华 中 科 技 大 学 硕 士 学 位 论 文

分 类 号 学号 M201471785

学校代码 10487 密级

華中科技大學

硕士学位论文

基于 Python 的自动化测试脚本管理平台的设计与实现

学位申请人：王 鸣

学 科 专 业：电子与通信工程

指 导 教 师：王玉明 副教授

答 辩 日 期：2016 年 5 月 20 日

华 中 科 技 大 学 硕 士 学 位 论 文

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Design and Implementation of a Python-Based Automated Testing Script Management Platform

Candidate : Wang Ming

Major : Electronics and Communication Engineering

Supervisor : Associate Prof. Wang Yuming

Huazhong University of Science and Technology

Wuhan, Hubei, 430074, P. R. China

May, 2016

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，　　在_____年解密后适用本授权书。
　　不保密□。
（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

随着科研管理信息化建设的发展，科研管理软件的需求日趋增长，软件规模越来越大，软件复杂度越来越高，如何保证科研管理软件的产品质量逐渐成为了近年来科研管理软件项目主要面临的问题之一，而软件测试是解决该问题的重要途径。故本文提出为科研管理软件搭建一个自动化测试脚本管理平台，实现对测试脚本的规范和统一管理，并引入自动化测试技术，改善现有的测试工作环境，进而更有效地保证科研管理软件的产品质量。

本文首先对科研管理软件项目的测试工作现状进行了调研，发现现有测试工作的不足之处并结合测试行业背景分析了改进方向，进而整理出测试脚本管理平台的需求；然后从脚本和数据分离的角度出发，提出“驱动器-用例-计划”的概念模型，并基于该测试模型设计出一种基于数据驱动脚本的自动化测试方案；接着基于测试模型和单元测试框架原理，采用 Python 语言，设计并实现了平台的脚本执行工具；同时采用 B/S 模式，以 MySQL 作为数据源，基于 web.py 服务端开发框架，基于 jQuery +Ajax+JSON+TrimPath 前端开发框架，设计并实现了平台的脚本管理系统；最后从功能、性能和兼容性三个角度对平台开展了测试，同时以科研管理移动端的接口测试工作为例讲述平台的应用过程。

该自动化测试脚本管理平台可以为科研管理软件提供各种类业务或类型的 Python 单元测试脚本的自动构建、自动运行、结果跟踪和分类管理功能，改善了脚本测试的技术方法和流程管理方式，提升了测试工作的效率，最终在缩短科研管理软件的开发周期和节省项目测试资源的同时，更好地保证了科研管理软件的产品质量。

关键词： 测试脚本管理，自动化测试，数据驱动，脚本执行工具

Abstract

With the development of information construction in scientific research management, requirements of software projects on scientific research management is in flux, the scale is getting larger and larger and the complexity is increasing, so how to ensure the quality of software is becoming a serious problem of software projects on scientific research management and software test is a major approach to solve the problem. As a result, the idea to establish an automated testing script management platform for software projects on scientific research management is presented.

Firstly, based on the research of current situation about test work, the deficiency of this work is analyzed and requirements of the testing script management platform is sorted out. Next, from the perspective to separate the script and data, the driver-case-plan model is presented and a data-driven test automation solution based the model is designed. Then, the script execution tool of the platform is designed and implemented based on the driver-case-plan model, principle of unit testing framework and Python. At the same time, the Web management system of platform is designed and implemented based on B/S structure, MySQL as the database, web.py as the server-side development framework and jQuery -Ajax-JSON-TrimPath as the front-end development framework, too. Lastly, to ensure the function, performance and compatibility of the platform, an overall test and result analysis are applied to the platform. At the same time, interface testing of the mobile applications on scientific research management is taken as an example to explain the application process of the platform.

The automated testing script management platform can provide automated building, auto running, result tracking and classification function for Python unit test scripts with various types and business. Consequently, the technique and workflow of software test has been improved, the test efficiency for the scientific research management projects has been raised. The quality of the scientific research management projects has ultimately been guaranteed, meanwhile, not only development period of the scientific research management projects has been shortened but also testing resources have been saved.

Keywords: Testing Script Management, Automated Testing, Data Driven, Script Execution Tool

目 录

缩略语

MVC - Model View Controller	模型－视图－控制器
CSS - Cascading Style Sheets	层叠样式表
DOM - Document Object Model	文档对象模型
AJAX - Asynchronous JavaScript and XML	异步 JavaScript 和 XML
JSON - JavaScript Object Notation	JavaScript 对象表示法
B/S - Browser/Server	浏览器/服务器

1 绪论

在科研管理信息化建设的过程中，科研管理软件项目越来越多，这也导致测试脚本不断积累而难以管理，也没有一套高效便捷的自动化测试方案，测试工作内容经常重复，测试效果不显著，测试效率低下。对测试脚本进行规范和统一管理并引入自动化测试框架技术可以极大地改善科研管理项目的测试工作，促进项目开发进度，保证软件产品质量，最终推动科研管理信息化建设。

本章首先介绍了课题的背景和来源，然后分析了课题的现状，接着阐述了课题的研究内容及意义，最后介绍了论文的组织结构。

1.1 课题背景及来源

随着科研管理信息化建设的不断发展，科研管理软件的需求日趋增长，软件规模日渐变大，软件复杂度日益变高，开发和维护工作变得越来越困难，如何保证科研管理软件的产品质量逐渐成为了近年来科研管理软件项目主要面临的问题，而软件测试是解决该问题的重要途径[1]。

在传统的软件工程理论中，测试通常被安排在需求、设计和开发之后进行。而近年来，软件工程界趋向“测试应该贯穿软件生命周期的每一阶段”的新观点，软件测试越来越被重视[2]。现阶段，自动化、规模化和通用化是软件测试的主要发展趋势[3]。随着国内的软件公司逐渐意识到测试的重要性，自动化测试成为他们的首选的改革方向。利用自动化测试技术，公司可以极大地改善测试工作，在确保软件质量的同时，增强软件在市场中的竞争力。

本文的研究内容服务于教育部重大委托课题（NO.10JZDW004）、教育部专项委托课题（NO.11JF005）以及华中科技大学自主创新基金项目（NO.2011WA001、NO.2013WZ020），旨在为这些课题的应用或系统搭建一个通用的自动化测试脚本管理平台，实现软件项目中各种类型、各种业务下的 Python 测试脚本的自动构建、运行和集中管理功能。

1.2 课题现状分析

为了解决软件测试工程中的诸多问题，“自动化”逐渐成为测试行业的主要潮流和趋势。而随着互联网的不断进步和快速发展，自动化测试技能已经逐步成为软件测试行业人才质量评判和业绩考核的重要技能指标[4]。

在国外，很多知名的互联网企业很早就开始研究并应用自动化测试技术，包括Google、Twitter、Facebook 在内的国际互联网巨头均推出过完善的自动化测试系统或工具，如 Autotest 是 Google 开发的自动化测试框架，它支持分布式运行测试任务并集成了大量的功能测试工具，专门用于构建 Chrome OS 的功能测试工作[5]；Diffy 是 Twitter 开发的开源自动化测试工具，旨在探测运行在生产环境中的 Apache Thrift 与其他 HTTP 服务器上的新代码所产生的潜在 Bug。根据相关统计资料，国外企业的软件测试工作量大约占整个开发工作量的 40%，测试经费占整个软件经费的 30%至 50%[6]。总之，国外的软件测试技术相当成熟，甚至已经被归为软件工程下的典型分支学科，产出了很多著名的测试技术方法和测试流程管理规范[7]。

在国内，软件测试在软件行业中属于劣势科目，测试活动在软件工程中的比重以及测试质量均与国外有着明显的差距。国内一半以上的软件公司（尤其是中小型互联网公司）还认为测试无足轻重，他们的测试活动仅仅涉及一些局部的功能测试。但随着近几年互联网的飞速发展，在一些大型公司的带领下，软件测试也逐渐引起了人们的重视。诸如阿里巴巴、腾讯和百度等大型互联网都成立了专门的质量保障（Quality Assurance）部门来专门负责测试开发工作，从而更好地研究和实现测试工具或测试平台。此外，一些中型公司也开始招聘专门的测试开发人员，自动化测试正在被越来越多的公司所接受和应用。

就科研管理软件的测试工作而言，科研管理软件现阶段只是进行了简单的 Python 脚本测试。这些测试脚本杂乱地分布在服务器上，代码格式混乱，代码内容重复性高，每次进行回归或迭代测试时需要手工执行。因此，设计并实现一个引用自动化测试技术的脚本管理平台是当务之急。

1.3 研究内容及意义

本课题的目的是为科研管理项目的应用或系统搭建一个通用的自动化测试脚本管理平台，具体的研究内容包括以下几个部分：

1）调研分析科研管理软件的测试工作现状，梳理平台的需求；

2）深入调研测试行业背景及测试相关技术，设计平台的自动化测试方案；

3）基于单元测试、自动化测试、测试脚本等技术设计并实现脚本执行工具；

4）基于 Web 前后端开发技术设计并实现脚本管理系统。

该脚本管理平台可以为 Python 测试脚本提供自动构建、自动运行、结果跟踪和分类管理功能。该平台下的测试脚本不受测试类型、业务类型的限制，具有较强的通用性，可以适用各类科研管理软件的脚本测试工作。该平台改善了脚本测试的技

术方法和流程管理方式，提高了脚本的重用性和可维护性，划清了项目相关人员的职责，改善了测试工作环境，简化了测试工作的内容，提升了测试工作的效率，最终在缩短科研管理软件的开发周期和节省项目测试资源的同时，更好地保证了科研管理软件的产品质量。

1.4 论文组织结构

论文总共分为六个章节，每个章节的内容概述如下：

第一章：绪论。本章节首先介绍了本课题的背景与来源，然后分析了课题的研究现状，接着在现状分析的基础上提出了课题的研究内容并阐述了意义价值所在，最后交代了论文的组织结构。

第二章：相关技术简介。本章节主要介绍了实现自动化测试平台时所用到的关键技术，主要包括软件测试技术和 Web 开发技术。对于软件测试技术，主要介绍了单元测试理论、自动化测试理论和常用的测试脚本技术。对于 Web 开发技术，主要从服务端和前端两个方面介绍了实现平台的相关框架或插件。

第三章：平台的分析与设计。本章节首先对脚本管理平台的需求进行了分析，然后根据需求设计了自动化脚本测试方案，并从整体对平台进行了层次结构设计、技术架构设计和数据库设计，接着详细讲述了脚本执行工具的设计过程，最后分别从功能、数据库和接口三个角度介绍了脚本管理系统的设计过程。

第四章：平台的实现。本章节首先介绍了平台的开发环境，然后讲述了脚本执行工具的实现过程，最后按照功能模块地划分讲述了脚本管理系统的实现过程。

第五章：平台的测试与应用。本章节首先介绍了平台的测试方案，然后按照测试方案分别对脚本执行工具和脚本管理系统进行了一系列的测试和结果分析，最后讲述了平台的一个具体应用实例。

第六章：总结与展望。本章节总结了课题的研究内容及成果，分析了平台的待改善点并展望了平台的后续工作。

2 相关技术简介

自动化测试脚本管理平台的开发主要涉及软件测试和 Web 开发这两方面的技术。软件测试为自动化测试方案和脚本执行工具的实现提供了支持，Web 开发技术为脚本管理系统的实现提供了支持。

本章首先对单元测试作简单的介绍，然后概述了自动化测试相关理论，接着着重介绍了一些自动化测试脚本技术，最后从 Web 服务端和 Web 前端两个角度分别介绍了实现平台时所用到的 Web 开发技术。

2.1 软件测试理论与技术

2.1.1 单元测试理论

单元测试又被称作模块测试，它负责对最小的软件设计单元进行验证，它使用项目设计文档中的模块描述作为依据，测试重要的程序分支以检测出软件的错误[8]。然而一个系统的单元测试通常比较繁多杂乱且难以手工进行，这就需要引入单元测试框架来进行自动化测试。单元测试框架通常包括以下四个方面的概念。

（1）测试用例（Test Case）：在测试过程中为特定目的而设计的一组测试输入、执行条件和预期的结果被称作测试用例，它是最小的执行实体[9]。

（2）测试套件（Test Suite）：以某种特性将测试用例组合到一起的用例集被称作测试套件。

（3）测试固件（Test Fixture）：测试主程序运行时所需要的一切东西，它可能是数据，可能是系统环境，也有可能是某个具体实例化类[10]。测试固件包括构建和销毁两个部分。测试固件既可以存在于测试用例层面，也可以存在于测试套件层面。

（4）测试运行器（Test Runner）：是测试程序的执行载体，掌控着测试任务的执行流程。

在自动化单元测试框架下，一个测试用例的执行过程包括：固件构建、执行测试用例主程序、固件销毁。而一个测试套件的执行过程包括：测试套件固件构建、测试用例 1 固件构建、执行测试用例 1 主程序、测试用例 1 固件销毁、测试用例 2 固件构建 ••• 测试用例 n 固件销毁、测试套件固件销毁[11]，其流程示意图如图 2.1 所示。

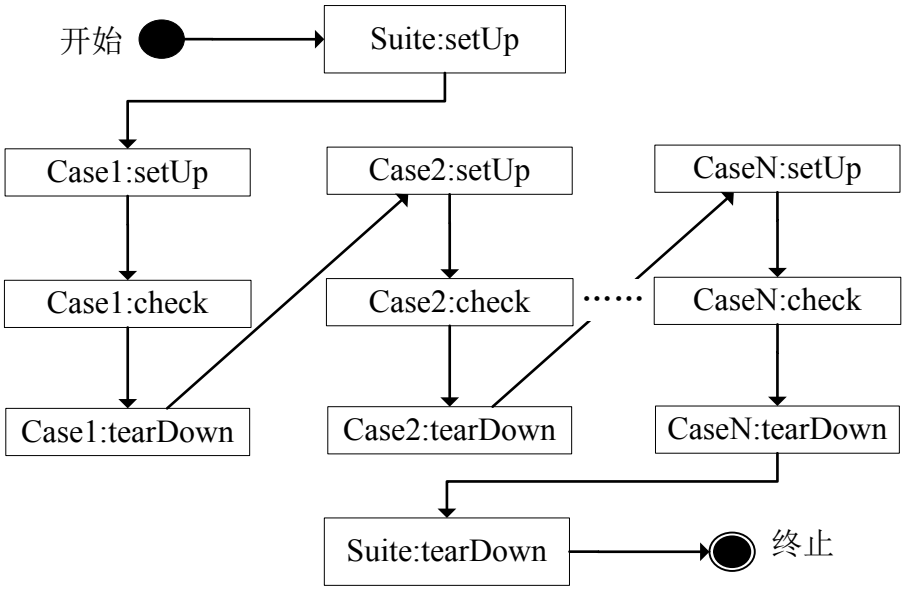


图 2.1 测试套件任务的执行流程示意图

现阶段，Java 语言下的 JUnit[12]、Python 语言下的 Unittest[13]以及 C++语言下的 CppUnit[14]等常用的单元测试工具均是采用这种框架实现的。

2.1.2 自动化测试理论

自动化测试就是采用某种策略或工具减少人工介入到非技术性、重复性、冗长的测试活动里，从而达到无人监守完成测试，自动产生测试报告，分析测试结果等一系列活动[15]。自动化测试是一种机制，它不仅指利用工具或框架进行自动化测试，而且包括如何确定自动化测试的方法，如何组织测试，以及如何管理测试的流程等。自动化测试的优点主要概括如下：

1）可以完成一些非功能性的、重复性的、冗长的或不可手工操作的测试，如压力测试、并发测试、海量数据测试等[16]。

2）测试内容更具有一致性、组织性，简化了测试工作，也使测试工作更具有可重用性和复用性。

3）可以大大节省测试开销和开发资源，尤其是可以大规模地节省回归测试的开销，从而提高测试效率，推进软件开发进度。

虽然自动化测试有很多优点，但并非所有测试都值得或能够转化为自动化测试。例如，对于敏捷开发或交付时间紧迫的项目会因为时间、资源和成本的限制而不宜采用自动化测试[17]。通常，我们可以从测试用例或测试工具本身的特性以及对比自动化测试前后的开销来分析自动化测试的可行性。此外，我们还可以采用静动态分

析执行程序、捕获和回放测试过程、引用测试脚本技术等方法来设计自动化测试方案。常用的自动化测试框架包括测试脚本模块化框架、测试库构架框架、数据驱动测试框架、关键字驱动测试框架、混合测试自动化框架等[18]。

2.1.3 自动化测试脚本技术

将自动化测试框架应用于脚本测试时所用到的相关技术被称作自动化测试脚本技术。合理利用脚本生成和脚本执行技术可以提高测试脚本的复用性、健壮性、可维护性、灵活性，进而改善测试工作环境，提高测试效率、节省测试开销。自动化脚本测试的流程一般如下：

1）测试设计：设计测试用例，搭建测试环境等。

2）脚本生成：根据测试设计自动生成测试脚本文件。

3）脚本运行：通过某种工具执行测试脚本，执行工程中会记录日志，脚本最后对测试内容进行核验，并返回最终的测试结果。

4）测试结果处理：对测试结果进行提取、分类、统计（成功率、测试时间）并记录。

5）测试报告生成：根据测试结果生成相关的测试报告。

6）测试通知发送：利用通讯工具将测试结果通知给项目相关人员[19]。

常用的自动化测试脚本技术包括以下几种：

1）线形脚本技术

由录制手工执行的测试用例得到的脚本通常是线形脚本。线形脚本的实现原理比较简单，适合用作演示或培训，但与软件本身的耦合性强，不能共享和重用，维护成本高。

2）结构化脚本技术

结构化脚本与结构化的程序类似，它含有控制脚本执行流程的指令。结构化脚本具有判断功能，适合包含重复性程序指令的测试用例，在一定程度上改善了脚本的灵活性和健壮性，但脚本内仍然捆绑测试的数据和逻辑，可定制性不强。

3）共享脚本技术

共享脚本技术使得脚本可以被多个测试用例共用。该技术复用性强，大大节省了测试开销，但耦合度高，多个测试用例复用同一个脚本的一致性要求导致维护性降低。

4）数据驱动脚本技术

数据驱动脚本技术将测试的数据信息单独存放文件中，同时在测试脚本中只存放逻辑控制信息，在测试执行时测试数据是从单独的文件而不是从脚本中输入的[20]。

数据驱动脚本技术实现了数据与脚本的分离，提升了脚本的灵活性、复用性和可维护性，但脚本的初始建立开销较大，对数据文件格式的要求较高，测试逻辑依然与脚本捆绑在一起。

5）关键字驱动脚本技术

关键字驱动脚本技术是对数据驱动脚本技术的一种延伸扩展，它用数据文件描述测试用例，用一系列的关键字描述执行的测试任务[21]。该技术真正实现了数据与脚本分离，测试逻辑与测试脚本分离，实现了测试的完全定制，但构建成本较大，且关键字和脚本的数量会随着软件复杂度的增加而大量增多。

6）混合测试脚本技术

结合以上两个或多个脚本技术协同使用，从而避免单一技术的局限性，并发挥多个技术的优点。

2.2 Web 开发技术

2.2.1 Web 服务端技术

web.py 是一个基于 Python 语言的轻量级 web 框架。web.py 的设计理念力求精简，因而 web.py 的源码相当简短，只提供一个 Web 框架所必须的模块，不依赖大量的第三方模块或类库，URL 处理模式非常简单，支持模板引擎的定制，也没有集成重量的数据控制模块。这些特性使得 web.py 十分灵巧，开发人员可以灵活地定制功能模块。

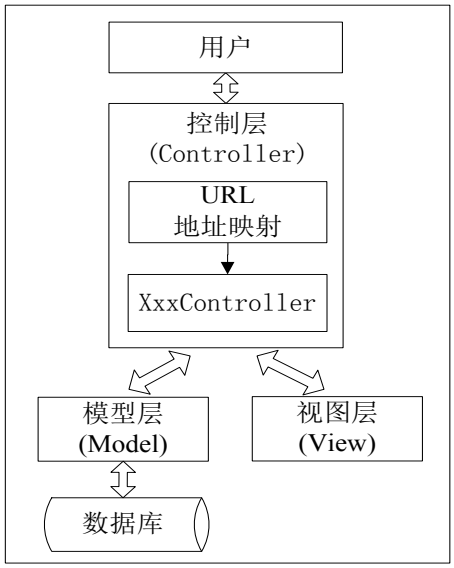


图 2.2 MVC 框架下 web.py 应用结构示意图

利用 web.py 的这些特性，web.py 可以很便捷地实现符合 RESTful[22]风格的 MVC

（Model-View-Controller）模式下 Web 应用[23]，其结构示意图如图 2.2 所示。

由图 2.2 可知，模型层能够对数据库作读写操作，负责系统业务逻辑处理功能；视图层主要负责数据的显示，它包括一些模板文件和前端静态资源；控制层是模型层和视图层的交互枢纽，它首先接收用户的 URL 请求，然后根据 URL 解析选定对应的控制层方法，接着控制层方法会调用模型方法进行业务处理，最后控制层方法会确定哪个模板来显示数据。基于 MVC 架构，web.py 应用可以将视图层和业务层分离，降低了代码的耦合度，使系统层次更加清晰，增加了软件的可维护性[24]。

利用 Nginx 和 FastCGI 可以快速部署发布 web.py 应用[25]。其中，Nginx 是一款轻量级的 Web 服务器、反向代理服务器及电子邮件代理服务器。FastCGI 即快速通用网关接口，是一种让交互程序与 Web 服务器通信的协议，致力于减少网页服务器与通用网关接口程序之间互动的开销，从而使服务器可以同时处理更多的网页请求。但是在部署发布 web.py 应用时，由于 Nginx 不能够直接调用或者解析外部程序，所有的外部应用必须通过 FastCGI 接口来调用，其运作流程如图 2.3 所示。

