计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(6):91−98 [doi: 10.15888/j.cnki.csa.008950]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

# 支持用例集并行测试的接口测试平台①

耿嘉祺, 李鑫丽, 祝小兰

(青海大学 计算机技术与应用系, 西宁 810016)
通信作者: 祝小兰, E-mail: zxlanscu@126.com

**摘　要**: 针对手工测试成本高、时效性弱和传统的接口自动化测试工具扩展能力不足问题, 提出了一个支持用例集并行测试的接口测试平台 (OLa). OLa 采用分层架构模式将系统分为用户展示层、应用逻辑层、数据服务层和用例执行层. 其中, 用户展示层基于 Vue 框架开发, 结合 Vue Router、Vuex 等工具实现单页应用; 应用逻辑层基于 Spring Boot 框架开发; 数据服务层基于 MyBatis-Plus 框架和 Spring Data 框架开发; 用例执行层使用 okhttp3、fastjson、jackson 等工具实现接口测试. 此外, 基于系统技术架构、Java 网络编程和面向抽象编程的思想, 创新性地提出了基于 C/S 模式的用例执行流程和基于参数识别的自动匹配校验方法, 解决了传统的自动化测试工具无法支持并行测试的问题. 实验结果表明本文设计与实现的 OLa 接口测试平台支持单用例测试、用例集的串行测试和并行测试, 能够自动识别用例参数并对接口响应内容进行校验, 提高了接口测试的灵活性和有效性, 降低了测试难度, 并能够在用例之间无相互依赖的情况下提高接口测试的效率.

**关键词**: 单用例测试; 用例集测试; 串行测试; 并行测试; 接口测试平台

引用格式: 耿嘉祺,李鑫丽,祝小兰.支持用例集并行测试的接口测试平台.计算机系统应用,2023,32(6):91–98. http://www.c-s-a.org.cn/1003-3254/8950.html

## Interface Test Platform Supporting Parallel Testing of Test Case Sets

GENG Jia-Qi, LI Xin-Li, ZHU Xiao-Lan

(Department of Computer Technology and Applications, Qinghai University, Xining 810016, China)

**Abstract**: Considering the high cost and poor timeliness of manual testing and the insufficient scalability of conventional automatic interface testing tools, an interface test platform (OLa) that supports the parallel testing of test case sets is proposed in this study. OLa is divided into the user presentation layer, application logic layer, data service layer, and test case execution layer according to a layered architecture model. Among them, the user presentation layer is developed on the basis of the Vue framework, which is combined with Vue Router, Vuex, and other tools to realize the single-page application. The application logic layer is realized with the Spring Boot framework, and the data service layer is based on the MyBatis-Plus framework and Spring Data framework. The test case execution layer uses okhttp3, fastjson, Jackson, and other tools to implement interface testing. In addition, according to the ideas of the technical architecture of systems, Java network programming, and abstraction-oriented programming, this study innovatively proposes the test case execution process with the C/S architecture and the automatic matching verification method based on parameter identification, which can solve the problem that some traditional automatic testing tools cannot support parallel testing. The experimental results show that OLa can support test case testing, serial testing, and parallel testing of test case sets and can automatically identify test case parameters and verify the interface response content, which improves the flexibility and effectiveness of interface testing. Moreover, it can reduce the difficulty of interface testing and improve the

① 基金项目: 国家自然科学基金 (61762074); 四川省科技厅重点研发项目 (2020YFS0575); 青海大学教育教学研究项目 (JY202127)
收稿时间: 2022-07-19; 修改时间: 2022-08-15; 采用时间: 2022-08-25; csa 在线出版时间: 2023-04-25
CNKI 网络首发时间: 2023-04-27

efficiency of interface testing without interdependence between test cases.

**Key words**: test case testing; test case set testing; serial testing; parallel testing; interface test platform

随着互联网技术的飞速发展, 敏捷开发模式逐渐地流行和成熟. 在敏捷开发的每次迭代中都需要对以前交付的功能进行回归测试. 手工回归测试不仅成本高, 而且测试难度也较大, 因此敏捷开发对自动化测试技术提出了强烈的需求[1,2].

传统的接口自动化测试工具很大程度上为自动化测试提供了方便, 但大部分测试工具的扩展能力不足, 例如无法将测试结果生成 HTML 形式的测试报告后发送到指定的邮箱[3]、无法并行测试等.

处理上述问题的核心在于如何利用自动化测试高效地完成一些重复性较强的接口测试工作, 同时具有良好的可扩展性, 以确保产品能够满足用户的需求和期望从而提高软件的质量[4]. 文献 [2,5,6] 基于测试数据和脚本分离的思想, 提出了不同的接口自动化测试框架, 减少了测试时间, 有效提高了测试效率. 文献 [7,8] 提出了不同的接口自动化测试方法, 采用不同的方式优化了测试用例的生成过程; 文献 [9–11] 设计并实现了针对不同环境下的接口测试平台, 用户可以方便地在平台中完成测试用例的编写、执行及测试结果的查看.

