南 阳 理 工 学 院

本科生毕业设计(论文)

学院(系)： 软件学院

专 业： 网络工程

学 生： 刘宇祥

指导教师： 惠征

完成日期 2020 年 04 月

南阳理工学院本科生毕业设计（论文）

基于Python的自动化测试系统的设计与实现

Design and Implementation of the  
Automated test System based on python

总 计：毕业设计(论文) 5页

表 格： 1个

图 片： 1个

南 阳 理 工 学 院 本 科 毕 业 设 计(论文)

基于Python的自动化测试系统的设计与实现

Design and Implementation of the  
Automated test System based on python

学 院(系)： 软件学院

专 业： 网络工程

学 生 姓 名： 刘宇祥

学 号： 1615935078

指导教师(职称)： 惠征 副教授

评 阅 教 师： 惠征

完 成 日 期： 2020年04月06日

南阳理工学院

Nanyang Institute of Technology

基于Python的自动化测试系统的设计与实现

网络工程 刘宇祥

1. 自动化测试指的是无需过多人工干预，从而实现测试的目的，减少人为误差。目前互联网行业的测试能力远远跟不上开发的进度和技术深度，往往是开发工程师完成开发后，测试工程师才开始挨个测试，效率低下。针对这样的问题，我们需要开发一种高效的便捷的测试方式。本系统后端使用Python的Flask Web框架，Python有利于提高开发速度和提高代码可读性。前端使用Vue和element-UI实现，大大降低了前端开发的难度，降低开发时间成本。数据库使用MySQL数据库，开源免费的，降低开发成本，简单好用。本系统不但能实现自动化测试，还在可视化方面提高用户交互。通过简单的操作就能实现复杂的测试。本系统支持批量执行测试，大大方便了大型后端系统的测试流程，提高研发效率。本系统有着操作简单、方便快捷的优点，代码量小，容易进行第三方开发特点。
2. MySql;Python;自动化测试

Design and Implementation of the  
Automated test System based on python

Network Engineering Major Liu yu xiang

1. Automated testing means that there is no need for too much manual intervention to achieve the purpose of testing and reduce human error. At present, the testing capability of the Internet industry can not keep up with the progress and technical depth of development. Often, after the development engineer completes the development, the test engineer starts testing one by one, which is inefficient. In response to such problems, we need to develop an efficient and convenient testing method. The back end of this system uses Python's Flask Web framework. Python is conducive to improving development speed and improving code readability. The front end is implemented using Vue and element-UI, which greatly reduces the difficulty of front-end development and reduces the cost of development time. The database uses MySQL database, which is open source and free, which reduces development costs and is easy to use. This system not only enables automated testing, but also improves user interaction in terms of visualization. Complex operations can be achieved with simple operations. This system supports batch execution testing, which greatly facilitates the testing process of large back-end systems and improves research and development efficiency. This system has the advantages of simple operation, convenience and quickness, small amount of code, and easy third-party development.
2. MySql;Python; [automated](javascript:;) [testing](javascript:;)

目 录

[1 绪论 1](#_Toc37837188)

[1.1 背景 1](#_Toc37837189)

[1.2 国内外研究现状分析 2](#_Toc37837190)

[1.3 实验内容 2](#_Toc37837191)

[1.4 实验目标 3](#_Toc37837192)

[1.5 论文整体结构 3](#_Toc37837193)

[2 相关工具及技术介绍 3](#_Toc37837194)

[2.1 开发环境 3](#_Toc37837195)

[2.2 编程语言 4](#_Toc37837196)

[2.2.1 Python 4](#_Toc37837197)

[2.2.2 JavaScript 4](#_Toc37837198)

[2.3 数据库存储 4](#_Toc37837199)

[2.4 前端展示技术 4](#_Toc37837200)

[2.5 Web技术 5](#_Toc37837201)

[3 系统分析 5](#_Toc37837202)

[3.1 需求分析 5](#_Toc37837203)

[3.2 功能分析 6](#_Toc37837204)

[3.3 可行性分析 6](#_Toc37837205)

[3.3.1 经济可行性 6](#_Toc37837206)

[3.3.2 技术可行性 6](#_Toc37837207)

[3.3.3 运行可行性 7](#_Toc37837208)

[4 系统设计 7](#_Toc37837209)

[4.1 业务流程设计 7](#_Toc37837210)

[4.1.1 用户登录方案 7](#_Toc37837211)

[4.1.2 测试服务方案 8](#_Toc37837212)

[4.1.3 测试用例方案 9](#_Toc37837213)

[4.1.4 测试用例执行方案 10](#_Toc37837214)

[4.2 系统整体设计 11](#_Toc37837215)

[4.3 数据库结构设计 12](#_Toc37837216)

[5 系统实现 16](#_Toc37837217)

[5.1 登录模块 16](#_Toc37837218)

[5.2 测试服务模块 18](#_Toc37837219)

[5.3 测试用例模块 21](#_Toc37837220)

[5.4 测试结果模块 23](#_Toc37837221)

[5.5 测试执行模块 24](#_Toc37837222)

[6 系统测试 26](#_Toc37837223)