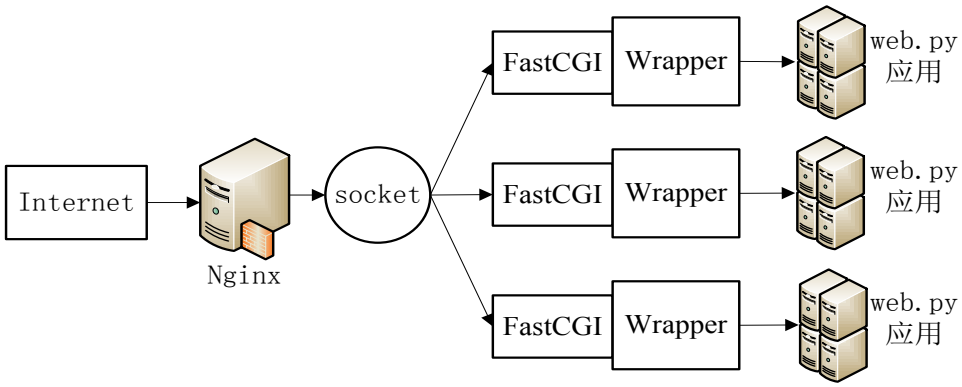


图 2.3 Nginx 和 FastCGI 部署 web.py 应用的运作流程图

在图 2.3 中，Wrapper 是 FastCGI 接口的包装器，用于启动另一个程序的程序。首先所有的网络请求会经由 Nginx 转发至 FastCGI 接口；然后 Wrapper 通过 FastCGI 接口接收该请求，并派生出一个新的线程，这个线程调用解释器或者外部程序处理脚本（即 web.py 程序）并读取返回数据；接着 Wrapper 再将 web.py 程序返回的数据通过 FastCGI 接口沿着固定的 socket 传递给 Nginx；最后 Nginx 将返回的数据发送给客户端。

2.2.2 Web 前端技术

Web 前端技术主要包括 HTML、CSS、JavaScript 以及衍生出来的前端工具、框架或类库[26]，这些技术被应用在浏览器端用来实现信息的展示和交互。随着前端技术的飞速发展，网页不再只是承载单一的文字和图片，各种富媒体让网页的内容更

加生动，网页上软件化的交互形式为用户提供了更好的使用体验。

出于页面响应速度、用户交互体验和软件开发难度等方面的考虑，本课题主要采用 jQuery +Ajax+JSON+TrimPath 的前端开发框架，该框架的原理结构如图 2.4 所示。

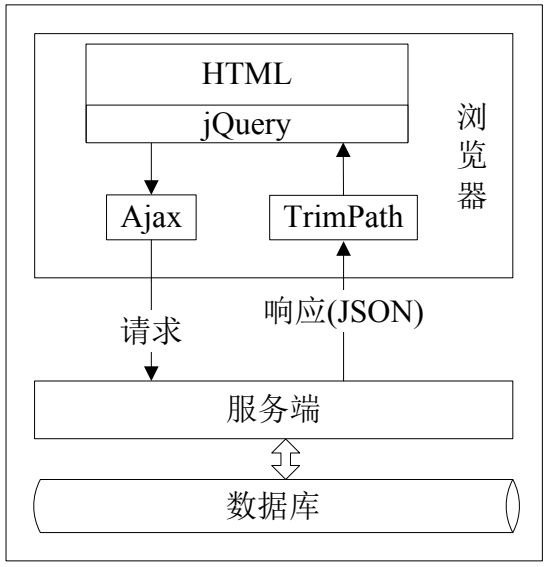


图 2.4 前端开发框架原理结构的示意图

其中，jQuery 插件对 JavaScript 方法进行封装，解决了不同浏览器解析 JavaScript 的兼容性问题，为前端开发提供了一个兼容主流浏览器的统一接口，提供 DOM（Document Object Model）支持、CSS（Cascading Style Sheets）效果、动画、事件、Ajax 支持等诸多功能[27]。

Ajax（Asynchronous JavaScript and XML）即异步 JavaScript 和 XML，它是一种创建交互式网页应用的网页开发技术，实现了对页面的局部动态刷新，使得用户能够以更好的方式获取最新的数据信息[28]。

JSON（JavaScript Object Notation）是一种数据格式，它为 Web 前后台数据交互提供了标准、简单、易用、可靠的数据交换格式[29]。

TrimPath 是一款基于 JavaScript 的、轻量级的、跨浏览器的前端模板引擎，它旨在解决前端显示控制的问题，通过分离前端页面的数据与逻辑，使得前端页面结构更加清晰。

该框架可以为用户提供漂亮的信息展示界面和良好的交互操作。此外，该框架还可以实现前后端的分离，改善前后端开发人员的协作方式。在开发时，后端开发人员只需要提供规范的数据接口，而前端开发人员不必关心后端程序的具体实现，只需要通过交互接口获得 JSON 格式的数据并利用 TrimPath 引擎写入模板从而展现

给用户。

2.3 本章小结

本章主要介绍了实现自动化测试平台时所用到的关键技术，主要包括软件测试技术和 Web 开发技术。对于软件测试技术，首先对单元测试作简单的介绍，着重介绍了自动化单元测试框架，然后介绍了一些自动化测试理论，如自动化的优缺点、可行性和流程等，最后举例了几种自动测试脚本技术，比较了它们的优缺点。对于 Web 开发技术，主要介绍了实现平台服务端的 web.py 框架技术和实现平台前端的 jQuery +Ajax+JSON+TrimPath 的混合开发框架。

3 平台的分析与设计

本章将在前期调研成果的基础上，首先对平台进行整体的需求分析，然后确定了平台的自动化测试方案，接着从层次结构、技术架构和数据库三个方面对平台进行了整体的设计，最后分别阐述了脚本执行工具和脚本管理系统的设计过程。

3.1 总体需求分析

软件需求分析是指通过与用户的充分沟通，对目标系统提出完整、准确、清晰、具体的要求，确定软件系统必须完成的任务，并尽可能地描述软件的功能和性能等软件属性。软件需求分析是软件开发周期的重要一环，是开发人员进行软件开发工作的参考点和驱动点，它紧密关系到软件工程的成败和软件产品的质量[30]。

现阶段，科研管理软件项目的测试工作大多以 Python 脚本的方式进行的。随着科研管理软件项目的需求日趋增长，软件规模越来越大，软件复杂度越来越高，测试脚本不断积累而难以管理，也没有一套便捷高效的自动化测试方案，测试工作内容经常重复，测试效率极其低下。针对这些现状并经过充分的调研后，从三个方面来梳理自动化脚本测试的需求。

1）测试脚本的集中管理

现阶段，科研管理软件项目的测试脚本杂乱地存储在服务器上，数量约至数百个，涉及各种测试类型，涵盖多个应用和系统。故设计一个集中管理脚本的 Web 平台，提供脚本的查看、详细说明、添加、编辑、删除以及统计等功能，并从测试业务或测试类型等角度加以分类，从而使脚本的管理工作更加直观、高效。

2）测试结果的记录与统计

现阶段，每次在对科研管理软件测试后，测试人员没有特别处理或记录测试结果。每当测试出现问题，测试人员只是简单地口头通知开发人员处理问题，而处理完问题后测试结果信息就被丢弃了。故需要对测试工作的脚本运行状况进行收集和记录，包括测试运行结果、运行日志、运行时间等信息，同时对测试结果进行统计与分析，如测试任务的成功率统计和回归测试的稳定性分析。合理地利用测试结果可以反映一个软件产品的运行和开发状况，从而改善软件质量和推动开发工作。

3）脚本测试的自动化

现阶段，科研管理软件项目的测试工作均为手工进行，测试脚本的复用性低，测试库功能重复，测试效率低下。故基于某种脚本生成技术和某种自动化测试框架技术设计自动化测试方案，改善测试方法和流程，使得测试工作更加高效和智能化。

具体的流程可规范为：自动生成脚本和测试用例文件，根据配置自动执行测试任务，自动处理并记录测试结果，自动生成测试报告，利用邮件自动发送测试结果通知。

3.2 自动化测试方案设计

本小节主要讲述平台的自动化测试方案的设计过程，首先提出一种数据与脚本分离的测试概念模型，然后在该模型的基础上设计自动化测试框架，最后介绍该框架下的自动化测试工作的流程。

3.2.1 测试概念模型

在传统的测试脚本中，测试数据与测试执行逻辑捆绑在一起的。为了提高测试脚本资源的复用性、灵活性和可维护性，课题基于数据驱动脚本技术提出“测试驱动器-测试用例-测试计划”的测试概念模型。该模型通过抽象出测试驱动器从而分离了测试控制逻辑和测试数据，测试模型的概念具体解释如下。

1）测试驱动器（Test Driver）对应实际的测试脚本。该脚本只包含测试的逻辑控制信息，而不涉及具体的测试数据信息，因此可以通过配置驱动数据使得驱动器脚本反复被利用。测试驱动器包括测试固件构建脚本、测试固件销毁脚本、测试执行主程序脚本三个部分。

2）测试用例（Test Case）对应实际的测试用例，它是根据驱动器配置相应的驱动参数生成的，因此测试用例文件只包含该用例所对应的驱动器名称和驱动参数信息，不包含逻辑执行程序。

3）测试计划（Test Plan）是具有相同特性的测试用例集，它通常按业务或测试类型划分归类。测试计划在执行结束会生成一个测试报告和发送测试结果通知。此外，还可以对测试计划进行“每日构建”，即配置定时器参数来让测试计划每天在固定的时间点自动执行。综述以上的信息，测试计划文件应该包含测试用例集、每日构建时间点和通知对象邮箱等信息。

图 3.1 是“测试驱动器-测试用例-测试计划”概念模型的对应关系图。一个测试驱动器可以被多个测试用例复用，一个测试用例可以被多个测试计划拥有，一个测试计划可以包括多个测试用例。

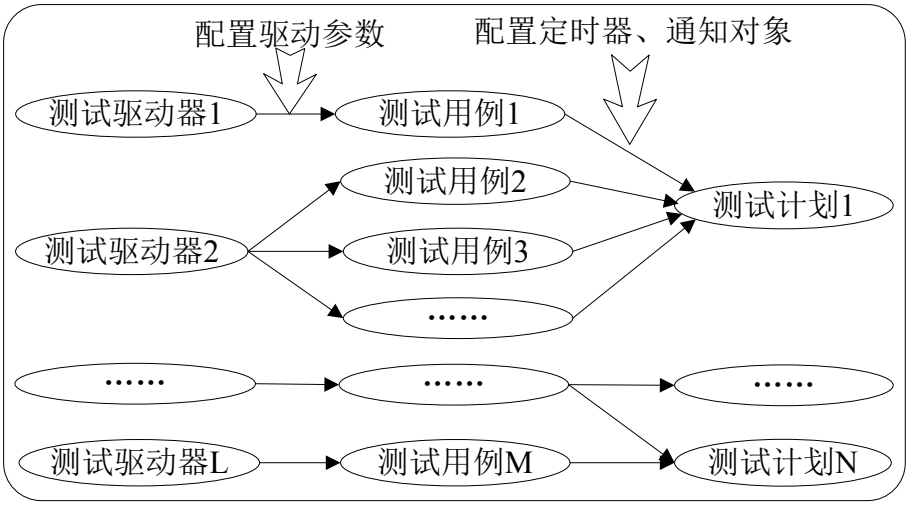


图 3.1 “测试驱动器-测试用例-测试计划”概念模型的对应关系图

3.2.2 自动化测试框架

基于“测试驱动器-测试用例-测试计划”模型，设计出一种基于数据驱动的自动化测试框架方案，其框架结构如图 3.2 所示。

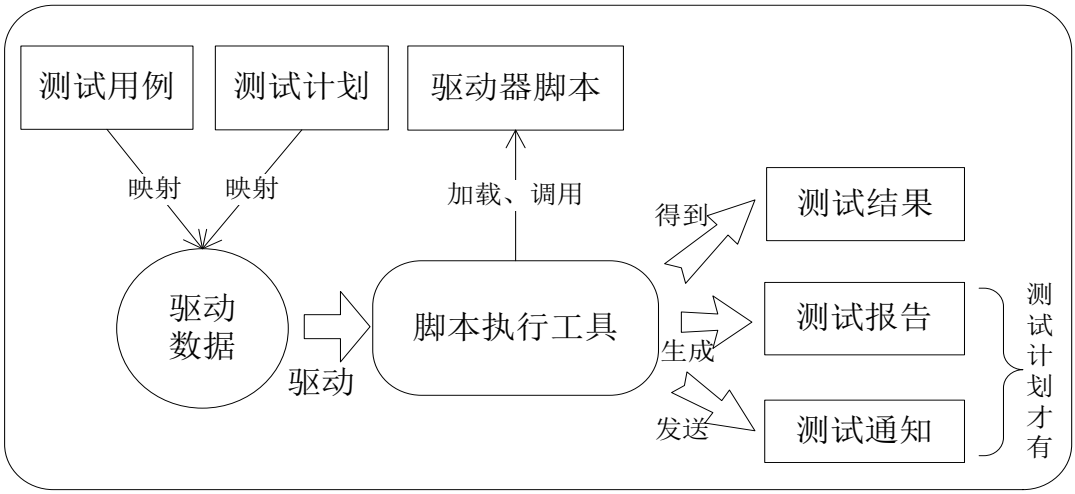


图 3.2 基于数据驱动的自动化测试框架结构示意图

由图 3.2 可知，脚本执行工具是整个框架的枢纽，控制着整个测试的流程。测试用例和测试计划是含有驱动参数的 JSON 文件，它可以驱动脚本执行工具执行测试任务。在执行过程中，脚本会根据驱动参数加载调用相应的驱动器脚本，最终获得测试结果。若执行的任务类型是测试计划，脚本执行工具还会生成测试报告，发送测试结果通知。

3.2.3 自动化测试工作流程

在数据驱动自动化测试框架的基础上，结合 Web 端在线管理脚本的需求，设计自动化测试的构建流程如图 3.3 所示。

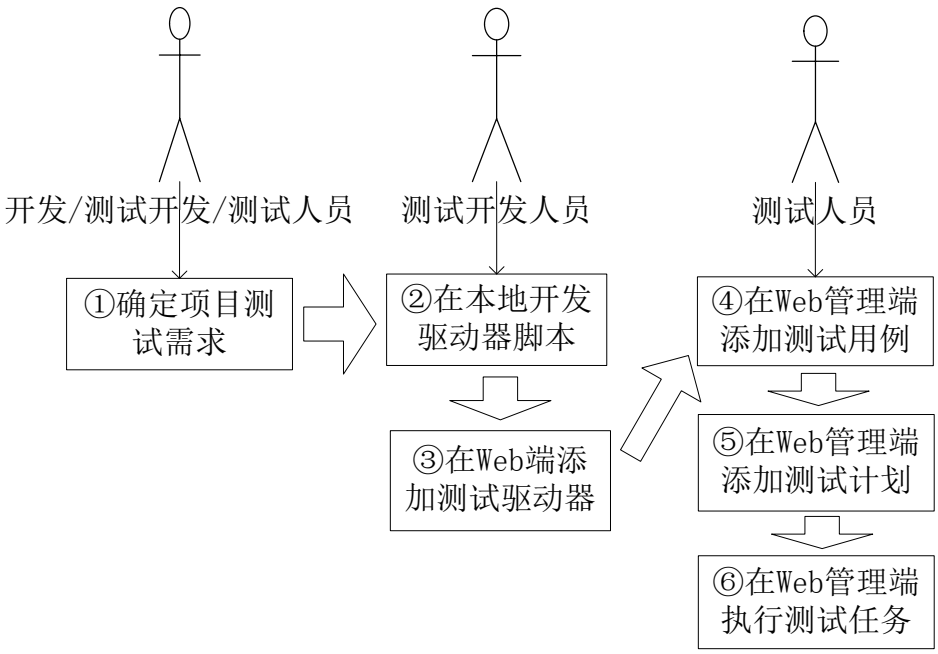


图 3.3 自动化测试工作的构建流程

自动化测试的构建流程具体说明如下：

1）首先，开发、测试开发及测试人员共同确定项目的测试需求。

2）然后，测试开发人员根据项目需求开发驱动器脚本。

3）其次，待驱动器脚本开发完成后，测试开发人员在脚本管理系统中添加该条驱动器脚本数据和默认的驱动参数信息。在添加时，脚本管理系统会自动生成驱动器脚本。

4）再其次，测试人员在脚本管理系统中选择测试驱动器并配置驱动参数从而添加测试用例数据。在添加时，脚本管理系统会自动生成描述测试用例的 JSON 文件。

5）接着，测试人员可以按测试类型或者业务类型将多个测试用例组织在一起并配置通知对象和定时器参数从而添加计划。在添加时，脚本管理系统会自动生成描述测试计划的 JSON 文件。

6）最后，测试人员可以在脚本管理系统中执行测试用例、执行测试计划和构建每日计划任务。执行计划时，脚本管理系统会调用脚本执行工具并获得测试结果信息和测试报告，必要时还可以发送测试结果的通知邮件。

在整个自动化测试的构建流程中，驱动器脚本能够被多个测试用例共用；测试

计划可以分类管理测试用例，集成测试任务；定时器功能可以使测试计划每日自动构建；通知功能使得测试工作更加即时高效；更重要的一点是，对于整个测试过程中，除了测试驱动器需要专门的测试开发人员开发，其他模块均不需要专门的人员去撰写代码，划清了测试工作的职责，使得测试的人力资源得到更好的利用。

3.3 平台整体设计

本小节将从整体介绍平台的设计方案，首先在自动化测试方案的基础上分别阐述平台的层次结构和功能结构，然后介绍平台的技术架构，最后阐述平台的数据库设计方案。

3.3.1 层次结构

根据平台的自动化测试方案，将平台划分为三个部分：测试相关数据、脚本执行工具和脚本管理系统，其层次结构示意图如图 3.4 所示。

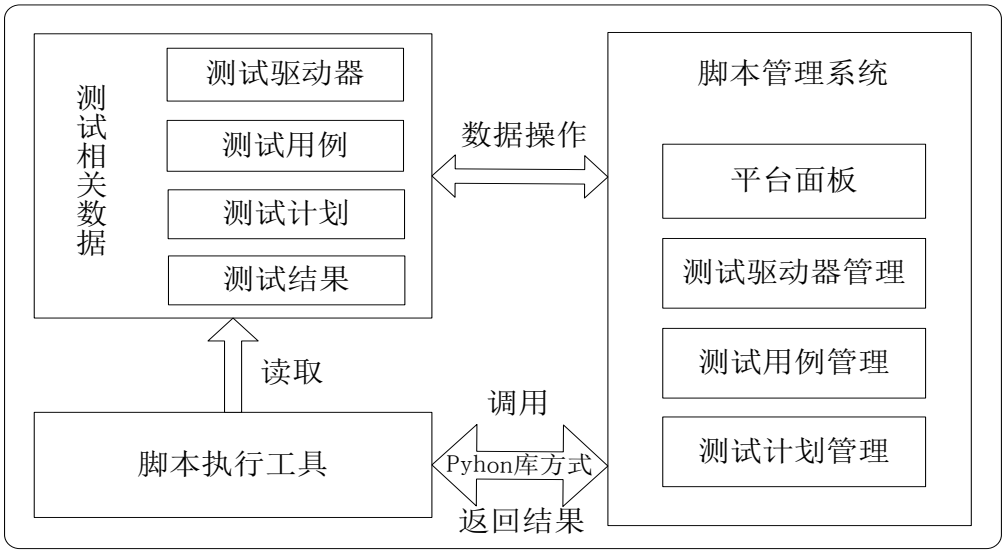


图 3.4 平台层次结构示意图

由图 3.4 可知，测试相关数据包括测试驱动器、测试用例、测试计划和测试结果四个部分，前三者以文件和数据库两种方式同步存在，测试结果只存在于数据库中。

脚本执行工具是测试任务的执行载体，具体主要包括执行测试用例、执行测试计划、生成测试报告和发送测试通知等功能。

脚本管理系统是指在浏览器端提供测试相关数据的业务管理功能，包括测试驱动器管理、测试用例管理、测试计划管理和平台面板四个功能模块。

脚本管理系统以 Python 库的方式调用该工具来执行测试任务并获得测试结果。

在该层次结构下，设计平台的整体功能结构如图 3.5 所示。

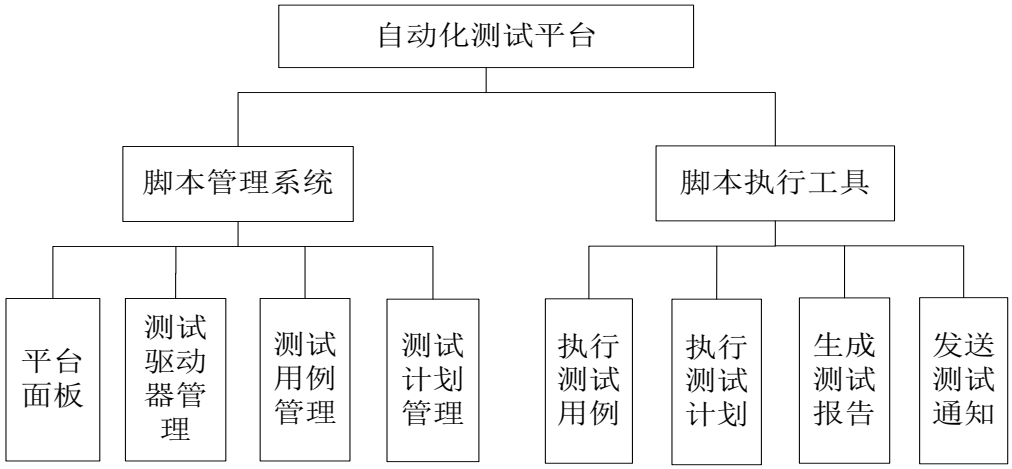


图 3.5 平台整体功能结构图

3.3.2 技术架构

平台采用 B/S 结构[31]，浏览器端通过 Http 接口[32]与服务端通信，服务端采用 web.py 框架，前端采用 jQuery +Ajax+JSON+TrimPath 的前端开发框架，数据库采用 MySQL[33]，平台的整体技术架构示意图如图 3.6 所示。