现有的自动化测试工具主要包括 3 类[12]. 以 JUnit、TestNg 为代表的开源单元测试框架, 能够支持开发人员编写和运行单元测试用例, 用户需要编写白盒测试代码, 要求用户具有一定的编程能力; 以 Postman 为代表的接口自动化测试工具, 可用于编写、管理和执行 HTTP 接口的测试用例, 也需要用户编写测试代码, 且只能进行串行测试; 以 Selenium、Appium 为代表的图形用户界面测试工具, 可录制用户在界面上的操作, 因为要求系统必须有用户界面, 所以不适用于中台和基础子系统的测试. 另外, 由于系统界面往往变更频繁, 因此已完成的用例可能在下个版本就不适用, 导致维护用例的成本较高.

考虑到 GUI 测试维护成本高、局限性较大的问题, 从接口测试切入自动化测试是较为可行的办法, 主要原因是: 接口测试的投入产出比相对较高, 能够实现较高的自动化率; 接口体现了系统的行为, 接口测试能够覆盖系统的业务规则; 接口易于测试, 且需要时可并行测试; 接口测试的适用范围较广, 适用于前台、中台、基础子系统等多种类型的系统的测试.

因此, 本文提出并实现了一个接口测试平台 OLa. 同时, 创新性地提出了基于 C/S 模式的用例执行流程和基于参数识别的自动匹配校验方法, 解决了传统的自动化测试工具无法并行测试的问题, 提高了测试的效率和灵活性.

相比于传统的自动化测试工具, OLa 的优势主要包括: 用例集支持并行测试, 在用例之间无相互依赖的情况下, 测试效率较高; 平台中预设了多种校验方式和校验方法, 能够满足不同情况下的测试要求, 具有较高的测试灵活性; 平台支持将测试评估结果生成 HTML 形式的报告并发送至指定的邮箱, 以便其他人员可以及时、方便地查看测试报告.

OLa 的主要功能包括: 能够在图形化界面中录入用例的参数、校验方法和内容, 支持实时地查看测试结果; 预设了多种校验方式和校验方法, 用户可以选择不同的校验方法对同一用例进行测试; 将用例编排为用例集, 可用于回归测试、新需求验收等多种场景; 用例集支持串行测试和并行测试; 实现了不同用户的用例数据隔离, 同时用户也可以主动地将自己的用例集共享到团队的项目中供其他项目成员使用.

## 1 系统架构

OLa 平台的 Web 端采用分层的架构模式, 由用户展示层、应用逻辑层和数据服务层构成, 如图 1 所示.

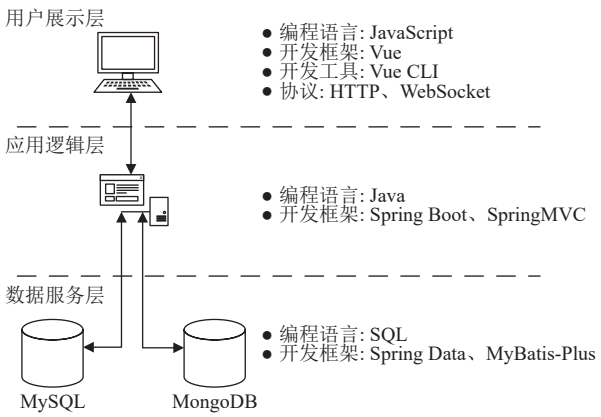


图 1 OLa 平台 Web 端的系统架构

用户展示层使用 Vue CLI 脚手架快速搭建 Vue 项目, 基于当下流行的前端开发框架 Vue 构建用户界面[13], 同时搭配 Vue Router 创建单页应用, 使用 Vuex 集中式存储和管理应用中所有组件的状态. Vue 组件库选用了 Ant Design Vue 和 Element UI, 在此基础之上结合 ECharts 图表库实现了数据的可视化. 在用例管理、团队和项目管理、请求消息发送模块中都使用了 Axios 库来发送异步请求, 以提高系统的及时反应能力[14]. 在测试客户端监控和测试客户端消息处理模块中采用了 HTML5 下的 WebSocket 技术, 实现了数据双向传输[15].

应用逻辑层基于 Spring Boot 开发, 由 Java 语言实现. Spring Boot 内嵌了 Tomcat 服务器, 通过 Maven 管理项目资源, 使得项目可以直接打成 jar 包, 从而简化了部署工作[16]. SpringMVC 框架是基于 MVC 设计模式的实现, 可以与 Spring Boot 框架无缝结合[17].

数据服务层基于 MyBatis-Plus 和 Spring Data 框架开发. MyBatis-Plus 内置 JDBC, 支持定制化 SQL、存储过程以及高级映射[18]. Spring Data 可以简化构建基于 Spring 框架应用的数据访问技术, 从而提高开发效率.

数据服务层中使用了关系型数据库 MySQL 以及非关系型数据库 MongoDB. 其中, MySQL 是一个开源的数据库管理系统, 具有性能高、跨平台的特点[19], 主要用于存储系统中的用例、用例集、团队和项目的基本信息以及中间表的数据; MongoDB 是一个面向文档存储的数据库[20], 主要用于存储系统中用例的请求信息.

