

河南工业大学

课 程 设 计

课程设计名称:

大数据综合应用开发实训

专 业 班 级:

数据科学与大数据技术 2106 班

学 生 姓 名:

学 号:

指 导 教 师:

李明旭

课程设计时间:

2024.05.06-2024.06.28

数据科学与大数据技术 专业课程设计任务书

学生姓名		专业班级	大数据 2106	学 号	
题 目	基于宝塔面板、GPT 网站后台和员工管理系统的测试报告				
课题性质	工程设计		课题来源	自拟课题	
指导教师	李明旭		同组姓名		
主要内容	<p>一、数据 GPT 网站后台数据库、公司架构和员工数据库。</p> <p>二、基本功能</p> <ol style="list-style-type: none">1. 使用宝塔面板完成基本的功能测试内容。2. 使用接口自动化测试的方法创建数据库,并继续用接口完成 SpringBoot 项目员工管理系统的部署和 Docker 项目 GPT 网站后台的部署。3. 使用功能测试和 UI 自动化测试的方法来对 GPT 网站后台的测试。4. 使用功能测试和 UI 自动化测试来实现对员工管理系统的测试。5. 使用性能测试方法分别对这三个平台进行性能测试。 <p>三、扩展功能</p> <ol style="list-style-type: none">1. 能够使用掌握 FTP 的相关测试方法,通过 FTP 接口进行文件的批量自动化上传。2. 能够掌握远程服务器的系统监控接口方法,实现在性能测试时的监控。				
任务要求	<ol style="list-style-type: none">1、能够根据用户需求确定设计目标。2、按照系统测试设计思想,结合软件系统测试流程,利用相关测试工具,能够系统分析所面对的工程问题,并进行概要设计和详细设计。3、能够按照设计要求进行系统开发项目的软件测试和评价。4、能够在设计过程中撰写规范的设计报告,在设计完成后通过答辩。				
参考文献	<p>[1] 李正言.计算机软件测试方法的研究[J].自动化应用,2024,65(02):199-201.D OI:10.19769/j.zdhy.2024.02.062.</p> <p>[2] 范飞燕.浅谈基于模块化的自动化软件测试框架技术[J].长江信息通信,2023,36(11):102-104.</p> <p>[3] 仇振安,王锋,任志伟.软件测试的难点及应对策略[J].电光与控制,2023,30(09):106-111.</p> <p>[4] 林生旭,盘茂杰.软件测试技术及其测试工具的研究与应用[J].现代计算机,2023,29(12):37-43.</p> <p>[5] Memon P,Bhatti S,Dewani A, et al.Scrutinizing Automated Load Testing Via Blazmeter, Apache Jmeter and HP Loadrunner[J]. International Journal of Advanced Studies in Computer Science and Engineering,2018,7(3TB):35-42.</p>				
审查意见	<p>同意。</p> <p>指导教师签字: 李明旭</p> <p>教研室主任签字: 苗建雨</p> <p>2024 年 4 月 22 日</p>				

目 录

1	概述	1
2	测试计划	2
2.1	功能测试	2
2.2	接口自动化测试	3
2.3	UI 自动化测试	3
2.4	性能测试	4
3	测试方案	4
3.1	功能测试方案	4
3.2	接口测试方案	5
3.3	UI 测试方案	6
3.4	性能测试方案	6
4	功能测试	7
4.1	用例设计	7
4.2	测试执行	8
5	接口自动化测试	13
5.1	用例设计	13
5.2	测试执行	21
6	UI 自动化测试	24
6.1	用例设计	24
6.2	测试执行	34
7	性能测试	35
7.1	用例设计	35
7.2	测试执行	37
8	缺陷记录	43
9	测试报告	45
10	总结	47
11	参考文献	48

1 概述

测试是通过人工或自动化方式运行软件系统，以验证其是否满足既定需求或预期结果。这一过程不仅在确保软件质量方面扮演着关键角色，同时也为整个开发生命周期提供了重要保障。测试的发展历史悠久，大致可以分为以下几个主要阶段：

探索阶段：早期探索阶段的测试工作主要由开发人员在编写代码的过程中自行进行。这个阶段的测试目的相对单一，主要是调试和修复已知的故障。虽然这种方式在小规模项目中能够奏效，但随着软件系统复杂度的增加，这种模式逐渐显得捉襟见肘。开发人员在忙于编写新代码的同时，还要兼顾测试工作，往往力不从心，难以全面覆盖所有可能的错误和故障。

专业化阶段：随着软件项目规模和复杂度的提升，测试逐渐从开发工作中独立出来，形成了一个专业化的职业领域。专业的测试人员开始承担起软件测试的重任，他们具备专门的测试技能和方法，能够更加系统和全面地进行测试。这一阶段的转变使得专业的测试人员能够设计更加全面和细致的测试用例，发现隐藏在复杂系统中的各种潜在问题。

标准化阶段：为了应对不断增加的软件复杂性和多样性，国际上开始制定各种测试标准和规范，这一阶段如 ISO/IEC 29119 系列标准相继出台，这些标准为测试方法、流程和工具提供了系统性的指导，确保了测试活动的规范性和一致性，使得不同组织和团队能够在统一的框架下进行高效的测试工作。

自动化阶段：随着计算机技术的进步，自动化测试工具逐渐普及。自动化测试不仅能够显著提高测试的效率，还能减少人为错误，确保测试过程的可重复性和可靠性。自动化测试的应用，使得大规模的回归测试和持续集成成为可能，极大地推动了软件质量的提升。

测试的核心意义在于发现问题，即找到软件的缺陷或不符合预期的地方。但测试不仅仅是“解决问题”的过程，它本身也是一种富有挑战性和趣味性的活动。通过测试发现并解决缺陷，不仅可以增加对软件质量的信心，确保软件在预期环境下的可靠性，还可以积累宝贵的测试经验，提升规避风险的能力，从而提高整体开发效率。

此外，测试还在以下几个方面有重要意义：

确保软件质量：软件测试是确保软件质量的关键步骤。通过测试，可以发现和修复软件中的缺陷，防止缺陷在用户使用产生影响。高质量的软件不仅能提高用户满意度，还能增强市场竞争力。

保障用户体验：通过全面和细致的测试，可以发现并修复可能影响用户体验的问题，确保最终交付的软件产品能够满足用户需求，并提供良好的使用体验。

提升软件安全性：在当前网络安全威胁日益严峻的背景下，测试工作在发现和修复安全漏洞方面具有重要作用。通过安全测试，可以有效防范潜在的安全威胁，保护用户数据和隐私。

降低后期成本：及时发现并修复软件缺陷，可以大幅降低后期维护成本。软件在开发阶段发现缺陷的成本远低于发布后修复缺陷的成本，因此，早期充分的测试能够显著降低总成本。

促进团队合作：测试工作需要开发、测试和运维等多个团队的紧密合作。通过测试活动，团队成员之间可以更好地沟通和协作，共同确保软件项目的成功。

支持持续改进：通过不断的测试和反馈，团队可以持续改进开发和测试流程，提升整体开发效率和软件质量。

测试是软件开发过程中不可或缺的一部分，从手工测试到自动化测试，从测试驱动开发到持续集成和持续交付，每个阶段都见证了技术和方法的进步。测试不仅仅是发现错误，更是确保软件质量、提高用户满意度、降低成本和风险的重要手段。随着软件复杂度和用户需求的不断增加，测试的重要性也将继续上升，推动软件测试技术和方法的进一步发展。

2 测试计划

2.1 功能测试

在测试的项目中，功能测试的主要目的是确保软件系统的功能符合需求和设计规范。这包括验证各功能的正确性，确认需求是否得到满足，发现并记录潜在缺陷，并提升系统的稳定性和用户体验。

测试背景：此次测试旨在验证服务器上多个系统的功能是否正常运行，涵盖所有主要功能，确保系统在各种操作场景下的可靠性和稳定性。

测试范围：测试将覆盖计划任务模块、系统安全模块和管理模块等主要场景。

测试内容请见下文对应章节的测试用例总表。

测试规则：包含前提条件和问题处理流程。

前提条件：所有功能模块已上线并通过单元测试。

问题处理：遇到问题时，测试将暂停；当问题修复并通过验证后，测试结束。

测试方法：通过手工测试，检查每个功能的实际运行情况。

测试要点：确保所有测试用例都已执行，并详细记录每个测试结果。

测试环境：测试服务器配置与生产环境相同，客户端设备为 Windows PC。

软件环境：Windows 10、Linux。

测试环境要求：具备稳定的网络连接，客户端安装必要环境。

测试设计：编写和审核测试用例，并设计相应的测试数据。

测试执行：按照测试用例执行测试，并记录测试结果。

测试总结：分析测试结果，编写并提交测试报告。

2.2 接口自动化测试

接口自动化测试的主要目的是确保系统接口的稳定性、正确性和性能。通过自动化测试，我们能够高效地发现接口存在的问题，从而提升系统的整体质量。

测试背景：在运维场景中，接口服务被广泛使用，其稳定性和正确性直接影响系统的可靠性和用户体验。因此，进行接口自动化测试是至关重要的。

测试范围涵盖主要系统管理接口，内容详见下文对应章节测试用例。

前提条件：所有接口均已上线并能够正常调用。

事件处理：遇到问题时，测试将暂停；当问题汇报并修复通过后，测试结束。

测试方法：使用 Python 脚本自动化调用，检查接口返回值是否符合预期。

测试环境和设计与功能测试部分相同；此外，Python 脚本将生成可视化的测试报告并实时发送到测试人员邮箱中。

测试总结：对测试结果进行分析，并提交测试报告。

2.3 UI 自动化测试

UI 自动化测试计划书旨在确保系统用户界面的稳定性和易用性。通过自动化测试，可以高效地发现用户界面存在的问题，提升系统的整体用户体验。

测试背景：系统的用户界面是用户与系统交互的直接窗口，其稳定性和易用

性直接影响到用户的满意度和使用效率。因此，进行 UI 自动化测试至关重要。

测试范围：测试将覆盖常用的 UI 界面，重点放在管理类操作上。

事件处理：遇到问题时，测试将暂停；当问题修复并通过验证后，测试结束。

测试方法：使用 Selenium 框架进行测试，验证设定的操作是否都能正确完成。

测试环境和设计与功能测试部分相同；测试用例详见下文对应章节。

测试总结：对测试结果进行分析，并提交测试报告。

2.4 性能测试

性能测试的主要目的是使用测试工具对系统进行性能评估，确保系统在高负载条件下的响应时间、吞吐量和稳定性，从而提升系统的整体性能。

测试背景：当服务器遭遇高并发用户访问和操作时，可能会出现性能瓶颈，影响用户体验和系统稳定性。因此，进行系统性能测试，发现潜在的性能问题并进行优化是必要的。

测试范围：测试将覆盖高频使用的接口，具体内容详见下文的测试用例章节。

前提条件：所有接口均已上线并能够正常调用，且基本接口测试无故障。

事件处理：遇到问题时，测试将暂停；当问题修复并通过验证后，测试结束。

测试方法：使用 JMeter 进行测试，并分析生成的结果报表。

测试总结：对测试结果进行详细分析，并提交测试报告。

3 测试方案

3.1 功能测试方案

测功能测试旨在验证系统的各个功能模块是否能够按照设计要求正常运行。这包括用户输入和输出功能、测试业务流程、系统数据处理和存储功能等。

测试系统介绍：

1. 宝塔面板：这是一款用于管理和监控 Linux 服务器的可视化工具，提供丰富的管理功能，如网站管理、数据库管理、文件管理等。

2. GPT 后台项目：这是一个用于管理 GPT 聊天站的后台管理系统，包含用户管理、对话管理、日志分析等功能。

3. 员工管理系统：一个通用的综合性管理系统，涵盖了用户管理、权限管理、

系统配置等各类管理功能。

测试目标：

1. 确保系统的各项功能能够按照设计要求正常运行。
2. 发现并修复所有严重及以上级别的缺陷，缺陷率控制在 0.5% 以内。
3. 确保核心业务流程的高可用性，确保系统稳定可靠。

功能测试主要包括以下几个方面：

1. 功能模块测试：针对每一个功能模块，编写详细的测试用例，验证其输入、处理和输出结果是否符合预期。例如，在宝塔 Linux 面板中，测试网站管理功能是否能够正确添加、删除和修改网站信息。

2. 业务流程测试：模拟用户的真实操作流程，验证系统在实际使用中的表现。例如，在 GPT 后台中，模拟管理员从登录到新增活动项目的全过程。

3. 边界值测试：测试系统在极限条件下的表现，例如输入最大值和最小值，文件上传的大小限制等。

具体的测试用例和测试数据见对应章节的附录。

3.2 接口测试方案

接口测试主要针对系统各个模块之间的接口，验证接口的正确性和稳定性。

测试系统介绍：宝塔面板：提供了丰富的 API 接口用于管理和监控服务器，包括网站管理接口、数据库管理接口等。

测试目标：

1. 确保测试的核心接口能够正常运行。
2. 发现并修复所有严重及以上级别的缺陷，缺陷率控制在 0.5% 以内。
3. 确保核心业务流程的高可用性，确保系统接口的稳定性和可靠性。

测试内容与数据：

1. 接口功能测试：验证接口的功能是否符合设计要求。例如，调用宝塔 Linux 面板的接口，验证其是否能返回需要的服务器信息。

2. 接口安全测试：验证接口的安全性，如认证机制、授权机制等，确保接口不会被非法访问和操作。

具体的测试用例和测试数据见对应章节附录。

3.3 UI 测试方案

UI 测试主要针对系统的用户界面，验证界面的功能、布局和交互设计是否符合要求。

测试系统介绍：

1. **GPT 后台项目：**用于管理 GPT 聊天站的后台管理系统，包含用户管理、对话管理、日志分析等功能。
2. **员工管理系统：**一个综合性管理系统，涵盖了用户管理、部门管理、权限管理等各类管理功能。

