



**实验报告（参考模板）**

课程名称 软件测试实验

学生学院 计算机学院

专业班级 软件工程22级 班

学   号 312888888

学生姓名

指导教师 廖朝辉

2024年1月4日

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 成文信息 | | | | | | | | |
| 文档名称： | | | 软件测试实验报告 | | | | | |
| 作者： | | | 陈婉瑜 | | | | | |
| 审核人： | | | 廖朝辉 | | | | | |
| 批准发布人： | | | 草稿[ 🗸 ] 内部[ ] 对外发布[ ] | | | | | |
| 文档类别： | | | 工程类[ 🗸 ] 管理类[ ] 支持类[ ] | | | | | |
| 发布信息 | | | | | | | | |
| 主送方： | | | 陈婉瑜 | | | 存档日期： | 2024年1 月 4 日 | |
| 抄送方： | | | 陈婉瑜 | | | 发布日期： | 2024年 1 月 4 日 | |
| 签收信息 | | | | | | | | |
| 发送方：陈婉瑜 | | 接收方：陈婉瑜 | | 接收方：陈婉瑜 | | | | 接收方：陈婉瑜 |
| 代表人：陈婉瑜 | | 代表人：陈婉瑜 | | 代表人：陈婉瑜 | | | | 代表人：陈婉瑜 |
| 日期：2024.1.4 | | 日期：2024.1.4 | | 日期：2024.1.4 | | | | 日期：2024.1.4 |
| 变更信息 | | | | | | | | |
| 版本 | 原因 | | | | 作者 | | | 日期 |
| Halo2.0.0 | 修复bug完成 | | | | 陈婉瑜 | | | 2024年 1月 4日 |

# 测试准备

1. **测试环境**

### 1.1硬件环境

处理器 AMD Ryzen 5 6600HS Creator Edition 3.30 GHz

机带 RAM 16.0 GB (13.7 GB 可用)

系统类型 64 位操作系统, 基于 x64 的处理器

版本 Windows 11

### 1.2软件环境

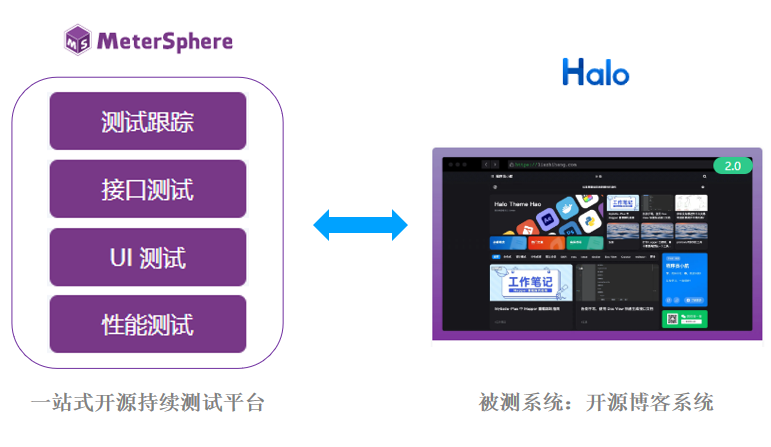


图1-2-1 测试平台和被测系统

测试工具平台地址：https://metersphere.topviewclub.cn/

Halo地址：10.21.32.183:8090

MeterSphere 是一站式的开源持续测试平台，遵循 GPL v3 开源许可协议，涵盖测试跟踪、接口测试、UI 自动化测试和性能测试等功能，全面兼容 JMeter、Selenium 等主流开源标准，有效助力开发和测试团队充分利用云弹性进行高度可扩展的自动化测试，加速高质量的软件交付。

Halo 技术栈：主流的 SpringBoot 微服务。

Halo Github 源码仓库：<https://github.com/halo-dev/halo>。

Halo 官网地址：<http://halo.run>。

### 1.3网络环境

Gdut校园网络

1. **测试分工及计划**

## 2.1 分组情况

组长：陈婉瑜

组员：陈婉瑜

## 2.2 测试计划

1：计划制定

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 阶段计划 | 任务 | 负责人 | 计划开始时间 | 计划完成时间 |
| 测试用例编写 | 测试用例编写 | 陈婉瑜 | 2024.11.1 | 2024.11.6 |
| 测试用例评审 | 陈婉瑜 | 2024.11.7 | 2024.11.9 |
| 系统功能测试 | 测试设计 | 陈婉瑜 | 2024.11.10 | 2024.11.13 |
| 设计执行 | 陈婉瑜 | 2024.11.14 | 2024.11.16 |
| 测试报告 | 陈婉瑜 | 2024.11.17 | 2024.11.18 |
| 接口测试 | 测试设计 | 陈婉瑜 | 2024.11.19 | 2024.11.21 |
| 设计执行 | 陈婉瑜 | 2024.11.22 | 2024.11.24 |
| 测试报告 | 陈婉瑜 | 2024.11.25 | 2024.11.26 |
| 接口自动化测试 | 测试设计 | 陈婉瑜 | 2024.11.27 | 2024.11.30 |
| 设计执行 | 陈婉瑜 | 2024.12.1 | 2024.12.3 |
| 测试报告 | 陈婉瑜 | 2024.12.4 | 2024.12.5 |
| 性能测试 | 测试设计 | 陈婉瑜 | 2024.12.6 | 2024.12.8 |
| 设计执行 | 陈婉瑜 | 2024.12.9 | 2024.12.12 |
| 测试报告 | 陈婉瑜 | 2024.12.13 | 2024.12.14 |
| 综合测试 | 测试设计 | 陈婉瑜 | 2024.12.15 | 2024.12.17 |
| 设计执行 | 陈婉瑜 | 2024.12.18 | 2024.12.22 |
| 测试报告 | 陈婉瑜 | 2024.12.23 | 2024.12.24 |

2：计划录入



# 测试执行

## 实验一：测试用例编写实验

### 实验目的

1. 掌握测试用例的编写规范。
2. 掌握根据功能设计编写测试用例方法。
3. 掌握基于 MeterSphere 专业测试云进行测试用例的编写及评审。

### 实验内容

1. 完成对 Halo 待测系统文章评论功能的测试用例设计。
2. 使用 MeterSphere 专业测试云完成测试用编写与评审。

### 实验步骤

采用场景法针对系统文章评论功能的测试用例设计如下：

首先，针对评论的完整步骤进行分析：

用户登录->打开文章->输入评论内容->提交评论

以上四步为必备步骤，设置为基本流。

评论模块中还包含有表情包发送功能，这个为选填项，设置为备选流。分析必备步骤中的异常操作也纳入备选流。

将这些备选流进行组合并与基本流进行组合以尽可能地覆盖大部分情况

1. 基本流

基本流1：用户成功登录系统。

基本流2：打开一篇文章

基本流3：用户在评论区输入符合规则的评论内容

基本流4：用户点击提交按钮

二、备选流

备选流 1：未登录评论

备选流 2：网络异常时点击提交评论

备选流 3：评论内容为空

备选流 4：点击输入表情包

这些基本流与备选流可以结合出以下的场景组合，涵盖了大部分的情况：

场景组合：

基本流1+基本流2+基本流3+基本流4

备选流1+基本流2+基本流3+基本流4

基本流1+基本流2+基本流3+备选流2

基本流1+基本流2+备选流3+基本流4

基本流1+基本流2+基本流3+基本流4+备选流4

实验结果



图1：用例列表以及评审结果



图2：评论成功（字符评论）



图3：评论成功（点击表情包评论）



图4：评论失败（未登录）



图5：评论失败（评论内容为空）



图6：评论失败（提交评论时网络异常）

### 实验总结

通过本次实验，我深刻体会到了测试用例设计和评审在软件测试中的重要性。合理的测试用例设计能够确保测试的全面性和准确性，而严格的评审过程则能够提升测试用例的质量。通过对文章评论功能的用例设计和评审，我了解了测试用例评审的流程和要点，提升了测试设计和评审的实践能力。但本次实验中测试用例的设计也存在一些不足，首先由于缺乏需求文档，没进行需求分析这一步，对于评论的测试需求不清晰，可能导致编写用例存在缺陷，不完备。

## 实验二 系统功能测试实验

### 实验目的

1. 掌握系统功能测试流程。
2. 掌握系统功能测试计划制定。
3. 掌握系统功能测试执行、缺陷管理和测试报告。

### 实验内容

1. 编写 Halo 开源建站工具测试计划。
2. 使用 MeterSphere 专业测试云完成以下操作：
   1. Halo 系统功能测试用例维护；
   2. 系统功能测试执行计划的创建；
   3. 系统功能测试执行任务分配；
   4. 系统功能测试执行；
   5. 系统功能测试总结和测试报告生成。

### 实验步骤

3.1测试范围和需求



图 3-1 Halo开源社区需求处理流程

Halo 开源社区用户反馈的缺陷经由测试人员复测确认是缺陷后，转给开发人员，由开发人员完成缺陷修复，然后测试人员完成测试工作。



图 3-2 Halo开源社区缺陷处理流程

测试人员将开发人员开发的需求和修复的缺陷作为确定的测试范围，分析需求和缺陷，设计测试用例。

以 Halo 2.2.1 Release 为例（ <https://github.com/halo-dev/halo/releases> ），该发布的需求列表和修复的缺陷列表如下：

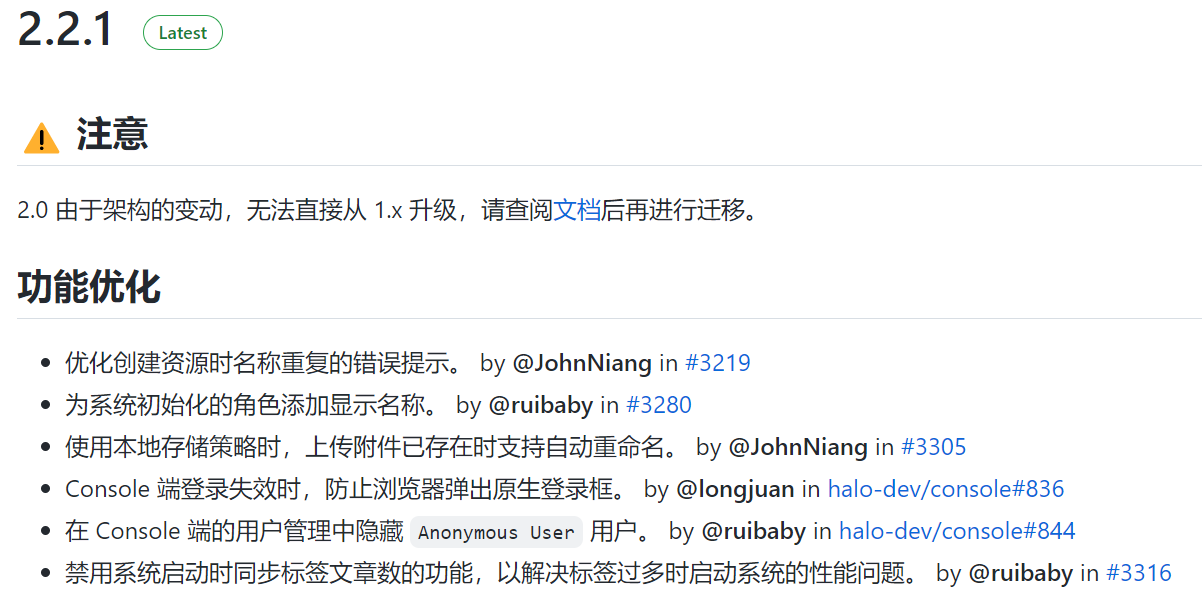


图 3-3 Halo 2.2.1需求列表



图 3-4 Halo 2.2.1缺陷列表

以上“功能优化”列表和“问题修复”列表，就是 Halo 2.2.1 的测试范围。

3.2**测试计划制定**

测试计划的目的是：规定测试活动的范围、方法、资源和进度。明确正在测试的项目、要测试的特性、要执行的测试任务、每个任务的负责人，以及与计划相关的风险。最终提高测试的工作效率，保障测试工作顺利、保质保量完成测试工作。

针对 Halo 2.2.1 发布的简版测试计划如下：

**测试范围：**

功能优化列表（见图3-3）；

问题修复列表（见图3-4）：

**测试方法：**

以功能测试为主，辅助以接口测试（参照实验五接口测试实验）、 Web UI自动化测试（参照实验四 Web UI 自动化测试实验）和性能测试（参照实验七性能测试实验）。

**测试环境：**Halo 环境。

表3-5 登录测试用例

|  |  |  |
| --- | --- | --- |
| URL | 账号 | 密码 |
| [http://10.21.32.183:8090](http://halo.edu.metersphere.com) | admin | TopView726 |

**测试参考文档：**

需求列表和缺陷列表，详情请访问 <https://github.com/halo-dev/halo/releases> 。

**测试时间计划：**

表3-6 测试时间计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试阶段 | 时间计划 | 主要任务 | 负责人 | 备注 |
| 计划制定 | 12月20日～12月22日 | 测试计划编写  评审测试计划 | 陈婉瑜 |  |
| 测试设计 | 12月23日～12月29日 | 需求分析  用例设计  用例评审 | 陈婉瑜 |  |
| 测试环境部署 | 12月30日～1月2日 | 部署Halo测试环境 | 陈婉瑜 |  |
| 测试执行 | 1月3日～1月10日 | 分配并执行测试用例  提交缺陷 | 陈婉瑜 |  |
| 测试报告 | x月x日～x月x日 | 编写测试报告 | 陈婉瑜 |  |