## 2 用例执行流程

本文提出的基于 C/S 模式的用例执行流程支持自动地将测试请求分配给空闲的测试客户端、自动地将跨端请求分配给对应的请求处理器、自动地识别用例参数并对接口响应内容进行校验、实时地将测试结果呈现给用户. 用例执行流程的流程图如图 2 所示.

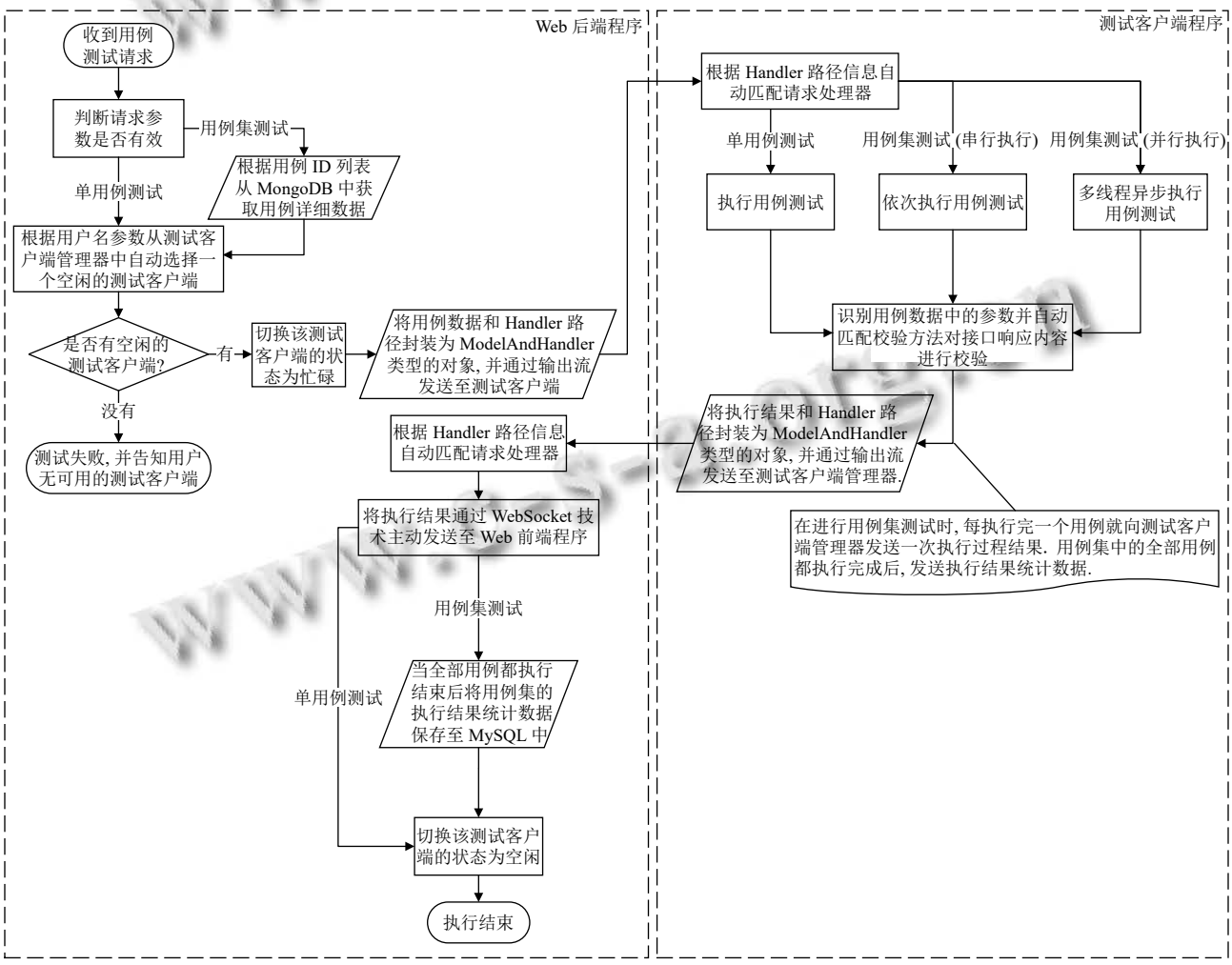


图 2 测试用例的执行流程图

在进行用例测试前, 需要启动至少一个测试客户端, 并成功连接至测试平台, 平台中的测试客户端管理器会为每一个连接成功的客户端分配一个线程进行通信.

### 2.1 单用例测试的执行流程

(1) 当用户点击浏览器页面中的发送测试按钮后, Web 前端程序会通过调用 JavaScript 生成为一个异步的 HTTP 请求, Web 后端程序接收到此请求后, 会从测试客户端管理器中自动选择该用户的一个空闲的测试客户端并通过输出流将执行用例所需的数据以及 Handler 路径封装为一个 ModelAndHandler 类型的对象发送至该客户端.