测试目标：

1. 确保用户界面能够按照设计要求正常运行。
2. 发现并修复所有严重及以上级别的缺陷。
3. 确保交互流程的稳定性，确保用户体验的一致性和流畅性。

测试内容与数据：

1. **界面布局测试：**验证各个界面的布局是否符合设计要求、元素是否对齐、布局是否合理。
2. **界面功能测试：**验证界面上的各个功能按钮、输入框、下拉菜单等是否能够正常工作。例如，在员工管理系统中是否能够正常添加一个用户并将其分配到正确的部门中。
3. **交互测试：**验证界面的交互设计是否合理，操作是否简便。例如，在 GPT 后台项目中，测试用户在多个界面之间切换的流畅性。
4. **界面兼容性测试：**验证界面在不同的浏览器（如 Chrome，Edge 等）、不同的分辨率下的显示效果，确保界面的一致性。

具体的测试用例和测试数据见对应章节附录。

3.4 性能测试方案

性能测试旨在模拟真实用户的操作行为，验证系统在高负载下的响应速度和稳定性。

测试系统介绍：

1. **宝塔面板：**可视化 Linux 管理工具，提供丰富的管理功能，如网站管理、

数据库管理、文件管理等。

2. **GPT 控制台**：用于管理 GPT 聊天站的后台管理系统，包含用户管理、对话管理、日志分析等功能。

3. **员工管理系统**：一个综合性管理系统，涵盖了用户管理、权限管理、系统配置等各类管理功能。

测试目标：

1. 模拟真实用户的操作行为，验证系统在高负载下的响应速度和稳定性。
2. 发现并修复所有严重及以上级别的缺陷，缺陷率控制在 0.5% 以内。
3. 确保核心业务流程的高可用性，确保系统在高负载情况下的稳定性和可靠性。

测试内容与数据：

1. **负载测试**：模拟大量用户同时访问系统，验证系统在高负载情况下的表现。例如，模拟大量用户同时登录宝塔 Linux 面板，测试系统的响应速度和稳定性。

2. **压力测试**：超过系统设计负载，测试系统在极端条件下的表现，观察系统是否会超时、崩溃或出现严重错误。

3. **并发测试**：模拟多个用户同时进行相同操作，验证系统的并发处理能力。例如，模拟多个用户同时在员工管理系统中进行操作。

具体的测试用例和测试数据见对应章节附录。

4 功能测试

4.1 用例设计



功能测试用例.xlsx

功能测试旨在验证软件系统的功能是否符合需求规范和设计文档中的要求。它关注的是系统的业务逻辑和功能实现，确保系统能够正确处理所有预期的输入并生成正确的输出。

在功能测试部分，我们分别分别针对宝塔 Linux 面板，Rhine Assistant 控制台、员工管理系统进行功能测试设计。首先理解功能需求和业务逻辑，确保测试用例覆盖所有常用功能点。在设计测试用例中，测试用例应描述清晰，包括前置

条件、输入数据、操作步骤和预期结果，还要考虑边界条件和极端情况，设计边界值测试用例同时进行等价类划分，将输入数据划分为不同的等价类，每个类中只需选择一个代表值进行测试，减少冗余的测试。在测试场景中，确保测试用例覆盖各种使用场景，包括正常流程和异常流程。功能测试的测试用例见附录 1。

4.2 测试执行

一、对宝塔面板进行测试执行

宝塔面板是一款服务器管理软件，不仅提供的功能多样，还可以通过 Web 端轻松管理服务器，提升运维效率。对于宝塔云面板编写手工测试用例进行功能测试，发现其存在以下方面需要完善。

首先是在计划任务模块中，当用户在新建计划任务时，如图 4-1 所示，其在最下方提示“shell 脚本中以下命令不可使用：shutdown, init 0, mkfs, passwd, chpasswd, --stdin, mkfs.ext, mke2fs”因为这些都是可能对服务器造成危险的命令。

添加任务

任务类型: Shell脚本

* 任务名称: 让服务器立即重启

执行周期: 每天 1 小时 30 分钟

每天的 01:30 执行一次

执行用户: 默认使用root用户执行

脚本内容: shutdown -r now

温馨提示 • 为了保证服务器的安全稳定，shell脚本中以下命令不可使用：shutdown, init 0, mkfs, passwd, chpasswd, --stdin, mkfs.ext, mke2fs

取消 确定

图 4-1 添加危险任务

显然，按照正常的逻辑，在保存添加时应该会通过系统对脚本内容的检查，因为 shutdown 命令为危险指令，不能保证每次重启的时候服务端没有正在运行的业务，面板限制不能保存这样的计划。但是在实际保存时，其没有进行脚本内容的检查，依然对这个计划任务进行了添加，如图 4-2 所示。



图 4-2 危险任务添加成功

解决方案：开发人员需要设置可以在提交添加时能自动拦截危险命令作为任务被提交的方法。

二、对 GPT 后台进行测试执行

这是一个国内搭建的 GPT 网站的后台系统面板。管理员可以在后台查看管理网站的注册会员，完善网站功能，并制定和推出套餐和优惠等服务。对于 GPT 后台编写手工测试用例进行功能测试，发现其存在以下方面需要完善。

首先是活动列表模块，在添加邀请活动时，在选择数字的时候可以选择所有非负整数来设定为这一项的值，或者使用-1 来代表不显示有效期，同时对小于-1 的输入进行报错，如图 4-3 所示。



图 4-3 拦截小于负一的负数

但是在进一步的测试中，发现其没有对输入方法进行甄别。比如当测试用例不输入阿拉伯数字，而是使用在编程中可以用来使用的科学计数法时会出现系统识别科学计数法的情况，如图 4-4 所示。



图 4-4 系统识别科学计数法

虽然在写代码时，`long a = 1e5` 可以设定 `a` 的值为 100000，但是按照程序设计

的角度，在前端页面输入这种值还能识别显然是不够完美的，而且很可能会引起其他不必要的问题。

而且不只是科学计数法，当我们使用其他进制数如十六进制进行输入时，也会被系统识别，如图 4-5 所示。



图 4-5 系统识别非十进制数

同样的，在 Java 中使用 `long a = 0x3f` 可以设定 `a` 的值为 63，但是按照程序设计的角度，在前端页面输入这种值还能识别显然是不够完美的，而且很可能会引起其他不必要的问题。

解决方法：将输入默认使用字符串形式，然后用 Long 类的 `parseLong` 方法将能转化成 long 类型，这样无法转化的就会报错不会被识别，从而杜绝了此类情况的再次发生。

三、对员工管理系统进行测试执行

员工管理系统是一个基于 **SpringBoot** 的前后端分离的后台管理系统，其实现了对整个公司部门，岗位，职位和员工的精细化管理。对于员工管理系统编写手工测试用例进行功能测试，发现其存在以下方面需要完善。

首先是在系统管理模块，在执行编辑角色列表的操作时，因为在员工管理系统中设计了很多权限名称，我们在加入角色时可以顺带给这个角色赋予他应该有的各种权限信息如图 4-6 所示。

修改角色

* 角色名称:

普通角色1

* 权限字符:

@RequiresRoles("system:role:add")

❗

控制器中定义的权限字符, 如: @RequiresRoles("")

* 显示顺序:

2

状态:

备注:

普通角色1

菜单权限:

☐ 展开/折叠

☐ 全选/全不选

☒ 父子联动

系统管理

系统监控

系统工具

若依官网

确定

关闭

图 4-6 给角色添加权限字符

但是系统无法检测自己内部有什么权限, 导致在输入错误的权限时, 这个不存在的权限信息也会被保存, 如图 4-7 所示。

修改角色

* 角色名称:

普通角色1

* 权限字符:

@RequiresRoles("1")

❗

控制器中定义的权限字符, 如: @RequiresRoles("")

* 显示顺序:

2

状态:

备注:

普通角色1

菜单权限:

☐ 展开/折叠

☐ 全选/全不选

☒ 父子联动

系统管理

系统监控

系统工具

若依官网

确定

关闭

图 4-7 给角色添加错误的权限字符

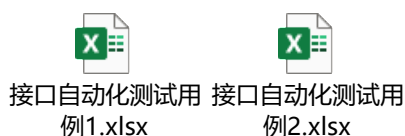
这里成功保存了一个不存在的权限字符, 但是系统没有给出报错信息, 但是后期这个角色在投入使用时会出现的错误。

解决方法: 在每一次使用这个方法时, 先检查系统的权限列表, 用复选框的

方式，让使用人员给角色添加正确的权限字符。

5 接口自动化测试

5.1 用例设计



对于接口自动化测试，从设计之初就有一个愿景，就是在经过接口测试之后，不仅要成功实现对所有被测接口的调用和测试，还能每个测试按先后顺序让测试接口和上一个接口的结果之间有逻辑上的联系。为了最大化对于宝塔面板的测试，我们先对基本的查看系统信息接口测试之后，我们设计了两条接口自动化测试主线：第一条接口自动化测试主线由于是对一个空的服务器，首先调用 FTP 模块接口创建 FTP 用户，然后调用接口将本地数据库文件和员工管理系统文件 jar 包进行上传，之后调用数据库模块的接口新建数据库并将数据表文件导入到数据库中，最后调用网站模块接口添加 jdk，最后将员工管理系统部署在这个服务器上并启动服务；第二条线因为数据库文件已经在服务器本地，所以直接调用数据库接口，完成数据库的创建并将数据表文件导入到数据库中，之后调用 Docker 模块的接口，导入编排模板并进行容器编排，最后成功启动运行 GPT 后台项目。

通过进行接口测试，不仅成功调用并测试了宝塔面板的绝大部分接口，还实现了全部通过调用接口的方式实现数据库的建立、SpringBoot 项目的部署和 Docker 项目的搭建。此外，整个项目两条主线均依赖同一个接口调用代码，完成不同任务时只需要在配置 Excel 中设置不同接口值，并设置要传递的参数即可，大大提升了测试项目代码的复用性，同时降低了耦合性。在测试之后，会生成测试报告，测试报告将以邮件的形式发送到测试人员的邮箱中，方便测试人员第一时间知道测试结果。接口自动化测试用例见附录 2 和附录 3。

员工管理系统作为一个 SpringBoot 项目，其部署流程并不繁琐，我们只需要准备数据库 sql 文件和项目 jar 包即可开始准备部署。其在接口自动化调用的时候主要涉及到两个重要的代码，一个是能适配并调用所有操作宝塔面板接口的宝塔操作代码，一个是在建立完 FTP 连接之后，使用 FTP 接口进行传输的相关代码。

在设计宝塔接口调用自动化代码时，首先封装 BT Panel API 请求：创建 `bt_api` 类来封装与宝塔面板 API 的交互，包含生成请求签名、发送 HTTP POST 请求以及处理返回结果的方法。然后读取测试数据：通过 `pandas` 库读取 Excel 文件中的测试用例数据，并将其转换为 Python 字典格式以便在测试中使用。最后设计数据驱动测试：利用 `ddt` 库实现数据驱动测试，逐条读取测试数据并执行相应的 API 请求，通过 `unittest` 框架验证请求结果。首先先建立一个实现调用宝塔接口的类，实现代码如下：

```
class bt_api:
    def __init__(self, bt_panel=None, bt_key=None):
        self.__BT_KEY = bt_key or
        'X1ifOvxIIXM5dxIpP1nU5afuGXCgZmKz'
        self.__BT_PANEL = bt_panel or 'http://117.50.195.127:10747'

    def __get_md5(self, s):
        m = hashlib.md5()
        m.update(s.encode('utf-8'))
        return m.hexdigest()

    def __get_key_data(self):
        now_time = int(time.time())
        p_data = {
            'request_token': self.__get_md5(str(now_time) + " +
self.__get_md5(self.__BT_KEY)),
            'request_time': now_time
        }
        return p_data

    def __http_post(self, url, p_data, timeout=1800):
        headers = {
            "accept": "application/json, text/plain, */*",
            "content-type": "application/x-www-form-urlencoded"
        }
        data = urllib.parse.urlencode(p_data).encode('utf-8')
        req = urllib.request.Request(url, data, headers=headers)
        response = urllib.request.urlopen(req, timeout=timeout)
        result = response.read()
        if isinstance(result, bytes):
            result = result.decode('utf-8')
        return result, response.getcode()
```

```

def api_request(self, url, data):
    p_data = self.__get_key_data()
    p_data.update(data)
    result, status_code = self.__http_post(url, p_data)
    return json.loads(result), status_code

def get_full_url(self, endpoint):
    return urllib.parse.urljoin(self.__BT_PANEL, endpoint)

```

因为宝塔面板有很强的加密属性，不能直接使用直接调用接口的方式来实现测试。为了能够实现自动化测试，我们需要先设定 key 和 panel，计算给定字符串的 MD5 值，然后生成请求签名和当前时间戳，最后收到测试的接口和传递的参数，将这些一并作为请求的内容对宝塔面板发起 Post 请求，这样就可以开始进行接口自动化测试了。

```

def read_excel(file, **kwargs):
    data_dict = []
    try:
        data = pd.read_excel(file, keep_default_na=False, **kwargs)
        # 检查请求参数是否为空，为空则设置为空字典
        for index, row in data.iterrows():
            if not row['请求参数'] or pd.isna(row['请求参数']):
                data.at[index, '请求参数'] = '{}'
        data_dict = data.to_dict('records')
    finally:
        return data_dict

# 读取 Excel 文件中的测试数据
Testdata = read_excel(r'data/宝塔数据库部署.xlsx', sheet_name='接口测试用例')

```

这段代码用于通过读取 Excel 表中的内容，获取接口自动化测试提供的接口和参数信息等内容。其不仅能成功遍历每一行来提取其中的内容，更针对于当表格中存在空值时如何在 Python 中进行正常识别进行了优化，使其能够正常读取当请求参数为空的情况。

```

@ddt.ddt
class TestMySQLAPI(unittest.TestCase):
    def setUp(self) -> None:
        self.api = bt_api()
        print("测试用例执行前操作")