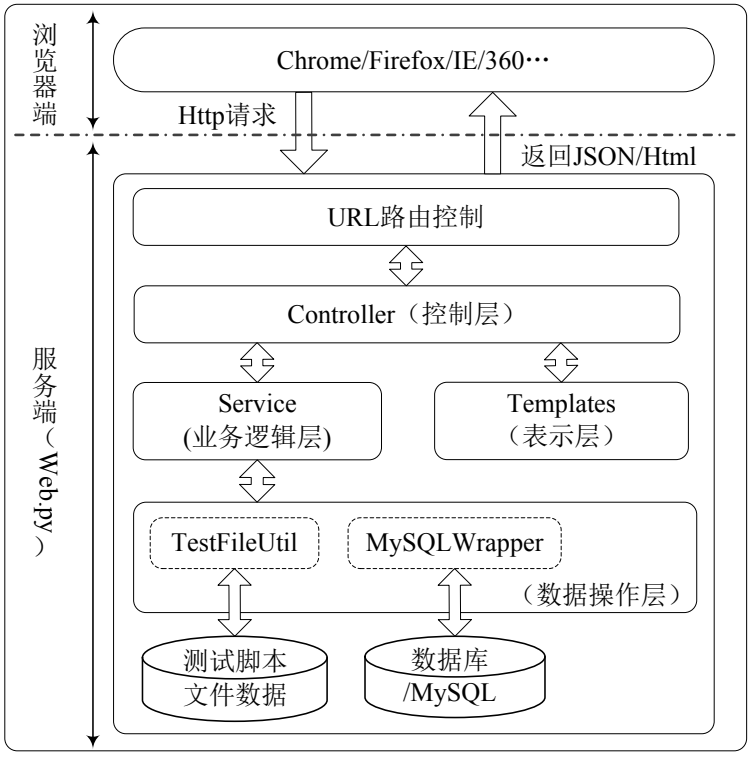


图 3.6 平台整体技术架构示意图

由图 3.6 可知，服务端采用“控制层-业务逻辑层-表示层-数据操作层”的开发结

构[34]，具体说明如下：

1）控制层：负责后台接口程序的执行流程。它首先会解析来自浏览器端的 URL，然后调用相应的业务逻辑处理方法，最后将业务处理后的数据经由表示层的模板渲染并返回。控制层串联了表示层和业务逻辑层。

2）业务逻辑层：负责脚本管理平台的全部业务处理功能。在执行脚本时，业务逻辑层需要调用脚本执行工具并获取测试结果。在做数据操作时，业务逻辑层需要调用数据操作层进行数据库读写操作。

3）表现层：负责将业务处理后的数据经由模板渲染后返回给浏览器。

4）数据操作层：负责对测试数据的读写操作功能，包括测试文件和传统数据库两个部分的数据。其中，测试文件的操作基于测试文件操作组件 TestFileUtil 实现的，TestFileUtil 是根据测试文件操作特性对 Python 自带的文件操作类库进行的一层接口封装；数据库的数据操作基于 MySQLWrapper 工具实现的，MySQLWrapper 是利用 Python 下的 MySQLConnector 类库对数据库常用操作进行的一层接口封装[35]。

该结构划分了各层的职责，降低了程序代码的耦合度，是 MVC 设计模式的一种具体实现。

3.3.3 数据库

平台采用 MySQL 作为数据源。MySQL 是一款体积小、速度快、免费、易维护、有着完善的安全性管理机制的关系型数据库管理系统，非常适合本系统。

整个数据库采用 InnoDB 引擎，它支持外键、事务处理和行级锁，具有高效的并发处理能力和数据恢复能力[36]。与此同时，数据表之间通过外键约束关系，外键可以保证数据的完整性，更可靠，外键关系也在一定程度说明了业务关系。平台的整体数据库结构关系如图 3.7 所示。

由图 3.7 可知，T_DRIVRE 是测试驱动器表，T_CASE 是测试用例表，测试用例表通过外键 DRIVER_ID 与测试驱动器表关联，T_PLAN 是测试计划表，测试计划表和测试用例表通过用例计划关联表 T_CASE_PLAN 关联，T_CASE_RESULT 是测试用例结果表，测试用例结果表通过外键 CASE_ID 关联测试用例表，T_PLAN_RESULT 是测试计划结果表，测试计划结果表通过外键 PLAN_ID 关联测试计划表。

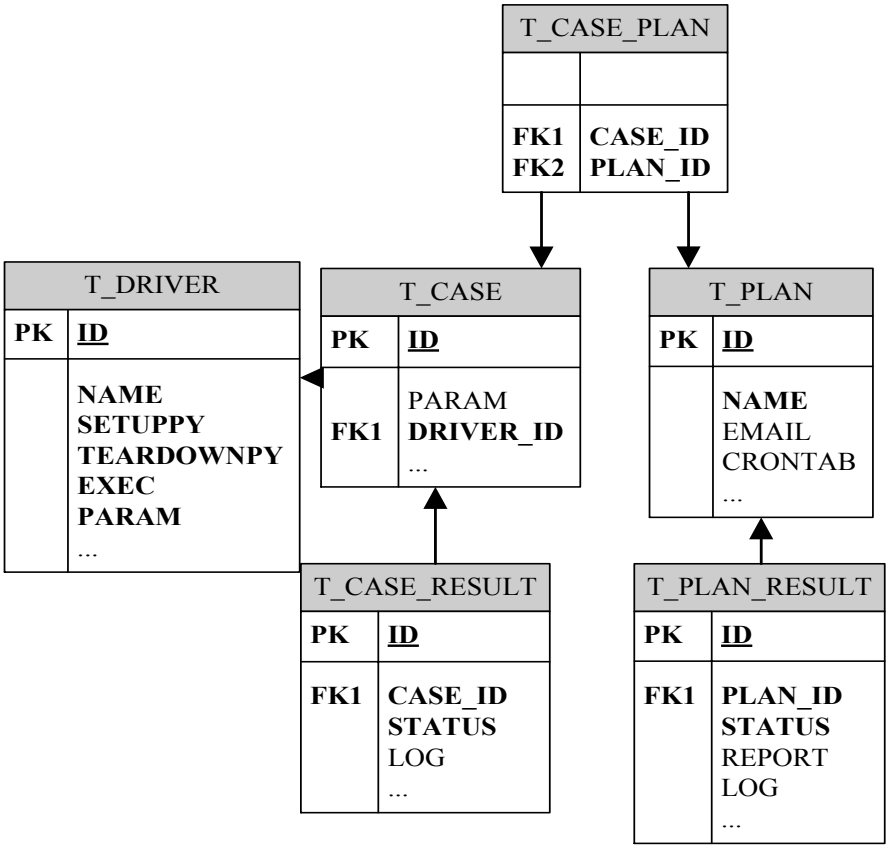


图 3.7 平台数据库的整体结构关系图

3.4 脚本执行工具的设计

平台的测试用例或测试计划均通过脚本执行工具执行驱动器脚本来完成的，该工具是平台的最基础也是最关键的构件，它主要包括执行测试用例、执行测试计划、生成测试报告和发送测试结果通知等功能。下面将在自动化测试方案的基础上阐述脚本执行工具的设计过程。

3.4.1 使用与安装方式

脚本执行工具可以被以 Python 库和终端命令这两种方式来调用。

其中，Python 库方式是为了能够在 Python 程序中调用脚本执行工具，脚本管理系统正是通过该方式来调用脚本执行工具执行测试任务的。脚本执行工具的 Python 库主要提供执行测试用例和执行测试计划两个接口，请求该接口后会返回一个测试结果对象，里面封装了测试任务的用例集概况、测试结果、测试报告和执行日志等信息。

由于测试开发人员不可能在脚本管理系统的 Web 界面中进行测试脚本的开发与

调试，他们需要在本地环境直接调用脚本执行工具。故脚本执行工具还提供终端命令方式执行测试任务的功能，从而方便测试开发人员撰写、调试和维护测试脚本数据。利用终端命令执行测试任务后，测试开发人员可以在终端屏幕上获取测试的执行日志和最终的执行结果。需要特别强调的是，利用终端命令执行测试任务的测试结果并不写入数据库，它只是对测试任务的一次“调试”。

此外，平台利用 Setuptools 工具对脚本执行工具的 Python 源程序进行打包[37]。Setuptools 是对 Python 工具包 distutils 增强版的集合，它可以使用一种更加透明的方法来查找、下载并安装依赖包；可以利用同一个包在系统中自由进行切换版本；可以声明对某个包的特定版本的需求。因此，只需要执行简单的终端命令即可安装或更新脚本执行工具。

3.4.2 脚本格式规范

本平台的脚本执行工具采用 Python 语言实现，因此测试驱动器脚本实际上也就是 Python 文件，而测试用例和测试计划是 JSON 格式的文本文件。

测试驱动器包括固件构建脚本、固件销毁脚本和测试逻辑执行脚本，分别命名为 setUp.py、tearDown.py 和 exec.py，它们的代码格式示例如图 3.8 所示。

```
'''以下是 setUp.py 的代码内容'''
def   setUp(ts):
      #固件构建代码
      Pass
'''以下是 tearDown.py 的代码内容'''
def   tearDown (ts):
      #固件构建代码
      Pass
'''以下是 exec.py 的代码内容'''
def   tc_case1 (ts):
      #case1 的测试逻辑代码
      pass
def   tc_case2(ts):
      #case2 的测试逻辑代码
      pass
```

图 3.8 测试驱动器脚本代码格式示例图

在 setUp.py 的 setUp 方法中可以进行测试环境准备、测试数据准备等固件构建操作。在 tearDown.py 的 tearDown 方法中可以进行测试环境销毁、测试数据销毁等固件销毁操作。在 exec.py 中可以定义具体的测试用例程序方法。若该固件下只有一

个测试用例，可以用默认的 runTest 方法。若该固件下有多个测试用例，可以以“tc_”开头定义多个方法，每个方法分别对应一条测试用例。在这三个脚本中，程序均通过传递测试套件实例变量“ts”来实现固件变量共享。在图 3.8 的示例代码中，tc_case1 和 tc_case2 拥有相同的测试套件固件，且通过变量 ts 传递共享套件实例内的变量。

测试用例的示例文件如图 3.9 所示。

```
{
    "Case Name ": "case_name",
    "Driver Name ": "driver_name",
    "Data": json_data
}
```

图 3.9 测试用例文件示例图

在测试用例文件中，“Case Name”是测试用例的名称；“Driver Name”是测试用例所对应的驱动器名称；“Data”是测试用例的驱动数据，它也是 JSON 格式的，且可以在测试套件实例中共享使用。

测试计划的文件如图 3.10 所示。

```
{
    "Plan Name":"plan_name",
    "Case List":"case_name1,case_name2",
    "Email List":"xxx@xxx.com"
}
```

图 3.10 测试计划文件示例图

在测试计划文件中，“Plan Name”是测试计划的名称；“Case List”是测试计划包含的测试用例合集；“Email List”是测试结果通知对象的邮箱。

3.4.3 接口设计

结合单元测试框架理论和自动化测试方案，设计脚本执行工具接口如图 3.11 所示。

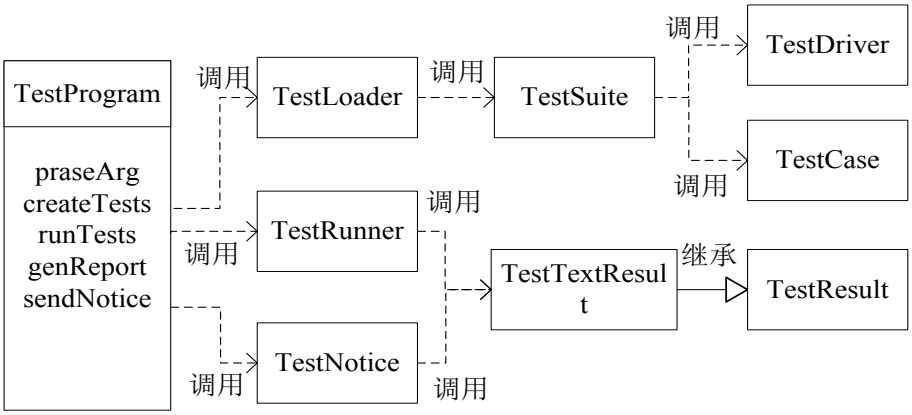


图 3.11 脚本执行工具接口设计图

1）TestProgram：测试工具的主程序，包含对终端命令的解析方法、Python 库的方法入口、执行测试任务流程的控制方法。

2）TestCase：测试用例对象，包括用例所对应的测试驱动器信息，用例名称和用例执行结果等信息。

3）TestSuite：测试套件对象，包含一个或多个测试用例对象信息，也可以嵌套包含一个或多个测试套件对象信息。

4）TestDriver：测试驱动器对象，包含驱动器名称、固件构建模块和固件销毁模块等信息。

5）TestLoader：测试实例加载器方法类，它会将某次执行任务内的所有测试用例实例初始化好。

6）TestRunner：测试实例执行器方法类，它会按一定规则执行 TestLoader 加载好的用例实例，并返回测试结果对象（TestResult）。

7）TestResult：测试结果对象基类，封装了测试任务的执行结果所包含的最基本信息，包括所测试对象的基本信息、测试结果、结果统计信息、测试报告信息、测试日志等。此外，测试工具还提供了 TestTextResult 对象，它继承于 TestResult，通过指定该测试结果对象，TestRunner 会在执行过程中将测试过程在终端屏幕上输出。

8）TestNotice：测试通知方法类，当测试计划执行结束后 TestRunner 会调用 TestNotice 来发送邮件通知。

3.5 脚本管理系统的设计

由平台层次结构的设计方案可知，脚本管理平台的脚本管理系统包括测试驱动器管理、测试用例管理、测试计划管理和平台面板四个功能模块。本小节将从详细

功能设计、数据库设计和程序接口设计三个角度分别阐述各个模块的设计过程。

3.5.1 测试驱动器管理模块

测试驱动器管理模块主要包括测试驱动器的查看、添加、修改和删除功能，其功能设计如图 3.12 所示。

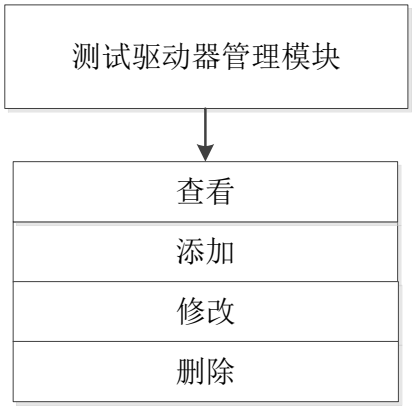


图 3.12 测试驱动器管理模块功能设计图

根据需求分析和功能设计方案，测试驱动器的表结构设计如表 3.1 所示。

表 3.1 测试驱动器对象数据表结构

名称	类型	属性	说明
ID	Int(4)	PK NOT NULL AUTO_INCREMENT	主键
NAME	CHAR(40)	NOT NULL UNIQUE	驱动器名称
DESCRIPTION	TEXT		驱动器说明
PARAM	TEXT	NOT NULL	默认的驱动数据（JSON）
SETUPPY	TEXT	NOT NULL	固件构建代码
TEARDOWNPY	TEXT	NOT NULL	固件销毁代码
EXECPY	TEXT	NOT NULL	主程序代码
TEST_TYPE	CHAR(100)		测试类型
PROJECT	CHAR(100)		所属项目名称

测试驱动器管理模块接口设计如图 3.13 所示。

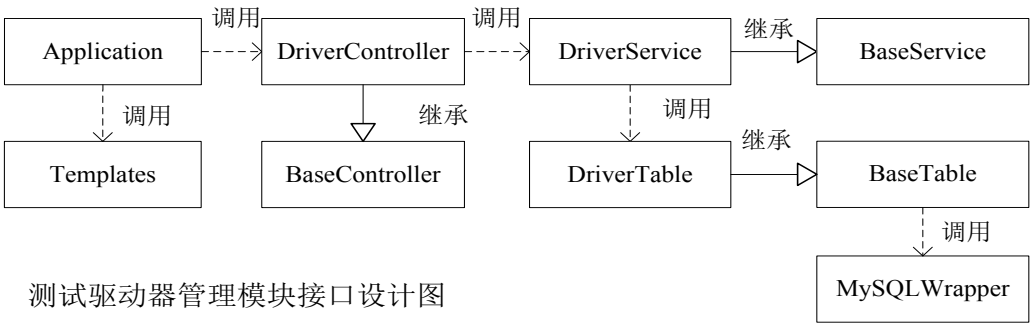


图 3.13 测试驱动器管理模块接口设计图

由图 3.13 知，DriverController 是驱动器管理模块的控制层方法类，它继承于控制层基类 BaseController。DriverService 是驱动器管理模块的业务逻辑层方法类，它继承于业务逻辑层基类 BaseService。DriverTable 是驱动器对象的数据操作层方法类，它继承于数据库对象基类 BaseTable。DriverService 通过调用 DriverTable 来进行数据库读写操作。BaseTable 通过数据操作工具 MySQLWrapper 来实现直接与数据库访问。Templates 是系统的模板资源，负责数据的页面显示。web.py 的内置 application 对象控制着系统的用户交互，它首先解析用户的测试驱动器管理请求，然后调用相应的业务逻辑层方法 DriverService，最后选择相应的模板资源 Templates 渲染并返回给用户。在下面的接口设计中不再赘述有关 Application、Templates、BaseController、BaseTable 和 MySQLWrapper 的内容。

3.5.2 测试用例管理模块

测试用例管理模块主要分为测试用例管理和测试用例执行结果管理两个部分，其功能设计图如图 3.14 所示。

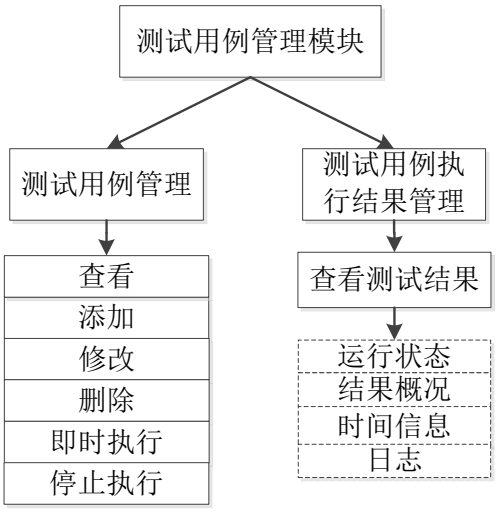


图 3.14 测试用例管理模块功能设计图

由图 3.14 可知，测试用例管理包括对测试用例的查看、添加、修改、删除、即时执行和停止执行功能；测试用例执行结果管理包括查看测试用例执行结果功能，能够查看的信息包括运行状态、运行概况、运行时间信息和运行日志等。

测试用例管理模块主要包括测试用例和测试用例执行结果两张表，它们的数据结构设计分别如表 3.2 和表 3.3 所示。

表 3.2 测试用例对象数据表结构

名称	类型	属性	说明
ID	Int(4)	PK NOT NULL AUTO_INCREMENT	主键
NAME	CHAR(40)	NOT NULL UNIQUE	测试用例名称
DESCRIPTION	TEXT		测试用例说明
DRIVER_ID	Int(4)	FK NOT NULL	用例对应的驱动器 ID，外键
PARAM	TEXT	NOT NULL	用例驱动数据（JSON）
TEST_TYPE	CHAR(100)		测试类型
PROJECT	CHAR(100)		所属项目名称

表 3.3 用例执行结果对象数据表结构

名称	类型	属性	说明
ID	Int(4)	PK NOT NULL AUTO_INCREMENT	主键
CASE_ID	Int(4)	NOT NULL	测试用例 ID，外键
DESCRIPTION	TEXT		运行结果概况

STATUS	CHAR(40)	NOT NULL	运行状态
LOG	CHAR(100)		日志地址
START_TIME	DATETIME		运行开始时间
STOP_TIME	DATETIME		运行结束时间
USER	CHAR(100)		执行人

测试用例管理模块接口设计如图 3.15 所示。

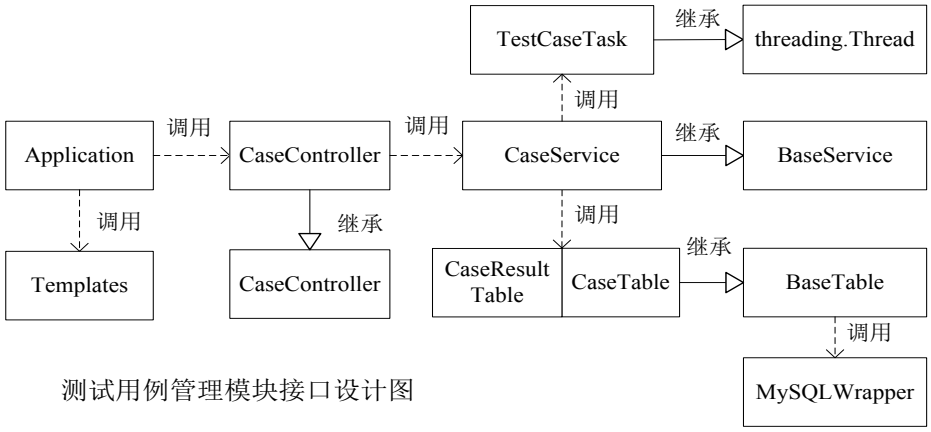


图 3.15 测试用例管理模块接口设计图

由图 3.15 可知，CaseController 是测试用例管理模块的控制层方法类，它继承于控制层基类 BaseController。CaseService 是测试用例管理模块的业务逻辑层方法类，它继承于业务逻辑层基类 BaseController。CaseTable 是测试用例对象的数据操作层方法类，CaseResultTable 是测试用例结果对象的数据操作层方法类，它们均继承于数据库操作层基类 BaseTable。CaseService 通过调用 CaseTable 和 CaseResultTable 来进行数据库读写操作。TestCaseTask 是执行测试用例任务时创建的多线程对象，它继承于 Python 自带的多线程类 threading.Thread[38]。CaseService 通过调用 TestCaseTask 来创建执行测试用例任务。