(2) 测试客户端收到测试请求后, 首先根据 Handler 路径信息自动匹配相应的请求处理器, 处理器负责执行用例并从测试客户端窗口中输出执行日志. 执行用例结束后测试客户端会将执行结果以及 Handler 路径封装为一个 ModelAndHandler 类型的对象发送至测试客户端管理器中与该客户端通信的线程, 该线程收到消息后, 首先根据 Handler 路径信息自动匹配相应的请求处理器, 然后通过 ServerEndpoint 中存储的用户的 WebSocket Session 将执行结果发送至 Web 前端程序. 此时, 前端程序中的 WebSocket 对象的 onmessage 事件会被触发, 从而接收到实时的执行结果, 最后调用 JavaScript 将执行结果分析并处理后呈现给用户.

### 2.2 用例集测试的执行流程

用例集测试的执行流程与单用例测试的执行流程基本相同, 不同之处如下.

(1) 执行用例集测试时, Web 后端程序首先需要根据测试请求内容中包含的用例 ID 列表从 MongoDB 中查询出对应的用例数据, 然后再请求用户的测试客户端执行用例集.

(2) 用例集中的全部用例都执行完成后, 测试客户端管理器需要调用 Web 后端服务将用例集的执行结果统计数据保存至 MySQL 中, 以便用户可以查询到用例集的历史执行结果.

用例集的执行方式有串行执行和并行执行两种. 串行执行是指按照用例在用例集中的位置顺序一个接着一个地执行; 并行执行是指通过多线程技术异步执行多个用例, 相比于接口测试工具 Postman 采用的串行测试而言, 并行测试能够降低用例集的执行耗时, 从而提高测试效率.

### 2.3 Web 后端程序与测试客户端之间的数据交互流程

Web 后端程序与测试客户端之间的数据交互流程参考了前端控制器设计模式, 如图 3 所示.

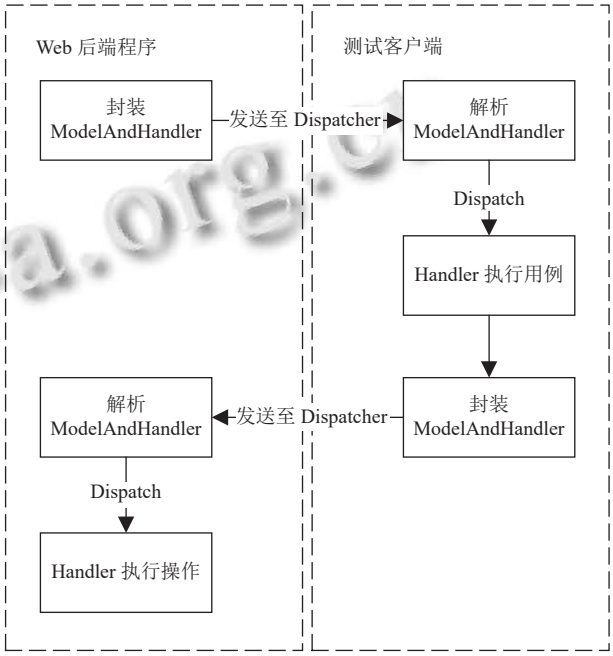


图 3 Web 后端程序与测试客户端之间的数据交互流程

在架构设计层面, 测试客户端独立于 Web 项目开发, 其主要功能是执行用例或用例集的同时在窗口中输出执行日志, 并将执行结果发送至 Web 后端程序. 测试客户端与 Web 后端程序之间通过 Socket 进行连接, 之后通过 Socket 的 IO 流进行通信.

(1) Web 后端程序向测试客户端发送测试请求之前, 会先将执行用例所需的数据以及 Handler 路径封装为一个 ModelAndHandler 类型的对象, 再将该对象通过输出流发送至测试客户端.

(2) 测试客户端中的 Dispatcher 通过输入流读取到 ModelAndHandler 类型的对象后, 会根据 Handler 路径信息自动匹配请求处理器, 同时将执行用例的任务分配给相应的 Handler. 执行用例结束后, 以相同的方式将 ModelAndHandler 类型的对象发送至 Web 后端程序.

(3) Web 后端程序中负责接收 ModelAndHandler 类型对象的 Dispatcher 也会先根据 Handler 路径信息自动匹配请求处理器, 再将通知执行结果的任务分配

给相应的 Handler.

这种设计的创新之处在于将 Web 后端业务逻辑与用例测试过程进行了充分的解耦, 使得测试客户端不但可以在服务器上运行, 也可以在用户电脑上运行, 从而不仅减少了 Web 服务器的压力, 而且提高了测试的灵活性. 用户只需要在电脑上运行测试客户端程序, 就可以开始测试本地应用.

## 3 基于参数识别的自动匹配校验方法

目前, 平台中提供了 3 种用于校验接口响应内容的方式, 每一种校验方式中包含一种或多种校验方法, 如图 4 所示. 对于某一个用例, 用户可以选择其中一种方法对接口的响应内容进行校验.

