试用例、接口场景用例、性能测试用例进行关联。进入测试用例详情页面，点击【关联测试】标签，选择希望与本条用例进行关联的用例关联测试测试类型，进入该类型的用例列表进行关联。



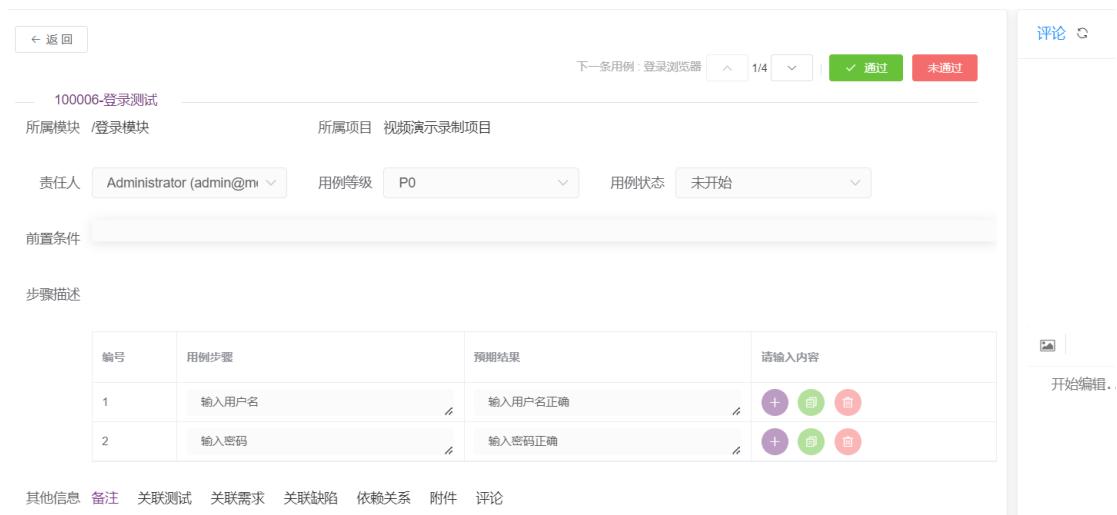
4.2. 创建测试用例评审

- 1) 创建相应的测试评审，可以对测试用例的内容进行评论或者上传附件等；
 - 点击【用例评审】可查看当前项目中所有的测试用例评审。
 - 进入测试用例评审详情页时，系统提供了两种用例显示模式：列表和脑图。
 - 点击【创建用例评审】，在弹出页面中编辑评审任务基础信息，点击【确认】保存至评审任务列表，用户可以在用例评审主页点击选定评审任务的【编辑】按钮重新进入编辑窗口。在任务创建页面，点击【规划&执行】将直接进入评审任务执行主页。
 - 在评审任务执行主页，点击右侧列表上方的【关联用例】按钮为评审任务关联用例。
 - 在弹出的用例选择列表中，选择要添加到此次用例评审任务中的测试用例，

点击【确定】完成添加。



- 2) 为评审任务添加完用例后，点击【开始评审】按钮进入用例详情页，默认从任务列表的第一条用例面实施评审，用户也可以点击任意一条用例的【编辑】按钮，从该条用例开始实施评审。
- 3) 进入用例评审详情页面，使用【翻页】按钮快速切换用例，使用标签快捷标注用例评审结果，标注后的评审结果将会在用例列表中显示。同时支持用户为此次评审发表评论。



4.3. 创建测试计划

- 1) 创建并执行测试计划，将相应的功能用例、接口用例、接口场景、性能测试等加入相应的测试用例；
 - 点击【创建测试计划】，填写信息。
 - 点击【确定】完成创建测试计划，并且页面跳转到测试计划列表信息页面，可在测试计划列表中查看到创建的测试计。

- 点击【规划&执行】完成创建测试计划，并且页面跳转到测试计划内容维护界面，可直接添加本次计划的测试范围和测试内容。



- 2) 在测试计划列表信息列表，点击某条计划，可以对测试计划详情进行维护；
- 3) 添加本次计划测试的范围：功能测试用例、接口测试用例、场景测试用例、性能测试用例等；



- 4) 点击【报告】按钮，可以查看测试计划的报告；【执行】按钮，可以执行测试计划；【...】按钮，在下拉操作菜单中，【删除】按钮，可以对测试计划进行删除；点击【定时任务】按钮，可以设置一个指定的执行时间；【复制】按钮，可克隆原计划生成新的计划，同时可对此新计划进行正常的维护 and 操作。



4.4. 缺陷创建

- 1) 缺陷管理，包括创建缺陷、修改缺陷（改变缺陷状态）等；
- 点击左上角的【创建缺陷】，可以在这里创建缺陷，但在此创建的缺陷需要手动关联测试用例。也可以在测试用例执行中创建缺陷或关联缺陷。

MeterSphere										
项目: 视频演示录制项目										
测试计划 接口测试 UI 测试 性能测试 报表统计 项目设置 系统设置										
项目: 视频演示录制项目 首页 功能用例 用例评审 测试计划 缺陷管理 报告										
创建缺陷 根据 ID/名称 搜索										
缺陷ID	缺陷标题	平台	创建人	缺陷来源	缺陷描述	用例数	创建时间	严重程度	处理人	状态
100002	网页密码复杂...	Local	Administrator	--	预览	1	2022-10-25 17:28:53	P0	Administr...	新建
100001	网页加载缓慢	Local	Administrator	--	预览	0	2022-10-24 14:09:10	P0	Administr...	已解决

4.5. 测试报告查看

1) 运行测试计划，查看测试报告：

名称	测试计划名称	创建人	创建时间	触发方式	状态	运行耗时(s)	成功率	操作
登录测试-2022-10-25 17:26:12	登录测试	Administrator	2022-10-25 17:...	手动触发	Completed	37.73	16.7%	查看测试报告
test-2022-10-24 15:36:08	test	Administrator	2022-10-24 15:...	手动触发	Completed	36.51	33.3%	
test-2022-10-24 15:35:20	test	Administrator	2022-10-24 15:...	手动触发	Completed	36.59	33.3%	

[返回](#)
[登录测试-2022-10-25 17:26:12](#)

[分享](#)
[导出](#)

概览

测试时间: 2022-10-25 17:26:12 ~ 2022-10-25 17:26:49
 测试总数: 6 执行率: 16.7% 通过率: 16.7%

报告总结

功能用例统计分析
[测试结果](#)
[失败用例](#)
[缺陷列表](#)
[所有用例 4](#)

功能用例数

4

4.6. 实验中遇到的问题与解决

1. 从本地 xmind 复制数据粘贴到 MeterSphere 用例脑图直接保存后没有数据。

解决：需要将相应的数据标记为模块或者用例才能保存。

2. 提交缺陷页面的字段想要自定义配置。

解决：添加自定义字段。具体操作在【项目设置】/【模版管理】/【缺陷模版】页面中，选择要添加字段的缺陷模版，打开【编辑缺陷模版】页面。在【编辑缺陷模版】页面，点击【设置自定义字段】按钮打开【创建字段】对话框，在对话框中填写字段名称、备注，选择【字段类型】，点击【确定】保存。

3. 测试用例误删除了，怎么找回来。

解决：在【测试跟踪-测试用例】模块，点击【回收站】找回。