3.5.3 测试计划管理模块

测试计划管理模块主要分为测试计划管理和测试计划执行结果管理两个部分，其功能设计如图 3.16 所示。

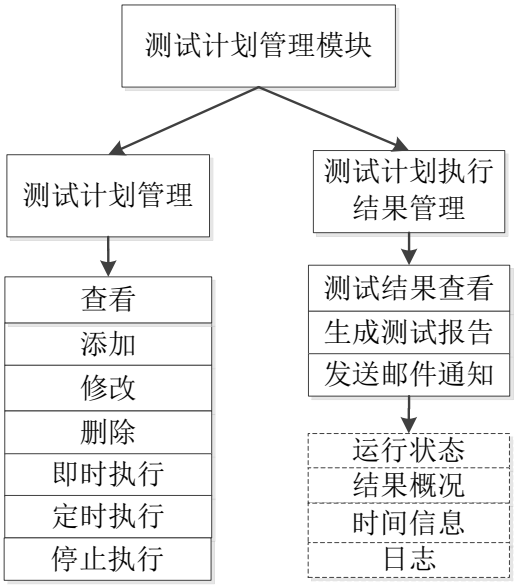


图 3.16 测试计划管理模块功能设计图

由图 3.16 可知，测试计划管理包括对测试计划的查看、添加、修改、删除、即时执行、定时执行和停止执行功能；测试计划执行结果管理包括查看测试计划执行结果、生成测试报告和发送测试通知功能，测试计划的执行结果信息包括运行状态、运行概况、测试报告、运行时间信息和运行日志等。

测试计划管理模块主要包括测试计划和测试计划执行结果两张表，它们的数据结构设计分别如表 3.4 和表 3.5 所示。

表 3.4 测试计划对象数据表结构

名称	类型	属性	说明
ID	Int(4)	PK NOT NULL AUTO_INCREMENT	主键
NAME	CHAR(40)	NOT NULL UNIQUE	测试计划名称
DESCRIPTION	TEXT		测试计划说明
CASE_LIST	TEXT	NOT NULL	计划包含的所有的用例的名称，以;隔开
CRONTAB	CHAR(100)		定时执行计划的定时器参数配置
PEOPLE	TEXT		通知对象邮箱，以;隔开
LAST_STATUS	CHAR(40)	NOT NULL	计划的上一次运行状态
LAST_TIME	DATETIME		计划的上一次运行时间
TEST_TYPE	CHAR(100)		测试类型
PROJECT	CHAR(100)		所属项目名称

表 3.5 测试计划执行结果对象数据表结构

名称	类型	属性	说明
ID	Int(4)	PK NOT NULL AUTO_INCREMENT	主键
PLAN_ID	Int(4)	NOT NULL	测试计划 ID，外键
DESCRIPTION	TEXT		运行结果概况
STATUS	CHAR(40)	NOT NULL	运行状态
LOG	CHAR(100)		日志地址
START_TIME	DATETIME		运行开始时间
STOP_TIME	DATETIME		运行结束时间
USER	CHAR(100)		执行人
REPORT	CHAR(100)		测试报告地址

测试计划管理模块接口设计如图 3.17 所示。

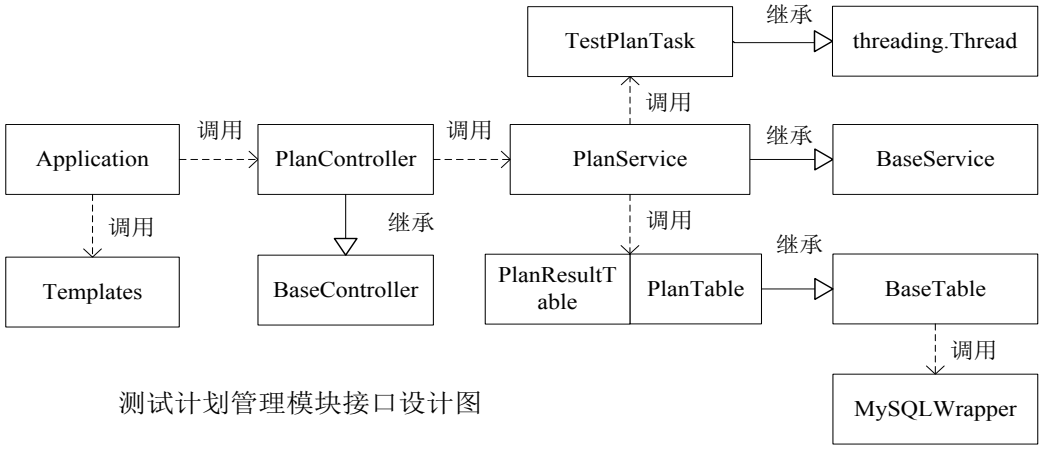


图 3.17 测试计划管理模块接口设计图

由图 3.17 知，PlanController 是驱动器管理模块的控制层方法类，它继承于控制层基类 BaseController。PlanService 是测试计划管理模块的业务逻辑层方法类，它继承于业务逻辑层基类 BaseService。PlanTable 是测试计划对象的数据操作层方法类，PlanResultTable 是测试计划结果对象的数据操作层方法类，它们均继承于数据库操作层基类 BaseTable。PlanService 通过调用 PlanTable 和 PlanResultTable 来进行数据库读写操作。TestPlanTask 是执行测试计划任务时创建的多线程对象，它继承于 Python 自带的多线程类 threading.Thread。PlanService 通过调用 TestPlanTask 来创建执行测试计划任务。

3.5.4 控制面板模块

测试平台面板模块是对平台整体状况的展示和管理，包括测试数据统计、测试

结果统计、测试工具配置和任务扫描器管理等功能，其功能设计如图 3.18 所示。

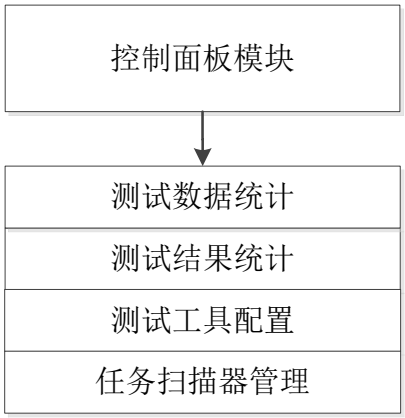


图 3.18 控制面板模块功能设计图

其中，测试数据统计是对测试驱动器、测试用例和测试计划数量的统计，它可以从侧面反映项目的测试覆盖率；测试结果统计是对测试用例或测试计划执行结果的运行状态统计，它可以在一定程度反映项目的质量和稳定性；测试工具配置功能是指对测试驱动器路径、测试报告模板、通知邮件服务器等工具配置的管理；任务扫描器管理是对扫描测试任务的线程的启动与停止功能。

控制面板模块的接口设计如图 3.19 所示。

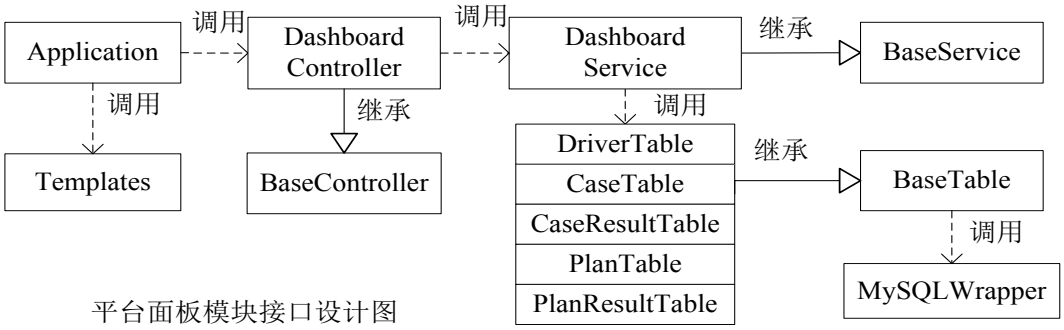


图 3.19 控制面板模块接口设计图

由图 3.19 知，DashboardController 是控制面板模块的控制层方法类，它继承于控制层基类 BaseController。DashboardService 是控制面板模块的业务逻辑层方法类，它继承于业务逻辑层基类 BaseService。必要时，DashboardService 会调用 DriverTable、CaseTable、CaseResultTable、PlanTable 或 PlanResultTable 来进行数据库读写操作。

3.6 本章小结

本章主要讲述了平台的分析和设计过程。首先对平台进行了整体的需求分析；

然后从自动化测试方案、层次结构、技术架构、数据库四个角度对平台进行了整体的方案设计；接着阐述了脚本执行工具的设计过程；最后按照业务功能模块划分介绍了平台的脚本管理系统的设计过程。

4 平台的实现

本章主要讲述平台的实现过程，首先介绍了所需要的开发环境配置，然后阐述了脚本执行工具的实现过程，最后介绍了脚本管理系统的实现过程。

4.1 开发环境配置

平台主要采用 Python 编程实现，其开发环境配置如表 4.1 所示。

表 4.1 平台开发环境配置表

开发工具	PyCharm 5.0
数据库	MySQL 5.6.21
浏览器	Chrome、Firefox、IE
Http 服务器	Nginx 1.7
Python 版本	Python 2.7.6
web.py 版本	web.py 0.37

4.2 脚本执行工具的实现

4.2.1 工具功能的实现

根据脚本执行工具的设计方案，实现脚本执行工具的执行流程如图 4.1 所示，下面将按照这个流程讲述执行测试任务功能的实现过程。

1）初始化配置文件

初始化脚本执行工具的全局配置，包括用例文件名的默认前缀、驱动器的存放路径、测试报告模板的存放路径、邮件服务器主机地址、邮件服务器账号和邮件服务器密码等信息。

2）解析用户请求

判断用户是执行测试用例还是执行测试计划，并根据解析结果调用测试加载器加载相应的测试目标。

其中，Python 库方式的用户请求是直接引用测试工具的 Python 库接口实现的，不需要特别作程序判断，若调用 runCase 接口则是执行测试用例，若 runPlan 接口则是执行测试计划。

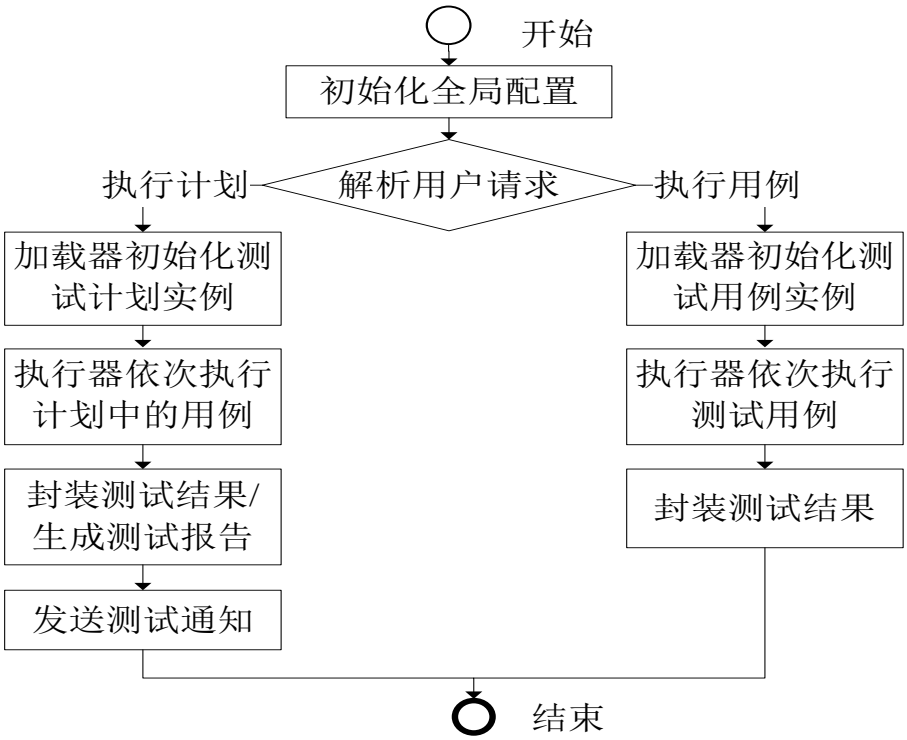


图 4.1 脚本执行工具的执行流程

终端命令方式的用户请求是基于 Python 的 getopt 库实现解析的[39]，首先用 getopt 库将终端命令分割为数组，然后判断数组是否包含指定的关键字，若含有短命令“-c”或长命令“--case”即执行测试用例，若含有短命令“-p”或长命令“--plan”即执行测试计划，其核心实现代码如图 4.2 所示。

```
def parseArgs(self, argv):
    long_opts = ["help", "version","case=", "plan=", "driver", "config"]   #长命令
    short_opts= "hHvc:p:d"                                                  #短命令
    try:
        options, args = getopt.getopt(argv, short_opts, long_opts)   #分割命令参数
        for opt, value in options:
            if opt in ('--case','-c'):
                #执行测试用例操作处理
            elif opt in ('--plan','-p'):
                #执行测试用例操作处理
            elif opt in ('-h','-H','--help'):、
                #显示工具帮助信息
            #省略一些次要的解析
    except getopt.error, msg:
        #命令解析异常处理
```

图 4.2 脚本执行工具解析终端命令的核心实现代码

3）加载器初始化测试实例

测试加载器根据测试目标将所有测试用例实例初始化好，并封装成一个测试套件实例返回，加载时具体分以下三种情况：

①若加载目标是测试用例文件名，首先利用测试文件操作工具 TestFileUtil 从 JSON 文本中解析出测试用例名称、测试驱动器名称和测试驱动参数等信息，然后根据驱动器名称初始化驱动器实例，接着根据以上信息初始化测试用例实例，最后将测试用例实例封装成一个测试套件实例并返回，其实现流程如图 4.3 所示，其核心实现代码如图 4.4 所示。

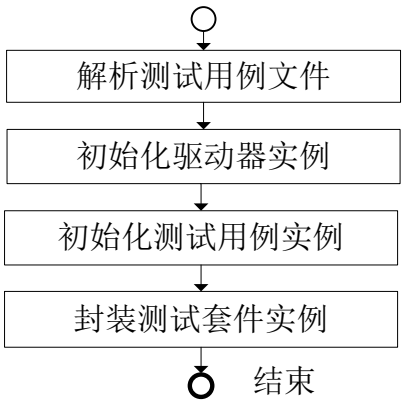


图 4.3 加载器加载测试用例文件的实现流程

```
#从测试用例文件加载测试用例实例
def loadTestsFromFile(self, path, prefix=None, projectName=None):
    case_json = TestFileUtil.prase(path, prefix= prefix)     #解析测试用例文件的 JSON
    driver_name = case_json ['Driver Name']
    case_name = case_json ['Case Name']
    driver = self.loadDriverFromName(driver_ame)          #初始化测试驱动器实例
    test = TestCase(case_ame,   driver)                   #初始化测试用例实例
    test.data = self.loadParam(case_json)                 #初始化测试用例驱动参数
    return TestSuite ([test],   projectName=projectName)
```

图 4.4 加载器加载测试用例文件的核心实现代码

②若加载目标是测试用例文件路径，则遍历搜索该路径下符合命名规范的测试对象（默认是以“tc_”前缀），并依次根据测试用例文件初始化测试用例实例，最后根据这些测试用例实例初始化一个测试套件实例并返回，其实现流程如图 4.5 所示，

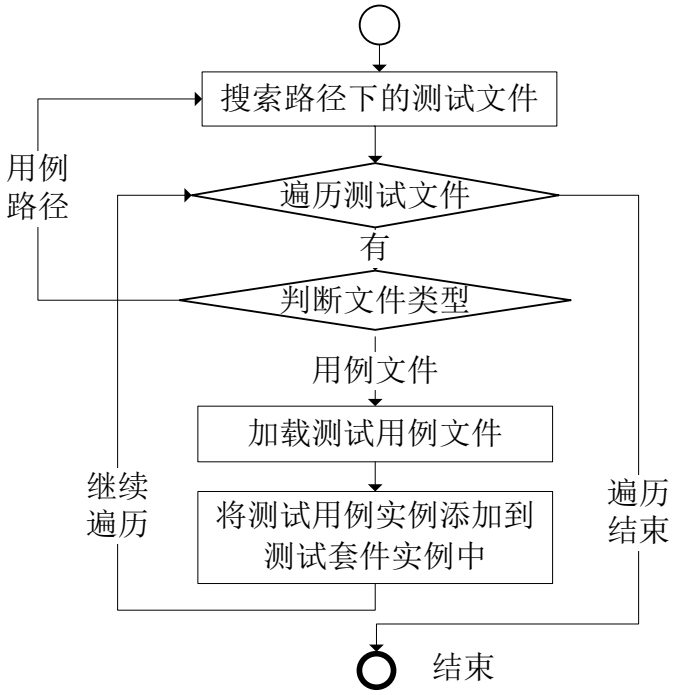


图 4.5 加载器加载测试用例文件路径的实现流程

③若目标是测试计划文件，则首先解析出 JSON 文本中的测试计划名称、测试用例集和通知人员邮箱等信息，然后根据测试用例集依次初始化为测试用例实例，最后根据这些测试用例实例初始化一个测试套件实例并返回，其实现流程如图 4.6 所示，其核心实现代码如图 4.7 所示。

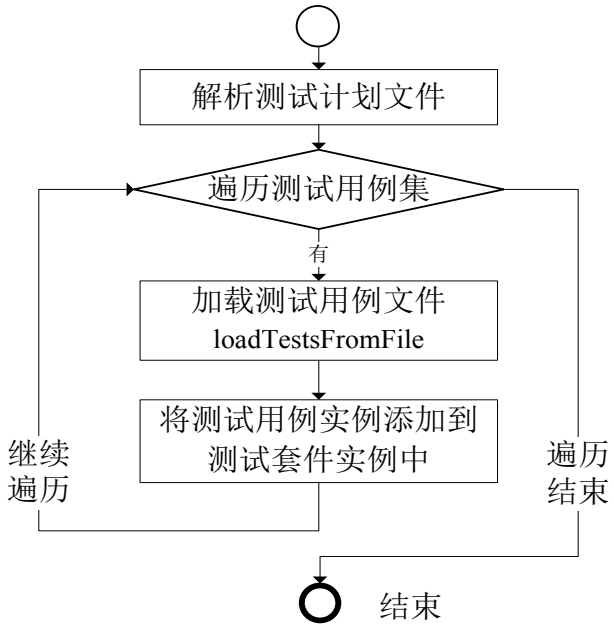


图 4.6 加载器加载测试计划文件的实现流程

```
#从测试计划文件加载测试用例实例
def loadTestsFromPlan(self, path):
    plan_json = TestFileUtil.prase(path)                    #解析测试用例文件的 JSON
    tests = []
    for caseName in plan_json["Case List"].split(","):
        caseFile = os.path.join(base_dir, caseName)
        if os.path.isdir(caseFile):
            tests.append(self.loadTestsFromDir(caseFile))     #从路径初始化用例实例
        elif os.path.isfile(caseFile):
            tests.append(self.loadTestsFromFile(caseFile))    #从路径初始化用例实例
    return TestSuite(tests, projectName=plan_json["Plan Name"])
```

图 4.7 加载器加载测试计划文件的核心实现代码

4）执行器执行测试用例

执行器按照自动化单元测试框架的流程依次执行测试用例，执行器的执行流程如图 4.8 所示，其核心实现代码如图 4.9 所示。

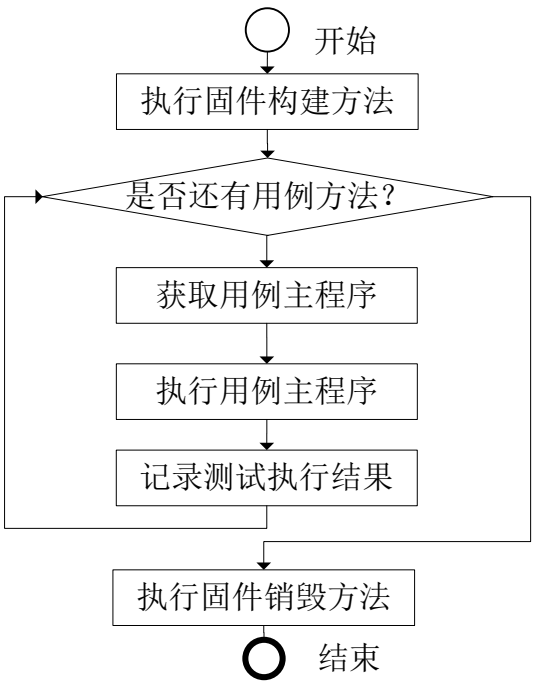


图 4.8 执行器的执行流程

由图 4.8 和图 4.9 可知，在执行一个测试驱动器下的测试用例时，首先执行固件构建方法；然后遍历执行“exec.py”中名为 runTest 或名称以“tc_”开头的方法，若在执行过程中抛出异常则执行失败，否则成功；最后执行固件销毁方法。

```
def run(self, result):
    self.setUp()          #固件构建
    for testMethod in self.testMethods:
        self.currentTestMethod = testMethod
        result.startTest(self)
        testFn = getattr(self._execModule, testMethod)    #获取测试主程序执行方法
        try:
            testFn(self)          #执行测试主程序
        except:
            result.addError(self, sys.exc_info())     #执行失败，省略一些异常处理代码
        else:
            result.addSuccess(self)         #执行成功
        finally:
            result.stopTest(self)
    self.tearDown()           #固件销毁
```