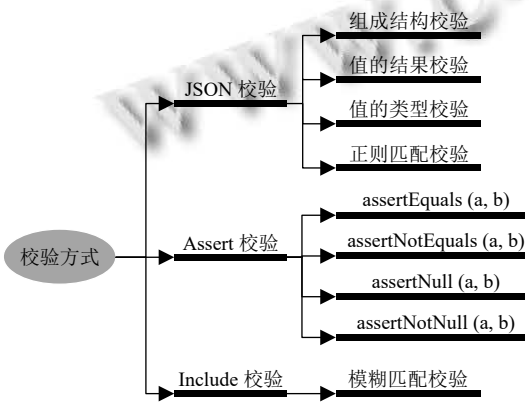


图 4 响应内容的校验方式

平台中提供的 3 种校验接口响应内容的方式分别是 JSON 校验、Assert 校验和 Include 校验. 其中, JSON 校验适用于响应内容是 JSON 格式的情况, 它可以校验响应内容的格式和组成结构、响应内容中值的结果和类型, 同时支持使用正则表达式对响应内容中的值进行匹配; Assert 校验适用于响应内容的格式比较简单, 但对结果的准确性要求较高的情况, 图 4 中列出了几个常用的断言方法; Include 校验适用于验证响应内容中是否包含指定的字符串的情况, 即对响应内容与指定的字符串进行模糊匹配.

用例测试请求的内容中包含两个用于自动匹配校验方式和校验方法的参数, 分别是校验方式的实现类名称和校验方法的编号.

(1) 在测试客户端的实现代码中使用了序列化和反序列化工具 jackson 中的@JsonTypeInfo 和@JsonSubTypes 两个注解用于识别请求内容中的参数、处理多态类型的序列化和反序列化, 从而达到自动匹配校验方式的目的.

(2) 校验方式匹配成功后, 再根据校验方法的编号自动选择相应的校验方法对响应内容进行校验. 这样做的好处是测试客户端在对响应内容进行校验时, 只需要调用校验类实例的校验入口方法, 即可获得校验结果, 而不需要关心抽象校验类型变量的具体子类型, 同时也便于扩展更多的校验方法.

## 4 系统实现

### 4.1 测试用例设计

测试用例的主要信息可由如下四元组表示:

$$< Identifier, Protocal, Init, Validation >$$

其中, *Identifier* 是用例编号, 唯一地标识一个用例; *Protocal* 是接口的协议类型; *Init* 是用例请求数据的初始化步骤; *Validation* 是响应校验, 按照预先设定的校验方法对接口的响应内容进行校验.

基于用例可以将用例集的主要信息表示如下:

$$\begin{cases} < TestCaseSet >::=< Ordered >< TestCaseItem > \\ < TestCaseItem >::=< TestCase >< Position > \\ Ordered ::= True|False \end{cases}$$

用例集是一组用例的集合, 其可以引用多个测试用例, 并指定其中用例的执行方式[12]. *Ordered* 默认为 *True*, 即串行执行, 适用于用例之间存在依赖的场景, 对于用例之间无相互依赖的场景, 可以将 *Ordered* 设置为 *False*, 即并行执行. *Position* 是用例在用例集中的位置, 用户可以在用例集测试页面中设置用例集中每个用例的位置.

### 4.2 单用例和用例集测试

用例是测试客户端执行测试的最小实体[21], 单用例和用例集测试基于 okhttp3、fastjson、jackson 等工具和 JavaScript 提供的 WebSocket 对象及其 API 实现.

#### 4.2.1 单用例测试

当用户在用例列表中选定某个用例后, Web 前端程序会根据用例的协议类型将浏览器页面切换到相应协议接口的编辑及测试视图, 如图 5 所示.

用户可以在此视图中完成请求信息和测试信息的录入、保存及执行用例测试等操作, 当用户输入有效的请求信息并点击发送按钮后, 平台就会开始执行该用例.

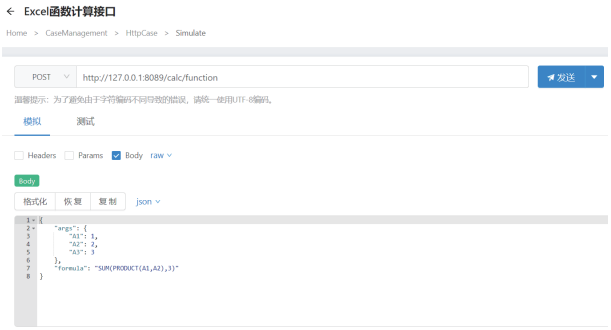


图 5 HTTP 用例编辑及测试视图

### 4.2.2 用例集测试

(1) 开始新的用例集测试前需要将待测的用例添加到用例集中. 为了记录每个用例在用例集中的位置, 在用例集与用例的中间表中增加了一个位置列, 表示用例在用例集中所处的位置, 向用例集中添加用例或更改用例在用例集中的位置时需要维护此列. 在用例集测试页面的编辑页签中可以搜索并添加用例至用例集、删除已添加的用例或调整已添加的用例的位置, 如图 6 所示.