```

```

@ddt.data(*Testdata)
def test_Api(self, testdata):
    print(f"Running test: {testdata}")
    self._testMethodDoc = testdata["用例描述"]
    url = self.api.get_full_url(testdata["接口地址"])
    data = json.loads(testdata["请求参数"])
    response, status_code = self.api.api_request(url, data)
    print(type(response))
    print(response)
    # 独立判断状态码
    print(status_code)
    self.assertEqual(status_code, 200, f"Expected status code 200, but
got {status_code}")

    def tearDown(self) -> None:
        print("测试用例执行后操作")

if __name__ == '__main__':
    unittest.main()

```

这段代码使用 `unittest` 框架和 `ddt` 库实现了一个数据驱动测试类，用于测试 `bt_api` 类的 API 调用。通过 `@ddt.ddt` 和 `@ddt.data` 装饰器，这个测试类可以使用不同的测试数据进行多次测试，从而完成接口测试的自动化。每次测试前，通过 `setUp` 方法初始化 `bt_api` 对象，并在测试中生成请求 URL、解析请求参数、调用 API，最后检查响应的类型、内容和状态码。在每次测试后，`tearDown` 方法会执行清理操作。

以上这段代码适用于所有宝塔面板的接口调用，对于不同的业务需求只需要在其配套的 Excel 表格中更改接口地址和传递的参数即可。

接下来我们设计使用宝塔 FTP 接口来实现文件传输的实现类。首先封装 FTP 上传功能：创建 `FTPUUploader` 类来封装 FTP 文件上传的操作，包括连接 FTP 服务器、上传文件及处理错误等。然后读取测试数据：通过 `pandas` 库读取 Excel 文件中的测试用例数据，并将其转换为 Python 字典格式以便在测试中使用。最后设计数据驱动测试：利用 `ddt` 库实现数据驱动测试，逐条读取测试数据并执行相应的上传操作，通过 `unittest` 框架验证上传功能的正确性。其具体实现代码如下：

```

class FTPUploader:
    def __init__(self):

```

```

self.server = "117.50.195.127"
self.port = 21
self.user = "francis"
self.password = "Y6xfDFBWxspNpjFn"

def upload_file(self, local_file, remote_file):
    try:
        # 连接到 FTP 服务器
        ftp = ftplib.FTP()
        ftp.connect(self.server, self.port)
        ftp.login(self.user, self.password)

        # 打开本地文件
        with open(local_file, 'rb') as file:
            # 使用 STOR 命令上传文件
            ftp.storbinary(f'STOR {remote_file}', file)

        # 关闭 FTP 连接
        ftp.quit()
        print("文件上传成功！")
    except ftplib.all_errors as e:
        print(f"文件上传失败: {e}")

```

这个类实现了将本地文件上传到远程 FTP 服务器的功能。它在初始化时设置 FTP 服务器的地址、端口、用户名和密码，并提供一个 `upload_file` 方法，该方法用于连接到 FTP 服务器并进行登录，随后上传指定的本地文件到远程路径，并在完成后关闭连接。如果在上传过程中发生错误，会捕获并输出错误信息。

```

def read_excel(file, **kwargs):
    data_dict = []
    try:
        data = pd.read_excel(file, **kwargs)
        data_dict = data.to_dict('records')
    finally:
        return data_dict

Testdata = read_excel(r'data/宝塔上传文件.xlsx', sheet_name='端口测试用例')

```

这段代码用于读取 Excel 文件并将其内容转换为字典列表。使用 `pandas` 库读取指定的 Excel 文件。将读取到的数据转换为字典列表，每行数据作为一个字典条目。

```

@ddt.ddt
class TestUpload(unittest.TestCase):
    def setUp(self) -> None :
        self.ftp = FTPUploader()
        print("测试用例开始执行")

    @ddt.data(*Testdata)
    def test_upload(self, testdata):
        print(f"Running test: {testdata}")
        self._testMethodDoc = testdata["用例描述"]
        local_file = testdata["本地文件路径"]
        remote_file = testdata["远程文件名"]
        self.ftp.upload_file(local_file, remote_file)

    def tearDown(self):
        print("测试用例执行结束")

```

这段代码使用 `unittest` 框架和 `ddt` 库实现了一个数据驱动的测试类，用于测试 `bt_api` 类的 API 调用。通过 `@ddt.ddt` 和 `@ddt.data` 装饰器，这个测试类可以使用不同的测试数据进行多次测试，从而完成接口测试的自动化。每次测试前，通过 `setUp` 方法初始化 `bt_api` 对象，并在测试中生成请求 URL、解析请求参数、调用 API，最后检查响应的类型、内容和状态码。在每次测试后，`tearDown` 方法会执行清理操作。

以上这段代码适用于使用向服务器传输文件。通过这段代码，我们可以自动化批量传输在本地的文件到服务器上。

在全套测试完成后，我们还需要将生成的测试报告通过邮件的形式来发送到测试人员的邮箱里，以便于其在第一时间获取测试信息。其设计思路是首先定义初始化方法，初始化邮件发送所需的参数，包括发件人、收件人、密码、SMTP 服务器地址和端口。然后设计邮件发送方法，创建邮件对象然后添加邮件正文和附件。最后连接到 SMTP 服务器，进行登录验证，并发送邮件。其具体实现代码如下：

```

class EmailSender:
    def __init__(self, context=None):
        self.sender = '1297289742@qq.com'
        self.receiver = '1297289742@qq.com'
        self.password = 'natbnoijjxbsjaii'
        self.smtp_server = 'smtp.qq.com'

```

```

        self.smtp_port = 465
        self.content = context
        logging.basicConfig(filename='email_log.log',
level=logging.INFO,
                                format='%(asctime)s - %(levelname)s
- %(message)s')

    def send_email(self):
        # 创建邮件对象
        msg = MIMEMultipart()
        msg['From'] = self.sender
        msg['To'] = self.receiver
        msg['Subject'] = '接口自动化测试报表'

        # 添加邮件正文
        msg.attach(MIMEText(self.content, 'html', 'utf-8'))

        # 添加附件
        for attachment in ['./testdemoreport.html']:
            with open(attachment, 'rb') as f:
                part = MIMEBase('application', 'octet-stream')
                part.set_payload(f.read())
                encoders.encode_base64(part)
                part.add_header('Content-Disposition', 'attachment',
filename=attachment)
                msg.attach(part)

        # 连接 SMTP 服务器并发送邮件
        try:
            smtp_obj = smtplib.SMTP_SSL(self.smtp_server,
self.smtp_port)
            smtp_obj.login(self.sender, self.password)
            smtp_obj.sendmail(self.sender, [self.receiver],
msg.as_string())
            logging.info('邮件发送成功')
            smtp_obj.quit()
        except smtplib.SMTPException as e:
            logging.error('邮件发送失败:  %s' % e)

# 使用示例
if __name__ == "__main__":
    email_sender = EmailSender()

```

```
email_sender.send_email()
```

最后，我们设计一个总的启动类，将这些不同分工的代码类的 Test 对象封装到一个套件中，然后按顺序执行。即可得到从文件上传到服务启动后发送测试报告全流程的执行流程。其具体实现代码如下：

```
def summary_format(result):
    summary = "\n" + u"<p>                测试结果汇总信息
</p>" + "\n" + \
        u"<p> 开始时间: " + result['beginTime'] + u" </p>" +
    "\n" + \
        u"<p> 运行时间: " + result['totalTime'] + u" </p>" +
    "\n" + \
        u"<p> 执行用例数: " + str(result['testAll']) + u" </p>"
    + "\n" + \
        u"<p> 通过用例数: " + str(result['testPass']) + u"
</p>" + "\n" + \
        u"<p> 失败用例数: " + str(result['testFail']) + u"
</p>" + "\n" + \
        u"<p> 忽略用例数: " + str(result['testSkip']) + u"
</p>" + "\n"
    return summary

def suite():
    suite = unittest.TestSuite()
    loader = unittest.TestLoader()
    suite.addTests(loader.loadTestsFromTestCase(TestFTPAPI))
    suite.addTests(loader.loadTestsFromTestCase(TestUpload))
    suite.addTest(loader.loadTestsFromTestCase(TestMySQLAPI))
    suite.addTest(loader.loadTestsFromTestCase(TestSpringBootAPI))
    return suite

if __name__ == '__main__':
    br = BeautifulReport(suite())
    br.report(filename='testdemoreport.html',description='测试报告
',report_dir='.')
    rpt_summary = summary_format(br.fields)
    es = EmailSender(rpt_summary)
    es.send_email()
```

在这里我们只需启动这个启动类，就可以完成接口的自动化测试，同时生成测试报告发送到测试人员邮箱中。

5.2 测试执行

在用例设计中，我们将整个的业务逻辑最后封装到一个文件名为 `app.py` 里的测试套件中，这个套件可以按顺序依次执行通过接口自动化测试完成员工管理系统的搭建工作，生成接口测试报告，最后将测试报告发送给测试人员。我们启动这个测试套件，其开始执行，通过控制台我们可以看到其调用接口时每一步的输出结果，结果如图 5-1~5-3 所示。

```
测试用例执行前操作
Running test: {'用例描述': '获取磁盘信息', '接口地址': '/system?action=GetDiskInfo', '请求参数': '{}', '是否通过': '查询成功', '备注': '无'}
<class 'list'>
[{'path': '/', 'size': ['39.76 GB', '11.33 GB', '28.41 GB', '28.51%', '16.00 MB'], 'filesystem': '/dev/vda1', 'type': 'ext4', 'inodes': [5160960, 233972, 4926988, '4.53 %']}]
200
测试用例执行后操作
测试用例执行前操作
Running test: {'用例描述': '添加ftp', '接口地址': '/ftp?action=AddUser', '请求参数': '{\n  "ftp_username": "francis",\n  "ftp_password": "Y6xfDFBwXspNpJFn",\n  "path": "/www/wwwroot/francis",\n  ...<class 'dict'>
{'status': True, 'msg': '添加成功'}
200
测试用例执行后操作
测试用例执行前操作
Running test: {'用例描述': '查询当前ftp列表', '接口地址': '/dataList/data/get_data_list', '请求参数': '{\n  "p": 1,\n  "limit": 10,\n  "search": "",\n  "table": "ftps"\n}', '是否通过': '查询成功'}
<class 'dict'>
{'where': '', 'page': '<div><span class="Pcurrent">1</span><span class="Pcount">共2条</span></div>', 'data': [{'id': 13, 'pid': 0, 'name': 'francis', 'password': 'Y6xfDFBwXspNpJFn', 'pat
200
测试用例执行后操作
测试用例开始执行
Running test: {'用例描述': '添加数据库表文件1', '本地文件路径': 'C:\\Users\\12972\\Desktop\\智能集成测试\\quartz.sql', '远程文件名': 'quartz.sql', '是否通过': '文件上传成功', '备注': '无'}
文件上传成功!
测试用例执行结束
测试用例开始执行
Running test: {'用例描述': '添加数据库表文件2', '本地文件路径': 'C:\\Users\\12972\\Desktop\\智能集成测试\\ry_20240112.sql', '远程文件名': 'ry_20240112.sql', '是否通过': '文件上传成功', '备注': '无'}
..文件上传成功!
测试用例执行结束
测试用例开始执行
Running test: {'用例描述': '添加若依jar包', '本地文件路径': 'C:\\Users\\12972\\Desktop\\智能集成测试\\origin.jar', '远程文件名': 'origin.jar', '是否通过': '文件上传成功', '备注': '无'}
.文件上传成功!
测试用例执行结束
```

图 5-1 建立 FTP 用户并上传数据库和 jar 包

```
测试用例执行前操作
Running test: {'用例描述': '增加新数据库', '接口地址': '/database?action=AddDatabase', '请求参数': '{\n  "name": "rytest1",\n  "db_user": "amosi",\n  "password": "root",\n  "dataAccess": "%",\n  "address": "%",\n  ...<class 'dict'>
{'status': True, 'msg': '添加成功'}
200
测试用例执行后操作
测试用例执行前操作
Running test: {'用例描述': '添加表1', '接口地址': '/database?action=InputSql', '请求参数': '{\n  "file": "www/backup/database/quartz.sql",\n  "name": "rytest1"\n}', '是否通过': '添加成功', '备注': '参数设置自己的'}
<class 'dict'>
{'status': True, 'msg': '导入数据库成功!'}
200
测试用例执行后操作
测试用例执行前操作
Running test: {'用例描述': '添加表2', '接口地址': '/database?action=InputSql', '请求参数': '{\n  "file": "www/backup/database/ry_20240112.sql",\n  "name": "rytest1"\n}', '是否通过': '添加成功', '备注': '参数设置自己的'}
..<class 'dict'>
{'status': True, 'msg': '导入数据库成功!'}
200
测试用例执行后操作
测试用例执行前操作
Running test: {'用例描述': '查询数据库', '接口地址': '/dataList/data/get_data_list', '请求参数': '{\n  "p": 1,\n  "limit": 10,\n  "table": "databases",\n  "search": "",\n  "order": ""\n}', '是否通过': '查询成功', '备注': ''}
<class 'dict'>
{'where': "LOWER(type) = LOWER('MySQL')", 'page': "<div><span class="Pcurrent">1</span><span class="Pcount">共3条</span></div>", 'data': [{'id': 34, 'pid': 0, 'name': 'rytest1', 'username': 'amosi', 'password': 'root
200
测试用例执行后操作
```