**测试资源投入：**

表3-7 测试资源投入计划

|  |  |  |  |
| --- | --- | --- | --- |
| 人员 | 姓名 | 计划人天投入 | 备注 |
| 测试经理 | 陈婉瑜 | 10人天 |  |
| 测试工程师 | 陈婉瑜 | 15人天 |  |
| 测试工程师 | 陈婉瑜 | 15人天 |  |

**测试输出：**

测试报告。

**风险因素：**

1. 测试人力资源不足；
2. 开发进度延期。

3.3 测试设计

3.4测试执行

测试用例编写和评审是软件测试中的重要工作。

开发人员完成功能开发，并自测通过后，提交测试人员测试。测试人员开启测试执行工作。测试执行工作需要依赖我们制定的测试计划，需要先完成测试计划制定，然后进行测试执行。

3.5测试报告生成和查看

在测试执行过程中通常需要每天和每周给产品经理、测试总监、开发经理汇报测试进度，以及测试过程中发现的问题。

MeterSphere 在测试执行过程中会动态产生实时执行报告发送给相关负责人。

### 实验结果

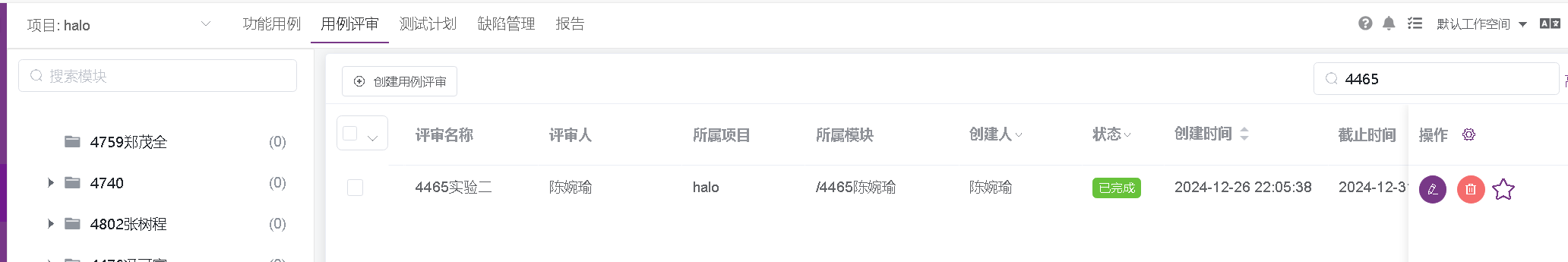


图4-1用例评审创建



图4-2测试计划创建

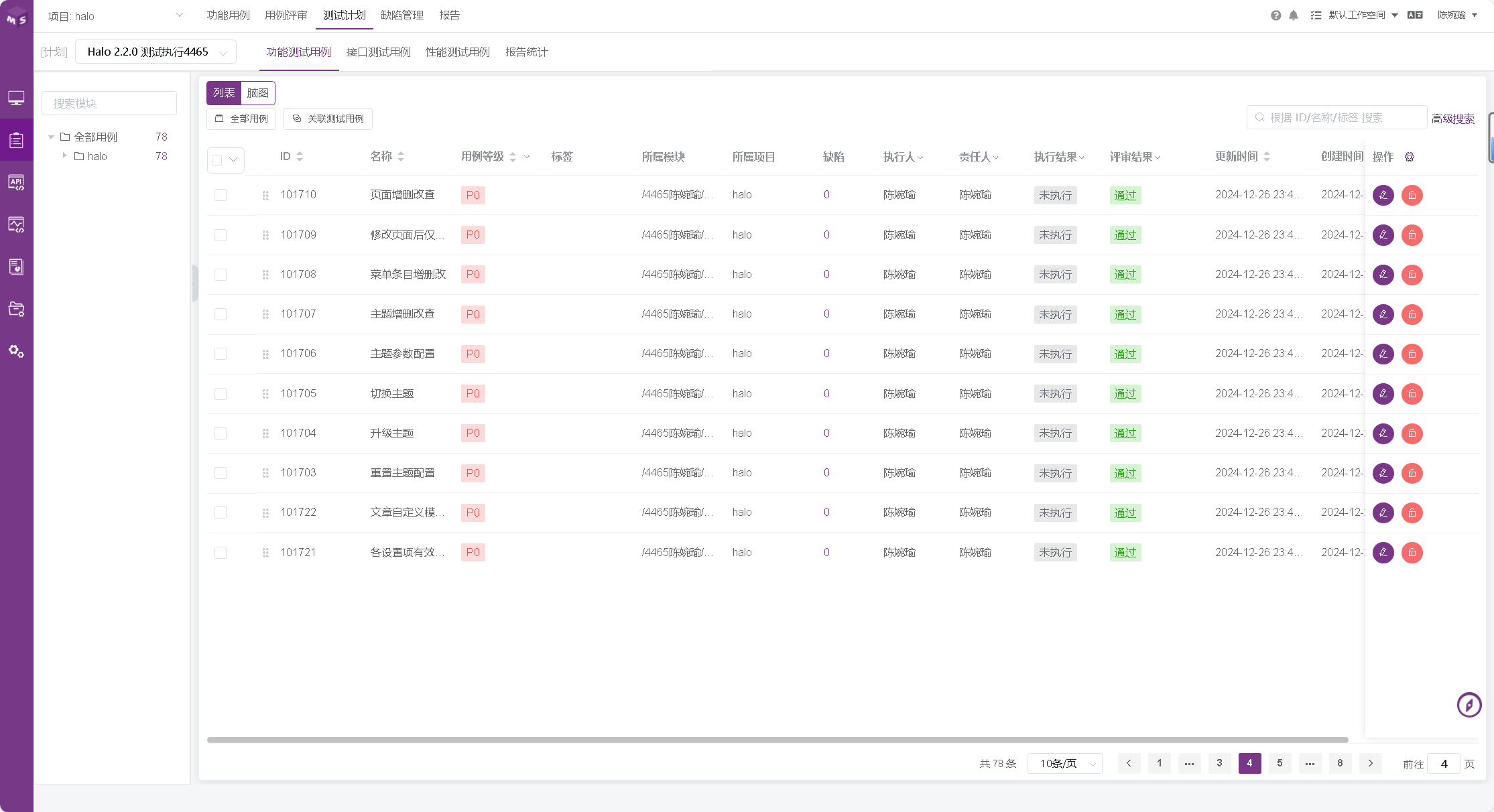


图4-3测试计划-关联测试用例



图4-4缺陷提交图

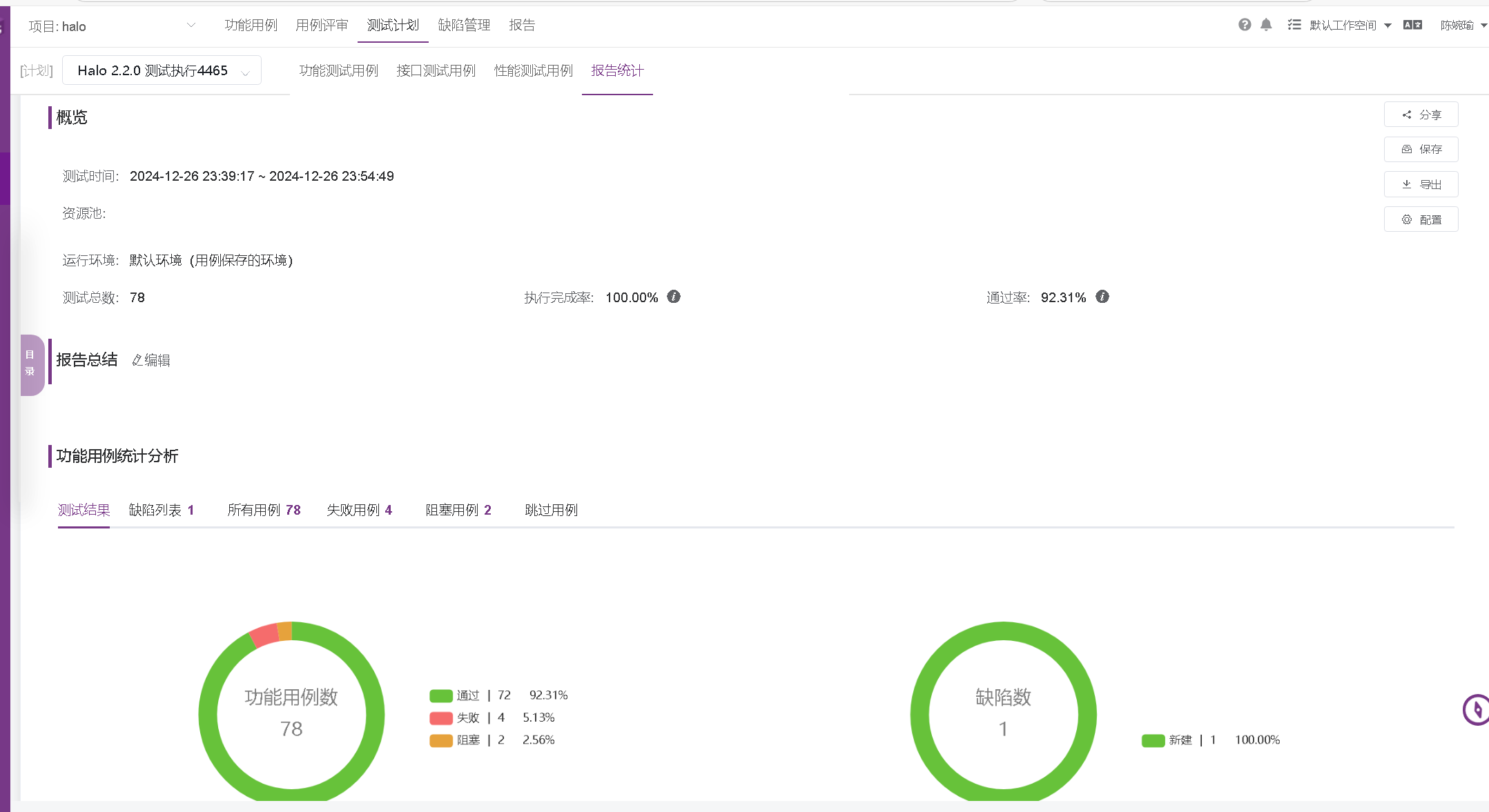


图4-5测试报告



图4-6测试结果



图4-7缺陷列表

**测试报告url:**

https://metersphere.topviewclub.cn/track/share-plan-report?shareId=2744ce39-60f4-49f0-8570-f9e515d1fd56

### 实验总结

通过本次实验，我深刻认识到了系统功能测试在软件开发过程中的重要性。掌握了系统功能测试的流程和方法，学会了如何制定详细的测试计划、编写测试用例、执行测试任务、缺陷管理以及编写测试报告。

未来，我将继续加强自己的测试技能和知识储备，不断提高自己的测试能力和水平。同时，积极参与更多的测试实践项目，将所学知识应用于实际工作中，为软件的质量保障贡献自己的力量。

## 实验三：接口测试

### 实验目的

1. 理解接口测试的作用以及常用请求参数含义。
2. 掌握接口定义的创建步骤和调试方法。
3. 能根据接口文档进行接口定义并能够转化为接口测试用例。
4. 会对项目环境进行基本的配置。

### 实验内容

1. 查阅 Halo 系统的接口测试文档。
2. 完成 Halo 系统登录、编写文章、发布文章和查看文章列表等接口的定义。
3. 将定义完成的接口转化为接口测试用例并进行调试。

### 实验步骤

本实验对Halo系统“文章发布”的场景进行接口定义并调试，“文章发布”场景的流程为：用户登录→编写文章→发布文章→获取文章列表。

首先阅读Halo系统的接口文档，弄清楚该场景的需要使用到的接口的功能和参数；然后在接口定义中完成接口定义及调试，最后将接口转化为测试用例并进行调试。

**3.1新建项目和运行环境**



图3-1-1环境配置(http配置)