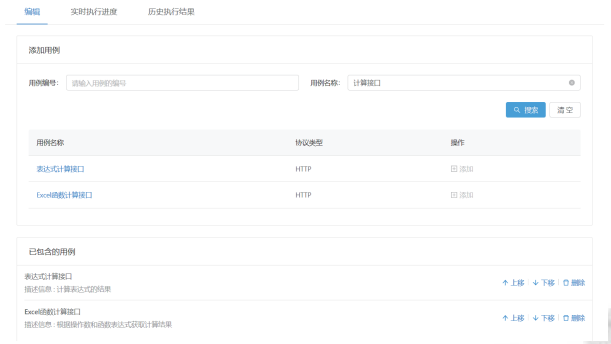


图 6 用例集测试页面的编辑页签

(2) 用户可以在用例集测试页面中选择用例集的执行方式 (串行执行或并行执行), 串行执行即一个用例执行完成后再执行下一个用例, 并行执行基于 JDK1.8 中引入的 CompletableFuture 类实现, 支持多个用例同时执行, 当所有用例都执行完成后, 返回执行结果的统计数据. CompletableFuture 类默认的执行器是 ForkJoinPool, 因此默认的并发线程数等于 Java 虚拟机可用的最大处理器数减一. 当用户选择执行方式并点击页面中的执行按钮后, 平台会按照用户选择的执行方式对用例集进行测试, 同时自动切换至实时执行进度页签, 如图 7 所示.

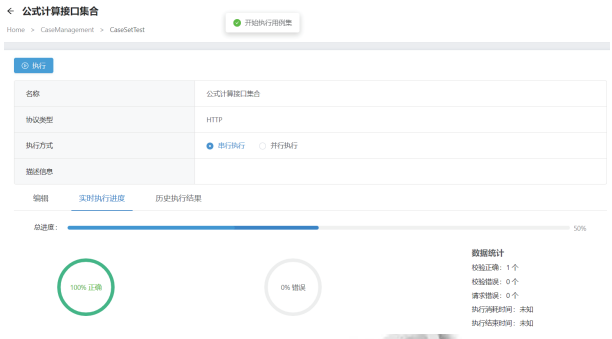


图 7 用例集测试页面的实时执行进度页签

(3) 用例集执行过程中, 用户可以在用例执行进度框内查看每个用例的执行进度和测试结果. 对于校验错误的用例, 用户可以将鼠标移至校验结果标签上查看提示信息, 方便用户了解校验错误的原因, 如图 8 所示.

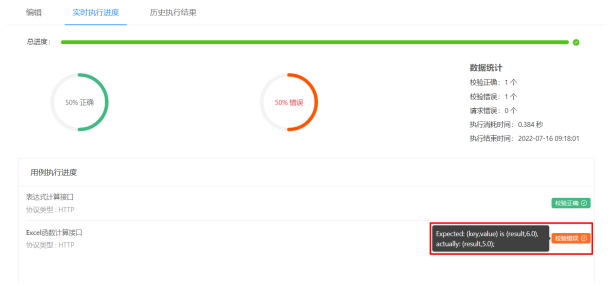


图 8 校验结果信息

## 4.3 实时监控和管理测试客户端

为了让用户可以实时地查看测试客户端的信息以及能够同时使用多个测试客户端执行用例或用例集, 通过 WebSocket 和 Callback (回调函数) 等技术实现了测试客户端的实时监控和管理.

(1) 当用户启动新的测试客户端并成功连接至服务器时, 负责与该客户端通信的线程会通过回调函数将用户名和客户端的信息发送至测试客户端管理器, 由其统一管理. 同时, 测试客户端管理器通过相应的 ServerEndpoint 将测试客户端上线的消息通知给 Web 前端程序, 前端程序收到通知后, 将数据更新并呈现给用户, 如图 9 所示.

(2) 当用户的某个测试客户端断开与服务器的连接时, 负责与该客户端通信的线程会通过回调函数将测试客户端下线的信息发送至测试客户端管理器, 测试客户端管理器会将原先存储的与该客户端相关的信息删除, 同时通过相应的 ServerEndpoint 将测试客户端下线的消息通知给 Web 前端程序. 前端程序收到通知

后, 将数据更新并呈现给用户.

除了直接关闭测试客户端以外, 平台中还提供了一个强制客户端下线的方式, 即通过点击浏览器页面中的下线按钮, 强制指定的测试客户端下线, 其基于 WebSocket 和 Axios 等技术实现.

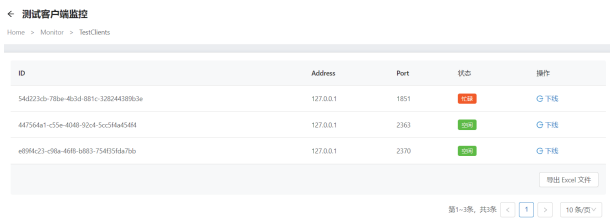


图 9 测试客户端监控视图

## 5 系统测试

为了验证接口测试平台 OLa 的有效性, 分别采用单用例测试和用例集测试对 OLa 平台的测试效率进行了评估.