图 5-2 添加数据库并导入表

测试用例执行前操作
Running test: {'用例描述': '添加jdk', '接口地址': '/project/java/install_jdk_new', '请求参数': '{\n "name": "jdk",\n "version": "jdk-11.0.19",\n "type": 1\n}', '是否通过': '添加成功', '备注': '无'}
<class 'dict'>
{'status': True, 'msg': '安装java成功'}
200
测试用例执行后操作
测试用例执行前操作
Running test: {'用例描述': '查询jdk信息', '接口地址': '/project/java/get_local_jdk_version', '请求参数': '{}', '是否通过': '查询成功', '备注': '无'}
<class 'list'>
[{'name': 'jdk1.8.0_371', 'path': '/www/server/java/jdk1.8.0_371/bin/java', 'operation': 1, 'is_current': False}, {'name': 'jdk-11.0.19', 'path': '/www/server/java/jdk-11.0.19/bin/java', 'ope
200
测试用例执行后操作
测试用例执行前操作
Running test: {'用例描述': '添加项目', '接口地址': '/project/java/create_project', '请求参数': '{\n "project_type": 3,\n "project_jar": "/www/wwwroot/Amos/ruoyi-admin.jar",\n "project_name": "ru
<class 'dict'>
200
测试用例执行后操作
测试用例执行前操作
Running test: {'用例描述': '查询项目信息', '接口地址': '/project/java/get_project_list', '请求参数': '{\n "p": 1,\n "limit": 10,\n "type_id": "",\n "search": ""\n}', '是否通过': '查询成功', '备注':
..<class 'dict'>
{'page': '<div>1共2条</div>', 'shift': '0', 'row': '10', 'data': [{'id': 17, 'name': 'ruoyi1', 'path': '/www/wwwroot/Amos/ruoyi-admin.j
200
测试用例执行后操作
测试用例执行前操作
Running test: {'用例描述': '设置项目启动', '接口地址': '/project/java/start_project', '请求参数': '{\n "project_name": "ruoyi1"\n}', '是否通过': '设置成功', '备注': '无'}
.
测试已全部完成, 可打开 C:\Users\12972\Desktop\智能集成测试\APIdemos\proj2\testdemoreport.html 查看报告
<class 'dict'>
{'status': True, 'msg': '启动成功'}
200
测试用例执行后操作

图 5-3 添加 jdk 并启动员工管理系统

此时我们邮箱中会收到测试报告，邮件内容和测试报告内容如图 5-4 和图 5-5 所示。

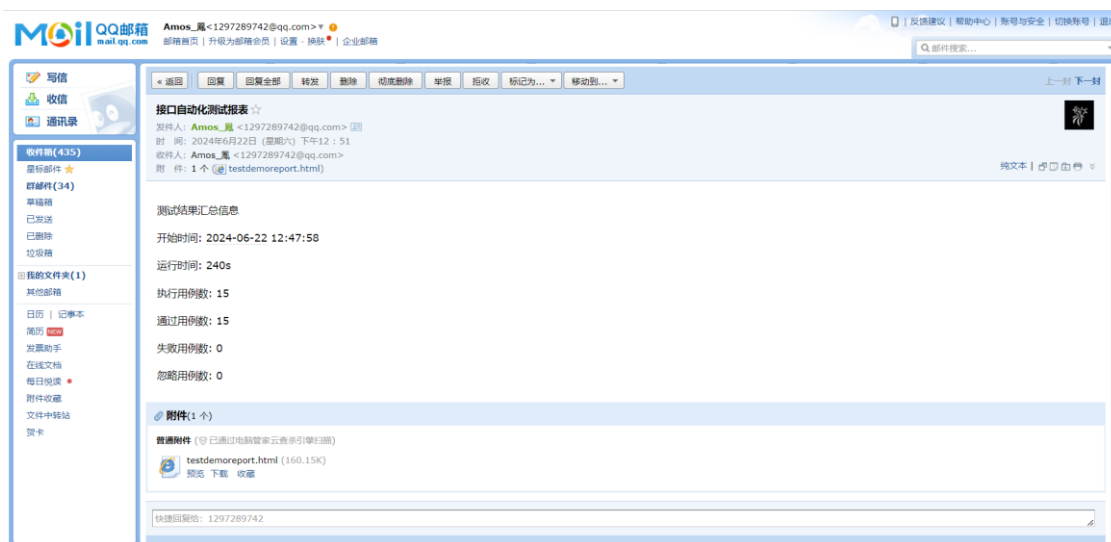


图 5-4 邮件内容



图 5-5 测试报告内容

在测试报告中，我们不仅可以看到用例数，用例通过、失败和跳过数量，测试开始时间和运行时长，还可以看到测试类、测试方法和用例描述，如图 5-6 所示。

之后我们验证接口自动化测试结果，输入正确的端口号，成功访问到了部署成功的员工管理系统，如图 5-7 所示。

测试类: ALL 结果: ALL 用例数: 15 成功: 15 失败: 0 跳过: 0						
编号	测试类	测试方法	用例描述	运行时长	结果	操作
1	TestFTPAPI	test_Api_1	获取磁盘信息	15.8 s	成功	展开
2	TestFTPAPI	test_Api_2	添加ftp	20.5 s	成功	展开
3	TestFTPAPI	test_Api_3	查询当前ftp列表	15.8 s	成功	展开
4	TestUpload	test_upload_1	添加数据库表文件1	0.711 s	成功	展开
5	TestUpload	test_upload_2	添加数据库表文件2	0.853 s	成功	展开
6	TestUpload	test_upload_3	添加若依jar包	22.9 s	成功	展开
7	TestMySQLAPI	test_Api_1	增加新数据库	20.4 s	成功	展开
8	TestMySQLAPI	test_Api_2	添加表1	6.64 s	成功	展开
9	TestMySQLAPI	test_Api_3	添加表2	21.4 s	成功	展开
10	TestMySQLAPI	test_Api_4	查询数据库	15.9 s	成功	展开
11	TestSpringBootAPI	test_Api_1	添加jdk	43.9 s	成功	展开
12	TestSpringBootAPI	test_Api_2	查询jdk信息	20.3 s	成功	展开
13	TestSpringBootAPI	test_Api_3	添加项目	4.67 s	成功	展开
14	TestSpringBootAPI	test_Api_4	查询项目信息	15.9 s	成功	展开
15	TestSpringBootAPI	test_Api_5	设置项目启动	14.0 s	成功	展开

图 5-6 测试具体内容

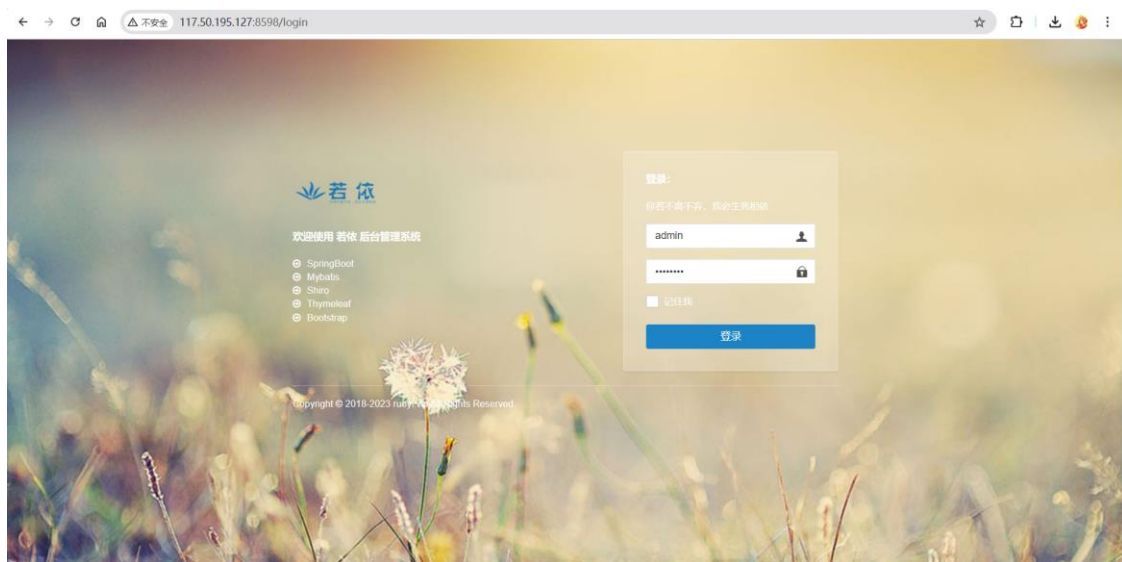


图 5-7 查看接口自动化测试结果

6 UI 自动化测试

6.1 用例设计

一、GPT 后台用例设计

UI 自动化测试利用 Selenium 框架,对 GPT 后台系统下述若干功能进行测试:

一、登录测试（前置操作，故用例编号从下一个操作开始）

```
def login(driver, url, username_str, password_str):  
    try:  
        # 打开网页  
        driver.get(url)  
        time.sleep(3)  
  
        # 找到用户名、密码输入框和登录按钮  
        username = driver.find_element(By.NAME, "username")  
        password = driver.find_element(By.NAME, "password")  
        login_button = driver.find_element(By.XPATH,  
        '//button[@type="button" and contains(@class, "el-button--primary") and  
        contains(@class, "el-button--mini")]')  
  
        # 输入用户名和密码  
        username.clear()  
        password.clear()  
        username.send_keys(username_str)
```

```

password.send_keys(password_str)

# 点击登录按钮
login_button.click()

# 等待页面加载
print("已执行登录")

# 关闭弹窗
try:
    # 等待弹窗出现
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CSS_SELECTOR,
            "div.el-message-box__wrapper")))
    # 点击确定按钮
    confirm_button = driver.find_element(By.CSS_SELECTOR,
        "div.el-message-box__btns button.el-button--primary")
    confirm_button.click()
    time.sleep(1)
except Exception as e:
    print(f"发生错误: {e}")
except Exception as e:
    print(f"发生错误: {e}")

```

该部分代码会找到用户名和密码的输入框，并填入设置的数据，然后找到登录按钮并点击，然后自动点击弹窗的“确定”按钮。

二、变更用户套餐可用性

```

def alter_b_available(driver):
    try:
        print("===== 测试用例 2: 更改用户套餐可  

        用性 =====\n")
        # 定位并点击“废除”按钮
        abolish_button = driver.find_element(By.XPATH,
            "//button[@class='el-button el-button--danger el-button--mini is-plain']")
        abolish_button.click()

        # 等待并点击确认框中的确定按钮
        confirm_button = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.XPATH,
                "//div[@role='dialog']/button[@class='el-button el-button--default  

                el-button--small el-button--primary ']))

```

```

    )
    confirm_button.click()
    success_message = WebDriverWait(driver, 10).until(
        EC.visibility_of_element_located(
            (By.CSS_SELECTOR,
"div.el-message.el-message--success p.el-message__content"))
    )
    print("执行禁用操作，捕获到弹窗:", success_message.text)
    time.sleep(2)

except Exception as e:
    print(f"操作过程中出现错误: {e}")

```

以上这段代码中，在这一步我们首先读取了用户额度信息列表，然后尝试将用户的现在的套餐执行废除的操作。接下来，我们将要尝试重写弃用这个用户套餐。

```

try:
    restore_button = driver.find_element(By.XPATH,
"//button[contains(@class, 'el-button--primary') and contains(@class,
'el-button--mini') and contains(@class,
'is-plain')]/span[normalize-space(text())='恢复']")
    restore_button.click()

    # 等待并点击确认框中的确定按钮
    confirm_button = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.XPATH,
"//div[@role='dialog']/button[@class='el-button el-button--default
el-button--small el-button--primary ']))
    )
    confirm_button.click()
    success_message = WebDriverWait(driver, 10).until(
        EC.visibility_of_element_located(
            (By.CSS_SELECTOR,
"div.el-message.el-message--success p.el-message__content"))
    )
    print("执行恢复操作，捕获到弹窗:", success_message.text)

    time.sleep(1)
    try:
        # 定位并点击包含“详情”字样的关闭按钮
        close_button = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.XPATH,

```

```

        "//button[contains(@aria-label, '详情')]))
    )
    close_button.click()
except Exception as e:
    print(f"操作过程中出现错误: {e}")
    print("\n===== 测试用例 2 结束
=====\\n")
    time.sleep(2)
except Exception as e:
    print(f"操作过程中出现错误: {e}")

```

上一步读取用户额度信息后，这段代码会尝试将用户废除后的套餐再启用，同时捕获操作结果的弹窗内容并输出到控制台。

预期结果：成功执行上述操作，并在控制台看到程序反馈。

三、添加管理员用户

```

def add_administrator(driver, nickname, username, password):
    # 检查菜单是否为 active 状态
    try:
        hamburger_menu = driver.find_element(By.CSS_SELECTOR,
        "svg.hamburger")
        if "is-active" not in hamburger_menu.get_attribute("class"):
            hamburger_menu.click()
            # print("发现菜单未展开，已展开")
            sleep(2) # 等待侧栏展开
    except NoSuchElementException:
        print("菜单未找到")

    try:
        print("===== 测试用例 3：添加管理员操作
=====\\n")
        # 等待页面出现
        WebDriverWait(driver,
        10).until(EC.presence_of_element_located((By.CSS_SELECTOR,
        "li.el-menu-item")))
        # 查找包含"管理员列表"文本的菜单项
        member_list_item = driver.find_element(By.XPATH,
        "//li[contains(@class, 'el-menu-item') and //span[text()='管理员列表']]")
        member_list_item.click()
        sleep(2)
    except Exception as e:
        print(f"发生错误: {e}")

```

```

try:
    # 定位并点击“新增”按钮
    add_button = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable(
            (By.XPATH, "//button[@class='el-button
el-button--primary el-button--mini']/span[text()='新增']"))
        )
    add_button.click()
except Exception as e:
    print(f"操作过程中出现错误: {e}")

```

以上这部分代码实现了点击 GPT 后台管理页面的管理员列表模块并点击了新增管理员的按钮，接下来这段代码，我们将实现把新的管理员信息填入这个弹出的表格。

```

try:
    # 等待弹出对话框的出现
    WebDriverWait(driver, 10).until(
        EC.visibility_of_element_located((By.XPATH,
        "//div[@role='dialog' and @aria-label='创建用户']"))
    )

    # 填入昵称
    nickname_input = driver.find_element(By.XPATH,
    "//label[text()='昵称']/following-sibling::div//input")
    nickname_input.send_keys(nickname)

    # 填入用户名(账号)
    username_input = driver.find_element(By.XPATH, "//label[text()='
用户名(账号)']/following-sibling::div//input")
    username_input.send_keys(username)

    # 填入密码
    password_input = driver.find_element(By.XPATH, "//label[text()='
密码']/following-sibling::div//input")
    password_input.send_keys(password)

    # 点击提交按钮
    submit_button = driver.find_element(By.XPATH, "//span[text()='
提交']/ancestor::button")
    submit_button.click()