图3-1-2环境配置（全局认证配置）

**3.2添加模块**



**3.3创建接口**



图3-3-1登录接口

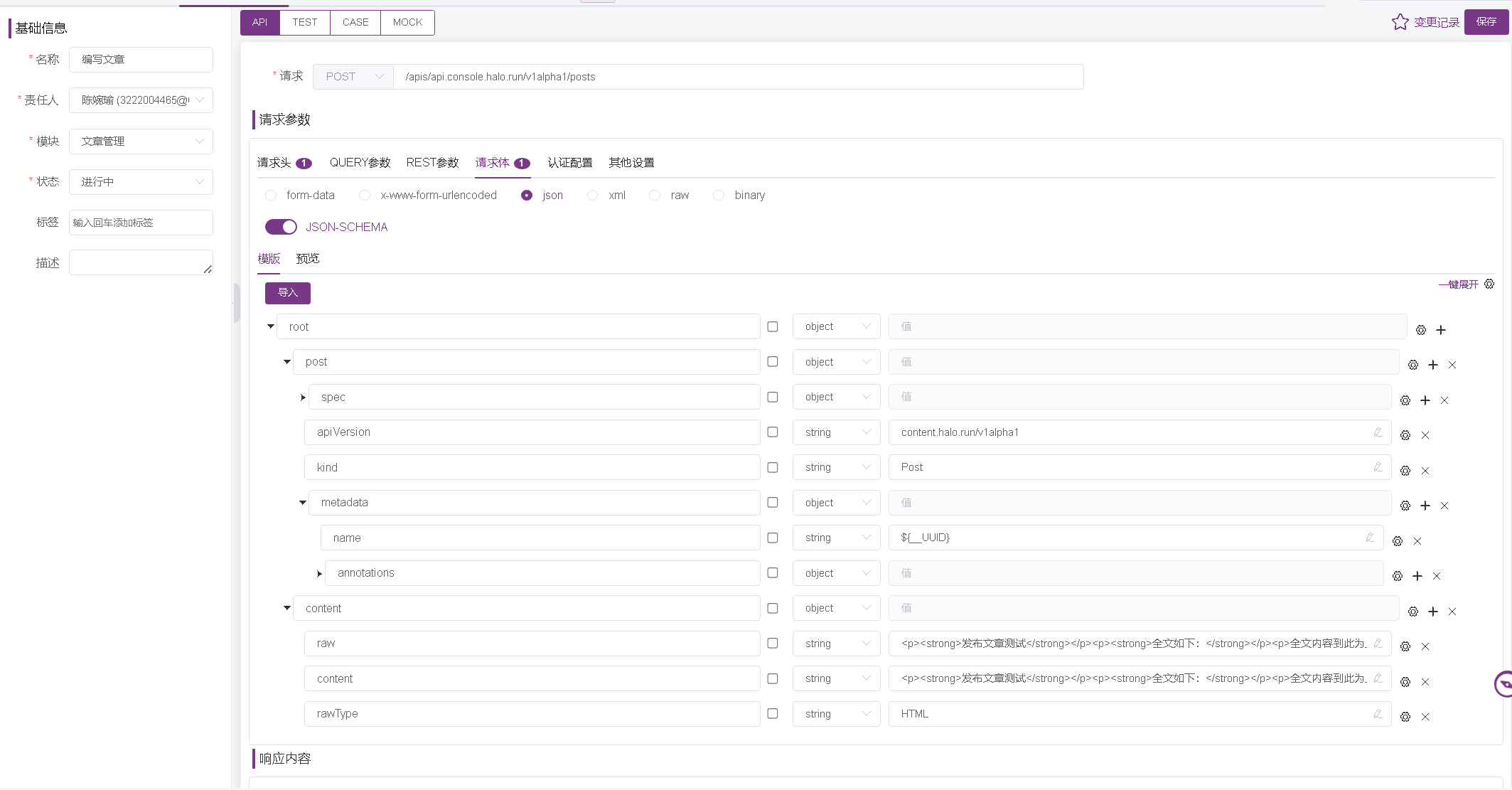


图3-3-2编写文章接口

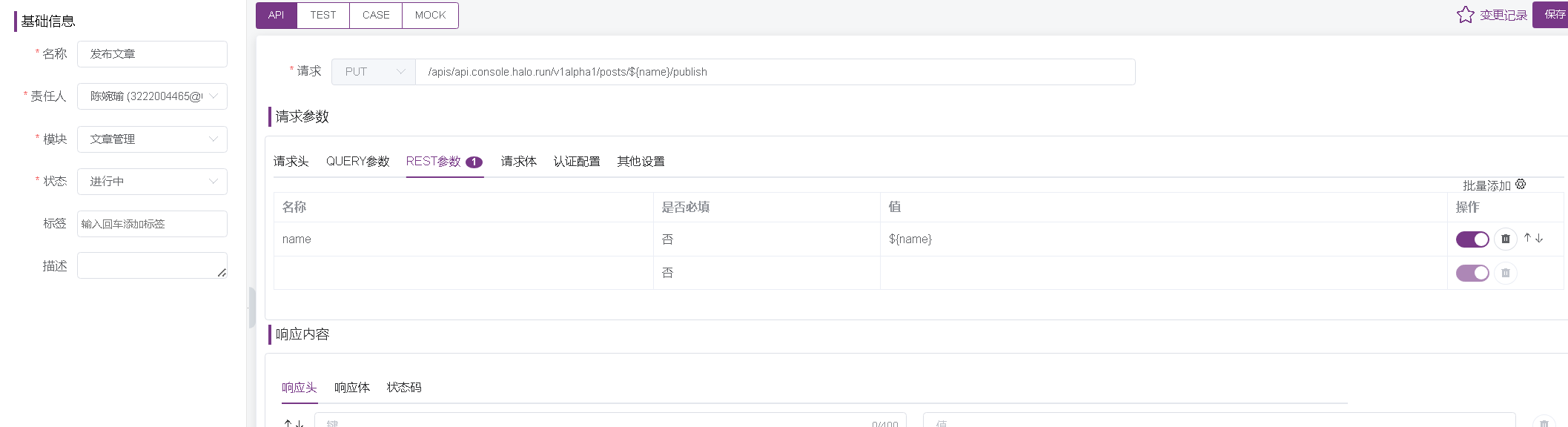


图3-3-3发布文章接口

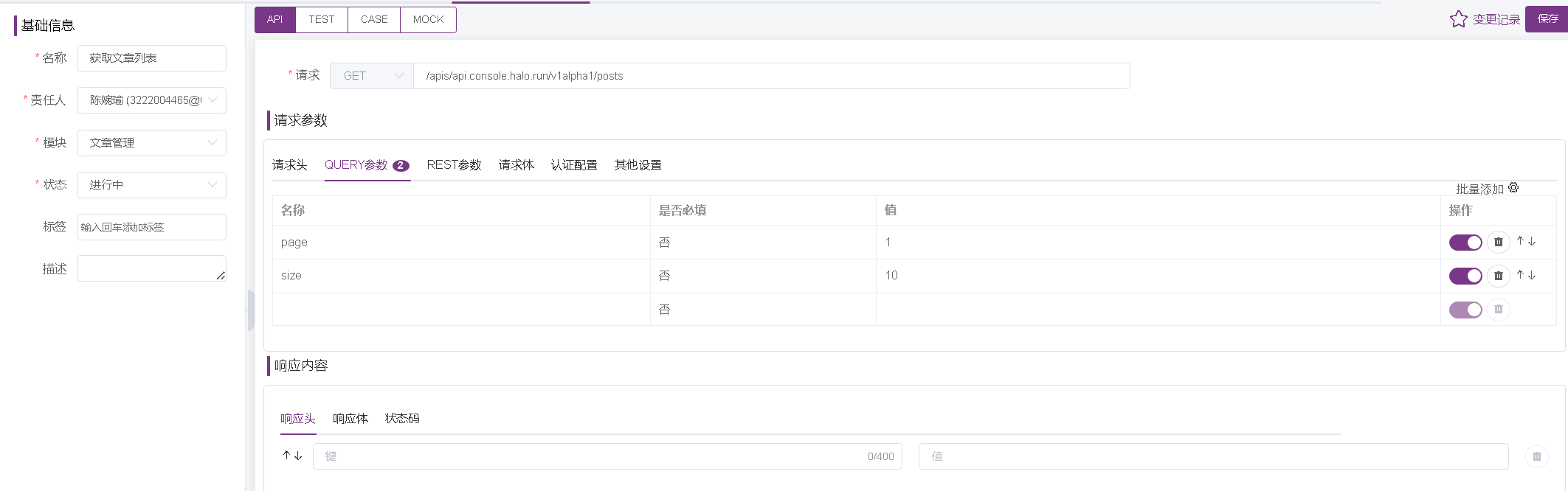


图3-3-4获取文章列表接口

**3.4接口调试**

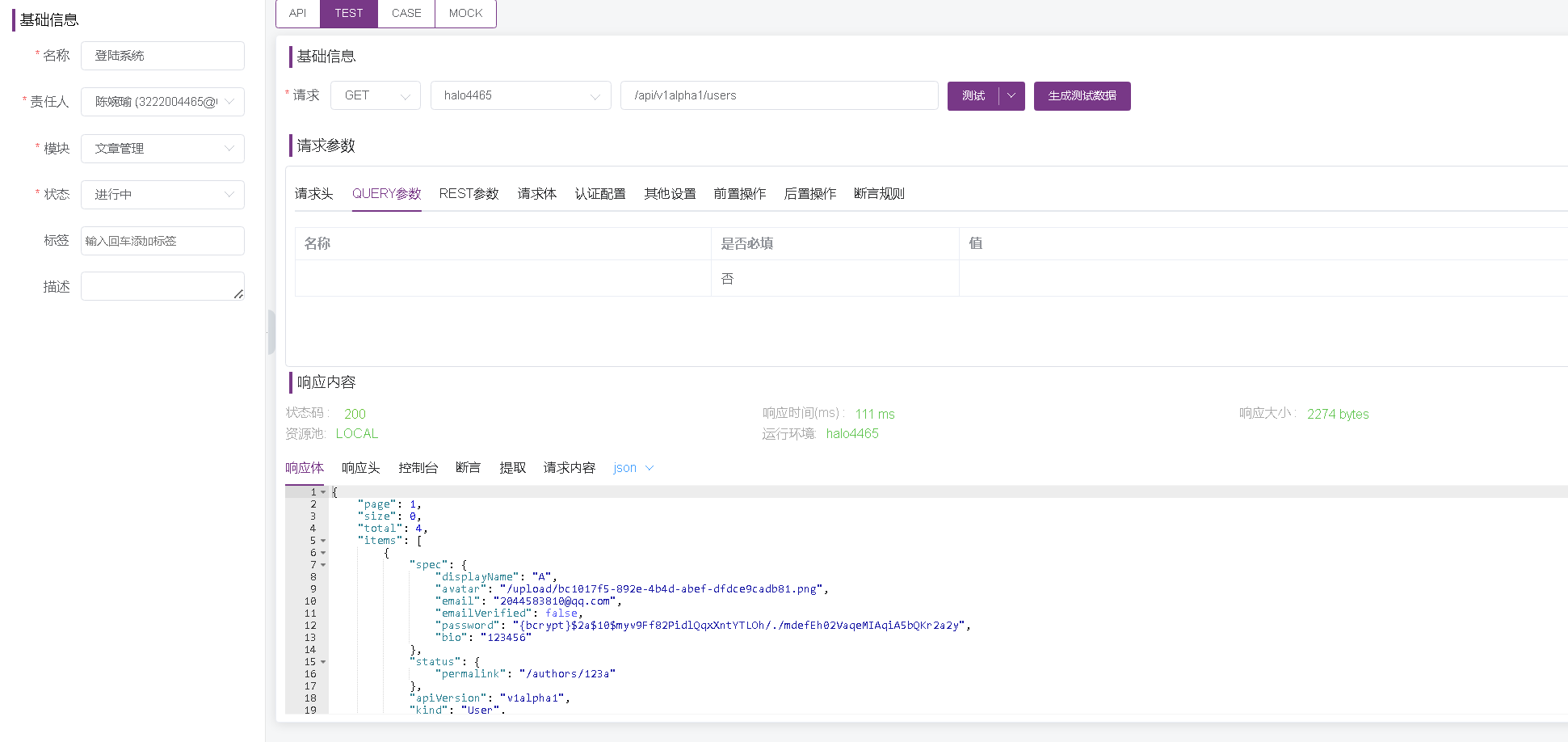


图3-4-1登录接口调试

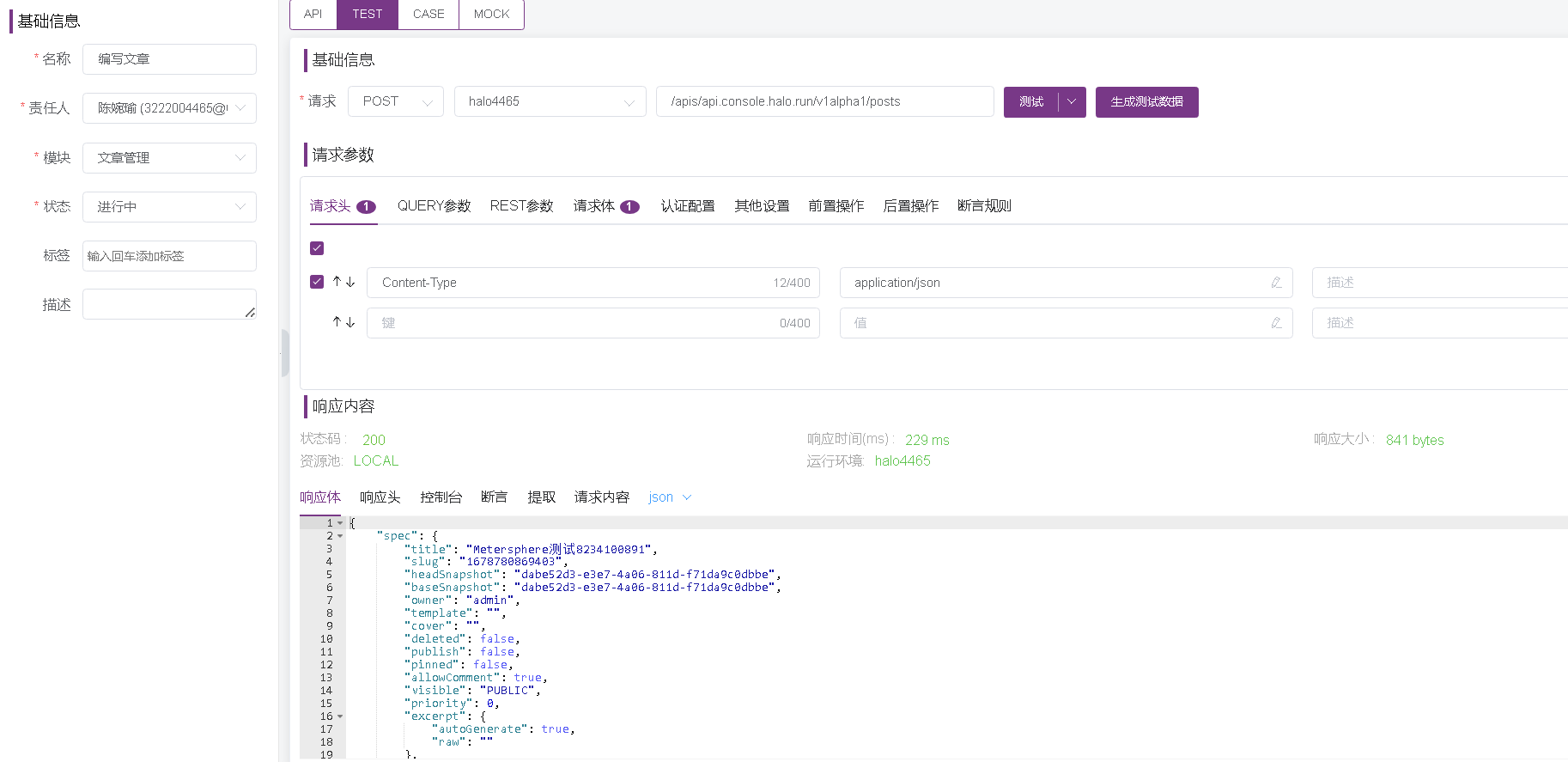


图3-4-2编写文章接口调试

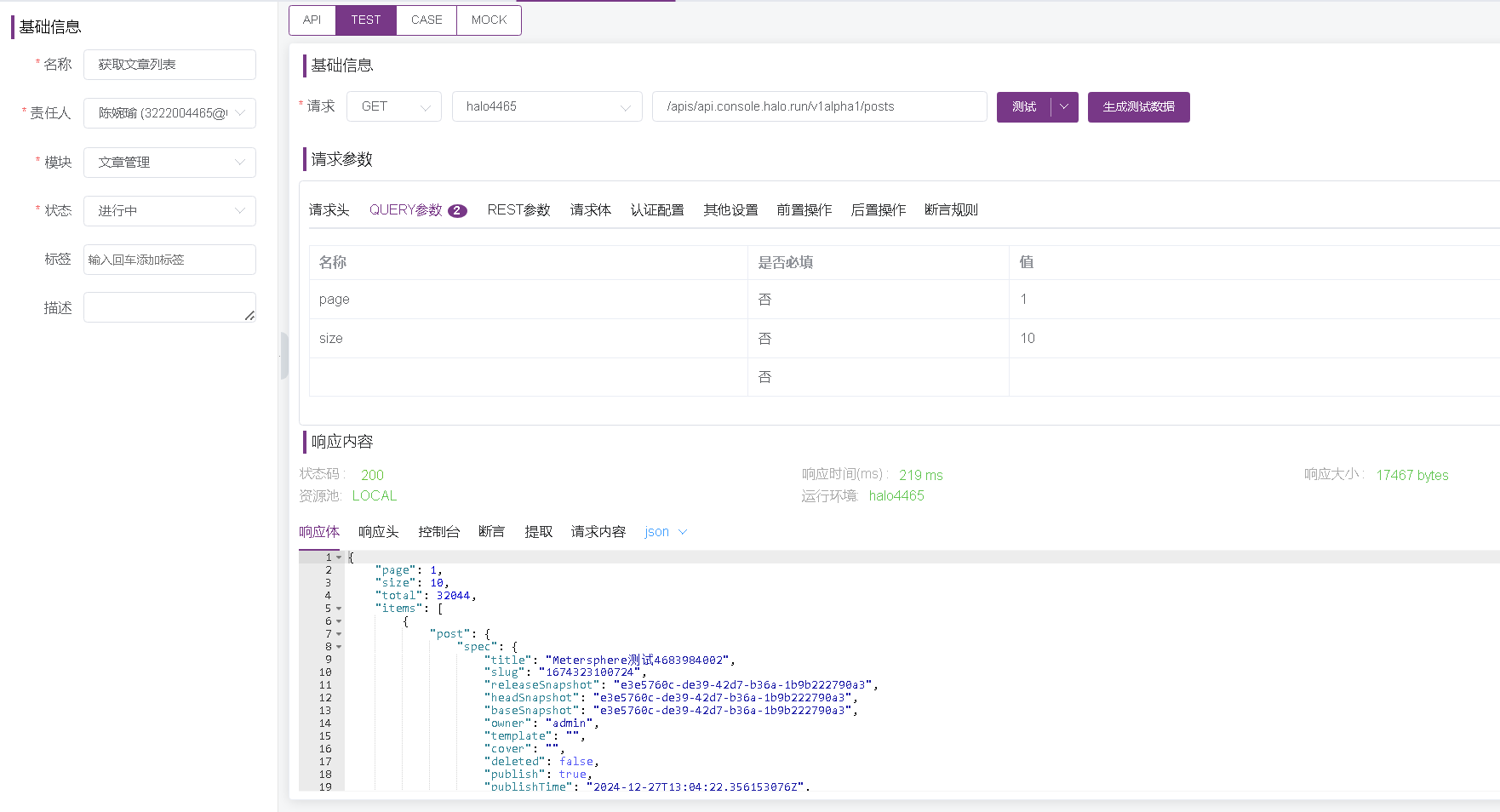


图3-4-3获取文章列表接口调试

**3.5接口用例**

针对登录接口设计测试用例：

首先对登录接口进行分析

1. **输入参数**：登陆必填参数用户名ussername以及密码passworld
2. **输出参数**：验证输入参数不同情况下接口返回的状态码：200/400/401
3. **前提条件**：网络状态正常
4. **测试步骤**：测试用例需要明确说明如何执行测试。这包括调用接口、传递参数、设置环境和验证输出等步骤。
5. **预期结果**：测试用例需要描述预期的输出结果。这有助于确保测试用例的正确性和准确性，并帮助开发人员识别和修复问题。