### 5.1 单用例测试

本次实验中, 选用 Excel 函数计算接口和表达式计算接口进行单用例测试.

(1) Excel 函数计算接口的协议类型是 HTTP, URL 为“http://ip:port/calc/function”, 请求方法为“POST”, 功能是根据操作数和 Excel 函数获取计算结果. 设置该用例的内容类型 (content type) 为“application/JSON”, 操作数为“A1=1.0、A2=2.0、A3=3.0”, Excel 函数为“SUM(PRODUCT(A1, A2), 3)”, 即计算“A1×A2+A3”, 设置校验方法为“JSON 校验方式中的值的结果校验”, 设置校验内容为“5.0”, 设置响应超时时间为“10 s”, 测试结果如图 10 所示.

(2) 表达式计算接口的协议类型是 HTTP, URL 为“http://ip:port/calc/expression”, 请求方法为“GET”, 功能是根据操作数和表达式获取计算结果. 设置该用例的请求参数“variables”和“expression”分别为“$x$=2, $y$=3”和“7×#{$y$}+(163−5×2)/#{$x$}+sin(0)”, 设置校验方法为“Assert 校验方式中的 assertEquals 校验”, 设置校验内容为“97.5”, 设置响应超时时间为“10 s”, 测试结果如图 11 所示.

从图 10 和图 11 中可以看出, 当用例执行结束后, 平台会给出用例的测试结果, 如果测试正确则会给出正确提示, 否则会提示失败并告知失败的具体原因.

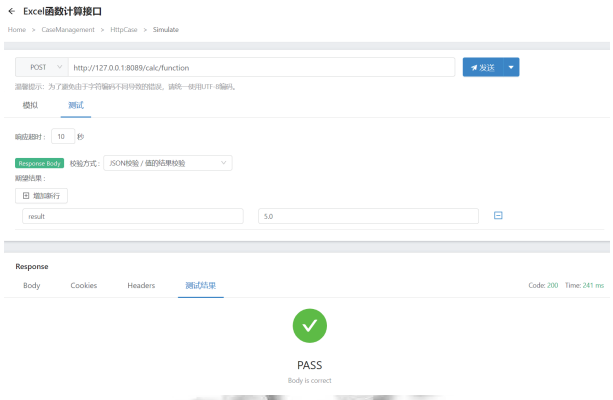


图 10 Excel 函数计算接口的测试结果

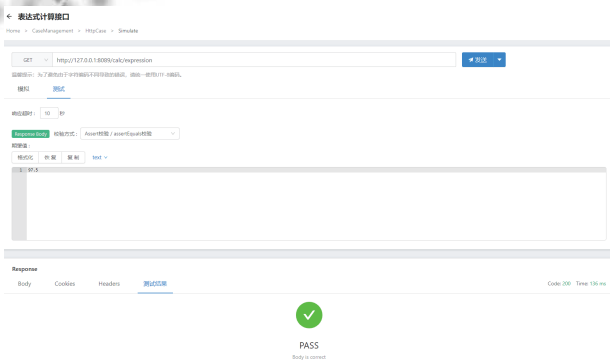


图 11 表达式计算接口的测试结果

### 5.2 用例集测试

本次实验中, 选用公式计算接口集合进行用例集测试, 该用例集中包含的用例为: Excel 计算接口和表达式计算接口.

在相同的测试环境下, 分别执行用例集中的用例各 1 次, 再使用串行执行和并行执行两种方式执行该用例集, 循环 10 次, 平均耗时如表 1 所示.

表 1 用例集的两种执行方式的测试耗时对比

| 用例/用例集名称 | 执行方式 | 平均耗时 (ms) |
|---|---|---|
| 表达式计算接口 | 执行一次 | 135 |
| Excel函数计算接口 | 执行一次 | 243 |
| 公式计算接口集合 | 串行执行 | 385 |
| 公式计算接口集合 | 并行执行 | 284 |

由于串行执行需要等待前一个用例执行结束后才能执行下一个用例, 而并行执行则支持多个用例同时执行, 在用例之间无相互依赖的场景下, 理论上执行效率会高于串行执行. 在本次实验中, 用例集采用并行执行的方式比串行执行在时间上平均缩短了约 26.2%.

通过以上实验证明, 本文实现的接口测试平台 OLa 具有一定的实用性, 提出的用例执行流程和自动匹配校验方法可行, 不仅能够降低接口测试的难度, 而且可以根据实际情况选择不同的用例执行方式, 提高了用例集测试的灵活性.

本文 OLa 与现有的自动化测试工具的对比结果如表 2 所示.

表 2 与现有的自动化测试工具的对比结果

| 名称 | 测试难度 | 适用范围 | 用例可维护性 | 测试效率 |
|---|---|---|---|---|
| JUnit | 较高 | 很广 | 很强 | 较高 |
| Postman | 一般 | 较广 | 较强 | 较高 |
| Selenium | 一般 | 一般 | 较弱 | 较低 |
| OLa | 较低 | 较广 | 较强 | 很高 |

从对比结果的表格中可以看出, OLa 相对于单元测试框架和传统的自动化测试工具来说, 不仅降低了测试难度, 而且具有更高的测试效率, 同时也有较强的用例可维护性.