图 4.9 执行测试用例的核心实现代码

5）封装测试结果、生成测试报告

当用例执行完毕后，执行器会对测试结果信息进行梳理统计并返回。如果是执行测试计划，执行器还会根据这些结果信息填充测试报告的 HTML 模板从而生成测试报告的静态 HTML 文件，其核心实现代码如图 4.10 所示，测试报告中包含的内容有测试项目名称、测试起始与结束时间、通知对象邮箱等信息、测试结果概况、用例成功率统计信息和测试日志。

```
def generateHtmlReport(self, result):
    template = file(self.emailTemplatePath).read()                    #读取测试报告模板
    template = template.replace('$planName',result.description)       #测试项目名称
    template = template.replace('$emailList',result.emailList)        #通知对象邮件
    template = template.replace('$duration','Total %s Seconds. ( %s -- %s )' %
(result.timeTaken, result.startTime, result.stopTime))                #测试起止时间
    template = template.replace('$statistics','Passed: %i,  Failed: %i,  Aborted: %i' %
(len(result.successes), len(result.failures), len(result.errors)))    #测试成功率统计
    template = template.replace('$log',result.log.getvalue())         #测试日志
    results = []
    for item in result.results:
        results.append('<tr><td class="%s">%s</td><td>%s</td></tr>' % (item[1], item[1],
        item[0]))
    template = template.replace('$results',''.join(results))          #测试结果概况
    return template
```

图 4.10 生成测试报告 HTML 文本的核心实现代码

5）发送测试通知

当测试计划执行完毕后，执行器会调用测试通知方法类将测试报告作为邮件内容发送给测试结果通知对象，其核心实现代码如图 4.11 所示。

```
def sendEmailNotice(self, result):
    template = self.generateHtmlReport(result)    #根据报告模板将测试结果生成测试报告
    msg = MIMEText(template, _subtype='html')
    msg['Subject'] = result.description
    msg['From'] = self.mail_from              #邮件主题
    msg['To'] = result.emailList              #邮件收件人，即测试结果通知对象
    self.smtp.connect(self.mail_host)         #连接邮件服务器
    self.smtp.login(self.mail_user, self.mail_pwd)          #登入邮件服务器
    self.smtp.sendmail(self.mail_from, result.emailList.split(','), msg.as_string()) #发送
    self.smtp.quit()
```

图 4.11 发送测试通知的核心实现代码

4.2.2 工具的打包与安装

根据脚本执行工具设计方案可知平台利用 Setuptools 对工具程序进行打包，脚本执行工具的打包安装代码如图 4.12 所示。

```
from setuptools import setup, find_packages
setup(name = 'stest',
      version = "1.0.0",
      packages = find_packages(),
      package_data = {'':['*.txt','*.html','*.md',]},
      entry_points = {'console_scripts': ['stest = stest:main',]} )
#添加默认的配置文件
fileUtil.copyfile('./stest/stest.conf.template', conf_file)
#添加默认的测试驱动器路径
fileUtil.mkdir(driver_path)
#添加默认的测试报告模板文件
fileUtil.copyfile('./stest/EmailTemplate.html',email_template_file)
```

图 4.12 脚本执行工具的打包安装代码

在图 4.12 的打包安装代码中，name 指定了工具的名称，version 是工具的版本信息，packages 指定了工具的依赖包，package_data 指定了包中所应包括或排除的文件，entry_points 指定了使用包时的程序入口，即“stest:main”程序。通过配置 entry_points 字段，Setuptools 在打包时不仅会生成 Python 第三方库文件还会在操作系统的环境变量路径下生成一个可执行文件。其中，Python 第三方库文件使得测试执行工具可以以 Python 库的方式使用，可执行文件使得测试执行工具可以以终端命令的方式使用。

安装测试执行工具时，首先需要在终端对工具的源码执行“python setup.py bdist_egg”命令来制作一个分发包，此时会在当前路径的 dist 文件夹生成一个尾缀为

egg 的文件；然后执行“python setup.py install”命令，该命令会将创建的 egg 包安装到 Python 默认的第三方库路径下，并将测试工具的可执行文件添加到操作系统的环境变量路径下，通常在“/usr/local/bin”文件夹下。

4.2.3 工具的使用

为了在方便程序代码或终端中调用脚本工具，将其命名为“stest”。而工具的具体使用方法说明如下：

1）Python 库方式

以 Python 库的方式调用脚本执行工具的示例代码如图 4.13 所示。其中，stest 是工具的 Python 库名称，runCase 是执行测试用例的方法接口，runPlan 是执行测试计划的方法接口，xxx 是测试用例资源或测试计划资源的路径。

```
import stest
#执行测试用例
test_result = stest.runCase(xxx)
#执行测试计划
test_result = stest.runPlan(xxx)
```

图 4.13 Python 库的方式调用脚本执行工具的示例代码

2）终端命令方式

脚本执行工具的命令提示信息如图 4.14 所示。

```
Usage: stest [options] [argv] [...]
options:                                                    #命令说明
    -h, --help          Show this help message
    -v, --version       Show tool version info
    -c, --case          run test cases
    -p, --plan          run test plans
    --conf              show configurations help message
Examples:                                                   #命令示例
stest -c xxx                    - run the case task
stest -p xxx.plan.json          - run the plan task
```

图 4.14 终端命令调用脚本执行工具的示例代码

由图 4.13 可知，可以通过短命令 stest -c xxx 或长命令 stest --case xxx 来执行测试用例，xxx 是测试用例文件或路径；可以通过短命令 stest -p xxx.plan.json 或长命令 stest --plan xxx.plan.json 来执行测试计划，其中 xxx.plan.json 是测试计划文件；可以通过短命令 stest -h 或长命令 stest -help 来获取工具的帮助信息；可以通过短命令 stest -v 或长命令 stest –verison 来获取工具的版本信息；可以通过长命令 stest –conf 来获取工具的配置信息。

4.3 脚本管理系统的实现

平台的脚本管理系统的后台部分采用的 web.py 框架，前台部分采用 jQuery +Ajax+JSON+TrimPath 的混合结构。脚本管理系统的 web.py 配置如图 4.15 所示，包括 URL 与控制层方法类的映射关系、 session 配置和前端模板配置等内容。

```
import web
#请求 URL 映射控制信息
urls = ("/driver/(\d+)","DriverController.ViewDriver",
    "/driver/(\d+)/mod","DriverController.ModifyDriver",
    #省略一些配置)
#web.py 配置
web.config.debug = False
app = web.application(urls, globals(), autoreload = True)
#SESSION 配置
if web.config.get('_session') is None:
    store = web.session.DiskStore('./sessionStore')
    session = web.session.Session(app, store)
else:
    session = web.config._session
#模板文件配置
render = web.template.render(cur_dir + '/template/', base='base', cache = False,
globals={'session': session})
```

4-15 图 web.py 基本配置的核心实现代码

本章节主要实现脚本管理平台的功能业务模块，下面将通过描述功能流程、列出实现代码等方式分别阐述测试驱动管理模块、测试用例管理模块、测试计划管理模块和平台面板模块的实现过程。

4.3.1 测试驱动器管理模块

根据 3.5.1 节的设计方案，实现测试驱动器管理模块的各个功能，下面将详细阐述其实现过程。

测试驱动器管理模块主要包括测试驱动器的查看、添加、修改和删除功能，根据这些功能配置测试驱动器管理模块的 URL 映射如图 4.16 所示。

```
urls = ("/driver","DriverController.ListDriver",
    "/driver/(\d+)","DriverController.ViewDriver",
    "/driver/(\d+)/mod","DriverController.ModifyDriver",
    "/driver/(\d+)/add","DriverController.AddDriver",
    "/driver/(\d+)/del","DriverController.DeleteDriver")
```

图 4.16 测试驱动器管理模块 URL 映射的配置代码

在图 4.16 中，URL 与控制层的方法相映射。当接收到浏览器端的 Http 请求时，web.py 会通过正则表达式匹配 urls 映射关系，并找到相应的控制层方法类进行业务逻辑处理返回视图资源。测试驱动器模块的控制层 DriverController 的部分实现代码如图 4.17 所示。

```
class ListDriver(Object):
    def GET(self, id):
        driver = plan_ctl.getDriver(id)
        return render.listDriver(driver)
    def POST(self, id):
        submit_data = web.input()
        result = plan_ctl.getDrivers(submit_data)
        return json.dumps(result)
class ViewDriver(Object):
    def GET(self, id):
        #……
        return render.viewDriver(driver)
class ModifyDriver(BaseController):
    #……
class AddDriver(Object):
    #……
class DeleteDriver(Object):
    #……
```

图 4.17 测试驱动器管理模块的控制层的部分实现代码

在图 4.17 中，DriverController 中的 ListDriver、ViewDriver、ModifyDriver、AddDriver、DeleteDriver 分别对应了驱动器列表、查看驱动器、修改驱动功能、添加驱动器和删除驱动器的功能类。请求 URL 的 Http 方法和控制层的功能类的方法相同，以图和图的映射关系为例，若是向“/driver”接口发送 GET 方法请求，则 web.py 则会映射至 DriverController 中的 ListDriver 类的 GET 方法执行相关程序。下面讲述测试驱动器管理模块功能的具体实现流程。

1）查看测试驱动器

用户进入测试驱动器列表页后，可以查看所有测试驱动器的基本信息点击测试驱动器名称进入测试驱动器详情查看页。

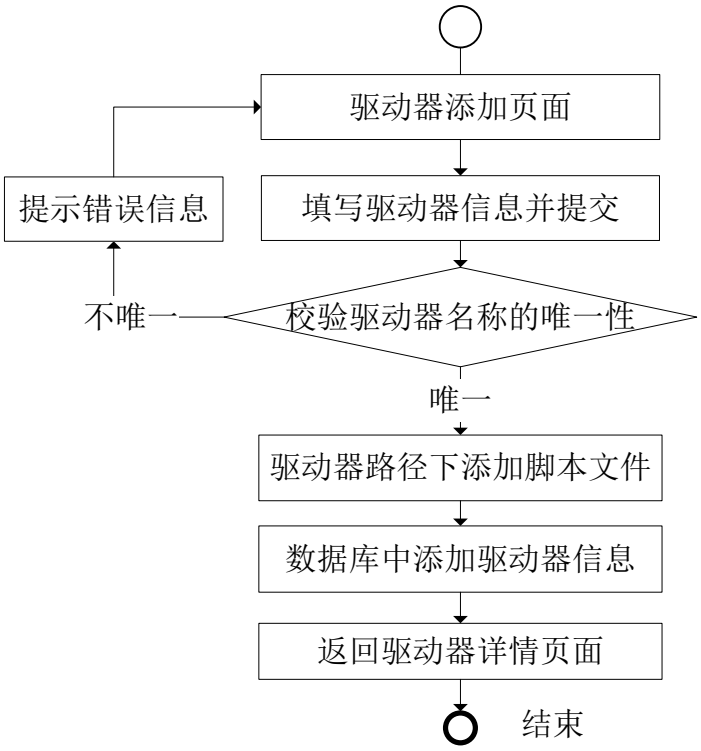


图 4.18 添加测试驱动的实现流程

2）添加测试驱动器

在驱动器列表页或驱动器详情页，点击”添加”按钮可以进入驱动器添加页。添加驱动器的流程如图 4.18 所示。添加测试驱动器的业务逻辑层核心实现代码如图 4.19 所示。

```
#添加测试驱动器
def addDriver(self,    form_data):
      driver_name = form_data['name']
            ret = self. driverTable .isUnique(driver_name)            #校验唯一性
      if not ret:
            return RT.ERR, 'driver [%s] exists in the system' % driver_name
            ret = TestFileUtil.addDriver(form_data)              #测试驱动器路径下添加脚本
文件
      if not ret:
            return RT.ERR, 'add script of driver [%s] failed' % driver_name
            ret = self.driverTable.insert(form_data)          #数据库中添加测试驱动器数据
      if not ret:
            return RT.ERR, 'add driver [%s] into db failed for [%s]' % (driver_name, ret[1])
```

图 4.19 添加测试驱动器业务逻辑层核心实现代码

由图 4.19 可知，DriverService 中的 addDriver 方法是添加测试驱动器的业务逻辑层实现。首先，它会调用测试驱动器的数据操作层对象 DriverTable 的 isUnique 方法进行测试驱动器名称的唯一性校验，若已存在，则返回添加页面并显示添加失败的提示信息；然后，调用测试脚本文件操作工具 TestFileUtil 的 addDriver 方法在测试驱动器路径下添加测试驱动器的脚本文件，包括：固件建立脚本文件、固件销毁脚本文件、测试主程序脚本文件和默认驱动参数的 JSON 文件；最后，调用 DriverTable 在 MySQL 数据库中添加该条测试驱动器数据。

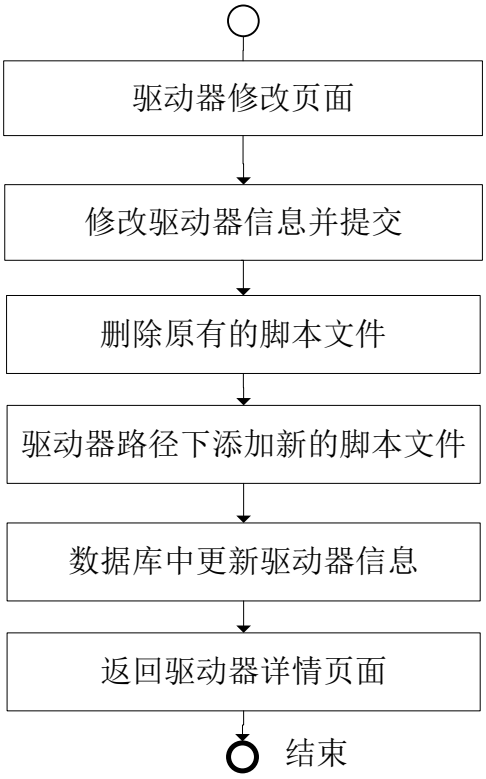


图 4.20 修改测试驱动器的实现流程图

3）修改测试驱动器

在测试驱动器修改页，用户可以修改测试驱动器的描述信息和脚本文件信息，且不能修改测试驱动器名称，其主要实现流程如图 4.20 所示。修改时，首先调用 TestFileUtil 删除原有测试驱动器的脚本文件；然后调用 TestFileUtil 在测试驱动器路径下添加新的驱动器脚本文件；最后调用 DriverTable 接口在 MySQL 数据库中更新该条测试驱动器数据。

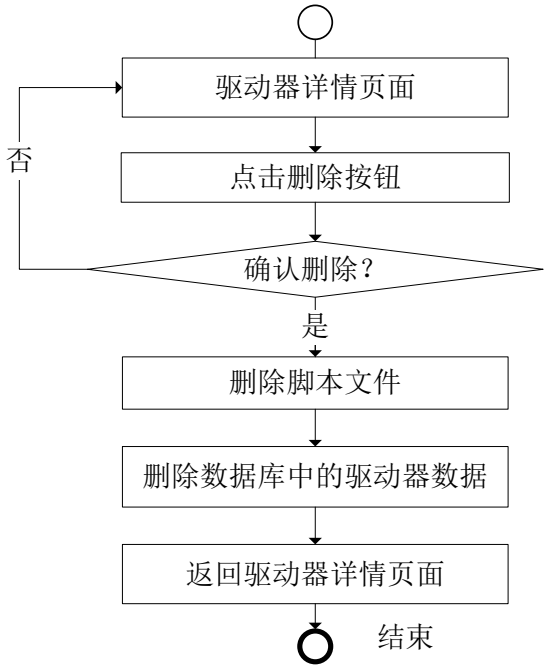


图 4.21 修改测试驱动的实现流程图

4）删除测试驱动器

在测试驱动器的详情页，用户可以点击”删除”按钮删除测试驱动器，其主要实现流程如图 4.21 所示。删除时，首先调用 TestFileUtil 删除该驱动器的脚本文件；然后调用 DriverTable 接口删除 MySQL 数据库中该条测试驱动器数据。

4.3.2 测试用例管理模块

根据 3.5.2 节的设计方案，实现测试用例管理模块的各个功能，下面将详细阐述其实现过程。

根据测试用例管理模块的功能设计方案配置测试用例管理模块的 URL 映射如图 4.22 所示。

```
urls = ("/case","CaseController.ListCase",
        "/case/(\d+)"," CaseController.ViewCase",
        "/case/(\d+)/mod","CaseController.ModifyCase",
        "/case/(\d+)/add","CaseController.AddCase",
        "/case/(\d+)/del","CaseController.DeleteCase",
        "/case/(\d+)/run"," CaseController.RunCase",
        "/case/(\d+)/stop"," CaseController.StopCase",
        "/case/(.+)/result"," CaseController.ListCaseResult")
```

图 4.22 测试用例管理模块 URL 映射的配置代码

由图 4.22 可知，CaseController 中的 ListCase、ViewCase、ModifyCase、AddCase、

DeleteCase、RunCase、StopCase、ListCaseResult 分别对应了测试用例列表、查看测试用例、修改测试用例、添加测试用例、删除测试用例、执行测试用例、停止用例任务和测试用例执行结果列表的功能类。

测试用例管理模块的增删改查功能的实现过程与测试驱动器管理模块类似，故本章节不再赘述，下面主要讲述执行测试用例功能的实现流程。

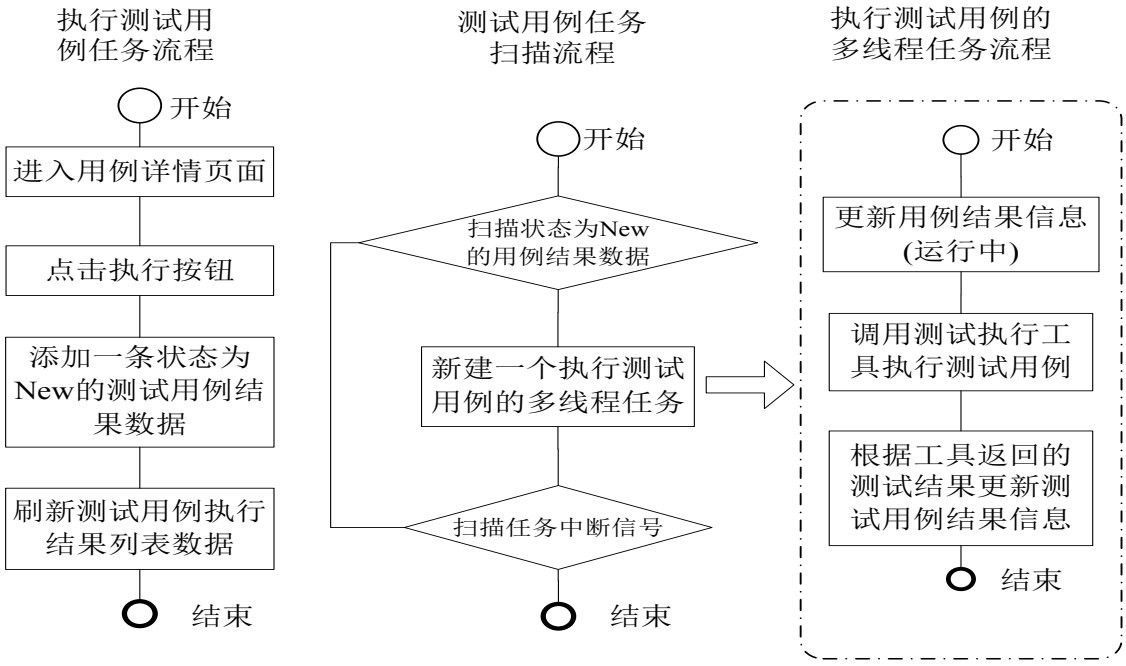


图 4.23 执行测试用例功能的实现流程图

用户进入测试用例详情页，点击”执行”按钮可以执行该测试用例。考虑执行测试用例的时间长短问题，平台采用多线程的方式执行测试用例任务，其实现流程如图 4.23 所示，具体流程说明如下：

1）首先，用户访问执行测试用例接口后，并不是立即添加一个测试用例多线程任务，而是添加一条状态为“New”的测试用例结果数据，其业务逻辑实现代码如图 4.24 所示。

```
def runCase(self, id):
    cs = self.getCase(id)
    case_result = {}
    case_result['case_name'] = cs['name']
    case_result['status'] = 'new'
    case_result['create_time'] = datetime.datetime.now()
    ret = self.caseResultTable.insert(case_result)
    if not ret[0]==RT.SUCC:
        return RT.ERR, 'insert case result to db failed and case name is [%s]' % cs['name']
    return RT.SUCC, 'start run case task successfully'
```

图 4.24 测试用例添加执行结果数据功能的业务层实现代码

2）然后，在平台部署时启动的测试用例任务扫描器会不断扫描状态为“New”的测试用例结果数据，若扫描到，则根据该条测试用例结果数据添加一个执行测试用例的多线程任务，其核心实现代码如图 4.25 所示。

```
def startCaseTask():
    caseResultTable = CaseResultTable()
    while True:
    #扫描状态为“New”的测试用例结果数据
        rt,tc_result_list = caseResultTable.selectAll({'status':'new'})
        if not len(tc_result_list)==0:
            #添加一个用例执行的多线程任务
            for tc_result in tc_result_list:
                tc_task = TestCaseTask(tc_result)
                tc_task.start()
        time.sleep(0.5)
```