以下为主要对登陆的正常场景、必填字段是否有传参异常场景的测试用例：



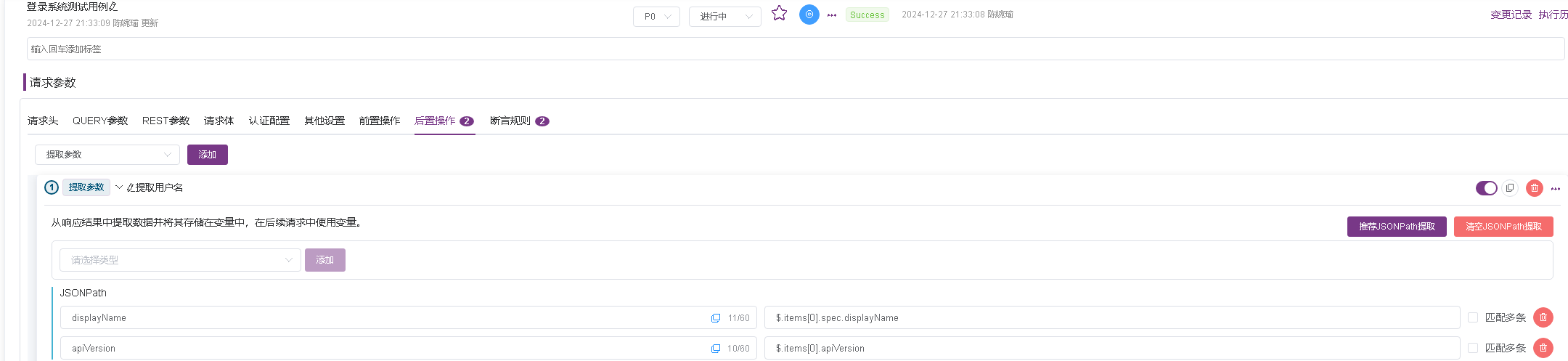
图3-5-1登录接口测试用例

**编写登录接口测试用例：**

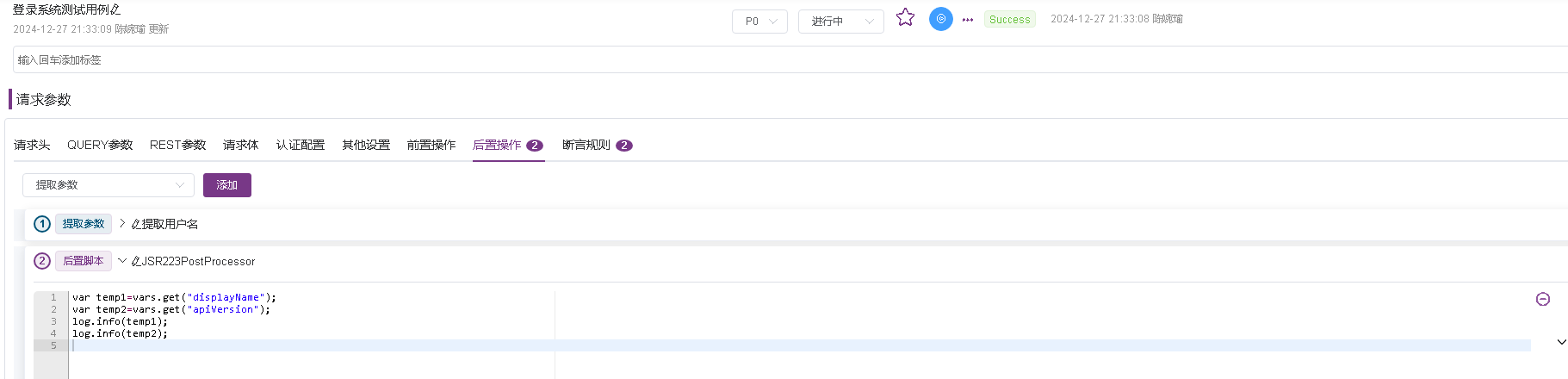
1**.**新建测试用例



2.添加提取参数：



3.添加后置脚本



4.添加断言



### 4.实验结果



图4-1：接口列表



图4-2：接口用例列表



图4-3：登录接口用例响应结果

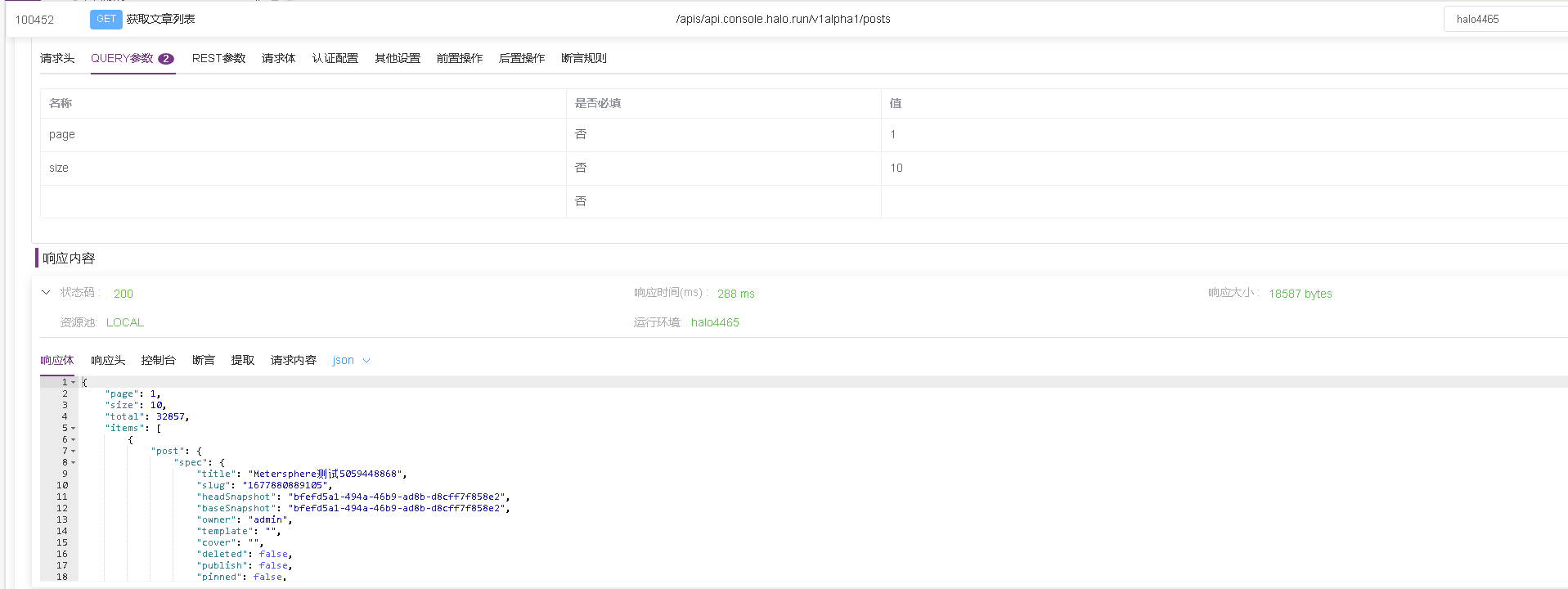


图4-4：获取文章列表接口用例响应结果

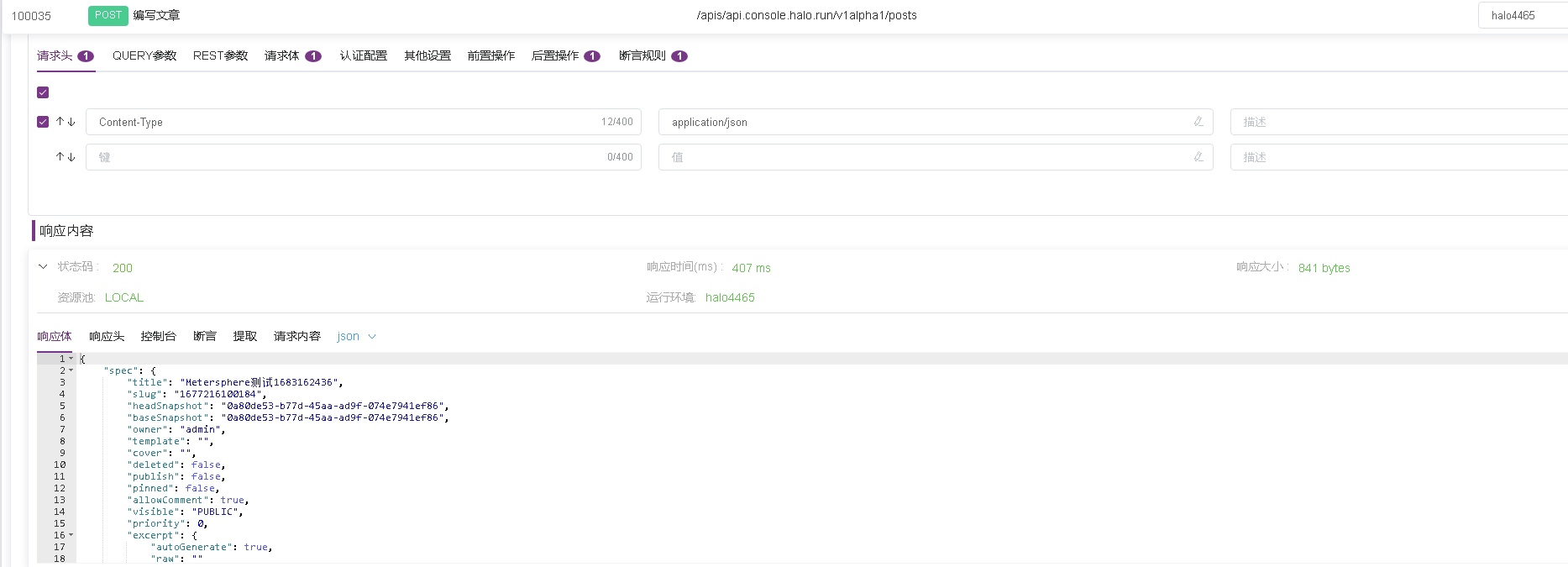


图4-5：编写文章接口用例响应结果图

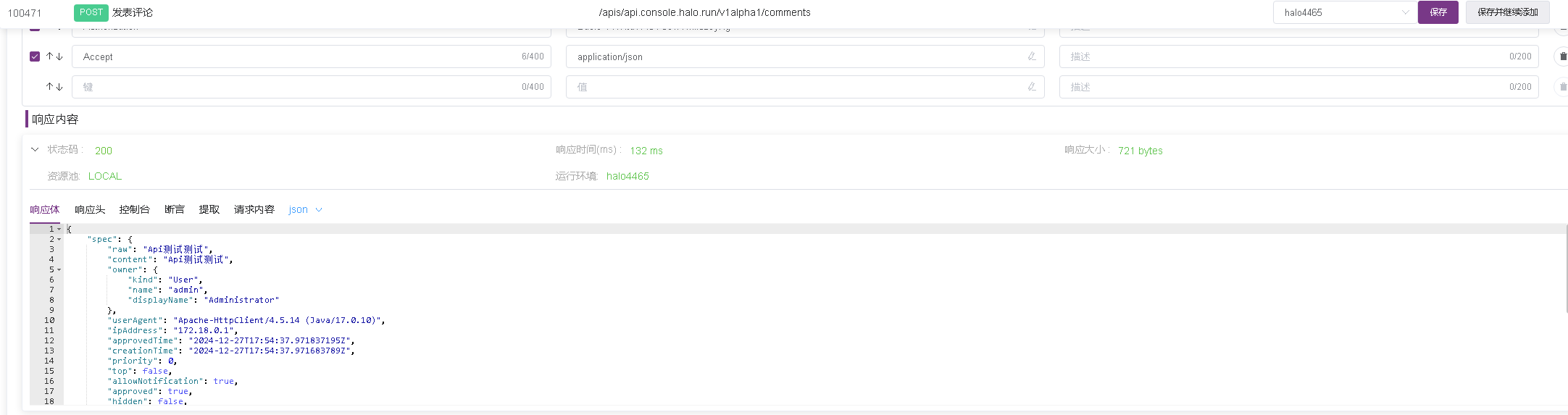


图4-6：发表评论接口用例响应结果

##### 5.实验总结

通过本次实验，我掌握了接口定义的创建流程与调试，学会从接口需求出发，依据接口文档规范，确定接口的名称、请求方式、参数列表、返回值类型等核心要素，构建清晰准确的接口定义。同时对 GET、POST 方法有了更精准的理解，也认识到了接口测试的重要作用：

1. 提高软件质量：接口是软件系统的重要组成部分，通过对接口进行全面、准确的测试，可以发现和修复潜在的漏洞和错误，提高软件的质量和稳定性。

2. 降低开发成本：通过接口测试，可以及早发现和解决问题，减少了后期修复的成本。同时，接口测试还可以帮助开发人员快速定位和修复问题，提高开发效率。

3. 提升用户体验：接口测试可以确保接口的正确性和稳定性，减少系统出错的可能性，提升用户的使用体验。

4. 支持持续集成和部署：接口测试可以与持续集成和部署流程结合，确保每次代码提交后都能进行自动化接口测试，保证系统的稳定性和一致性。这对于大型项目和团队协作非常重要。

总之，接口测试的使用成效和价值主要体现在提高软件质量、降低开发成本、提升用户体验和支持持续集成和部署等方面，对于保证软件系统的稳定性和可靠性具有重要意义。

## 实验四：接口自动化测试

### 实验目的

1. 理解接口自动化测试的使用场景
2. 掌握接口自动化测试的操作过程
3. 掌握接口自动化测试报告的问题反馈

### 实验内容

1. 完成对 Halo 系统文章成功发布的业务流程设计。
2. 使用 MeterSphere 专业测试云进行接口自动化测试。
3. 通过对接口自动化测试报告的分析，发现系统存在的问题。

### 实验步骤

此次实验“文章成功发布”场景的流程如下：

用户登录→编写文章→发布文章→获取文章列表→回收文章→删除回收站文章

在这个过程中，需要创建以下接口，登录接口、发布文章接口、获取文章列表接口、回收文章接口、删除回收站文章接口