```

```

        try:
            success_message = WebDriverWait(driver, 10).until(
                EC.visibility_of_element_located(
                    (By.CSS_SELECTOR,
                     "div.el-message.el-message--success p.el-message__content")))
            )
            print("执行添加管理员操作，捕获到弹窗:",
                  success_message.text)
            print("\n===== 测试用例 3 结束\n")
        except Exception as e:
            print(f"操作过程中出现错误: {e}")
        except Exception as e:
            print(f"操作过程中出现错误: {e}")

        sleep(2)

```

这段代码完成了根据设置的数据新建一个管理员账户，最后捕获操作结果的弹窗内容并输出到控制台。

预期结果：成功添加管理员账户并看到程序反馈。

二、员工管理系统用例设计

UI 自动化测试利用 **Selenium** 框架，对员工管理系统下述若干功能进行测试：

一、登录测试（前置操作，故用例编号从下一个操作开始）

```

def login(driver, url, username_str, password_str):
    try:
        # 打开网页
        driver.get(url)
        time.sleep(3)

        # 找到用户名、密码输入框和登录按钮
        username = driver.find_element(By.NAME, "username")
        password = driver.find_element(By.NAME, "password")

        # 输入用户名和密码
        username.clear()
        password.clear()
        username.send_keys(username_str)
        password.send_keys(password_str)

```



```

# 等待页面加载并定位按钮
submit_button = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "btnSubmit"))
)

# 点击按钮
submit_button.click()

# 等待页面加载
print("已执行登录")

# 关闭弹窗
try:
    # 等待页面加载并定位“取消”按钮
    cancel_button = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.CSS_SELECTOR,
".layui-layer-btn1"))
    )
    # 点击“取消”按钮
    cancel_button.click()
    time.sleep(1)
except Exception as e:
    print(f"发生错误: {e}")
except Exception as e:
    print(f"发生错误: {e}")

```

这部分代码会定位登录页面的输入框并输出设定的信息，然后寻找登录按钮并点击，接着点击登录后弹窗。

预期结果：成功登录并点击弹窗。

二、读取系统信息

```

def read_system_info(driver):
    try:
        # 等待页面加载并定位“系统工具”菜单项
        system_monitor_menu = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.XPATH, "//span[text()='系
统监控']"))
        )
        # 点击“系统监控”菜单项以展开子菜单
        system_monitor_menu.click()
        sleep(1)
    except Exception as e:

```

```
print(f"发生错误: {e}")
```

以上这部分代码将定位到员工管理系统的监控系统模块并点击下拉菜单，接下来我们将实现点击并读取服务监控中返回的内容。

```
try:
    # 等待并点击“服务监控”子菜单项
    service_monitoring_menu = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH, "//a[text()='服务
监控]")))
    )
    service_monitoring_menu.click()
    sleep(5)
except Exception as e:
    print(f"发生错误: {e}")

try:
    driver.switch_to.frame(driver.find_element(By.XPATH,
        "//iframe[@name='iframe15']"))
    data = {}
    rows = driver.find_elements(By.XPATH, "//tbody/tr")
    # print(rows)
    print("===== 测试用例 1: 读取系统监控信
息 =====\n")

    for row in rows:
        cells = row.find_elements(By.XPATH, "//tbody/tr/td")
        contents = []
        key = "系统监控"
        for i in range(0, len(cells)):
            contents.append(cells[i].text.strip())
            data[key] = contents

    # 打印结果
    import json
    print(json.dumps(data, indent=4, ensure_ascii=False))
    print("\n===== 测试用例 1 结束
===== \n")

    sleep(1)
    driver.switch_to.default_content()
    sleep(2)
except Exception as e:
```

```
print(f"发生错误: {e}")
```

这部分代码会先找到并进入“服务监控”页，然后读取页面中的系统信息并输出到控制台。

预期结果：控制台输出指定信息。

三、添加用户

```
def user_add(driver, username, loginname, password):
    try:
        # 等待页面加载并定位“系统管理”菜单项
        system_monitor_menu = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.XPATH, "//span[text()='系
统管理']")))
        # 点击“系统监控”菜单项以展开子菜单
        system_monitor_menu.click()
        sleep(1)
    except Exception as e:
        print(f"发生错误: {e}")

    try:
        # 等待并点击“用户管理”子菜单项
        service_monitoring_menu = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.XPATH, "//a[text()='用户
管理']")))
        service_monitoring_menu.click()
        sleep(2)
    except Exception as e:
        print(f"发生错误: {e}")
```

以上这部分代码将定位到员工管理系统里系统管理模块并点击下拉菜单，接下来我们将实现点击用户管理并进行新增用户操作。

```
try:
    print("===== 测试用例 2: 新建用户
=====\\n")
    # 切换到用户管理 iFrame
    driver.switch_to.frame(driver.find_element(By.XPATH,
        "//iframe[@name='iframe2']"))

    # 等待并点击“新增”按钮
    add_button = WebDriverWait(driver, 10).until(
```

```

        EC.element_to_be_clickable((By.XPATH,
        "//a[contains(@class, 'btn-success') and contains(., '新增')]"))
    )
    # 点击“新增”按钮
    add_button.click()
    sleep(2)

    # 切换到用户添加 iFrame
    driver.switch_to.default_content()
    driver.switch_to.frame(driver.find_element(By.XPATH,
    "//iframe[@data-id='/system/user/add']"))

    # 找到用户名、密码输入框和登录按钮
    username_input = driver.find_element(By.NAME, "userName")
    loginname_input = driver.find_element(By.NAME, "loginName")
    password_input = driver.find_element(By.NAME, "password")

    username_input.clear()
    loginname_input.clear()
    password_input.clear()

    username_input.send_keys(username)
    loginname_input.send_keys(loginname)
    password_input.send_keys(password)

    save_button = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH,
        "//button[@type='button' and contains(@class, 'btn-primary') and contains(.,
        '保存')]"))
    )
    save_button.click()

    sleep(1)
    driver.switch_to.default_content()
    driver.switch_to.frame(driver.find_element(By.XPATH,
    "//iframe[@name='iframe2']"))
    try:
        # 使用显式等待查找包含特定文本的元素
        element = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.XPATH,
            f"//*[contains(text(), {username})]"))
        )
        print(f"用户已添加！")

```

```

except Exception as e:
    print("用户未正常添加", e)

driver.switch_to.default_content()
print("\n===== 测试用例 2 结束
=====\\n")

sleep(2)
except Exception as e:
    print(f"发生错误: {e}")

```

这部分代码会找到并进入“用户管理”页面，然后执行用户添加操作。
预期结果：用户添加成功。

6.2 测试执行

一、GPT 后台测试执行

1. 登录测试：

结果：通过，如图 6-1 所示。

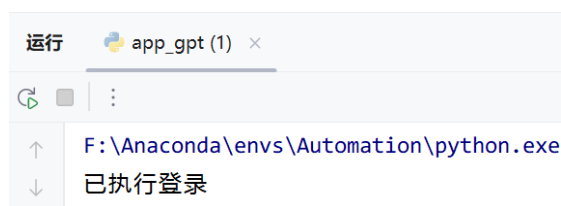


图 6-1 登录测试输出

2. 读取用户额度信息：

结果：通过，如图 6-2 所示。

```

===== 测试用例 1：读取用户套餐信息 =====

用户名（账号）：rauser
昵称：rauser
tokens: 114514
普通聊天次数：19
高级聊天次数：19
绘画次数：810

===== 测试用例 1 结束 =====

```

图 6-2 读取用户额度信息

3. 变更用户套餐可用性：

结果：通过，如图 6-3 所示。

```
===== 测试用例 2：更改用户套餐可用性 =====

执行禁用操作，捕获到弹窗：操作成功！
执行恢复操作，捕获到弹窗：操作成功！

===== 测试用例 2 结束 =====
```

图 6-3 变更用户套餐可用性

二、员工管理系统测试执行

1. 登录测试：

结果：通过，如图 6-4 所示。

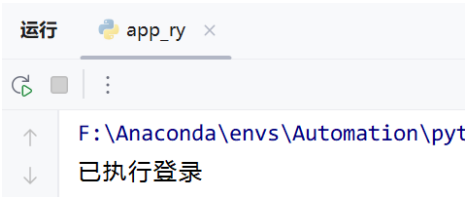


图 6-4 登录测试

2. 读取系统信息：

结果：通过，如图 6-5 所示。

```
===== 测试用例 1：读取系统监控信息 =====

{
  "系统监控": [
    "核心数",
    "4",
    "用户使用率",
    "0.25%",
    "系统使用率",
    "0.0%",
    "当前空闲率",
    "99.75%",
  ]
}
```

图 6-5 读取系统信息

3. 添加用户：

结果：通过，如图 6-6 所示。

```
===== 测试用例 2：新建用户 =====

用户已添加！

===== 测试用例 2 结束 =====
```

图 6-6 添加用户

7 性能测试

7.1 用例设计

一、宝塔面板用例设计

对于宝塔面板，我们模拟其对查看网络的请求、查看系统日志、查看防火墙状态这三个接口进行性能测试。

首先使用 JMeter 在宝塔线程新建 HTTP 信息头管理器，分别设置接口调用时在请求头需要携带的 Content-Type、Cookie、x-http-token 信息。然后新建 HTTP 请求默认值配置文件添加每次请求的默认值，简便后续使用。之后新加三个 HTTP 取样器来实现对接口调用的返回值的取样。然后添加同步定时器来模拟并发情况。最后添加聚合报告和查看结果树的监听器，来实现对结果的监听。宝塔面板线程组设计如图 7-1 所示。

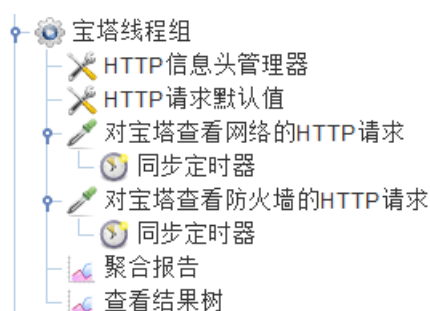


图 7-1 宝塔系统的线程组设计

我们分别检测其在 5、10、30、50、100 个并发数下的并发用户数与事务响应情况与并发用户数与服务器性能情况进行测试。

二、GPT 后台用例设计

对于 GPT 后台，我们模拟其对后台登录功能、查看高级管理员列表、查看注册增额信息这三个接口进行性能测试。

首先使用 JMeter 在 GPT 后台线程新建 HTTP 信息头管理器，分别设置接口调用时在请求头需要携带的 Content-Type、Authorization 信息。然后新建 HTTP 请求默认值配置文件添加每次请求的默认值，简便后续使用。之后新加三个 HTTP 取样器来实现对接口调用的返回值的取样。然后添加同步定时器来模拟并发情况。最后添加聚合报告和查看结果树的监听器，来实现对结果的监听。GPT 后台线程组设计如图 7-2 所示。

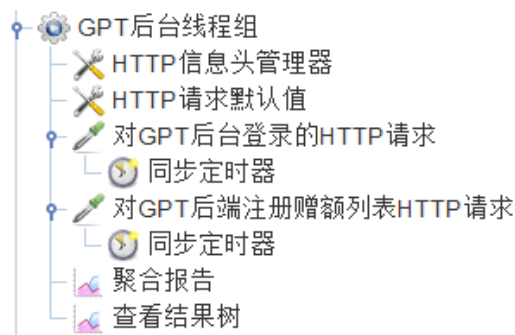


图 7-2 GPT 后台的线程组设计

我们分别检测其在 5、10、30、50、100 个并发数下的并发用户数与事务响应情况与并发用户数与服务器性能情况进行测试。

三、员工管理系统用例设计

对于员工管理系统，我们模拟其对查看角色管理列表、查看部门管理列表、查看通知公告信息这三个接口进行性能测试。

首先使用 JMeter 在 GPT 后台线程新建 HTTP 信息头管理器，分别设置接口调用时在请求头需要携带的 Content-Type、Cookie 信息。然后新建 HTTP 请求默认值配置文件添加每次请求的默认值，简便后续使用。之后新加三个 HTTP 取样器来实现对接口调用的返回值的取样。然后添加同步定时器来模拟并发情况。最后添加聚合报告和查看结果树的监听器，来实现对结果的监听。员工管理系统线程组设计如图 7-3 所示。