## 6 结语

本文采用分层开发的思想设计并实现了一个支持用例集并行测试的接口测试平台 (OLa), OLa 平台中只需要录入用例的参数、校验方法以及校验内容后即可进行测试, 相比于手工测试来说, 不仅节省了编写大量测试代码的时间, 而且降低了测试的难度. 另外, 平台采用 SpringMVC、MyBatis-Plus、Vue 等框架进行开发, 降低了系统各模块之间的耦合度, 提高了系统的可维护性, 解决了传统的接口自动化测试工具扩展能力不足的问题.

平台测试结果表明, OLa 不仅具有较高的测试效率, 而且具有良好的交互体验, 但是平台仍然还有一些不足和需要优化的地方, 在接下来的工作里将会为接口测试平台添加组合校验, 接口参数自动生成, 不同协议接口测试等功能, 以满足更多的测试需求.

## 参考文献


1 林萍. 数据分层的接口自动化测试框架设计与应用. 信息技术, 2016, (5): 37–40.

2 王娜. 基于 Python 的接口自动化测试框架设计. 电脑知识与技术, 2020, 16(12): 246–248.

3 马春宇. 自动化测试管理与实践. 网络安全技术与应用, 2019, (6): 85–87. [doi: 10.3969/j.issn.1009-6833.2019.06.051]

4 Anjum H, Babar MI, Jehanzeb M, *et al*. A comparative analysis of quality assurance of mobile applications using automated testing tools. International Journal of Advanced Computer Science and Applications, 2017, 8(7): 249–255.

5 刘智. 一种数据驱动的软件接口自动化测试框架的设计与实现. 信息化研究, 2015, 41(1): 75–78.

6 杨清玉, 李金丽, 陈吉兰, 等. HTTP 接口自动化测试方法研究. 微型机与应用, 2016, 35(18): 22–25.

7 卓欣欣, 白晓颖, 许静, 等. 服务接口测试自动化工具的研究. 计算机研究与发展, 2018, 55(2): 358–376. [doi: 10.7544/issn1000-1239.2018.20160721]

8 赵明明, 周静, 补冲. Robot Framework 在软件接口自动化测试中的研究与应用. 电信工程技术与标准化, 2018, 31(10): 78–82. [doi: 10.3969/j.issn.1008-5599.2018.10.017]

9 张士恒, 赵小盼, 祁鹏. 应用于 Web Service 服务接口的自动化测试策略研究及应用. 电脑与电信, 2010, (1): 65–67.

10 国建胜, 张亚楠, 张雪石. 基于 Java 的自动化测试接口测试系统. 信息与电脑, 2019, 31(3): 109–110.

11 李艳丽, 张宗勇, 冯捷, 等. 一种分布式可视化 Dubbo 接口测试平台. 华东师范大学学报 (自然科学版), 2019, (4): 120–132, 143.

12 蔡高扬, 马化军, 李挥. aSIT: 面向接口的分布式自动化测试系统. 广东通信技术, 2020, 40(4): 38–43, 63. [doi: 10.3969/j.issn.1006-6403.2020.04.010]

13 朱二华. 基于 Vue.js 的 Web 前端应用研究. 科技与创新, 2017, (20): 119–121.

14 郭文丽, 赵晓晔, 周婕. 基于 Ajax 的图书馆讲座预约系统构建. 现代图书情报技术, 2010, (5): 84–88.

15 宋衍, 傅赛. 基于受限网络应用层协议的物联网应用代理研究与实现. 计算机应用, 2013, 33(11): 3010–3015.

16 张峰. 应用 Spring Boot 改变 Web 应用开发模式. 科技创新与应用, 2017, 7(23): 193–194.

17 李洋. SSM 框架在 Web 应用开发中的设计与实现. 计算机技术与发展, 2016, 26(12): 190–194.

18 何军, 陈倩怡. Vue+Spring Boot+MyBatis 开发消费管理系统. 电脑编程技巧与维护, 2019, (2): 87–88, 102. [doi: 10.3969/j.issn.1006-4052.2019.02.025]

19 肖文娟, 王加胜. 基于 Vue 和 Spring Boot 的校园记录管理 Web APP 的设计与实现. 计算机应用与软件, 2020, 37(4): 25–30, 88. [doi: 10.3969/j.issn.1000-386x.2020.04.005]

20 宗平, 李雷. PostgreSQL 与 MongoDB 处理非结构化数据性能比较. 计算机工程与应用, 2017, 53(7): 104–108, 170. [doi: 10.3778/j.issn.1002-8331.1508-0203]

21 王妍. 基于 TestLink 的软件测试自动化管理. 智能计算机与应用, 2019, 9(2): 159–161.


(校对责编: 孙君艳)