以下过程为接口的创建以及接口自动化的实现：

1、创建登录接口



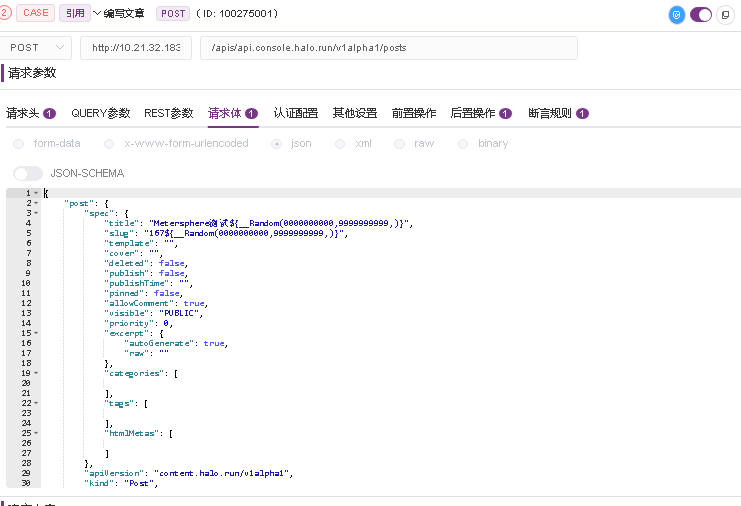
图3-1接口自动化-创建登录接口

1. 创建编写文章接口（接口中包括请求头，请求体，后置操作，断言规则）

编写文章接口的请求体是用 JSON 格式的方式，该接口请求体中的 slug 和 name 参数必须唯一，因此使用 JMeter 工具的函数小助手进行了参数化的设置。

该接口执行后，会生成唯一的文章 name 字段，并且作为参数进行后续发布文章接口的传递，因此需要在测试用例的后置操作进行提取。提取参数的类型为 JSONPath。







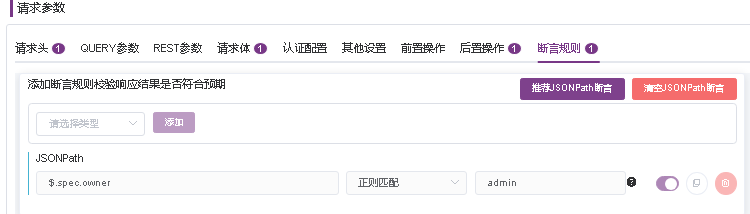


图3-2接口自动化-创建文章编写接口

1. 创建发布文章接口



图3-3接口自动化-创建发布文章接口

4、创建获取文章列表接口



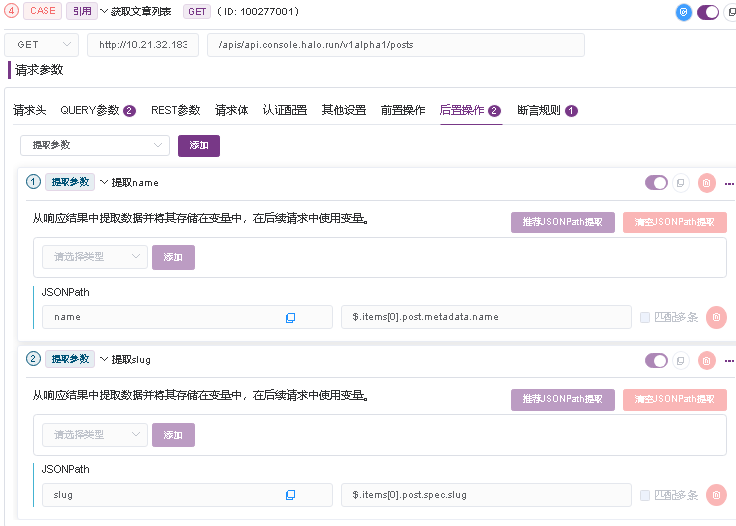


图3-4接口自动化-创建发布文章接口

5、创建回收文章接口







图3-5接口自动化-创建回收文章接口

6、创建删除回收站文章接口





图3-6接口自动化-创建删除回收站文章接口

### 实验结果



图3-8-接口自动化-文章发布场景

1.此步骤为登录接口运行成功截图。



图3-9-接口自动化-登录接口运行成功

2.此步骤为编写文章接口运行成功截图。



图3-10-接口自动化-编写文章接口运行成功

3.此步骤为发布文章接口运行成功截图。



图3-11-接口自动化-发布文章接口运行成功

4.此步骤为获取文章列表接口运行成功截图。



图3-12-接口自动化-获取文章列表接口运行成功

5.此步骤为回收文章接口运行成功截图。



图3-13-接口自动化-回收文章接口运行成功

6. 此步骤为清空回收站文章接口自动化运行成功截图



图3-14-接口自动化-清空回收站文章接口自动化运行成功

7. 此步骤为接口自动化运行成功截图

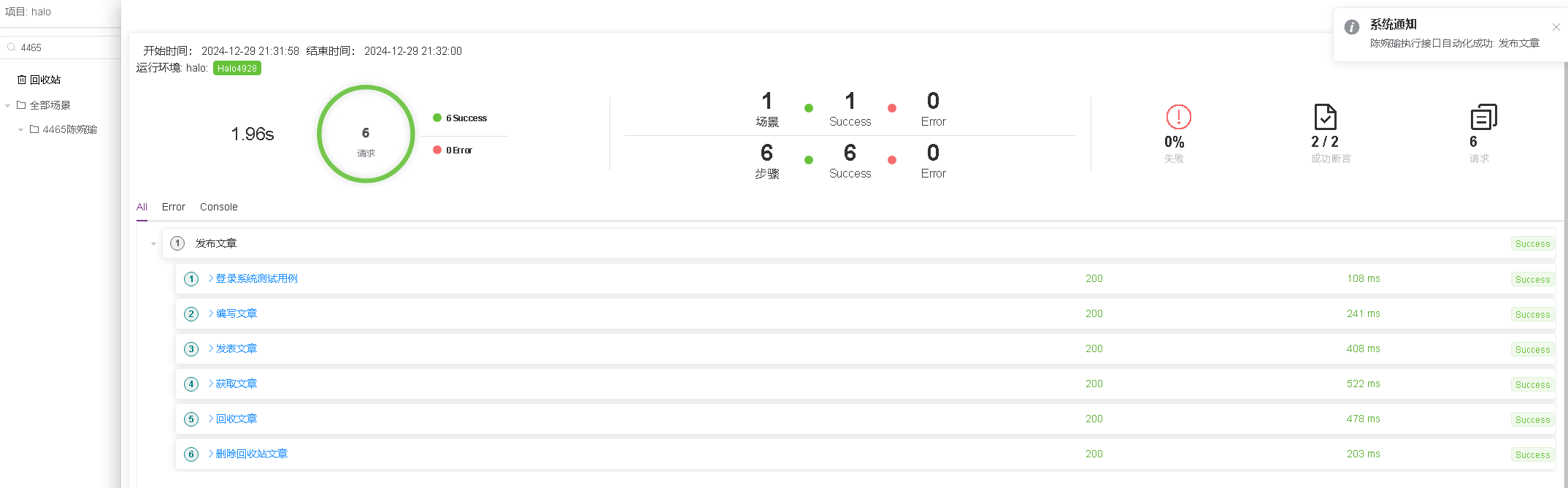


图3-14-接口自动化-接口自动化运行成功

### 实验总结

经过本次实验，熟悉了自动化接口测试的流程，清晰认识到接口自动化测试在现代软件开发周期中的不可或缺性。以 Halo 系统为例，随着系统频繁更新迭代，文章发布功能作为核心模块，每一次变更都涉及大量的回归测试工作。手工测试不仅效率低下，且容易因人为疏忽遗漏关键场景，而自动化测试能够按照预设脚本快速、精确且重复性地执行测试流程，无论是在开发阶段的频繁自测，还是上线前的全面回归，都能有效节省人力成本、缩短测试周期，确保系统稳定性，切实感受到其在保障软件质量、提升交付效率方面的关键支撑作用。