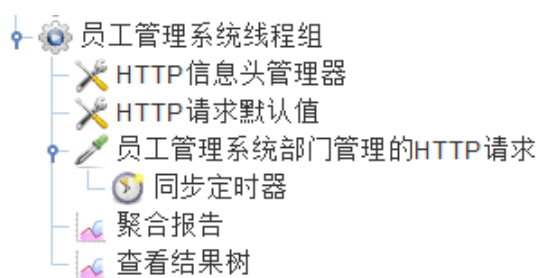


图 7-3 员工管理系统的线程组设计

我们分别检测其在 5、10、30、50、100 个并发数下的并发用户数与事务响应情况与并发用户数与服务器性能情况进行测试。

7.2 测试执行

一、宝塔面板测试执行

对宝塔面板进行性能测试，在测试之后查看宝塔线程组的结果树，结果

如图 7-4 所示。

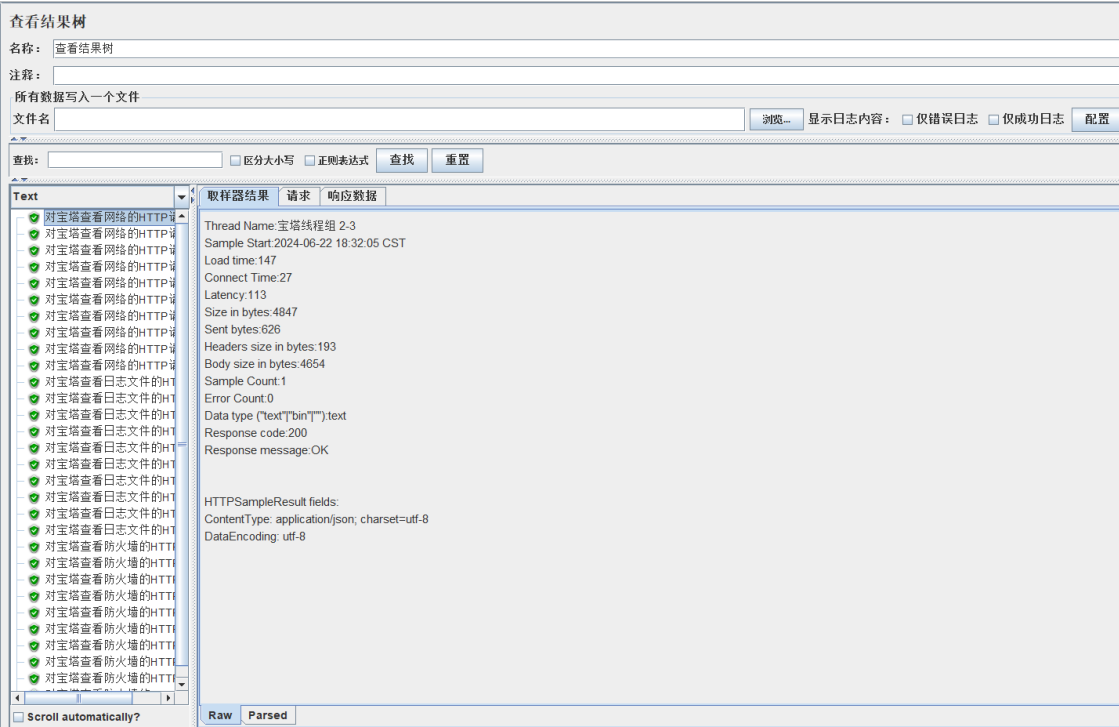


图 7-4 查看宝塔面板的结果树

分别在指定并发数下进行执行，将执行后的聚合报告情况和服务器性能情况进行记录，记录情况如表 7-1~表 7-2 所示

表 7-1 查看宝塔系统网络

用例名称	查看宝塔系统网络			
用例编号	test001			
用例描述	登录的 TPS 达到 20 时，响应时间在 3S 内			
前置条件	用户及相关数据配置完成			
用例步骤	动作	期望的性能		
1	进入宝塔面板，查看系统网络情况	<=3S		
并发用户数与事务响应				
并发用户数	事务平均响应时间	事务最大响应时间	平均每秒处理事务数(TPS)	事务成功率
5	217ms	237ms	20.7	100%
10	219ms	227ms	44	100%
30	363ms	551ms	54.2	100%
50	427ms	677ms	73.4	100%
100	1024ms	1428ms	69.5	100%
并发用户数与服务器性能				
并发用户	CPU 利用率	内存利用率	磁盘 IO 情况	其他参数

数				
5	12.3	73.6	18kb/s	
10	12.9	73.3	25kb/s	
30	14.7	73.5	29kb/s	
50	16.8	74.1	35kb/s	
100	19.6	74.8	43kb/s	

表 7-2 查看宝塔系统防火墙

用例名称	查看宝塔系统防火墙			
用例编号	test002			
用例描述	登录的 TPS 达到 20 时，响应时间在 3S 内			
前置条件	用户及相关数据配置完成			
用例步骤	动作	期望的性能		
1	进入宝塔面板， 查看系统防火墙 情况	<=3S		
并发用户数与事务响应				
并发用户数	事务平均响应时间	事务最大响应时间	平均每秒处理事务数(TPS)	事务成功率
5	229ms	243ms	21.1	100%
10	231ms	246ms	41.9	100%
30	362ms	556ms	54.1	100%
50	426ms	690ms	70.1	100%
100	1015ms	1398ms	72.2	100%
并发用户数与服务器性能				
并发用户数	CPU 利用率	内存利用率	磁盘 IO 情况	其他参数
5	12.6	73.2	18kb/s	
10	12.7	73.4	24kb/s	
30	13.9	74.2	28kb/s	
50	15.6	74.5	36kb/s	
100	18.9	74.9	43kb/s	

二、GPT 后台测试执行

对 GPT 后台进行性能测试，在测试之后查看 GPT 后台线程组的结果树，结果如图 7-5 所示。

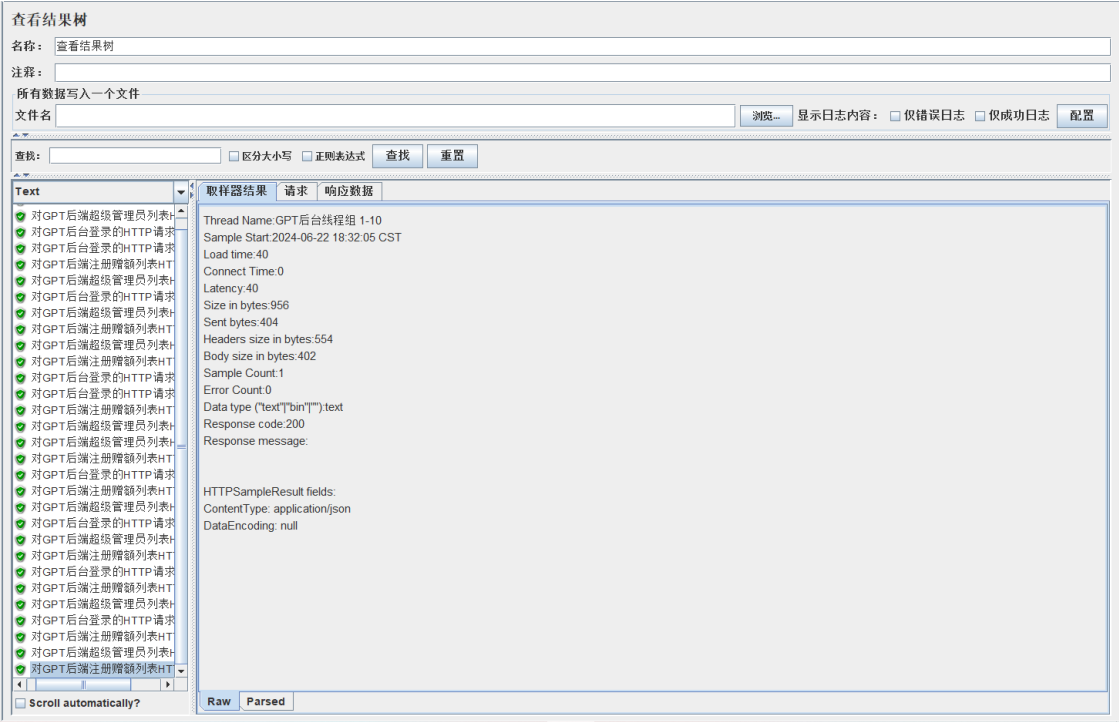


图 7-5 查看 GPT 后台的结果树

分别在指定并发数下进行执行，将执行后的聚合报告情况和服务器性能情况进行记录，记录情况如表 7-3~表 7-4 所示。

表 7-3 登录 GPT 后台

用例名称	登录 GPT 后台			
用例编号	test003			
用例描述	登录的 TPS 达到 20 时，响应时间在 3S 内			
前置条件	用户及相关数据配置完成			
用例步骤	动作	期望的性能		
1	进入登录页面， 输入账号密码， 点击登录	<=3S		
并发用户数与事务响应				
并发用户数	事务平均响应时间	事务最大响应时间	平均每秒处理事务数 (TPS)	事务成功率
5	211ms	230ms	4.7	100%
10	196ms	215ms	9.6	100%
30	379ms	719ms	28	100%
50	380ms	669ms	36.4	100%
100	1532ms	2322ms	39.8	100%
并发用户数与服务器性能				
并发用户数	CPU 利用率	内存利用率	磁盘 IO 情况	其他参数
5	12.3	73.6	18kb/s	

10	12.9	73.3	28kb/s	
30	14.7	73.5	30kb/s	
50	16.8	74.1	32kb/s	
100	19.6	74.8	45kb/s	

表 7-4 查询 GPT 后台注册赠额

用例名称	查询 GPT 后台注册赠额			
用例编号	test004			
用例描述	登录的 TPS 达到 20 时，响应时间在 3S 内			
前置条件	用户及相关数据配置完成			
用例步骤	动作	期望的性能		
1	进入主页，查看后台注册赠额信息	<=3S		
并发用户数与事务响应				
并发用户数	事务平均响应时间	事务最大响应时间	平均每秒处理事务数 (TPS)	事务成功率
5	199ms	228ms	5.4	100%
10	230ms	265ms	8.9	100%
30	384ms	721ms	28.6	100%
50	420ms	697ms	36.4	100%
100	1603ms	2563ms	39.8	100%
并发用户数与服务器性能				
并发用户数	CPU 利用率	内存利用率	磁盘 IO 情况	其他参数
5	12.6	73.4	14kb/s	
10	13.1	73.3	26kb/s	
30	14.5	73.8	29kb/s	
50	16.3	74.3	37kb/s	
100	18.8	74.9	44kb/s	

三、员工管理系统项目测试执行

对员工管理系统进行性能测试，在测试之后查看员工管理系统线程组的结果树，结果如图 7-6 所示。

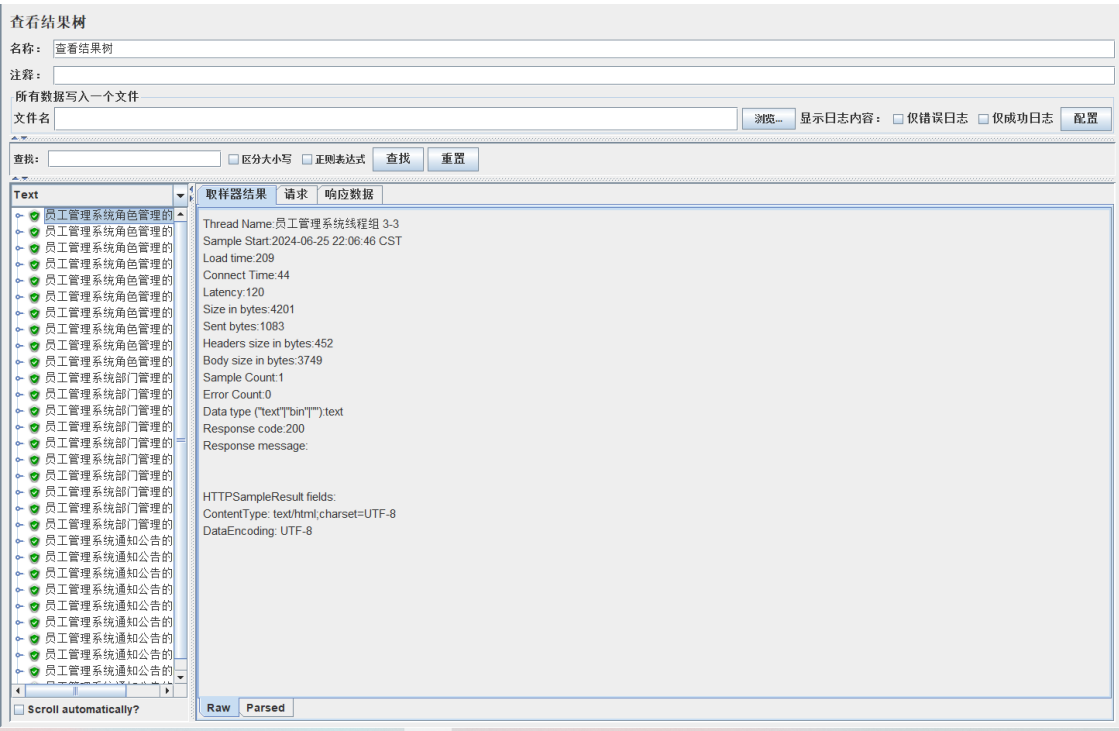


图 7-6 查看员工管理系统的结果树

分别在指定并发数下进行执行，将执行后的聚合报告情况和服务器性能情况进行记录，记录情况如表 7-5 所示。

表 7-5 查看员工管理系统部门管理

用例名称	查看员工管理系统部门管理			
用例编号	test005			
用例描述	登录的 TPS 达到 20 时，响应时间在 3S 内			
前置条件	用户及相关数据配置完成			
用例步骤	动作	期望的性能		
1	进入员工管理系统，查看系统部门管理信息	<=3S		
并发用户数与事务响应				
并发用户数	事务平均响应时间	事务最大响应时间	平均每秒处理事务数 (TPS)	事务成功率
5	119ms	157ms	31.6	100%
10	108ms	112ms	103.9	100%
30	130ms	172ms	177.9	100%
50	211ms	298ms	180.2	100%
100	270ms	459ms	216.8	100%
并发用户数与服务器性能				
并发用户数	CPU 利用率	内存利用率	磁盘 IO 情况	其他参数

5	13.2	73.3	17kb/s	
10	13.2	73.4	22kb/s	
30	13.9	73.8	27kb/s	
50	15.4	74.5	34kb/s	
100	17.9	74.9	48kb/s	

8 缺陷记录

项目名称: 智能集成测试综合实训

测试人员: [REDACTED]

测试时间: 2024 年 5 月 6 日-2024 年 6 月 22 日

系统: 宝塔面板、GPT 后台、员工管理系统项目

1. 宝塔面板缺陷记录

缺陷编号: BT-01
模块: 计划任务
优先级: 高
测试标题: 新建计划任务时未能拦截危险指令
预置条件: 用户已登录并进入计划任务页面
步骤描述:
点击侧边栏的计划任务菜单
点击“添加任务”按钮
点击“确定”按钮
测试数据:
任务类型: Shell 脚本
任务名称: FTP 审计日志切割任务
执行周期: 每天 0 小时 1 分钟
执行用户: root
脚本内容: shutdown -r now
预期结果: 不能保存, 因为其为危险指令, 不能保证每次重启的时候服务端没有正在运行的业务, 面板限制不能保存这样的计划
实际结果: 成功保存, 并且能查询到计划任务
测试版本号: v8.2.0