[结束语 27](#_Toc37837224)

[参考文献 28](#_Toc37837225)

[致谢 30](#_Toc37837226)

# 绪论

随着互联网行业发展，公司系统Web API越来越多。一个小的销售系统都有近100个API，每次版本迭代测试都是严峻的挑战，需要专门的测试团队进行测试。人力测试的时间成本和人力成本很高，而且人为存在失误，导致用户体验差的情况，投诉率居高不下。

本系统很好的解决了这个问题，系统只需要在每次系统API开发完成后，录入测试用例，即可一键批量测试API。完美解决时间成本和人力成本的问题，避免人为操作失误的问题。

## 背景

API测试用于确定API是否针对各种可行请求返回正确的响应（以预期的格式），对诸如失败和意外等极端输入之类的极端情况做出正确反应，在可接受的时间内交付响应，以及安全地应对潜在的安全攻击[1][4]。服务虚拟化与API测试结合使用，以隔离被测服务，并通过模拟无法进行测试的API /服务来扩展测试环境访问[6]。

API测试通常包括测试REST API或SOAP Web服务以及通过HTTP，HTTPS，JMS和MQ发送的JSON或XML 消息有效负载[2][7]。它还可以包括消息格式，例如SWIFT，FIX，EDI和类似的固定长度格式，CSV，ISO 8583和通过传输协议（例如TCP / IP，ISO 8583）发送的 协议缓冲区，MQTT，FIX，RMI，SMTP，TIBCO集合点和FIX[8][9]。

API测试通常涉及以下实践：

1、单元测试：测试单个操作的功能。

2、功能测试：测试更广泛场景的功能，通常使用单元测试作为端到端测试的构建块。包括测试用例定义，执行，验证和回归测试。

3、负载测试：常通过重用功能测试用例来验证负载下的功能和性能。

4、运行时错误检测：监视应用程序的自动或手动测试的执行，以暴露诸如争用条件，异常和资源泄漏之类的问题。

5、安全测试：包括渗透测试和模糊测试以及验证身份验证，加密和访问控制。

6、Web UI测试：作为端到端集成测试的一部分执行，该测试还涵盖API，使团队可以在较大事务的上下文中验证GUI项目。

7、互操作性测试：（SOAP only）检查是否符合Web服务互操作性概要文件。

8、WS-\*符合性测试：（SOAP only）检查是否符合WS- \*标准，例如WS-Addressing，WS-Discovery，WS-Federation，WS-Policy，WS-Security和WS-Trust。

9、渗透测试：测试计算机系统，网络或Web应用程序，以发现攻击者可以利用的漏洞。

10、模糊测试：强制将大量纯随机数据（有时称为"noise"或"fuzz"）输入系统，以尝试强制崩溃，溢出或其他负面行为。这样做是为了在绝对极限下测试API，并且在某种程度上充当"worst case scenario"。

## 国内外研究现状分析

近年来Web服务领域发生着翻天覆地的变化，继传统的XML-RPC风格的Web Service之后，一种新的风格REST被应用于Web Service。

REST是Representational State Transfer(表述性状态转移)的缩写，它最初是Roy Thomas Fielding 于2000 年在他的博士论文中提出的，是针对分布式系统的软件架构风格。

目前很多公司采用传统的网络应用软件测试技术对REST架构软件进行测试，这种测试技术主要针对网络软件的User Interface层，由测试人员根据经验编写测试用例，测试方式以手动测试为主。通常情况下这种测试方式并不能够保证完全覆盖到底层的所有接口，不易发现软件深层次的逻辑问题，在发现问题时不易进行准确定位，繁琐和重复的手动测试也会降低网络应用软件的开发效率，导致网络应用软件的测试完备性和测试充分性不高，软件的性能得不到很好的保证。

API测试被认为比GUI测试更适合于测试自动化和连续测试（尤其是与敏捷软件开发和DevOps结合使用的自动化）[3][4]。引用的原因包括：

系统复杂度： GUI测试无法充分验证与多层体系结构关联的功能路径和后端API /服务。API被认为是与被测系统最稳定的接口。

具有快速反馈循环的短发布周期：敏捷和DevOps团队在进行短迭代和快速反馈循环后发现GUI测试需要大量的返工才能跟上频繁变化的步伐。API层的测试不那么脆弱，更易于维护。

由于这些原因，建议团队在降低对GUI测试的依赖的同时提高其API测试的级别。建议将API测试用于绝大多数的测试自动化工作和尽可能多的边缘测试。然后保留GUI测试，以在系统级别，移动测试和可用性测试中验证典型用例[3][4][10]。

总体来说，需要一套高效的软件测试框架对REST架构软件进行自动化测试，提高软件生命周期中的测试环节的效率，避免低级BUG的产生，提高API可靠性和稳定性。减少软件后期的经济支出，节约软件后期维护成本。

## 实验内容

通过Python编写自动化测试系统，用于Web API的功能性测试，帮助项目迭代时进行的大量接口测试。使用VUE和element-UI建立前端组件，用于更好的可视化体验。