## 实验五：性能测试

### 实验目的

1. 理解 MeterSphere 系统性能测试原理
2. 熟悉 MeterSphere 系统性能测试基本流程
3. 掌握 MeterSphere 系统三种性能测试方式
4. 掌握性能测试系统调优

### 实验内容

（1）完成如下 Halo 接口对应的接口用例的性能测试，查看并分析接口性能测试报告。

表5-2-1接口用例

|  |  |  |
| --- | --- | --- |
| 编号 | 接口用例 | 备注 |
| 1 | 登录Halo控制台 | 接口测试实验生成的接口用例 |
| 2 | 编写文章 |
| 3 | 获取用例列表 |

（2）完成 “发布 Halo 文章” 接口自动化场景的性能测试，测试 “用户登录→编写文章→发布文章→获取文章列表” 多个接口组成的业务场景性能。

表5-2-2发布 Halo 文章接口自动化场景接口组成

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 步骤编号 | 接口 | 接口用例 | 复制/引用 | 备注 |
| 1 |  | 登录Halo控制台 | 引用 | 接口自动化测试实验生成的接口自动化场景。 |
| 2 |  | 编写文章 | 引用 |
| 3 | 发布文章 |  | 复制 |
| 4 |  | 获取用例列表 | 引用 |

### 实验步骤

**3.1接口用例转性能测试**

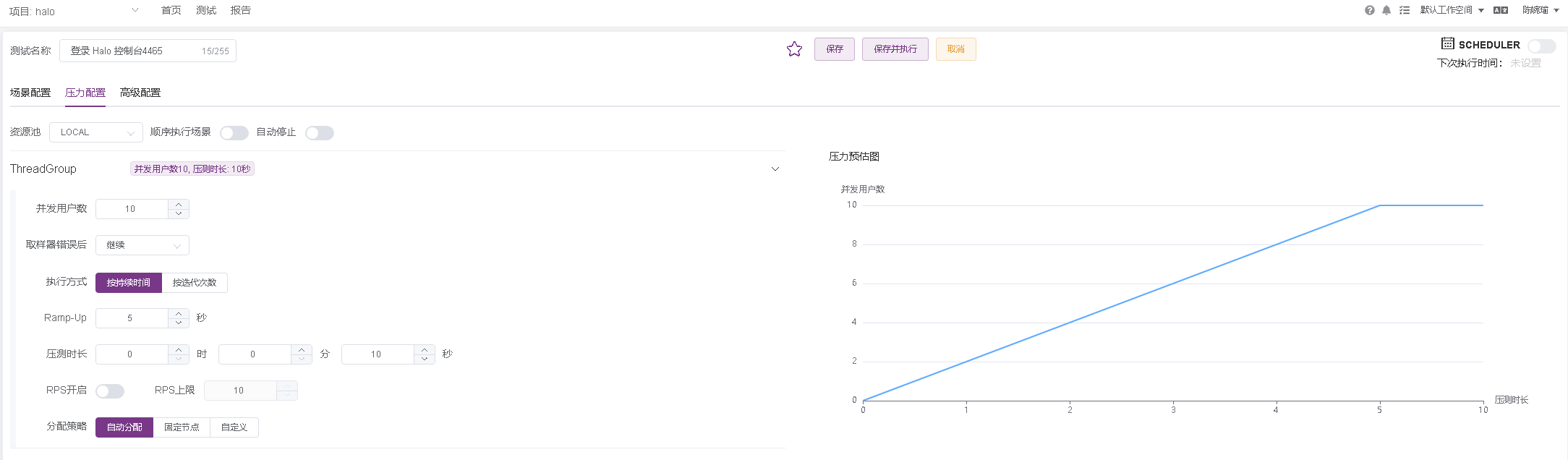


图3-1-1登陆halo控制台用例-性能测试任务

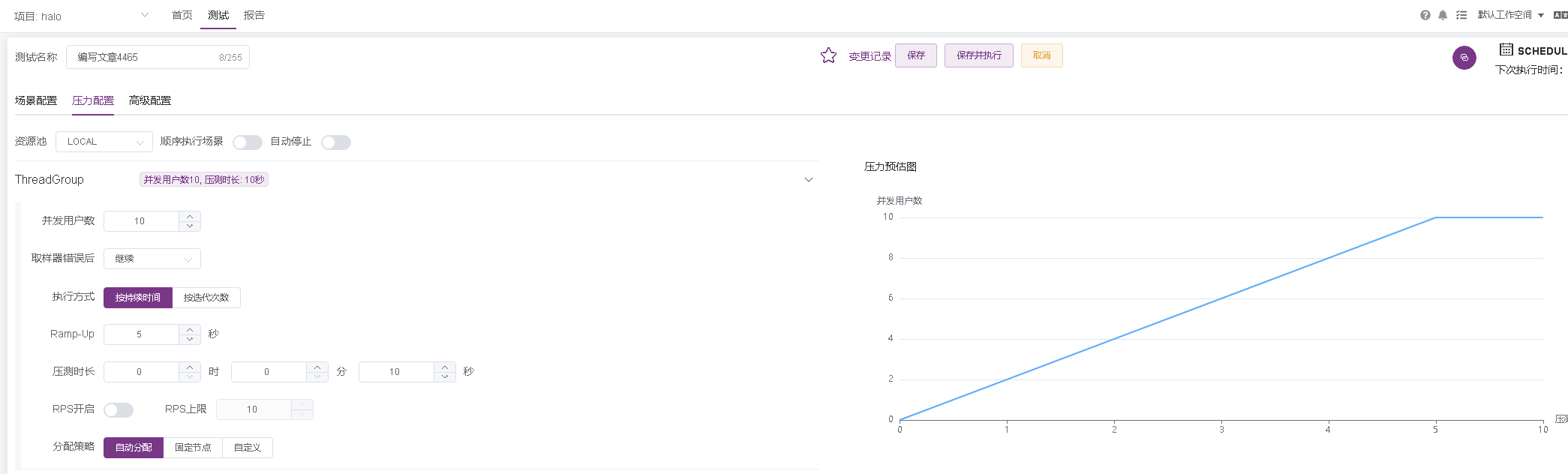


图3-1-2编写文章用例-性能测试任务

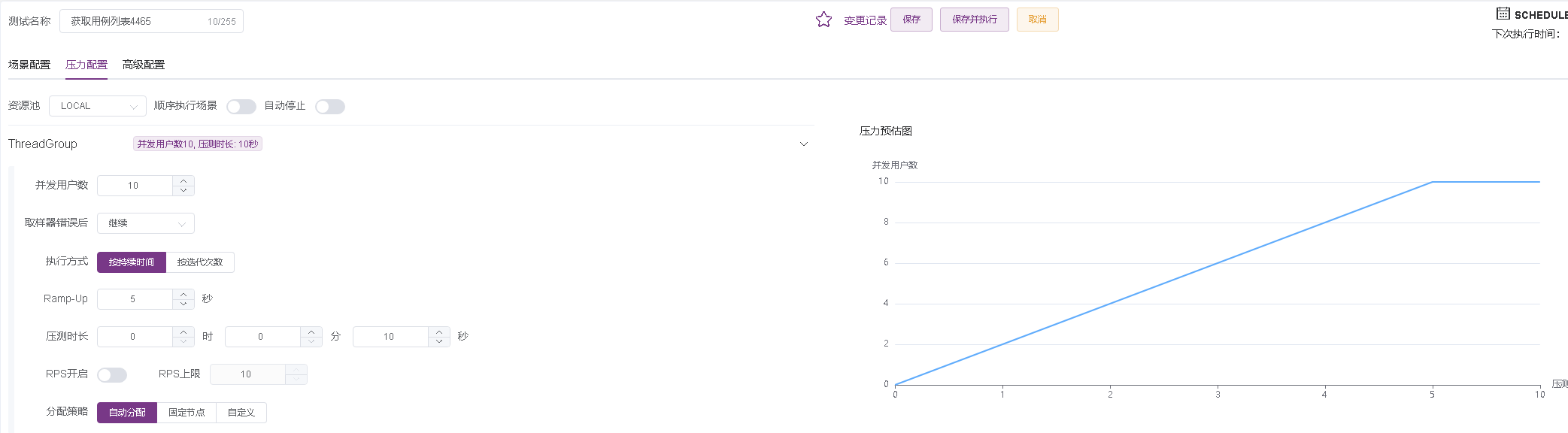


图3-1-3获取用例列表用例-性能测试任务

**3.2接口自动化测场景测试转性能测试**

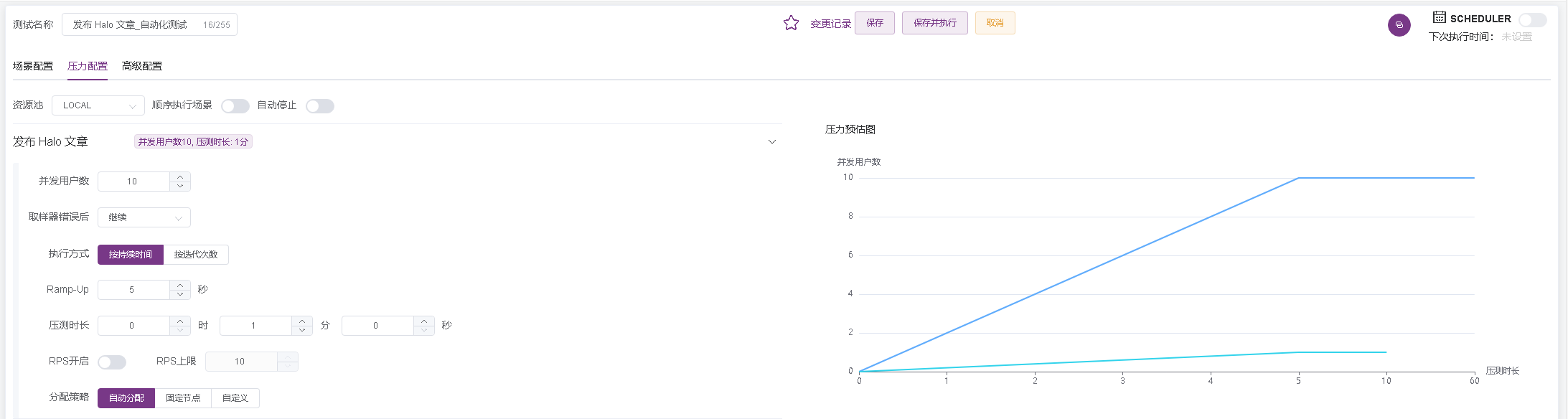
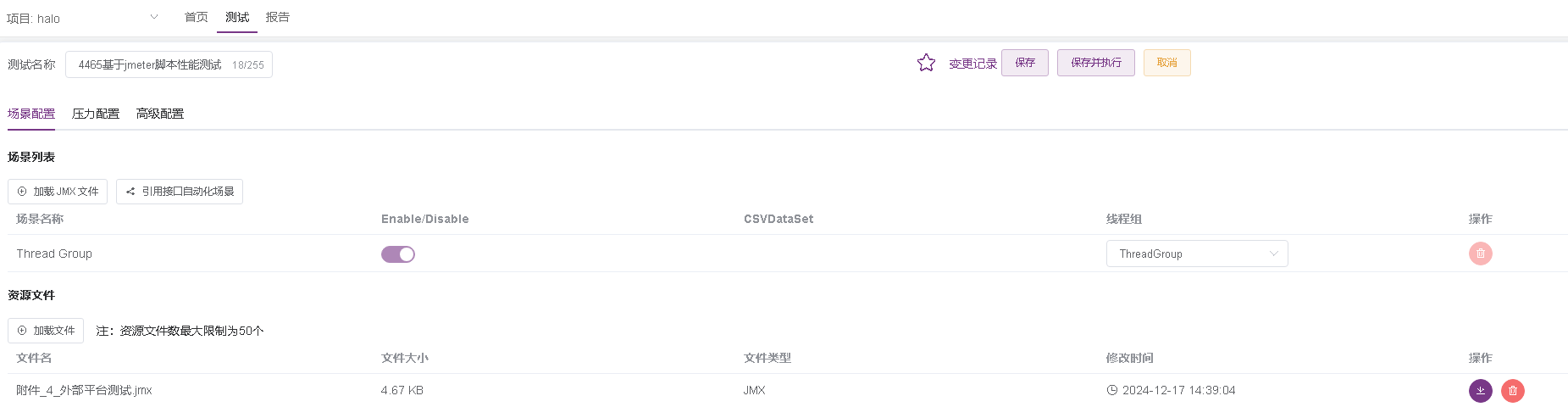