图 4.25 测试用例扫描器的核心实现代码

3）最后，执行测试用例的多线程任务会调用测试工具执行测试用例并根据返回的测试结果更新数据库中的测试用例结果信息。测试用例多线程实现类是继承自 Python 自带的多线程库 threading.Thread，通过重写该类的 run()方法可以自定义线程的执行流程，通过调用 start()方法启动该线程，它的主要实现代码如图 4.26 所示。在执行时，首先更新测试用例执行结果状态为运行中，然后调用测试执行工具库执行测试目标，最后根据工具库返回的测试结果更新测试执行结果信息，具体包括测试状态、测试结果、执行时间和执行日志等信息。

```
class TestCaseTask(threading.Thread):
    def __init__(self, case_result):
        #省略参数初始化操作
        super(TestCaseTask, self).__init__()
    def prepareTask(self):
        #准备任务操作
    def stopTask(self):
        #停止任务操作
    def runTask(self):
        #执行任务操作
        #更新测试结果状态
        self.caseResultTable.update({'id': self.case_result_id}, {'status': 'Running'})
        #调用测试工具执行测试用例
        test_result = stest.runCase(self.case_path)
        #状态更新、记录结果和执行时间、记录执行日志
        ret = self.caseService.updateResult(self.case_result_id,test_result)
    def run(self):
        self.prepareTask()
        self.runTask()
        self.stopTask()
```

图 4.26 执行测试用例多线程任务的核心实现代码

此外，停止用例任务功能是根据线程 ID 杀死后台进程并更新测试用例执行结果信息来实现的。

4.3.3 测试计划管理模块

根据 3.5.3 节的设计方案，实现测试计划管理模块的各个功能，下面将详细阐述其实现过程。

依据功能设计方案配置测试计划管理模块的 URL 映射如图 4.27 所示。

```
urls = ("/plan","PlanController.ListPlan",
        "/ plan /(\d+)"," PlanController.ViewPlan",
        "/ plan /(\d+)/mod"," PlanController.ModifyPlan",
        "/ plan /(\d+)/add"," PlanController.AddPlan",
        "/ plan /(\d+)/del"," PlanController.DeletePlan",
        "/ plan /(\d+)/run"," PlanController.RunPlan",
        "/ plan /(\d+)/stop"," PlanController.StopPlan",
        "/ plan /(.+)/result"," PlanController.ListPlanResult")
```

图 4.27 测试计划管理模块 URL 映射的配置代码

由图 4.27 可知，PlanController 中的 ListPlan、ViewPlan、ModifyPlan、AddPlan、DeletePlan、RunPlan、StopPlan、ListPlanResult 分别对应了测试计划列表、查看测试

计划、修改测试计划、添加测试计划、删除测试计划、执行测试计划、停止计划任务何测试用例执行结果列表的功能类。

测试计划管理模块的增删改查功能的实现过程与测试驱动器模块类似，执行和停止测试计划任务功能的实现过程与测试用例管理模块类似，故本章节不再赘述，下面主要讲述定时执行测试计划功能的实现流程。

APScheduler 是基于 Quartz 的一个 Python 定时任务框架，提供了基于日期、固定时间间隔以及 crontab 类型的定时任务，并且可以持久化任务[40]，本平台的测试计划定时任务功能就是基于 APScheduler 实现的，其实现代码如图 4.28 所示。

```
#定时执行测试计划
def startPlanSchedulerTask():
    planTable = PlanTable()
    #查询出所有的有定时属性的测试计划
    rt, plans = planTable.selectAll({'timed': 1})
if not len(plans) == 0:
        #导入 apscheduler 库
        from apscheduler.schedulers.background import BackgroundScheduler
        #初始化后台调度器
        scheduler = BackgroundScheduler()
        #往调度器中添加作业
        for plan in plans:
            start_time = datetime.datetime.strptime(plan['crontab'],
"%Y-%m-%d %H:%M:%S")
            scheduler.add_job(runPlan, 'interval', days=1, next_run_time=start_time,
args=[plan])
        #启动调度器任务
        scheduler.start()
def runPlan(plan):
    planResultTable = PlanResultTable()
    plan_result = {}
    plan_result['plan_name'] = plan['name']
    plan_result['status'] = 'new'
    plan_result['type'] = '1'
    ret = planResultTable.insert(plan_result)
```

图 4.28 定时执行测试计划功能的核心实现代码

当在平台部署时会通过执行 startPlanSchedulerTask()方法来启动定时任务，其实现流程如图 4.29 所示。

1）首先，从数据库中查询出所有具有定时配置的测试计划。

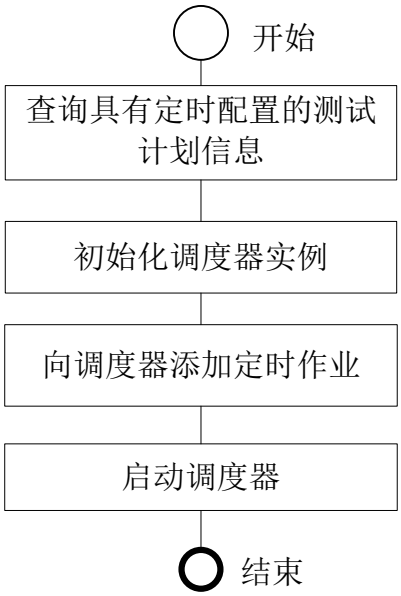


图 4.29 测试计划定时任务实现流程图

2）然后，初始化一个 BackgroundScheduler 调度器实例，BackgroundScheduler 是 APScheduler 下的一种能在后台执行定时作业的调度器，利用该调度器可以统一管理定时作业。

3）接着，按照测试计划信息向调用器中添加定时作业。其中，作业的类型是“interval”，即间隔调度作业，间隔时间是一天，这样测试计划可以每天在固定的时间执行，从而达到测试的“每日构建”。作业的执行方法是 runPlan，即当任务触发时调度器会执行 runPlan 方法来执行作业。在 runPlan 方法中会根据测试计划信息向数据库中添加一条状态为“New”的测试用例执行结果数据。与此同时，测试计划任务扫描器会扫描到该条数据并由此添加执行测试计划的多线程任务。

4）最后，启动调度器从而启动定时任务。

4.3.4 控制面板模块

根据 3.5.4 节的设计方案，实现控制面板模块的各个功能，下面将详细阐述其实现过程。

依据功能设计方案配置控制面板模块的 URL 映射如图 4.30 所示。

```
urls = ("/dashboard","DashboardController.Dashboard",
        "/ tool /conf"," DashboardController.ToolConf",
        "/ task/(.+)/start"," DashboardController. StartTask",
        "/ task/(.+)/stop"," DashboardController.StopTask")
```

图 4.30 控制面板模块 URL 映射的配置代码

由图 4.30 可知，PlanController 中的 Dashboard、ToolConf、StartTask 和 StopTask 分别对应了进入控制面板首页、测试工具配置、启动任务扫描器和停止任务扫描器功能的控制层方法类。

其中，在控制面板首页展示的内容具体包括对测试驱动器、测试用例和测试计划数量统计，所有的测试用例执行结果的状态统计、所有的测试计划执行结果的状态统计和 “每日构建计划”执行结果的状态统计等信息。

在对执行结果的状态统计信息作展示时，平台采用了 Morris.js 前端插件，Morris.js 是一个专门用于图表显示的轻量级 JavaScript 库[41]。“每日构建计划”执行结果的状态统计图表显示的实现代码如图 4.31 所示。

```
<div id="daily_plans"></div><div class="chart_name">每日计划执行结果</div>
<script>
    Morris.Donut({
      element: 'daily_plans',
      data: [
        {label: "Success", value: $statisticInfo['succDailyPlans']},
        {label: "Failed", value: $statisticInfo['errDailyPlans']},
        {label: "Running", value: $statisticInfo['rngDailyPlans']}
      ],
      colors: ['rgb(50, 200, 50)','rgb(255, 55, 55)','rgb(255, 245, 0)'],
    });
</script>
```

图 4.31 “每日构建计划”状态统计图的核心实现代码

Morris.js 插件拥有折线、面积、条形和饼状四种图表显示方式，可以在 JavaScript 中分别通过初始化 Line、Area、Bar 和 Dount 对象实例来控制图表的显示。由图 4.31 可知，“每日构建计划”执行结果的状态统计信息是以饼状图实现的。其中，element 指定了图表在 HTML 页面中的位置，即在 id 为“daily_plans”的 div 元素出；data 指定了饼状图各个饼块的数据信息，succDailyPlans、errDailyPlans、rngDailyPlans 分别对应了执行成功、执行失败和正在执行的“每日构建计划”的数量；colors 指定个了每个饼块的颜色。

此外，在控制面板首页还提供了测试工具配置、启动和停止任务扫描器功能的界面操作入口。测试工具配置功能是通过文件操作修改测试工具默认的配置文件实现的。启动任务扫描器功能是新建一个扫描器后台进程任务实现的，停止任务扫描器功能是根据线程 ID 杀死后台的扫描器进程实现的，而任务扫描器主要包括测试用例任务扫描器和测试计划任务扫描器两种。

4.4 本章小结

本章节主要讲述了平台的实现过程。首先介绍了平台的开发环境配置，然后讲述了脚本执行工具的实现过程，包括工具执行功能的实现和工具的打包与使用两个方面的内容，最后分讲述了脚本管理系统的测试驱动器管理、测试用例管理、测试计划管理和控制面板模块的实现过程。

5 平台的测试与应用

本章首先将从功能、性能、兼容性三个方面对平台进行测试，具体从测试方案和结果分析两个角度阐述，然后利用平台构建科研管理客户端的接口测试工作并讲述平台的应用过程。

5.1 测试方案

平台的测试工作主要采用黑盒测试为主白盒测试为辅的测试方案[42]，该方案首先通过黑盒测试发现问题，在发现问题后通过白盒测试定位问题的具体所在并加以修正[43]。平台的测试工作内容主要包括平台的功能测试、性能测试、兼容性测试和稳定性测试，测试环境配置如表 5.1 所示。

表 5.1 测试环境配置

服务端调试工具	PyCharm 5.0
前端调试工具	Firebug
数据库	MySQL 5.6.21
浏览器	Chrome、Firefox、IE、360
Http 服务器	Nginx 1.7
Python 版本	Python 2.7.6
web.py 版本	web.py 0.37

下面将分别讲述脚本执行工具和脚本管理系统的测试方案。

5.1.1 脚本执行工具的测试方案

脚本执行工具主要包括功能测试、性能测试和兼容性测试三个方面的测试工作，并以 Python 集成开发环境 PyCharm 作为程序的调试工具，具体测试方案说明如下。

1）功能测试

脚本执行工具的功能测试工作主要以手工操作和人为判断的方式进行，其主要测试内容包括：能够执行测试用例或测试计划并获得正确的测试结果信息；在执行测试计划时能够生成测试报告、发送结果通知；在执行过程中能够对错误的测试驱动器脚本、错误的测试用例文件、错误的测试计划文件和错误的测试工具配置文件进行异常处理；能够正确地解析测试工具的终端命令；能够被以 Python 第三方包的形式被引用。

2）性能测试

该部分的测试工作主要以 Python 脚本的方式进行，通过在脚本中批量执行不同时间长短的测试用例和测试计划并记录实际执行的时间，将实际的执行时间与理想的执行时间进行对比，根据对比结果分析异常时间的原由，进而改善脚本执行工具的程序。这些测试用例或测试计划所对应的驱动器脚本均只含有进程休眠函数，即 time.sleep()函数，由于不包含其他执行程序可以将进程的睡眠时间看作脚本的理想执行时间，因此可以通过控制休眠时间长短控制脚本执行时间长短。

3）兼容性测试

脚本执行工具的兼容性测试是指脚本执行工具能够在不同的操作系统安装使用，因此该部分的测试工作是：在 Lniux、Windows 或 Mac OS 等系统上安装并使用脚本执行工具，主要以手工操作和人为判断的方式进行。

5.1.2 脚本管理系统的测试方案

脚本管理系统主要包括功能测试、性能测试和兼容性测试三个方面的测试工作，并以 Python 集成开发环境 PyCharm 作为服务端程序的调试工具，以 Firebug 插件作为前端程序的调试工具。具体测试方案说明如下。

1）功能测试

脚本管理系统的功能测试工作主要以手工操作和人为判断的方式进行，其主要测试内容包括：能够查看、添加、修改、删除测试驱动器数据；能够查看、添加、修改、删除测试用例数据，执行测试用例，停止测试用例执行任务，查看测试用例执行结果，查看用例执行日志；能够查看、添加、修改、删除测试计划数据，执行测试计划，定时执行测试计划，停止测试计划执行任务，查看测试计划执行结果信息，查看计划执行日志，查看测试报告，发送测试结果通知；能够查看测试数据和测试结果统计信息，修改脚本执行工具配置，启动和停止用例任务扫描器，启动和停止计划任务扫描器。

2）性能测试

脚本管理系统的性能测试主要是指在浏览器页面中各个操作请求均有合理的响应时间，该部分的主要通过手工点击和 Firebug 插件查看响应时间进行。

3）兼容性测试

脚本管理系统的兼容性测试主要指脚本管理系统能够在各大主流浏览器中正常使用，该部分的测试工作主要是通过手工点击和肉眼观察的方式进行。

5.2 测试结果分析

5.2.1 脚本执行工具测试结果分析

1）功能测试

根据 5.1.1 小节中描述的功能测试方案对脚本执行工具进行功能测试，测试结果结果如表 5.2 所示。

表 5.2 脚本执行工具的功能测试结果

序号	功能	输入与动作	预期响应结果	实际响应结果	是否通过
1	执行测试用例	正确的用例文件	正常执行并返回结果	正常执行并返回结果	是
2	执行测试用例	错误的用例文件	提示并记录该异常信息异常并继续执行	提示并记录该异常信息异常并继续执行	是
3	执行测试用例	错误的驱动器脚本	提示并记录该异常信息异常并继续执行	提示并记录该异常信息异常并继续执行	是
4	执行测试用例	错误的工具配置文件	提示并记录该异常信息异常并继续执行	提示并记录该异常信息异常并继续执行	是
5	执行测试计划	正确的计划文件	正常执行并返回结果	正常执行并返回结果	是
6	执行测试计划	错误的计划文件	提示并记录该异常信息异常并继续执行	提示并记录该异常信息异常并继续执行	是
7	执行测试计划	错误的驱动器脚本	提示并记录该异常信息异常并继续执行	提示并记录该异常信息异常并继续执行	是
8	执行测试计划	错误的工具配置文件	提示并记录该异常信息异常并继续执行	提示并记录该异常信息异常并继续执行	是
9	生成测试报告	执行测试计划	成功生成测试报告	成功生成测试报告	是
10	发送测试通知	执行测试计划	收到测试计划执行结果的通知 Email	收到测试计划执行结果的通知 Email	是

通过分析表 5.2 的测试结果可以得出，脚本执行工具的各项功能运行正常。其中，

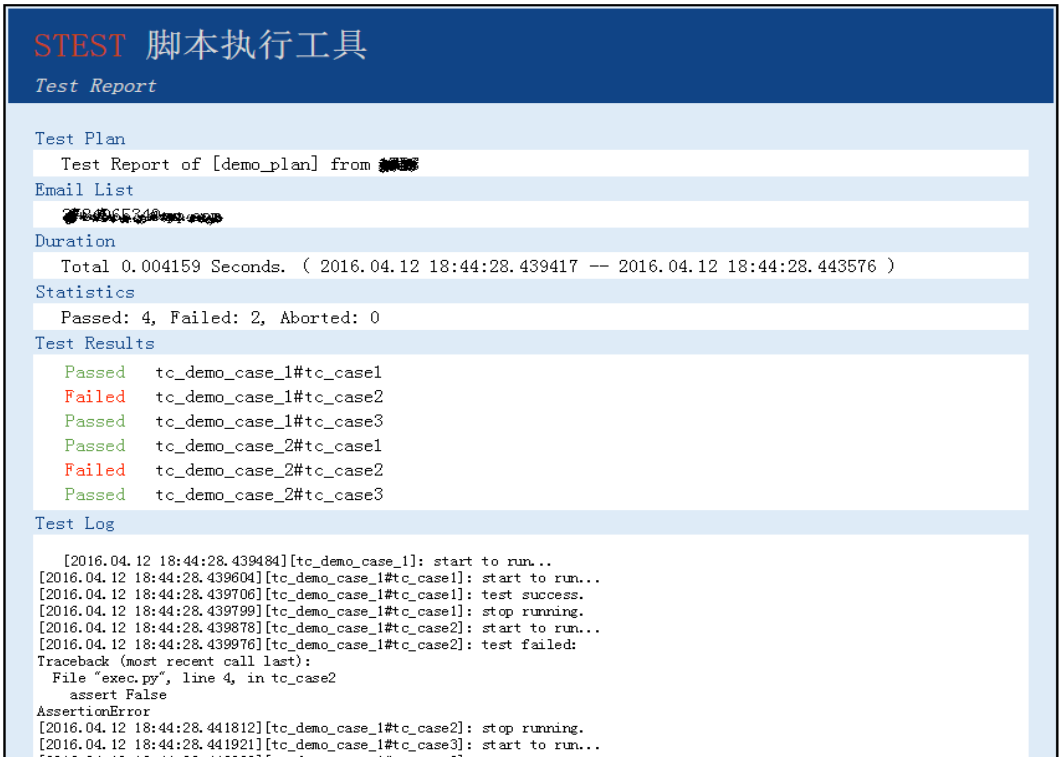
在终端下通过脚本执行工具执行测试用例或计划的终端屏幕效果如图 5.1 所示。

```
[root@webapp case]# stest -c tc_caculate_1_2
[2016.04.12 18:49:52.838110][tc_caculate_1_2]: start to run...
[2016.04.12 18:49:52.838208][tc_caculate_1_2#tc_add]: start to run...
[2016.04.12 18:49:52.838264][tc_caculate_1_2#tc_add]: test success.
[2016.04.12 18:49:52.838337][tc_caculate_1_2#tc_add]: stop running.
[2016.04.12 18:49:52.838388][tc_caculate_1_2#tc_minus]: start to run...
[2016.04.12 18:49:52.838436][tc_caculate_1_2#tc_minus]: test failed:
Traceback (most recent call last):
  File "exec.py", line 4, in tc_minus
    assert (tc.a-tc.b)==tc.minus_out
AssertionError
[2016.04.12 18:49:52.852242][tc_caculate_1_2#tc_minus]: stop running.
[2016.04.12 18:49:52.852294][tc_caculate_1_2]: stop running.

Test Build          /root/atrs/atrs/trunk/workbench/case/tc_caculate_1_2
Start Time          2016.04.12 18:49:52.838088
Stop Time           2016.04.12 18:49:52.852330
Duration            0.014242 seconds
Statistics
    All=2 Passed=1 Failed=1 Aborted=0
```

图 5.1 脚本执行工具执行测试任务的终端屏幕效果图

由图 5.1 可知，脚本执行工具能够详细地记录脚本执行的整个过程，在执行过程中能够应对各种异常状况，在脚本执行完成后能够对本次的测试任务进行总结和统计。当执行测试计划时，脚本执行工具还会生成测试报告和发送测试结果通知，通知邮件的内容如图 5.2 所示。



STEST 脚本执行工具

Test Report

Test Plan

Test Report of [demo_plan] from [redacted]

Email List

[redacted]

Duration

Total 0.004159 Seconds. (2016.04.12 18:44:28.439417 -- 2016.04.12 18:44:28.443576)

Statistics

Passed: 4, Failed: 2, Aborted: 0

Test Results

Passed	tc_demo_case_1#tc_case1
Failed	tc_demo_case_1#tc_case2
Passed	tc_demo_case_1#tc_case3
Passed	tc_demo_case_2#tc_case1
Failed	tc_demo_case_2#tc_case2
Passed	tc_demo_case_2#tc_case3

Test Log

```
[2016.04.12 18:44:28.439484][tc_demo_case_1]: start to run...
[2016.04.12 18:44:28.439604][tc_demo_case_1#tc_case1]: start to run...
[2016.04.12 18:44:28.439706][tc_demo_case_1#tc_case1]: test success.
[2016.04.12 18:44:28.439799][tc_demo_case_1#tc_case1]: stop running.
[2016.04.12 18:44:28.439878][tc_demo_case_1#tc_case2]: start to run...
[2016.04.12 18:44:28.439976][tc_demo_case_1#tc_case2]: test failed:
Traceback (most recent call last):
  File "exec.py", line 4, in tc_case2
    assert False
AssertionError
[2016.04.12 18:44:28.441812][tc_demo_case_1#tc_case2]: stop running.
[2016.04.12 18:44:28.441921][tc_demo_case_1#tc_case3]: start to run...
```

图 5.2 脚本执行工具发送的通知邮件内容

由以上的分析，可知脚本执行工具不仅功能完善，而且具有较好的健壮性。

2）性能测试

根据 5.1.1 小节中描述的性能测试方案对脚本执行工具进行性能测试，测试结果如表 5.3 所示。

表 5.3 脚本执行工具性能测试结果

序号	功能	执行对象数量	用例的时间构成（ms）			理论执行时间（ms）	实际执行时间(ms)	是否合理
			固件构建	主程序	固件销毁			
1	执行用例	1 个用例	0	100	0	100	101	是
2	执行用例	1 个用例	100	100	0	200	201	是
3	执行用例	1 个用例	100	100	100	300	302	是
4	执行用例	1 个用例	0	1000	0	1000	1002	是
5	执行用例	1 个用例	1000	1000	0	2000	2003	是
6	执行用例	1 个用例	1000	1000	1000	3000	3005	是
7	执行计划	计划(20 个用例)	0	100	0	2000	2020	是
8	执行计划	计划(20 个用例)	100	100	0	4000	4021	是
9	执行计划	计划(20 个用例)	100	100	100	6000	6030	是
10	执行计划	计划(20 个用例)	0	1000	0	20000	20035	是
11	执行计划	计划(20 个用例)	1000	1000	0	40000	40064	是
12	执行计划	计划(20 个用例)	1000	1000	1000	60000	60082	是
13	执行计划	计划(50 个用例)	0	100	0	5000	5051	是
14	执行计划	计划(50 个用例)	100	100	0	10000	10060	是
15	执行计划	计划(50 个用例)	100	100	100	15000	15078	是
16	执行计划	计划(50 个用例)	0	1000	0	50000	500092	是
17	执行计划	计划(50 个用例)	1000	1000	0	100000	100134	是
18	执行计划	计划(50 个用例)	1000	1000	1000	150000	150218	是