2. GPT 后台缺陷记录

缺陷编号: GPT-01
模块: 活动列表
优先级: 中
测试标题: 添加邀请活动时未能识别科学计数法

预置条件: 用户已登录并进入活动列表页面
步骤描述:
点击侧边栏的活动列表菜单
点击邀请注册栏的“配置”按钮
点击“确定”按钮
测试数据:
邀请人&被邀请人天数: 1e5
tokens: 2
普通聊天次数: 2
高级聊天次数: 2
AI 绘图次数: 2
类型: 总额
预期结果: 保存失败, 系统无法识别 1e5 这个数字的含义
实际结果: 成功保存, 系统识别 1e5 为 100000, 并且能查询到本条活动
测试版本号: v1.1.0

缺陷编号: GPT-02
模块: 面具管理
优先级: 高
测试标题: 新建面具管理列表时未能拦截负数话题新鲜度
预置条件: 用户已登录并进入面具管理页面
步骤描述:
点击侧边栏的面具管理菜单
点击面具管理栏的“新建”按钮
点击“确定”按钮
测试数据:
名称: 新建面具
语言: 中文
类型: 无
模型: gpt-3.5-turbo
随机性: -1
话题新鲜度: -1
预期结果: 保存失败, 话题新鲜度不能为负数
实际结果: 成功保存, 并且能查询到本条面具
测试版本号: v1.1.0

3. 员工管理系统缺陷记录

缺陷编号: RY-01
模块: 系统管理
优先级: 中
测试标题: 编辑用户列表时未能校验电话号码格式

预置条件: 用户已登录并进入系统管理页面
步骤描述:
点击侧边栏的系统管理菜单
点击系统管理菜单下的用户管理菜单
点击“编辑”按钮
测试数据:
用户名称: Amos_Liu
手机号码: 11451419198
登录账号: amos
邮箱: amos@exmaple.com
角色: 普通用户
预期结果: 保存失败, 不是正常的邮箱格式
实际结果: 成功保存, 并且能查询到本条用户信息
测试版本号: v4.7.8

缺陷编号: RY-02
模块: 系统管理
优先级: 高
测试标题: 编辑角色列表时未能校验权限字符的有效性
预置条件: 用户已登录并进入系统管理页面
步骤描述:
点击侧边栏的系统管理菜单
点击系统管理菜单下的角色管理菜单
点击“编辑”按钮
测试数据:
角色名称: 普通角色 1
权限字符: @RequiresRoles("1")
显示顺序: 1
预期结果: 保存失败, 因为没有这个权限
实际结果: 成功保存, 并且能查询到本条角色信息
测试版本号: v4.7.8

总结: 在本次测试中, 我们对宝塔面板、GPT 后台和员工管理系统进行了详细的功能测试, 发现并记录了多个系统缺陷。这些缺陷主要集中在输入校验、危险操作的拦截以及权限和数据有效性的验证上。针对这些问题, 建议开发团队尽快修复, 并在修复后进行回归测试, 确保系统的稳定性和安全性。

9 测试报告

一、概述

本项目包括三个系统：宝塔 Linux 面板、Rhine Assistant 控制台和员工管理系统。宝塔 Linux 面板是一款可视化 Linux 管理工具，Rhine Assistant 控制台是 GPT 聊天站的后台管理系统，员工管理系统是一个综合性管理系统。测试目的是确保这些系统的各项功能能够按照设计要求正常运行，发现并修复所有严重及以上级别的缺陷，确保核心业务流程的高可用性。

测试需求涵盖功能测试、接口测试、UI 测试、性能测试、安全测试、回归测试和自动化测试。测试目标是发现并修复所有严重及以上级别的缺陷，确保系统的稳定性和可靠性。

二、测试过程

1. 评审记录

在测试开始前，进行了详细的需求评审和设计评审，确保所有测试需求和设计方案符合预期。

2. 测试范围

功能测试：覆盖宝塔 Linux 面板、Rhine Assistant 控制台和员工管理系统的所有主要功能模块。

接口测试：涵盖宝塔 Linux 面板的核心接口。

UI 测试：重点测试了 Rhine Assistant 控制台和员工管理系统的用户界面。

性能测试：测试了宝塔 Linux 面板，GPT 控制台和员工管理系统的高频使用接口，模拟高负载环境下的系统响应速度和稳定性。

三、测试情况

1. 宝塔 Linux 面板：所有测试用例已测试通过，出现的问题已反馈；

2. Rhine Assistant 控制台：所有测试用例已测试通过，出现的问题已反馈；

3. 员工管理系统：所有测试用例已测试通过，出现的问题已反馈。

四、测试统计

1. 资源统计

测试人员：2 名

测试时间：2024 年 5 月 6 日至 2024 年 6 月 22 日

测试环境：Linux 服务器

2. 执行情况（所有测试人员总和）

测试用例总数：110

功能测试：用例总数：60；已执行：60；通过：50；未通过：10。

接口测试：用例总数：30；已执行：30；通过：30；未通过：0。

UI 测试：用例总数：10；已执行：10；通过：10；未通过：0。

性能测试：用例总数：10；已执行：10；通过：10；未通过：0。

3. 问题统计

严重缺陷：3

一般缺陷：4

小缺陷：2

4. 问题列表

（仅展示严重缺陷，其他缺陷见上一章节）

严重缺陷 1：宝塔 Linux 面板的安全模块新增端口时拦截功能设计有漏洞，可以新增负数的端口号，已反馈修复。

严重缺陷 2：Rhine Assistant 控制台的新增套餐功能存在范围性漏洞，无法正常排出-1 以下的数，并且对于如 1e5、0x3f 这类的非法输入无法进行拦截，已反馈修复。

严重缺陷 3：员工管理系统在系统管理操作时不能像用户管理时一样对手机号码是否正确做出正确判断，已反馈修复。

五、测试总结

本次测试涵盖了宝塔 Linux 面板、Rhine Assistant 控制台和员工管理系统的所有主要功能模块和核心接口。通过功能测试、接口测试、UI 测试、性能测试、和自动化测试，发现并提交修复了所有缺陷。

所有测试用例均已执行，覆盖率达到 100%。

所有严重和一般级别的缺陷均已修复，小缺陷有部分未修复，但不影响核心功能的正常运行。

通过本次测试，宝塔 Linux 面板、Rhine Assistant 控制台和员工管理系统的各项功能、接口和性能均达到了设计要求。系统稳定性和可靠性得到了有效验证，可以投入实际使用。

10 总结

在本次测试项目中，我们针对宝塔面板、GPT 后台管理系统和员工管理系统

进行了全面的功能测试、接口自动化测试、UI 自动化测试以及性能测试。这些测试活动旨在确保系统的稳定性、可靠性和用户体验。

在功能测试部分，我们通过详细的用例设计，涵盖了宝塔面板的任务管理、安全管理、GPT 后台的活动管理、用户管理及员工管理系统的用户管理等核心模块。在功能测试过程中，发现了一些关键缺陷，如负数处理、输入校验、权限管理等问题，并及时记录和反馈，确保开发团队能够迅速响应和修复。

在接口自动化测试部分，我们用 Python 脚本进行接口自动化测试，覆盖了主要的系统管理接口。通过自动化调用和结果校验，有效地验证了接口的稳定性和正确性，而且还成功使用接口自动化测试内容成功搭建数据库并部署员工管理系统的 SpringBoot 项目和 GPT 后台的 Docker 项目。

在 UI 自动化测试部分，我们用 Selenium 框架进行 UI 自动化测试。其中重点测试了系统的登录功能、用户管理功能等。通过模拟用户操作，确保了系统界面的易用性和稳定性。测试结果表明，系统的 UI 功能流畅，满足用户的使用需求。

在性能测试部分，性能测试主要针对高并发场景下系统的响应时间和稳定性进行评估。通过 JMeter 工具，我们模拟了不同并发用户数下的系统行为，找到了系统在高负载条件下的性能瓶颈，并提出了相应的优化建议。

此次测试覆盖面广，涉及功能、接口、UI 和性能多个方面。通过系统的测试活动，我们发现并记录了多个系统缺陷，提出了优化建议，为后续的系统优化和稳定性提升提供了有力支持。建议开发团队根据测试报告中的问题，及时修复并进行回归测试，确保系统在上线后能够稳定、高效地运行。

总之，测试作为软件开发的重要环节，通过系统化、规范化的测试活动，不仅保障了软件质量，也为软件项目的成功奠定了坚实基础。未来，我们将继续完善测试流程和方法，不断提升测试效率和效果，为软件开发和运维提供更强有力的支持。

11 参考文献

- [1] 李正言. 计算机软件测试方法的研究[J]. 自动化应用, 2024, 65(02): 199-201. DOI: 10.19769/j.zdhy.2024.02.062.
- [2] 范飞燕. 浅谈基于模块化的自动化软件测试框架技术[J]. 长江信息通信, 2023, 36(11): 102-104.
- [3] 仇振安, 王锋, 任志伟. 软件测试的难点及应对策略[J]. 电光与控制, 2023, 30(09): 106-111.

- [4] 林生旭,盘茂杰.软件测试技术及其测试工具的研究与应用[J].现代计算机,2023,29(12):37-43.
- [5] Memon P,Bhatti S,Dewani A, et al.Scrutinizing Automated Load Testing Via Blazmeter, Apache Jmeter and HP Loadrunner[J].International Journal of Advanced Studies in Computer Science and Engineering,2018,7(3TB):35-42.

附录1

ID	模块	优先级	测试标题	预置条件	步骤描述	测试数据	预期结果	测试结果	测试版本号	测试人员	备注
BT01	计划任务	低	查看计划任务	登陆成功	1、点击侧边栏的计划任务菜单；		查看成功，页面显示所有计划任务信息	查看成功，页面显示所有计划任务信息	v8.2.0		无
BT02	计划任务	低	新建计划任务	登陆成功	1、点击侧边栏的计划任务菜单； 2、点击“添加任务”按钮； 3、点击“确定”按钮	任务类型 Shell脚本 任务名称 FTP审计日志切割任务 执行周期 每天0小时1分钟 执行用户 root 脚本内容 /usr/bin/btpython /www/server/panel/script/ftplogs_cut.py	成功保存，并且能查询到计划任务	成功保存，并且能查询到计划任务	v8.2.0		无
BT03	计划任务	高	新建计划任务	登陆成功	1、点击侧边栏的计划任务菜单； 2、点击“添加任务”按钮； 3、点击“确定”按钮	任务类型 Shell脚本 任务名称 FTP审计日志切割任务 执行周期 每天0小时1分钟 执行用户 root 脚本内容 shutdown -r now	不能保存，因为其危险指令，不能保证每次重启的时候服务端没有正在运行的业务，面板限制不能保存这样的计划	成功保存，并且能查询到计划任务	v8.2.0		危急



BT04	计划任务	低	修改计划任务	登陆成功	1、点击侧边栏的计划任务菜单； 2、点击“编辑”按钮；	任务类型 Shell脚本 任务名称 FTP审计日志切割任务 执行周期 每天1小时1分钟 执行用户 root 脚本内容 /usr/bin/btpython /www/server/panel/script/ftptlogs_cut.py	成功保存，并且能查询到计划任务	成功保存，并且能查询到计划任务	v8.2.0		无
BT05	计划任务	低	执行计划任务	登陆成功	1、点击侧边栏的计划任务菜单； 2、点击“执行”按钮；		成功执行，并且能查询到执行结果	成功执行，并且能查询到执行结果	v8.2.0		无
BT06	安全	低	查看端口规则	登陆成功	1、点击侧边栏的安全菜单；		查看成功，页面显示所有端口规则信息	查看成功，页面显示所有端口规则信息	v8.2.0		无
BT07	安全	低	添加端口规则	登陆成功	1、点击侧边栏的安全菜单； 2、点击“添加端口规则”按钮； 3、点击“确定”按钮	协议 TCP 端口 1234 来源 所有IP 方向 入站（默认） 备注 添加端口1234	成功保存，并且能查询到本端口规则	成功保存，并且能查询到本端口规则	v8.2.0		无



BT08	安全	低	添加端口规则	登陆成功	1、点击侧边栏的安全菜单; 2、点击“添加端口规则”按钮; 3、点击“确定”按钮	协议TCP 端口0 来源所有IP 方向入站（默认） 备注 添加端口0	保存失败，端口输入范围不正确，请重新输入	保存失败，端口输入范围不正确，请重新输入	v8.2.0		无
GPT01	登录	低	登录	未登录	1、进入GPT后台管理页面; 2、点击“登录”按钮	用户名 ratest 密码 ratest123456	登录成功	登录成功	v1.1.0		无
GPT02	会员列表	低	查看活动列表	登陆成功	1、点击侧边栏的会员列表菜单;		查看成功，页面显示所有会员信息	查看成功，页面显示所有会员信息	v1.1.0		无
GPT03	活动列表	低	查看活动列表	登陆成功	1、点击侧边栏的活动列表菜单;		查看成功，页面显示所有活动	查看成功，页面显示所有活动	v1.1.0		无

GPT04	活动列表	低	添加邀请活动	登陆成功	1、点击侧边栏的活动列表菜单； 2、点击邀请注册栏的“配置”按钮； 3、点击“确定”按钮	邀请人&被邀请人 天数 1 tokens 1 普通聊天此处 1 高级聊天次数 1 AI绘图次数 1 类型 总额	成功保存，并且能查询到本条活动	成功保存，并且能查询到本条活动	v1.1.0		无
GPT05	活动列表	低	添加邀请活动	登陆成功	1、点击侧边栏的活动列表菜单； 2、点击邀请注册栏的“配置”按钮； 3、点击“确定”按钮	邀请人&被邀请人 天数 -1 tokens -1 普通聊天此处 -1 高级聊天次数 -1 AI绘图次数 -1 类型 总额	成功保存，系统成功识别-1为无限制，并且能查询到本条活动	成功保存，系统成功识别-1为无限制，并且能查询到本条活动	v1.1.0		无