图3-2-1发布文章场景接口自动化-性能测试任务

**3.3基于 JMeter 脚本新建性能测试**



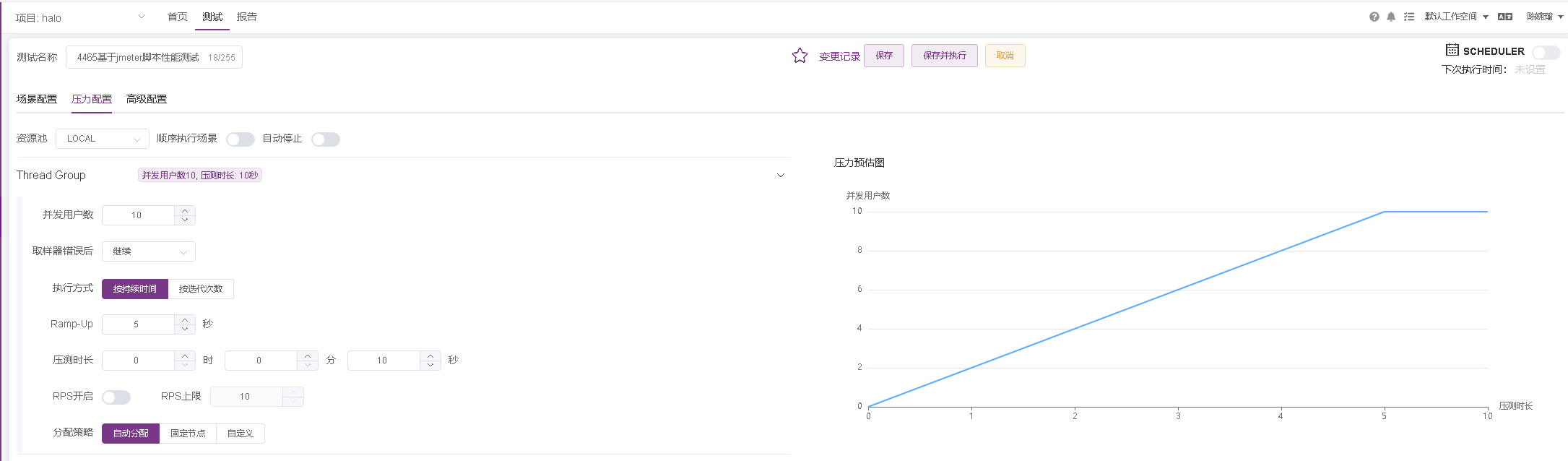


图3-3-1发布文章场景-基于JMeter 脚本性能测试任务

### 四、实验结果

1、登录halo控制台接口性能测试



图4-1登录halo控制台接口-性能测试报告

从图表中可以看出，当负载达到 20 左右时，事务数的增长开始变缓，而响应时间开始增加。这表明系统在负载为 20 左右时可能开始出现性能瓶颈。

2、编写文章接口性能测试

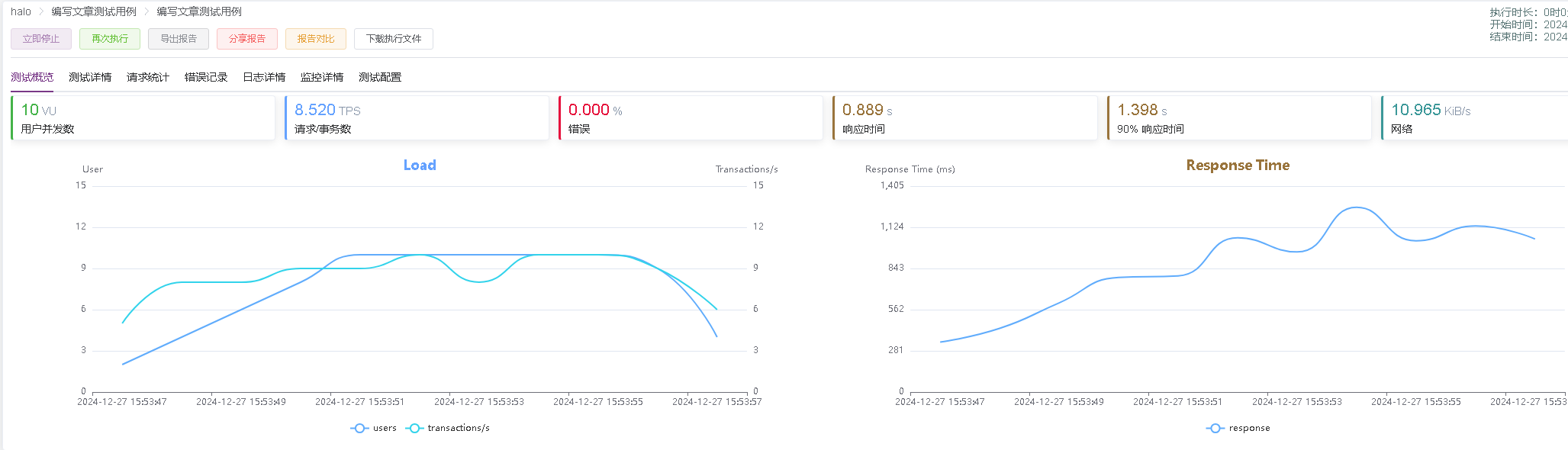


图4-2编写文章接口-性能测试报告

从图表中可以看出，当负载达到 5 左右时，事务数的增长开始变缓，而响应时间开始增加。这表明系统在负载为 5 左右时可能开始出现性能瓶颈

3、获取用例列表接口性能测试

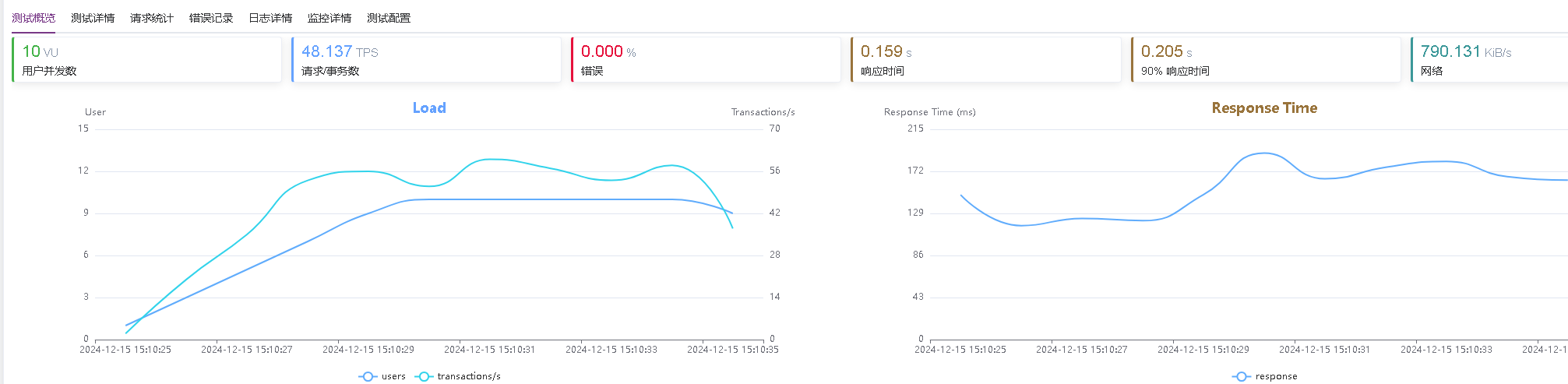


图4-3获取用例列表接口-性能测试报告

从图表中可以看出，当用户并发数达到 10 时，每秒事务数和响应时间趋于稳定。这表明系统在用户并发数为 10 时可能达到了性能瓶颈。

4、发布文章场景性能测试（JMeter 脚本创建）

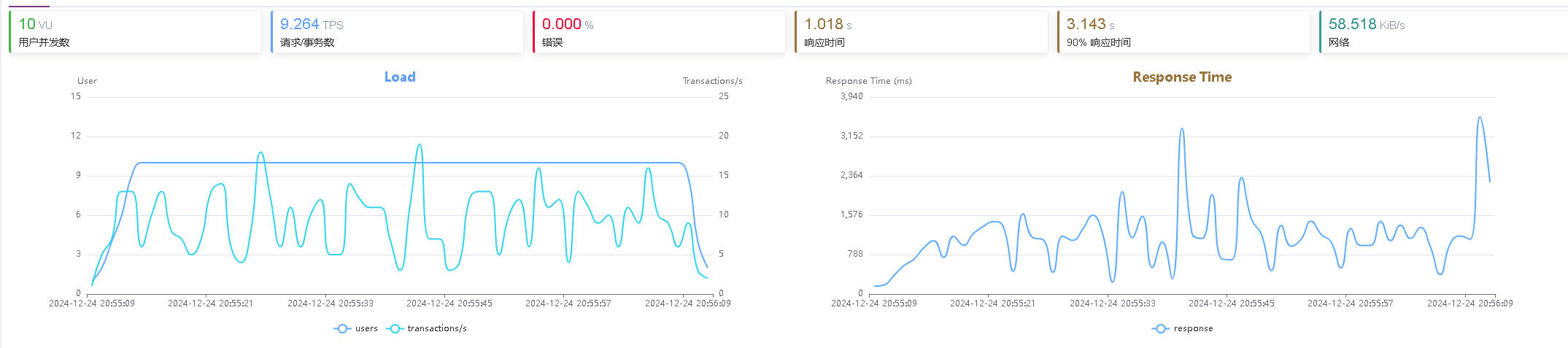


图4-4发布文章场景-性能测试报告

从图表可知每秒请求数为 9.264 TPS，错误率为 0.000%，网络吞吐量为 58.518 KiB/s。响应时间波动较大，有时达到 3 秒多，这可能是系统的性能瓶颈所在。

### 实验总结

针对当前实验结果进行总结，对被测试系统提出优化建议。

1.总结

实验流程：通过MeterSphere成功地对halo的接口进行了性能测试。

测试方法：使用ConcurrencyThreadGroup进行并发测试，通过阶梯式加压策略，逐步增加并发用户数，以找出接口的性能拐点。

测试结果：虽然部分接口在低负载情况下性能表现尚可，但在高并发场景下仍存在诸多性能问题亟待解决，发布文章场景的性能测试中响应时间波动较大。

2.优化建议

性能调优：

针对数据库：检查数据库查询优化，是否有可以改进的索引、查询语句或数据库配置。

针对代码：分析代码逻辑，查看是否有不必要的计算或数据库操作，减少接口响应时间。

缓存策略：考虑使用缓存技术，如Redis或Memcached，来缓存常用查询结果，减少数据库压力。

负载均衡：如果系统部署在单一服务器上，可以考虑部署多个实例，并使用负载均衡器进行分发，提高整体吞吐量。

监控与日志：

加强系统监控，实时观察接口响应时间、服务器负载等指标，以便及时发现问题并进行处理。

完善日志记录，记录接口请求、异常等信息，方便后续问题排查。

压力测试与容量规划：

定期进行压力测试和容量规划，确保系统能够应对预期的用户量增长和突发流量。

根据测试结果进行针对性的优化和扩容。

用户体验优化：

在接口性能下降时，考虑向用户展示友好的提示信息，避免用户因等待时间过长而产生不满。

对于高频次、高并发的操作，考虑使用异步处理、消息队列等技术来优化用户体验。

安全性加固：

在进行性能优化的同时，不要忽视系统的安全性。确保所有安全措施都得到妥善实施，如输入验证、访问控制等。

定期进行安全审计和漏洞扫描，确保系统没有已知的安全漏洞。

本次实验主要基于MeterSphere 平台对halo系统进行性能测试。通过实践，深刻理解了性能测试原理，熟悉了从测试计划制定到结果分析的基本流程。

在实验中，掌握了 MeterSphere 的三种性能测试方式，包括不同负载场景下的测试操作。观察到随着负载增加，系统事务处理能力和响应时间的变化，找到了可能的性能瓶颈点。同时，也学习了性能测试系统调优相关知识，了解到可从系统架构、数据库、应用程序等多方面进行优化，为今后提升系统性能奠定了坚实基础。掌握了性能测试的流程，并学会分析性能测试报告并提出优化建议。

**实验六 测试计划综合实验**

### 实验目的

1. 熟悉 MeterSphere 测试计划的设计。
2. 掌握 MeterSphere 测试计划的使用。
3. 掌握 MeterSphere 测试计划报告的查看和分析。

### 实验内容

1. 创建 MeterSphere 测试计划。（前面已做，这里可以不做）
2. 关联功能测试用例到测试计划。
3. 关联接口测试用例到测试计划。
4. 关联接口自动化测试场景到测试计划。
5. 关联性能测试任务到测试计划。
6. 执行功能测试、接口测试、和性能测试。
7. 查看并分析测试报告。