由表 5.3 的统计结果可知，当测试用例数量较小时，测试任务的实际执行时间与理论执行时间极为接近；当随着测试用例数量增大时，测试任务的实际执行时间与理论执行时间的间隔会随测试用例数量等比例变大；当用例的理论执行时间增大时，实际与理论执行时间比无限接近于 1。根据上面一系列的分析可知，测试工具在执行测试用例或计划时不会造成太多的额外时间开销，具有良好的时间性能。

3）兼容性测试

根据 5.1.1 小节中描述的兼容性测试方案对脚本执行工具进行测试，测试结果如表 5.4 所示。

表 5.4 脚本执行工具兼容性测试结果

序号	操作系统类型	版本	安装结果	使用结果	是否兼容
1	Windows	XP	正常	正常	是
2	Windows	Win 7	正常	正常	是
3	Windows	Win 8	正常	正常	是
4	Linux	CentOS 6.5	正常	正常	是
5	Linux	Ubuntu 14.04	正常	正常	是
6	Mac OS	10.9 Mavericks	正常	正常	是
7	Mac OS	10.10 Yosemite	正常	正常	是

由表 5.4 可知，脚本执行工具可以兼容各种常见的操作系统。

5.2.2 脚本管理系统测试结果分析

1）功能测试

根据 5.1.2 小节中描述的功能测试方案分别对脚本管理系统进行功能测试。

①测试驱动器管理模块

测试驱动器管理模块的功能测试结果如表 5.5 所示。

表 5.5 测试驱动器管理模块的功能测试结果

序号	功能	输入与动作	预期响应结果	实际响应结果	响应时间(ms)	是否通过
1	查看驱动器列表信息	点击“驱动器”导航菜单	进入驱动器列表页	成功进入驱动器列表页	74	是
2	查看驱动器详情	在列表页点击驱动器名称	进入驱动器详情页	进入驱动器详情页	112	是
3	添加驱动器	点击”添加”按钮，填写信息并提交	添加成功并转入详情页	添加成功并转入详情页	192	是
4	修改驱动器	点击”修改”按钮，修改信	修改成功并转入详情页	修改成功并转入详情页	243	是

		息并提交				
5	删除驱动器	点击”删除”按钮	确认删除后转入列表页	删除成功并转入列表页	185	

其中，测试驱动器列表页面的实现效果如图 5.3 所示。

图 5.3 测试驱动器列表页面

在测试驱动器列表页面，点击”添加”按钮进入测试驱动器添加页面，测试驱动器的添加页面如图所示。

图 5.4 测试驱动器添加页面

在测试驱动器添加页面，点击“+”按钮可以添加驱动参数，点击“x”按钮可以删除驱动参数。在填写好测试驱动器信息后，点击”提交”按钮可以添加一条测试驱动器数据。

在测试驱动器列表页面，点击驱动器名称进入测试驱动器详情页面，其实现效果如图 5.5 所示。

图 5.5 测试驱动器详情页面

在测试驱动器详情页面，点击”修改”按钮可以进入测试驱动器修改页面并对测试驱动器信息进行修改，点击”删除”按钮可以删除该条测试驱动器数据。

②测试用例管理模块

测试用例管理模块的功能测试结果如表 5.6 所示。

表 5.6 测试用例管理模块的功能测试结果

序号	功能	输入与动作	预期响应结果	实际响应结果	响应时间(ms)	是否通过
1	查看测试用例列表信息	点击“用例”导航菜单	进入测试用例列表页	成功进入用例列表页	91	是
2	查看测试用例详情	在列表页点击用例名称	进入测试用例详情页	进入测试用例详情页	177	是
3	添加测试用例	点击”添加”按钮，填写相关信息并提交	添加成功并转入详情页	添加成功并转入详情页	185	是

4	修改测试用例	点击”修改”按钮，修改信息并提交	修改成功并转入详情页	修改成功并转入详情页	231	是
5	删除测试用例	点击”删除”按钮	确认删除后转入列表页	删除成功并转入列表页	168	是
6	查看测试用例执行结果	进入用例详情页	详情页下部有执行结果的列表信息	详情页下部有执行结果的列表信息	/	是
7	查看用例执行日志	点击日志查看标签	新标签弹出日志信息页面	新标签弹出日志信息页面	/	是
8	执行测试用例	点击”执行”按钮	执行成功并有一条测试结果	执行成功并有一条测试结果	103	是
9	停止用例执行任务	点击”停止”按钮	用例停止执行且测试结果状态更新为中止	用例停止执行且测试结果状态更新为中止	159	是

其中，测试用例列表页面如图 5.6 所示。

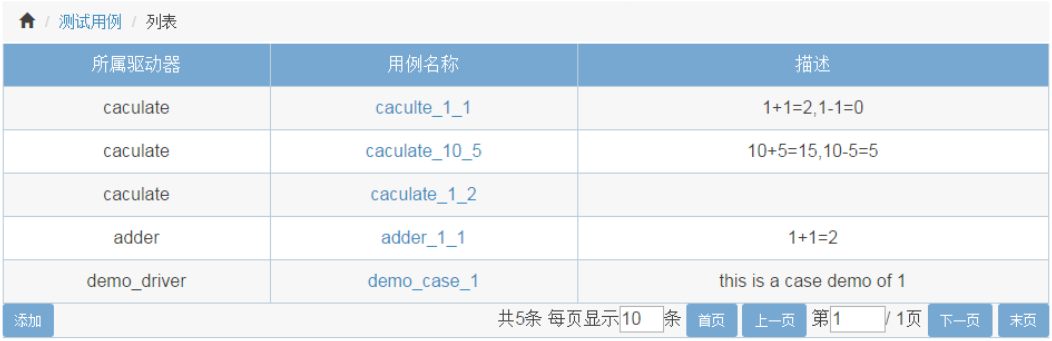


图 5.6 测试用例列表页面

在测试用例列表页面，点击”添加”按钮进入测试用例添加页面，测试用例的添加页面如图 5.7 所示。

在测试用例添加页面，切换所属驱动器时测试用例的驱动参数也随着变化。在填写好所有测试用例信息后，点击”提交”按钮可以添加一条测试用例数据。

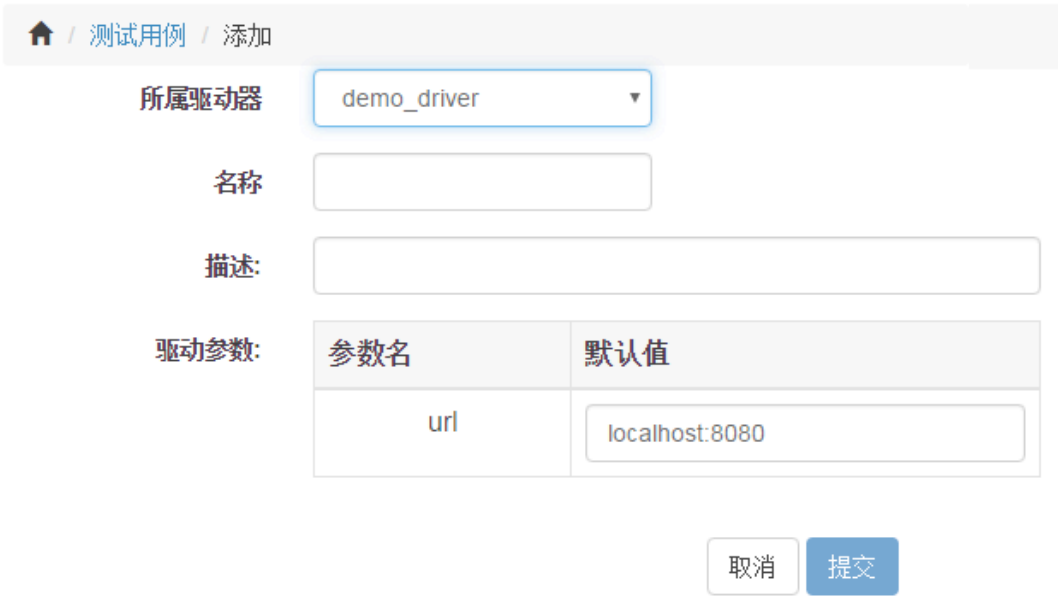


图 5.7 测试用例添加页面

在测试用例列表页面，点击测试用例名称进入测试用例详情页面，如图 5.8 所示。

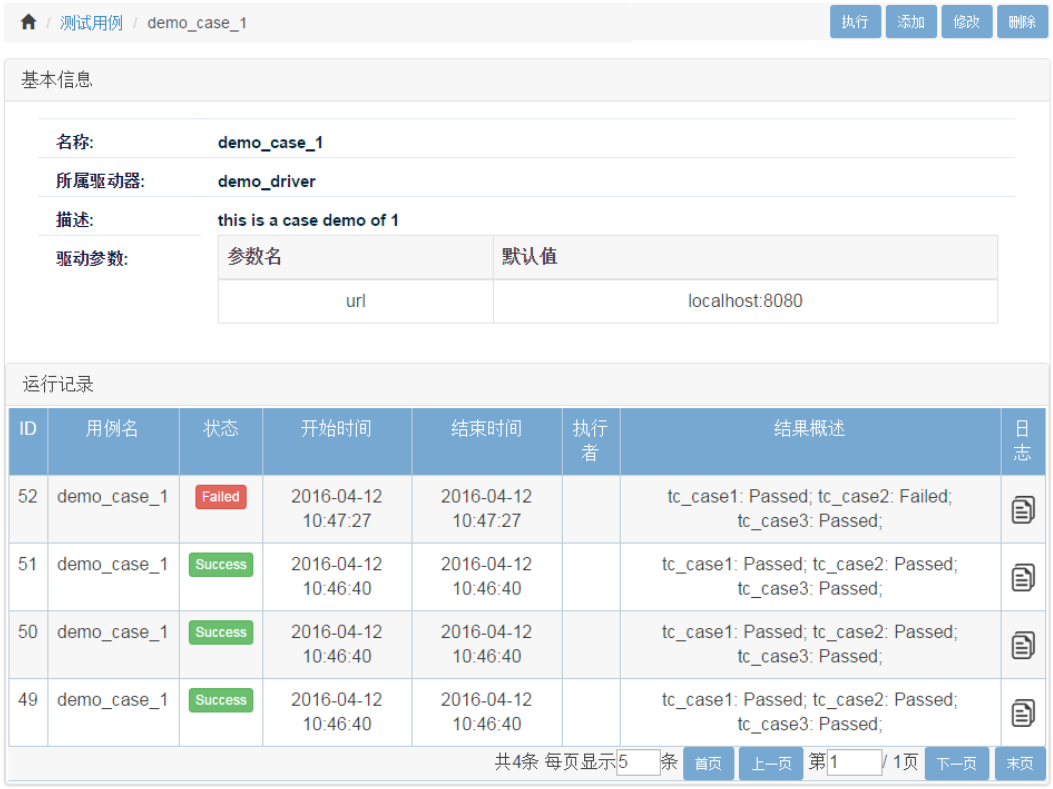


图 5.8 测试用例详情页面

由图 5.8 可知，测试用例详情页面的上半部是测试用例的基本信息，下半部是该测试用例的执行结果记录列表信息。在该页面，点击”执行”按钮可以执行该测试

用例，点击”修改”按钮可以进入测试用例修改页面并对测试用例信息进行修改，点击”删除”按钮可以删除该条测试用例数据，点击日志图标可以查看测试用例的执行日志。

③测试计划管理模块

测试计划管理模块的功能测试结果如表 5.7 所示。

表 5.7 测试计划管理模块功能测试结果

序号	功能	输入与动作	预期响应结果	实际响应结果	响应时间(ms)	是否通过
1	查看测试计划列表信息	点击“计划”导航菜单	进入测试计划列表页	成功进入计划列表页	90	是
2	查看测试计划详情	在列表页点击计划名称	进入测试计划详情页	进入测试计划详情页	156	是
3	添加测试计划	点击”添加”按钮，填写相关信息并提交	添加成功并转入详情页	添加成功并转入详情页	175	是
4	修改测试计划	点击”修改”按钮，修改相关信息并提交	修改成功并转入详情页	修改成功并转入详情页	214	是
5	删除测试计划	点击”删除”按钮	确认删除后转入列表页	删除成功并转入列表页	143	是
6	查看测试计划执行结果	进入计划详情页	详情页下部有执行结果的列表信息	详情页下部有执行结果的列表信息	/	是
7	查看计划执行日志	点击日志查看标签	新标签弹出日志信息页面	新标签弹出日志信息页面	/	是
8	查看测试报告	点击报告查看标签	新标签弹测试报告页面	新标签弹测试报告页面	50	是
9	执行测试计划	点击”执行”按钮	执行成功，增加一条结果数据，生成测试报告，收到邮件通知	执行成功，增加一条结果数据，生成测试报告，收到邮件通知	97	是
10	定时执行测试计划	添加计划的定时器参数	到定时时间后会执行计划	到定时时间后会执行计划	/	是
11	停止计划执行	点击”停止”	用例停止执行	用例停止执行	124	是

	任务	按钮	且测试结果状态更新为中止	且测试结果状态更新为中止		

其中，测试计划列表页面如图 5.9 所示。

图 5.9 测试计划列表页面

在测试计划列表页面，点击”添加”按钮进入测试计划添加页面，测试计划的添加页面如图 5.10 所示。

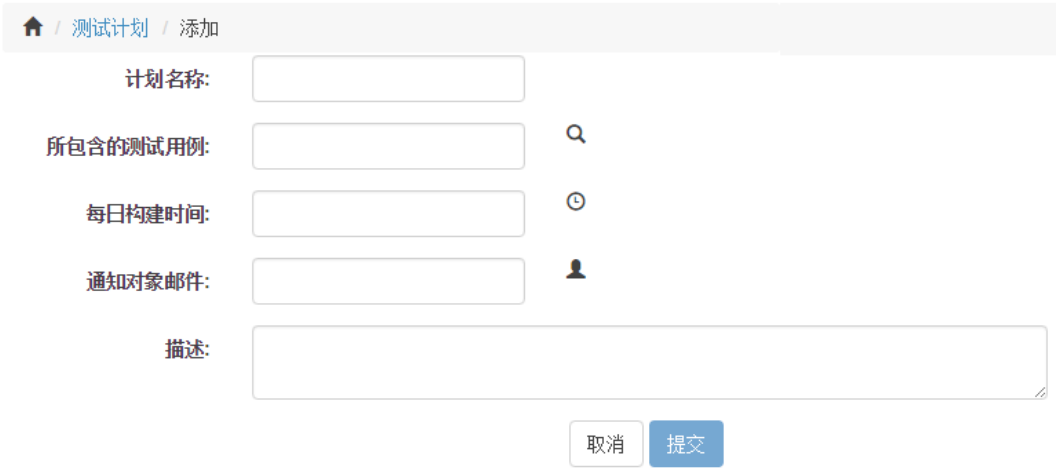


图 5.10 测试计划添加页面

在测试计划添加页面，点击“搜索”图标可以弹出层选择测试用例，点击“时间”图标可以弹出层选择时间点，点击“用户”图标可以弹出层选择用户邮件。在填写好所有测试计划信息后，点击”提交”按钮可以添加一条测试计划数据。

在测试计划列表页面，点击测试计划名称进入测试计划详情页面，如图 5.11 所示。

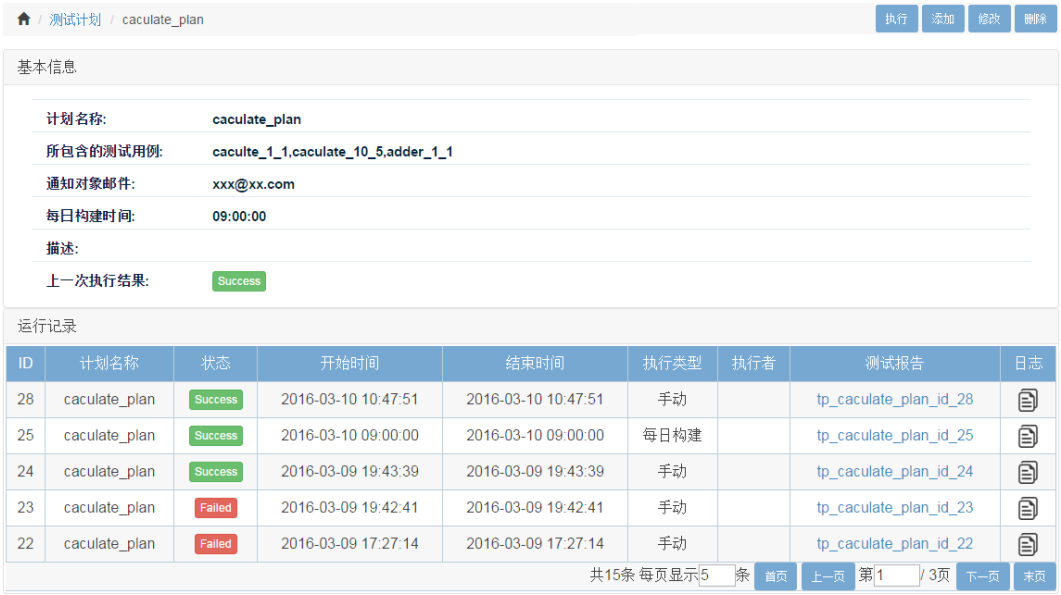


图 5.11 测试计划详情页面

由图 5.11 可知，测试计划详情页面的上半部是测试计划的基本信息，下半部是该测试计划的执行结果记录列表信息。在该页面，点击”执行”按钮可以执行该测试计划，点击”修改”按钮可以进入测试计划修改页面并对测试计划信息进行修改，点击”删除”按钮可以删除该条测试计划数据，点击测试报告名称可以查看该条执行结果的测试报告页面，测试报告页面与图 5.2 相同，点击日志图标可以查看测试计划的执行日志。

④控制面板模块

控制面板模块的功能测试结果如表 5.8 所示。

表 5.8 测试计划管理模块功能测试结果

序号	功能	输入与动作	预期响应结果	实际响应结果	响应时间(ms)	是否通过
1	查看测试数据统计信息	点击“首页”，进入控制面板	查看到测试数据统计信息	查看到测试数据统计信息		是
2	查看测试结果统计信息	点击“首页”，进入控制面板	查看到测试结果统计饼状图	查看到测试结果统计饼状图		是
3	配置测试工具参数	在控制面板页点开脚本执行配置标签，点击“铅笔”图表进行修改	成功修改测试工具的配置参数	成功修改测试工具的配置参数		是

4	启动用例任务扫描器	在控制面板页点击用例任务扫描器的”启动”按钮	成功启动用例任务扫描器	成功启动用例任务扫描器		是
5	启动计划任务扫描器	在控制面板页点击计划任务扫描器的”启动”按钮	成功启动计划任务扫描器	成功启动计划任务扫描器		是
6	停止用例任务扫描器	在控制面板页点击用例任务扫描器的”停止”按钮	成功停止用例任务扫描器	成功停止用例任务扫描器		是
7	启动计划任务扫描器	在控制面板页点击计划任务扫描器的”启动”按钮	成功启动计划任务扫描器	成功停止计划任务扫描器		是

其中，控制面板的信息展示界面如图 5.12 所示。

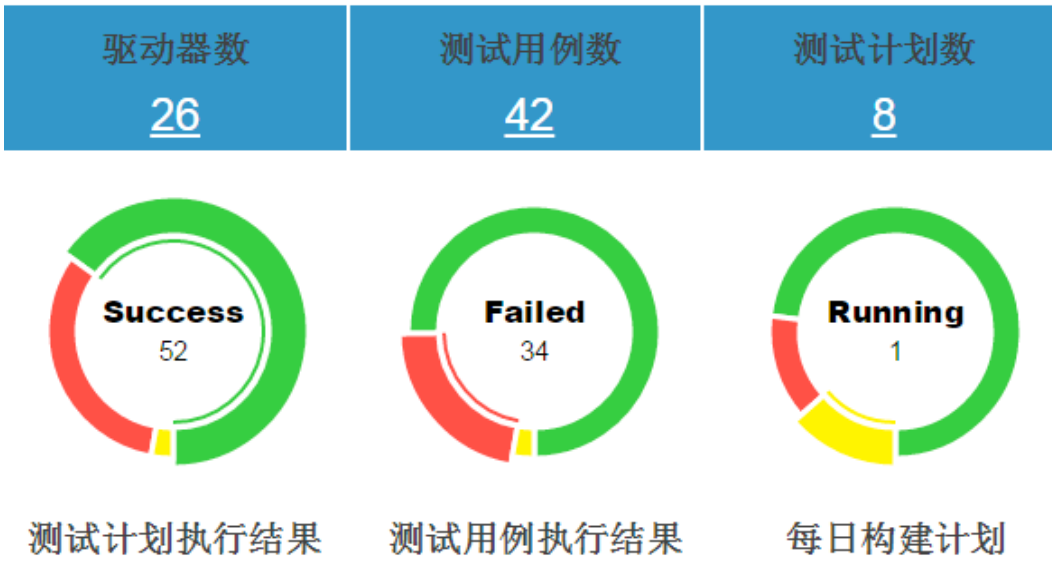


图 5.12 测试数据和测试结果统计信息的展示界面

脚本执行工具配置和任务扫描器的管理界面如图 5.13 所示。

图 5.13 测试工具配置和任务扫描器的管理界面

总结以上的测试结果可知，脚本管理系统的所有功能均能正常使用。

2）性能测试

对脚本管理系统功能测试结果表的响应时间进行统计和分析，可知各个功能的响应时间均在合理范围内，保证了用户在业务操作时有良好的用户体验。

3）兼容性测试

根据脚本管理系统的功能测试流程，分别在多个浏览器下进行页面操作，并记录出现 bug 的功能，其结果记录如表 5.9 所示。

表 5.9 脚本管理系统的兼容性测试结果

序号	浏览器类型	出现异常或错误的功能或页面				是否兼容
		驱动器管理模块	测试用例管理模块	测试计划管理模块	控制面板模块	
1	Chrome	无	无	无	无	是
2	Firefox	无	无	无	无	是
3	IE	无	无	无	无	是
4	360	无	无	无	无	是