GPT06	活动列表	低	添加邀请活动	登陆成功	1、点击侧边栏的活动列表菜单； 2、点击邀请注册栏的“配置”按钮； 3、点击“确定”按钮	邀请人&被邀请人天数 -2 tokens -1 普通聊天此处 -1 高级聊天次数 -1 AI绘图次数 -1 类型 总额	保存失败，系统无法保存小于-1的整数	保存失败，系统无法保存小于-1的整数	v1.1.0		无
GPT07	活动列表	中	添加邀请活动	登陆成功	1、点击侧边栏的活动列表菜单； 2、点击邀请注册栏的“配置”按钮； 3、点击“确定”按钮	邀请人&被邀请人天数 1e5 tokens 2 普通聊天此处 2 高级聊天次数 2 AI绘图次数 2 类型 总额	保存失败，系统无法识别1e5这个数字的含义	成功保存，系统识别1e5为100000，并且能查询到本条活动	v1.1.0		修复



GPT08	活动列表	中	添加邀请活动	登陆成功	1、点击侧边栏的活动列表菜单； 2、点击邀请注册栏的“配置”按钮； 3、点击“确定”按钮	邀请人&被邀请人天数 0x3f tokens 2 普通聊天此处 2 高级聊天次数 2 AI绘图次数 2 类型 总额	保存失败，系统无法识别0x3f这个数字的含义	成功保存，系统识别0x3f为63，并且能查询到本条活动	v1.1.0		修复
GPT09	管理员列表	低	查看管理员列表	登陆成功	1、点击侧边栏的管理员列表菜单；		查看成功，页面显示所有管理员列表	查看成功，页面显示所有管理员列表	v1.1.0		无
GPT10	套餐列表	低	查看套餐列表	登陆成功	1、点击侧边栏的套餐列表菜单；		查看成功，页面显示所有套餐列表	查看成功，页面显示所有套餐列表	v1.1.0		无

GPT11	套餐列表	低	添加套餐	登陆成功	1、点击侧边栏的套餐列表菜单; 2、点击套餐列表栏的“新建”按钮; 3、点击“确定”按钮	标题 套餐1 副标题 套餐1 类型 总额 tokens 1 普通聊天次数 (GPT3.5) 1 高级聊天次数 (GPT4) 1 价格 (元) 1	成功保存, 并且能查询到本条套餐	成功保存, 并且能查询到本条套餐	v1.1.0		无
GPT12	套餐列表	低	修改套餐	登陆成功	1、点击侧边栏的套餐列表菜单; 2、点击套餐列表栏的“编辑”按钮; 3、点击“确定”按钮	标题 套餐1 副标题 套餐1 类型 总额 tokens -1 普通聊天次数 (GPT3.5) -1 高级聊天次数 (GPT4) -1 价格 (元) 1	成功保存, 系统成功识别-1为无限制, 并且能查询到本条活动	成功保存, 系统成功识别-1为无限制, 并且能查询到本条活动	v1.1.0		无

GPT13	套餐列表	高	修改套餐	登陆成功	1、点击侧边栏的套餐列表菜单； 2、点击套餐列表栏的“编辑”按钮； 3、点击“确定”按钮	标题 套餐1 副标题 套餐1 类型 总额 tokens -1 普通聊天次数 (GPT3.5) -1 高级聊天次数 (GPT4) -1 价格（元） -1	保存失败，价格不能是负数	成功保存，并且能查询到本条活动	v1.1.0		危急
GPT14	套餐列表	高	修改套餐	登陆成功	1、点击侧边栏的套餐列表菜单； 2、点击套餐列表栏的“编辑”按钮； 3、点击“确定”按钮	标题 套餐1 副标题 套餐1 类型 总额 tokens -2 普通聊天次数 (GPT3.5) -1 高级聊天次数 (GPT4) -1 价格（元） 1	保存失败，系统无法保存小于-1的整数	成功保存，并且能查询到本条活动，其中tokens数为-2	v1.1.0		危急

RY01	登录	低	登录	未登录	1、进入若依管理页面; 2、点击“登录”按钮	用户名 admin 密码 admin123	登录成功	登录成功	v4.7.8		无
RY02	首页	低	查看首页	登录成功	1、点击侧边栏的首页菜单;		查看成功, 页面显示首页介绍内容	查看成功, 页面显示首页介绍内容	v4.7.8		无
RY03	系统管理	低	查看用户列表	登录成功	1、点击侧边栏的系统管理菜单; 2、点击系统管理菜单下的用户管理菜单;		查看成功, 页面显示所有用户信息	查看成功, 页面显示所有用户信息	v4.7.8		无

RY04	系统管理	低	新增用户列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的用户管理菜单； 3、点击“新增”按钮 4、点击“保存”按钮	用户名称 Amos_Liu 手机号码 15135461481 登录账号 amos 用户性别 男 岗位 人力资源 归属部门 财务部门 邮箱 liuamos866@gmail.com 密码 amos123	成功保存，并且能查询到本条用户信息	成功保存，并且能查询到本条用户信息	v4.7.8		无
RY05	系统管理	低	编辑用户列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的用户管理菜单； 3、点击“编辑”按钮	用户名称 Amos_Liu 手机号码 1145141919810 登录账号 amos 用户性别 男 岗位 人力资源 归属部门 财务部门 邮箱 liuamos866@gmail.com 密码 amos123	保存失败，手机号码长度超出限制	保存失败，手机号码长度超出限制	v4.7.8		无



RY06	系统管理	低	编辑用户列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的用户管理菜单； 3、点击“编辑”按钮	用户名称 Amos_Liu 手机号码 11451419198 登录账号 amos 用户性别 男 岗位 人力资源 归属部门 财务部门 邮箱 liuamos866@gmail.com 密码 amos123	保存失败，不是正确的手机号码格式	保存失败，不是正确的手机号码格式	v4.7.8		无
RY07	系统管理	低	编辑用户列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的用户管理菜单； 3、点击“编辑”按钮	用户名称 Amos_Liu 手机号码 15135461481 登录账号 amos 用户性别 男 岗位 人力资源 归属部门 财务部门 邮箱 123 密码 amos123	保存失败，不是正确的邮箱格式	保存失败，不是正确的邮箱格式	v4.7.8		无

RY08	系统管理	中	编辑用户列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的用户管理菜单； 3、点击“编辑”按钮	用户名称 Amos_Liu 手机号码 11451419198 登录账号 amos 用户性别 男 岗位 人力资源 归属部门 财务部门 邮箱 1@1.com 密码 amos123	保存失败，不是正常的邮箱格式	成功保存，并且能查询到本条用户信息	v4.7.8		完善
RY09	系统管理	低	查看角色列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的角色管理菜单；		查看成功，页面显示所有角色信息	查看成功，页面显示所有角色信息	v4.7.8		无
RY10	系统管理	低	新增角色列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的角色管理菜单； 3、点击“新增”按钮 4、点击“保存”按钮	角色名称 普通角色1 权限字符 @RequiresRoles("system:role:add") 显示顺序 3 备注 普通角色 菜单权限 父子联动	成功保存，并且能查询到本条用户信息	成功保存，并且能查询到本条用户信息	v4.7.8		无

RY11	系统管理	低	编辑角色列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的角色管理菜单； 3、点击“编辑”按钮	角色名称 普通角色1 权限字符 admin 显示顺序 3 备注 普通角色 菜单权限 父子联动	成功保存，并且能查询到本条用户信息	成功保存，并且能查询到本条用户信息	v4.7.8		无
RY12	系统管理	高	编辑角色列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的角色管理菜单； 3、点击“编辑”按钮	角色名称 普通角色1 权限字符 @RequiresRoles("1") 显示顺序 3 备注 普通角色 菜单权限 父子联动	保存失败，因为没有这个权限	成功保存，并且能查询到本条用户信息	v4.7.8		危急
RY13	系统管理	低	查看部门列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的部门管理菜单；		查看成功，页面显示所有部门信息	查看成功，页面显示所有部门信息	v4.7.8		无

RY14	系统管理	低	新增部门列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的角色管理菜单； 3、点击“新增”按钮 4、点击“保存”按钮	上级部门 若依科技 部门名称 研发部门1 显示排序 2 负责人 Amos 联系电话 15135461481 邮箱 liuamos866@gmail.com 部门状态 正常	成功保存，并且能查询到本条部门信息	成功保存，并且能查询到本条部门信息	v4.7.8		无
RY15	系统管理	低	编辑部门列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的角色管理菜单； 3、点击“编辑”按钮	上级部门 若依科技 部门名称 研发部门1 显示排序 2 负责人 Amos 联系电话 15135461481 邮箱 liuamos866@gmail.com 部门状态 停用	成功保存，并且能查询到本条部门信息	成功保存，并且能查询到本条部门信息	v4.7.8		无

RY16	系统管理	中	编辑部门列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的角色管理菜单； 3、点击“编辑”按钮	上级部门 若依科技 部门名称 研发部门1 显示排序 2 负责人 Amos 联系电话 1145141919810 邮箱 liuamos866@gmail.com 部门状态 停用	保存失败，联系电话长度不正确	成功保存，并且能查询到本条部门信息	v4.7.8		修复
RY17	系统管理	低	查看岗位列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的岗位管理菜单；		查看成功，页面显示所有岗位信息	查看成功，页面显示所有岗位信息	v4.7.8		无
RY18	系统管理	低	新增岗位列表	登录成功	1、点击侧边栏的系统管理菜单； 2、点击系统管理菜单下的岗位管理菜单； 3、点击“新增”按钮 4、点击“保存”按钮	岗位名称 董事长 岗位编码 ceo 显示顺序 1 岗位状态 正常	成功保存，并且能查询到本条岗位信息	成功保存，并且能查询到本条岗位信息	v4.7.8		无

RY19	系统监控	低	查看在线用户列表	登录成功	1、点击侧边栏的系统监控菜单； 2、点击系统管理菜单下的在线用户菜单；		查看成功，页面显示所有在线用户信息	查看成功，页面显示所有在线用户信息	v4.7.8		无
RY20	系统监控	低	强行踢出在线用户	登录成功	1、点击侧边栏的系统监控菜单； 2、点击系统管理菜单下的在线用户菜单； 3、点击“强退”按钮	会话编号 http://117.50.195.127:6343/monitor/online### 登录名称 admin 部门名称 研发部门	强退成功，显示已经踢出	强退成功，显示已经踢出	v4.7.8		无

附录2

用例描述	本地文件路径	远程文件名	是否通过	备注
添加数据库表文件1	C:\Users\12972\quartz.sql	quartz.sql	文件上传成功	本地文件上传
添加数据库表文件2	C:\Users\12972\ry_20240112.sql	ry_20240112.sql	文件上传成功	本地文件上传
添加数据库表文件3	C:\Users\12972\rhassistant.sql	rhassistant.sql	文件上传成功	本地文件上传
添加员工管理系统jar包	C:\Users\12972\origin.jar	origin.jar	文件上传成功	本地jar包上传
添加GPT后台Docker文件	C:\Users\12972\docker-compose.yml	docker-compose.yml	文件上传成功	本地文件上传

附录3

用例描述	接口地址	请求参数	是否通过	备注
获取磁盘信息	/system?action=GetDiskInfo		查询成功	无
添加ftp	/ftp?action=AddUser	{ "ftp_username": "francis", "ftp_password": "Y6xfDFBWxspNpjFn", "path": "/www/wwwroot/francis", "ps": "francis" }	添加成功	无
查询当前ftp列表	/datalist/data/get_data_list	{ "p": 1, "limit": 10, "search": "", "table": "ftps" }	查询成功	无
增加新数据库	/database?action=AddDatabase	{ "name": "rytest1", "db_user": "amos1", "password": "root", "dataAccess": "%", "address": "%", "codeing": "utf8mb4", "dtype": "MySQL", "ps": "rytest1", "sid": 0, "listen_ip": "0.0.0.0/0", "host": "" }	添加成功	无

添加表1	/database?action=InputSql	{ "file": "/www/backup/database/quartz.sql", "name": "rytest1" }	添加成功	和数据库名对应
添加表2	/database?action=InputSql	{ "file": "/www/backup/database/ry_20240112.sql", "name": "rytest1" }	添加成功	和数据库名对应
查询数据库	/datalist/data/get_data_list	{ "p": 1, "limit": 10, "table": "databases", "search": "", "order": "" }	查询成功	无
添加jdk	/project/java/install_jdk_new	{ "name": "jdk", "version": "jdk-11.0.19", "type": 1 }	添加成功	无
查询jdk信息	/project/java/get_local_jdk_version		查询成功	无

添加项目	/project/java/create_project	{ "project_type": 3, "project_jar": "/www/wwwroot/Amos/ruoyi-admin.jar", "project_name": "ruoyi1", "port": "6343", "project_jdk": "/www/server/java/jdk1.8.0_371/bin/java", "project_cmd": "/www/server/java/jdk1.8.0_371/bin/java -jar -Xmx1024M -Xms256M /www/wwwroot/Amos/ruoyi-admin.jar --server.port=6343", "nohup_log": true, "jmx_status": false, "run_user": "www", "use_project_watch": false, "auth": false, "project_ps": "ruoyi-admin.jar", "domains": [], "jmx_port": "java", "debug": false, "is_separation": false, "release_firewall": true, "api_url": "", "host_url": "http://127.0.0.1:6343", "static_path": "/www/wwwroot/Amos" }	添加成功	无
查询项目信息	/project/java/get_project_list	{ "p": 1, "limit": 10, "type_id": "", "search": "" }	查询成功	无
设置项目启动	/project/java/start_project	{ "project_name": "ruoyi1" }	设置成功	和项目名对应