### 实验步骤

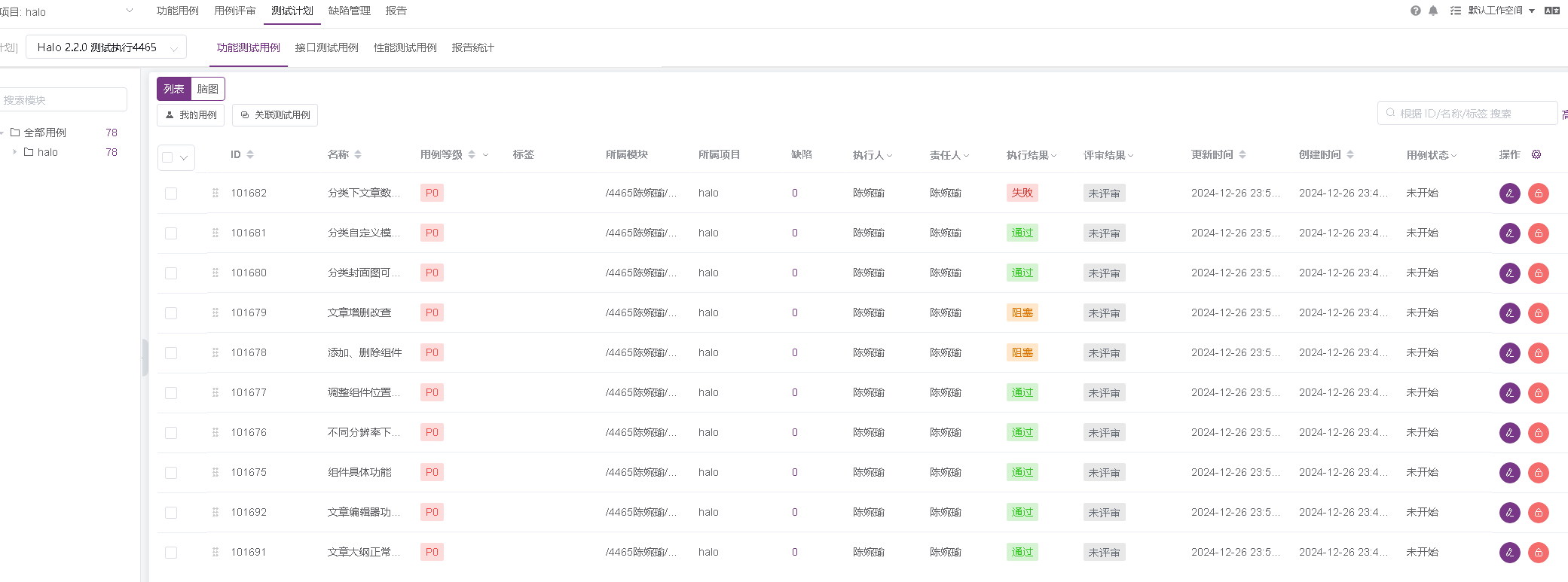


图3-1测试计划（功能测试）



图3-2测试计划（接口测试）



图3-3测试计划（接口场景测试）



图3-4测试计划（性能测试）

### 实验结果

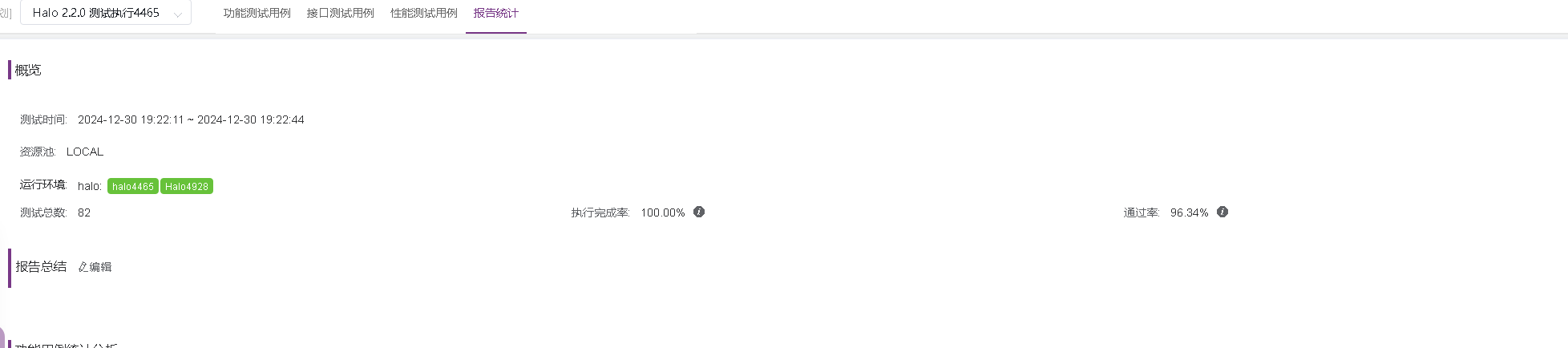


图4-1测试报告概览

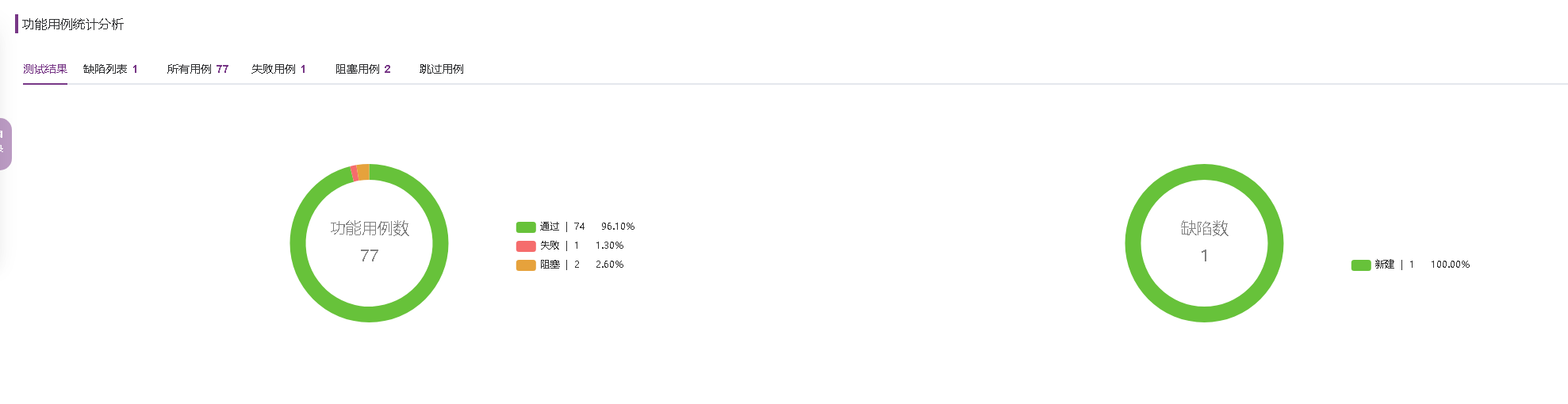


图4-2功能用例统计分析



图4-3接口用例统计分析

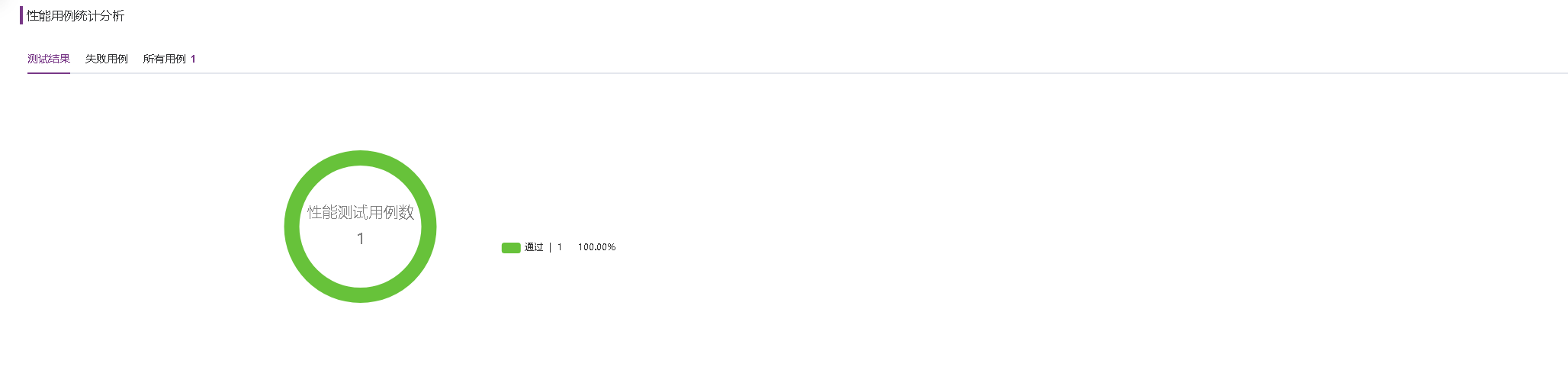


图4-2性能用例统计分析

### 实验总结

最终测试通过率96.34%，功能测试中提交bug\*1。但在创建接口用例调试过程中曾出现错误，最终通过查看接口响应状态码、排查日志都一一解决了。以下是此次实验积累的排查错误心得：

一、接口测试常见错误情况及排查思路：

（1）常见状态码及含义：

1.400 Bad Request：通常表示客户端发送的请求有语法错误，例如请求参数格式不正确、缺少必要参数等。排查时需仔细核对请求参数的类型、数量以及格式是否与接口文档要求一致，检查接口测试用例中构造请求的逻辑是否正确。

2.401 Unauthorized：意味着请求需要用户认证，但认证信息未提供或者不正确。要查看请求头中是否添加了正确的认证信息（如 token 等），确认认证机制（例如 OAuth、Basic Auth 等）的配置是否准确，是否过期等情况。

3.403 Forbidden：表明服务器理解请求客户端的请求，但是拒绝执行此请求。可能是客户端没有足够的权限访问该接口资源，需要检查用户角色权限的配置，确认测试账号所具备的权限范围是否涵盖了当前请求的接口操作。

4.404 Not Found：说明服务器无法根据客户端的请求找到对应的资源。需确认请求的 URL 是否正确，是否存在拼写错误或者路径配置错误等情况，同时检查接口所对应的后端服务路由是否正确映射。

5.500 Internal Server Error：这是服务器内部错误，一般是服务端代码出现了异常，比如业务逻辑处理出错、数据库操作失败等。此时需要结合响应体中的报错信息（如果有），联合开发人员排查服务端代码逻辑、数据库连接以及相关配置是否存在问题。

（2）接口性能相关问题（响应时间过长）

排查思路：查看接口请求的开始时间和结束时间记录在日志中的时间戳，计算出响应时间。若响应时间超出了性能指标要求，首先检查网络状况，是否存在网络延迟、带宽不足等问题。同时分析接口内部的逻辑，比如是否存在复杂的数据库查询、大量的数据处理或者循环操作等导致性能瓶颈。可以借助性能分析工具（如 Jmeter 中的性能分析插件等）来进一步定位具体的代码执行缓慢的位置。

二、性能测试常见错误情况及排查思路

（1）响应时间过长（整体系统响应缓慢）

排查思路：结合接口测试中排查响应时间过长的方法，从网络、数据库、代码逻辑等多个层面进行分析。此外，在性能测试场景下，还要考虑并发压力对系统的影响。查看在高并发情况下，各个组件（如应用服务器、数据库服务器等）的资源利用率情况，如果 CPU 使用率过高接近 100%，可能是存在 CPU 密集型的业务操作，需要优化算法或者进行服务器硬件升级；如果内存使用率持续攀升，可能存在内存泄漏问题，需检查代码中是否存在对象未及时释放等情况；对于网络带宽占用过高，要分析是否有大量不必要的数据传输或者可以对传输的数据进行压缩优化等。

（2）吞吐量未达到预期

排查思路：吞吐量是指单位时间内系统处理的请求数量，若未达到预期，首先检查测试环境的配置是否合理，比如服务器的硬件资源是否限制了系统的处理能力，网络带宽是否足够支持大量请求的传输。分析应用程序内部的业务逻辑，是否存在同步阻塞、线程池配置不合理等情况导致请求无法及时被处理。对比不同负载水平下的吞吐量变化情况，判断系统的性能扩展能力以及是否存在性能瓶颈点，如在某个并发用户数之后吞吐量不再增长甚至下降，要重点排查此时对应的组件资源使用情况以及代码执行逻辑。

（3）资源利用率异常（过高或过低）

排查思路：对于资源利用率过高的情况，如前面提到的 CPU、内存等资源使用过度，按照相应资源对应的排查方法去定位问题，寻找代码中的性能优化点。而资源利用率过低也可能存在问题，比如服务器配置了过高的硬件资源但实际应用程序并没有充分利用，可能是应用程序的架构设计不合理，无法充分发挥服务器的性能优势，或者存在不必要的资源限制（如线程池设置过小等），需要对应用架构和相关配置进行评估和调整。

# 实验总结

在完成本次软件测试实验后，我对软件测试这一重要环节有了更为深入且全面的认识，学会了设计测试用例、掌握了功能测试、接口测试、接口自动化、性能测试流程，收获了满满的知识与实践经验。

本次实验以halo作为测试对象，它涵盖了用户管理、文章管理等功能模块。此次实验开始我先制定了测试计划，规划好测试范围、测试策略以及各阶段的时间安排，为后续工作有序开展奠定基础。接着，依据计划运用多种方法设计测试用例。功能测试中，通过等价类划分、边界值分析等技巧，全面考量各种可能的输入情况，确保能精准检测功能是否符合预期。执行测试时，我利用MeterSphere平台，严谨记录每一项测试结果。一旦出现测试不通过的情况，便仔细查看测试日志、精准定位缺陷，分析其产生原因并判定严重程度和优先级，做好详细记录。

通过这次实验，我熟练掌握了软件测试的基本流程与常用方法，明白了测试在保障软件质量方面的关键作用。当然，实验中也遇到过如测试环境不稳定、部分复杂功能用例设计不完善等挑战，但都在不断探索中得以克服。未来，我会继续精进软件测试技能，为软件质量把关贡献更多力量。