由表 5.9 可知，脚本管理系统能够在各种主流浏览器中正常使用，具有良好的兼容性。

5.3 应用实例

前面的章节已经完成了平台的全部实现。平台能够服务于多个科研管理软件的

测试工作，它们的作用关系如图 5.14 所示。

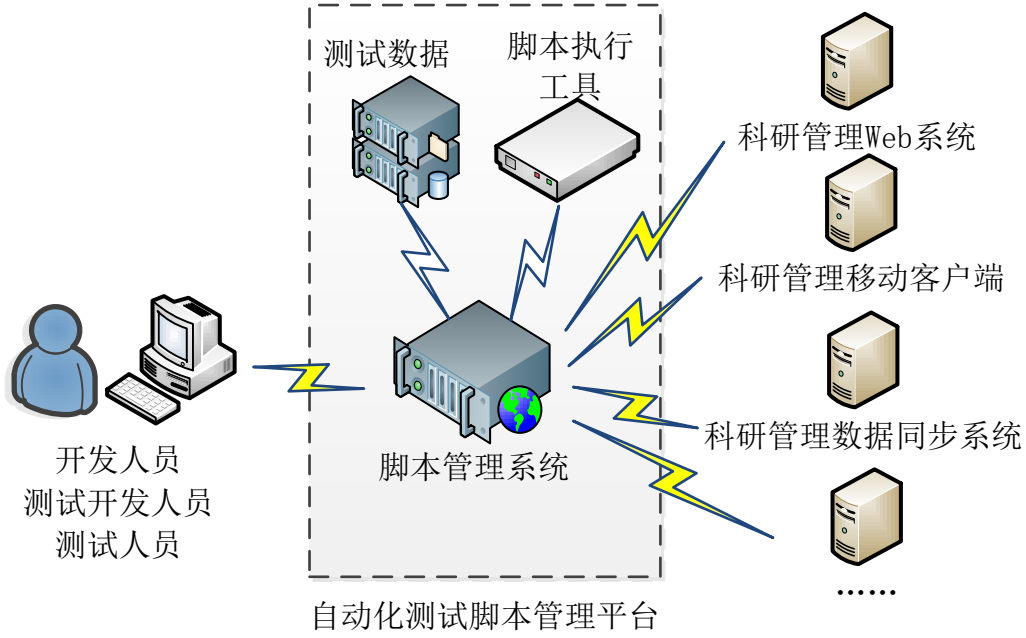


图 5.14 应用平台测试科研管理软件的作用关系示意图

下面主要以科研管理系统移动端接口的测试工作和科研管理数据同步系统的部分测试工作为例具体讲述平台的应用过程。

5.3.1 科研管理系统移动端

科研管理系统移动客户端是基于科研管理 Web 系统并结合移动设备特性设计并实现的，它旨在方便科研管理人员在移动端访问科研管理数据库和做科研管理日常工作。本次测试内容是对科研管理移动客户端的 Http 接口做功能测试，利用脚本管理平台建立起迭代测试和回归测试。

首先根据移动端的测试需求和测试特点进行驱动器脚本开发，其设计思路如图 5.15 所示，思路说明如下：

1）首先在测试固件构建脚本中进行用户登入和访问功能接口操作。

2）然后在测试主程序脚本中进行功能验证。

3）最后在测试固件销毁脚本中进行用户退出操作。

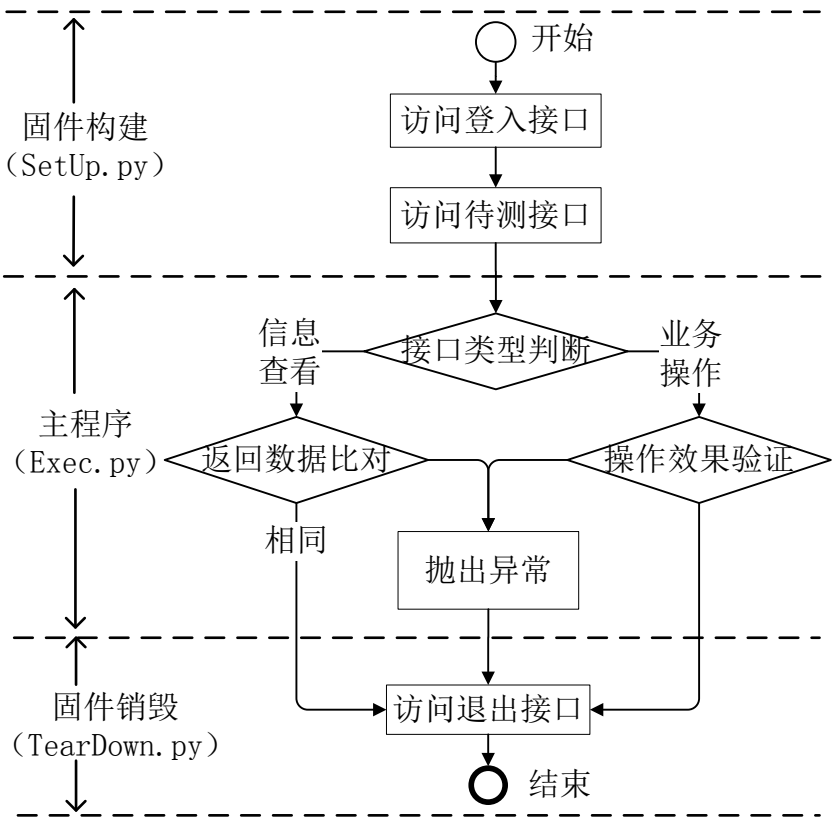


图 5.15 移动端接口测试的驱动器脚本设计思路

在驱动器脚本开发完成后便在脚本管理系统中添加相应的驱动器数据，根据驱动器信息配置驱动数据从而添加测试用例数据，分类组合测试用例配置定时器和邮件信息从而添加测试计划数据。

最后便可以脚本管理系统中直接执行该部分测试相关的测试计划，其测试结果如表 5.10 所示。

表 5.10 科研管理移动客户端的 Http 接口的功能测试结果

测试计划名称	模块名称	接口数量	测试用例数量	测试结果
系统功能计划（xxx_system_plan）	个人信息	7	10	成功：10，失败：0，中止：0
	系统设置	5	13	成功：13，失败：0，中止：0
	新闻通知管理	12	12	成功：12，失败：0，中止：0
业务功能计划（xxx_ business _plan）	社科人员管理	6	6	成功：6，失败：0，中止：0
	社科机构管理	6	6	成功：6，失败：0，中止：0
	社科项目管理	11	16	成功：16，失败：0，中止：0
	社科成果管理	10	14	成功：14，失败：0，中止：0

	社科奖励管理	10	14	成功：14，失败：0，中止：0

由以上内容可知，相关人员利用平台很方便地为科研管理系统移动端软件构建了测试工作。平台能够在保证移动客户端接口功能正常的同时，提高科研管理移动应用的质量。

5.3.2 科研管理数据同步系统

科研管理系统数据同步系统旨即时更新或同步多个科研机构间的科研管理项目数据。本次测试内容是数据同步系统的数据交换接口对同时做功能和性能测试，利用脚本管理平台建立起迭代测试和回归测试。

首先根据数据数据同步系统的测试需求和测试特点进行驱动器脚本开发，其设计思路如图 5.16 所示。思路说明如下：

1）首先在测试固件构建脚本中进行准备同步数据源和访问数据交换接口操作。

2）然后在测试主程序脚本中进行功能验证和性能判断。

3）最后在测试固件销毁脚本中进行恢复同步数据源操作。

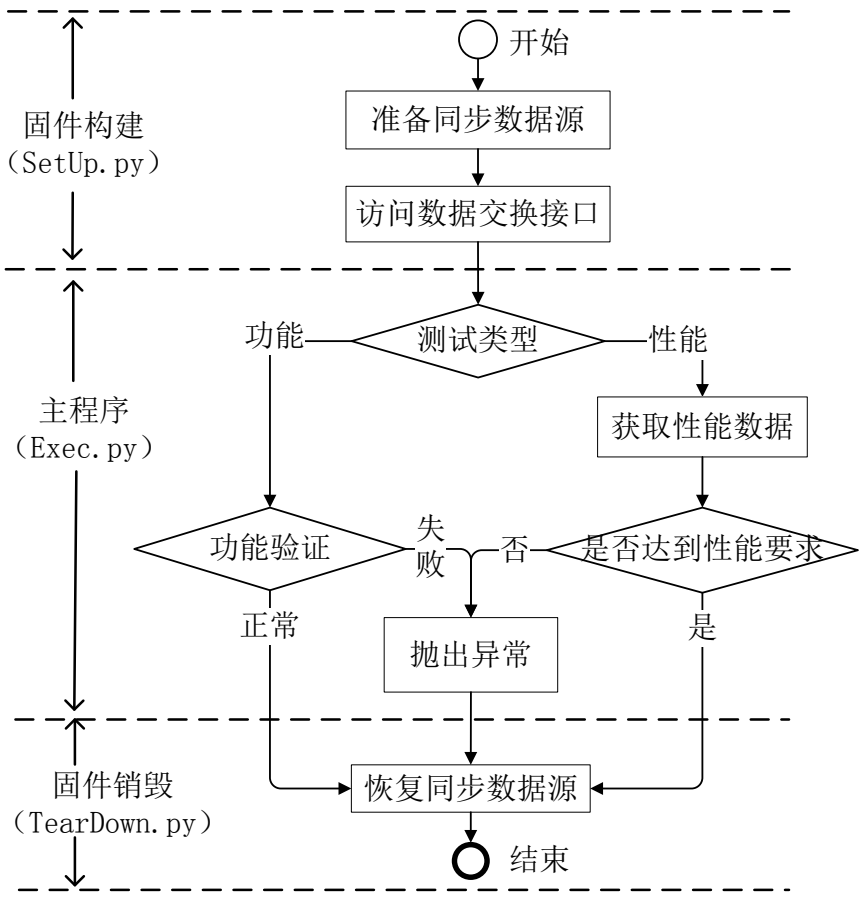


图 5.16 移动端接口测试的驱动器脚本设计思路

接下来的测试工作步骤与科研管理系移动端的测试工作步骤类似，这里不再赘述，执行该部分的测试计划得到测试结果如表 5.11 所示。

表 5.11 科研管理数据同步系统的功能和性能测试结果

测试计划名称	模块名称	接口数量	测试用例数量	测试结果
数据同步计划（data_sync_plan）	项目数据	4	8	成功：8，失败：0，中止：0
	人员数据	1	2	成功：2，失败：0，中止：0
	机构数据	1	2	成功：2，失败：0，中止：0
	成果数据	1	2	成功：2，失败：0，中止：0
	奖励数据	1	2	成功：2，失败：0，中止：0

由以上内容可知，相关人员利用平台很方便地为科研管理数据同步系统构建了测试工作。平台能够在保证科研管理数据同步系统功能和性能正常的同时，提高科研管理系统的产品质量。

5.4 本章小结

本章节对平台的测试与应用工作做了详细介绍。首先从整体介绍了平台的测试内容、测试工具和测试环境，然后分别介绍了脚本执行工具和脚本管理系统的测试方案，接着依据测试方案开展测试工作并对测试结果进行详细的分析，最后以科研管理软件的部分测试工作为例讲述平台的应用过程。从以上的内容可知，平台不仅有完善的功能、良好的性能、兼容性和用户体验，还能快速有效地应用于科研管理软件的测试工作。

6 总结与展望

6.1 论文总结

本文针对科研管理项目的测试工作现状，提出科研管理项目的脚本管理平台的需求，设计并实现了自动化测试脚本管理平台，其主要工作内容和结论总结如下：

1）从脚本和数据分离的角度出发，提出“驱动器-用例-计划”的测试模型，并基于该测试模型设计出一种基于数据驱动的自动化测试方案。在该方案下的自动化测试流程中，测试逻辑和测试数据分离，因此测试脚本具有可配置性和高复用性；可以按照测试类型或者业务类型将测试用例归类为测试计划，有效地对测试脚本和数据行进行统一组织管理；定时执行测试计划功能满足测试的迭代测试需求；邮件通知功能使得测试工作能够被即时反馈。

2）结合单元测试组织概念和自动化单元测试框架原理，实现了平台的脚本执行工具。该工具既可以以终端命令的方式也可以以 Python 第三方库的方式调用；该工具能够在各种常见操作系统中安装使用，具有良好的兼容性；该工具的源码程序结构划分清晰，接口具有高可扩展性。

3）基于 Web 开发技术实现测试驱动器脚本、测试用例数据和测试结果数据的在线管理。该脚本管理系统采用 B/S 模式，以 MySQL 作为数据源，以 web.py 作为 Web 服务端的开发框架，以 jQuery +Ajax+JSON+TrimPath 作为 Web 前端的开发框架。该脚本管理系统结合脚本执行工具实现了各种类型、各种业务下 Python 单元测试脚本的自动构建、自动运行、结果跟踪和分类管理功能，改善了软件测试的技术方法和流程管理方式，提高了测试效率，缩短了科研管理项目的开发周期，保证了科研管理产品的质量。

6.2 论文展望

到目前为止，本文的研究工作已告一段落，平台也达到了初期既定的目标，但是由于研究时间和相关条件的限制，平台仍存在一些需要扩展和改进的地方 ，具体如下：

1）在驱动器脚本方面，可以为测试驱动器脚本的查看或编辑功能增加 Python 语法的高亮和校验，语法高亮可以使脚本内容显示更加明了清晰，语法校验可以避免提交错误的脚本；可以提供驱动器脚本的在线调试功能，这个功能使得测试人员在对脚本作简单或少量修改时不需要特意去本地调试脚本；可以提供脚本的版本控

制功能，当驱动器脚本修改后，原有测试用例的驱动参数可能发生变化而不可用，可以加入版本控制功能保证测试用例与驱动器间的驱动参数的一致性。

2）在测试结果数据方面，平台现阶段只是对测试结果做了简单的统计，随着越来越多的软件项目在平台上构建测试工作且测试结果数据积累的越来越多，可以对这些数据作进一步的统计与分析甚至是数据挖掘，进而发现数据中潜在信息，如从成功率变化与分布总结出产品的质量、稳定性和开发效率；可以利用图表统计对测试结果进行统计，丰富测试报告的内容。

3）此外，在脚本执行工具方面，可以逐步建立起测试库。测试库是指在因频繁使用而专门封装的用于驱动器脚本的 Python 库文件，该测试库可以随着脚本执行工具安装并使用。随着测试库的不断丰富，测试开发人员在撰写驱动器脚本时可以专心于测试业务本身，测试构建工作会变得越来越简单，驱动器脚本的代码内容也会变得越来简单明了，建立完善的测试库具有重大意义。

致 谢

在论文的末尾，我要感谢在我撰写论文期间为我的学习与生活提供帮助和支持的人。

衷心感谢我的导师王玉明副教授。在论文撰写期间，无论是论文的开题、研究方案的确定还是论文的收稿，王老师都为我作了悉心的指导。借此机会再次向王老师表示感谢。

感谢我的实习同事张弛和李景成。在写论文的过程中，我经常与他们进行技术探讨，他们为我的论文提供很多重要建议。

感谢陪伴我读研生涯的实验室同学们。他们和我一起学习，一起生活，一起玩耍，和他们相处的日子是我一生难忘的经历。我们即将各奔东西，由衷希望大家在以后的日子里一帆风顺。

感谢我的好友们。他们经常关心和支持我，尤其是在我遇阻时给了我莫大的鼓励，希望我们以后能经常聚聚。

最后，我要特别感谢我的父母。我的父母为我创造出很好的成长环境，对我做的任何事作无条件支持，对我无时无刻地关心和爱护，没有他们就没有现在的我。

参考文献

[1] Crispin L, Gregory J. 敏捷软件测试: 测试人员与敏捷团队的实践指南. 孙伟峰,

[2] 肖丰佳, 李立新. 软件测试技术研究. 工业控制计算机, 2012, 01: 75-76

[3] 梁家安. 自动化软件测试技术研究: [硕士学位论文]. 无锡: 江南大学, 2011.

[4] 王碧. 公司测试工程师绩效考核研究: [硕士学位论文]. 北京: 中国地质大学, 2014.

[5] Whittaker J A, Arbon J, Carollo J. How Google tests software. New Jersey: Addison-Wesley, 2012. 1-14

[6] 曹文婷. 软件测试用例生成及管理系统的设计和实现: [硕士学位论文]. 吉林: 吉林大学, 2012.

[7] Anand S, Burke E K, Chen T Y, et al. An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 2013: 1978-2001

[8] 张秋杰, 张晓莹. 基于 Python 的单元测试框架. 中国科技论文在线, 2009: 1-7

[9] 李洁. 软件测试用例设计. 电脑编程技巧与维护, 2010, 04: 17-19+23

[10] 杨玚. 深入理解 gtest:C/C++单元测试经验谈. 信息安全与技术, 2011, 01: 91-96

[11] Beck K. Simple smalltalk testing: With patterns. The Smalltalk Report, 1994, 4: 16-18

[12] Massol V, Husted T. Junit in action. Shelter Island: Manning Publications, 2003. 1-25

[13] Turnquist G L. Python Testing Cookbook. Packt Publishing Ltd, 2011. 5-39

[14] Madden B. Using cppunit to implement unit testing. Game Programming Gems, 2006, 6: 1-2

[15] 李晓会. Web 系统自动化功能测试框架研究与实践: [硕士学位论文]. 北京: 北京邮电大学, 2011.

[16] 李玮. 软件自动化测试混合框架的研究与实现: [硕士学位论文]. 北京: 北京交通大学, 2007.

[17] 张秋杰. 基于 PyUnit 框架的企业级软件自动化测试技术的研究: [硕士学位论文]. 北京: 北京邮电大学, 2010.

[18] 张玲. Web 应用自动化测试框架的研究和应用: [硕士学位论文]. 广州: 华东理工大学, 2014.

[19] 刘慕涛, 张磊, 王艳等. 基于 XML 的 API 自动化测试工具设计与实现. 计算机工程, 2007, 13: 96-98
[20] 凌永发, 张云生, 郭秀萍. 软件测试自动化中的脚本技术. 云南民族学院学报(自然科学版), 2002, 01: 544-548
[21] 王莉, 殷锋, 李奇. 软件自动化测试脚本设计研究. 西南民族大学学报(自然科学版), 2006, 02: 357-360
[22] Fielding R T. Architectural styles and the design of network-based software architectures. University of California, 2000.
[23] Leff A, Rayfield J T. Web-application development using the model/view/controller design pattern. In: Enterprise Distributed Object Computing Conference, 2001: 118-127
[24] 高昂, 魏惠茹, 李晓东等. MVC 设计模式研究. 电脑知识与技术, 2016, 1: 88-89
[25] Rhodes B, Goerzen J. Web Applications. Foundations of Python Network Programming, 2010: 179-196.
[26] Zakas N C. Professional javascript for web developers. New Jersey: John Wiley & Sons, 2009. 562-595
[27] Bibeault B, Kats Y. jQuery in Action. New Delhi: Dreamtech Press, 2008. 103-120
[28] T. Holdener. Ajax: The Definitive Guide, O'Reilly Media Publishing House, 2008
[29] Crockford, D. JSON: The fat-free alternative to XML. In: Proc. of XML 2006, Boston, 2006. 12-18
[30] 董文彬. 浅论软件需求分析. 科技视界, 2012, 26: 236-237.
[31] 李云云. 浅析 B/S 和 C/S 体系结构. 科学之友, 2011, 01: 6-8
[32] Fielding R, Gettys J, Mogul J, et al. Hypertext transfer protocol--HTTP/1.1. 1999
[33] Forta B. MySQL crash course. India: Pearson Education, 2006. 56-78
[34] 沈华. 邮件处理系统的设计与实现: [硕士学位论文]. 成都: 电子科技大学, 2013.
[35] Forta B, 刘晓霞, 钟鸣. MySQL 必知必会[M]. 人民邮电出版社, 2009.
[36] 王威. MySQL 数据库源代码分析及存储引擎的设计: [硕士学位论文]. 南京: 南京邮电大学, 2012.
[37] Ziadé T. Expert Python Programmin. Packt Publishing Ltd, 2008. 25-55
[38] Barry P. Head First Python. O'Reilly Media, 2010. 588-618
[39] Langtangen H P. Advanced Python. Springer Berlin Heidelberg, 2004. 299-429
[40] Alex Grönholm. User guide — APScheduler 3.1.0.dev1 documentation.

http://apscheduler.readthedocs.org/en/latest/userguide.html, 2016-4-11
[41] Olly Smith. Pretty time-series line graphs. http://morrisjs.github.io/morris.js/, 2013
[42] 向润. 黑盒测试方法探讨. 软件导刊, 2009, 01: 33-35
[43] 胡静. 浅析黑盒测试与白盒测试. 衡水学院学报, 2008, 01: 30-32