针对不同的Web API进行测试，可查询测试结果，测试结果在前端展示。

## 实验目标

根据用户提供的测试用例，进行批量测试，记录测试结果。用户可查测试结果，可用测试定时任务，定期测试指定API。

每次测试记录用户信息和测试结果，用于报表使用。测试结果可以导出，便于项目汇报。

## 论文整体结构

第一章，绪论。本章节介绍了自动化API测试的研究背景及其意义，并且进一步提出本文课题实验目标。

第二章，本章节描述了本次系统设计与开发需要用到的技术和工具。如开发语言和开发环境。

第三章，系统分析，分析了现阶段国内外学者们在自动化API测试方法上的研究现状，以及现阶段依然存在的问题，探索系统实现方式，保证系统的根本逻辑是行得通的。

第四章，系统设计，最后设计测试用例，对系统的核心模块进行设计。

第五章，系统实现，前端实现和后端实现合并在此章节。前端实现截图和后端实现代码。

第六章，系统测试，本次系统设计的结论，系统优缺点描述，改进方向。还有自动化测试目前发展空间描述。

# 相关工具及技术介绍

## 开发环境

开发过程中，搭建开发环境是该系统开发的起步过程，同时标志着该实验开始的标志，是非常重要的一步，这一部分重点讲解如何配置开发使用的环境。

本次操作系统

本次系统开发过程使用的开发环境如表2-1所示：

表2-1 开发环境表

| 开发环境 | 操作系统：Microsoft windows 10 enterprise 64Bit |
| --- | --- |
| 编程语言：Python for Windows 3.6.4 64Bit |
| 开发IDE： PyCharm 2019.1 professional edition 64Bit |
| 三方库： flask、pymysql、json、yaml、time、logging、requests、ast、flask\_cors、flask\_apscheduler、datetime、VUE、element-UI |

## 编程语言

### Python

Python是一种解释，高层次的，通用的 编程语言。Python由Guido van Rossum创建并于1991年首次发布，Python的设计理念通过显着使用大量空白来强调代码的可读性。它的语言构造和面向对象的方法旨在帮助程序员为大型和大型项目编写清晰的逻辑代码[11]。

Python是动态类型化并被垃圾回收的。它支持多种编程范例，包括过程，面向对象和 函数式编程。由于Python具有完善的标准库，因此通常被称为“batteries included”的语言[13]。

### JavaScript

JavaScript通常缩写为JS，是一种编程语言符合的ECMAScript的规范。

JavaScript是高级的，通常是即时编译的，并且是多范例的。它具有花括号语法，动态类型，基于原型 的面向对象和一流的功能。

除了HTML和CSS，JavaScript是万维网的核心技术之一。JavaScript支持交互式网页，并且是Web应用程序的重要组成部分。绝大多数网站将其用于客户端页面行为，并且所有主要的网络浏览器都有专用的JavaScript引擎来执行它[15]。

## 数据库存储

数据库是一个有组织的集合的数据，通常存储和从计算机系统电子地访问。在数据库更复杂的地方，通常使用正式的设计和建模技术来开发它们[17]。

该数据库管理系统（DBMS）是软件与交互的最终用户，应用程序和数据库本身来捕获和分析数据。DBMS软件还包含为管理数据库而提供的核心功能。数据库，DBMS和关联的应用程序的总和可以称为“数据库系统”。通常，术语“数据库”还用于宽松地指代任何DBMS，数据库系统或与数据库关联的应用程序[18]。

MySQL是开源 关系数据库管理系统（RDBMS）。根据GNU通用公共许可证的条款，MySQL是免费的开源软件，还可以根据各种专有许可证使用。MySQL由瑞典MySQL AB拥有和赞助，该公司由Sun Microsystems（现为Oracle Corporation）收购[18]。

2010年，当Oracle收购Sun时，Widenius 派遣了开源 MySQL项目来创建MariaDB。

## 前端展示技术

Vue是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

## Web技术

万维网（WWW），俗称的网络，是一个信息系统，其中的文件和其它网络资源被确定统一资源定位符，这可能是通过超文本相互链接，并且可以通过Internet访问。WWW的资源是通过超文本传输​​协议（HTTP）进行传输的，并且用户可以通过称为Web浏览器的软件应用程序对其进行访问。并由称为Web服务器的软件应用程序发布[19]。

英国科学家Tim Berners-Lee于1989年发明了万维网。他于1990年在瑞士日内瓦附近的欧洲核子研究组织（CERN）工作时编写了第一个网络浏览器。该浏览器于1991年在CERN外部发布，从1991年1月开始首先发布给其他研究机构，然后在1991年8月发布给公众。万维网一直是信息时代发展的中心，并且是数十亿人在互联网上进行交互的主要工具[20]。

Web资源可以是任何类型的下载媒体，但是网页是已经用超文本标记语言（HTML）格式化的超文本媒体。这种格式允许包含URL的嵌入式超链接，并允许用户导航到其他Web资源。除文本外，网页还可以包含对图像，视频，音频和软件组件的引用，这些信息在用户的 Web浏览器中显示为多媒体内容的连贯页面[21]。

一个具有共同主题，共同域名或同时具有两者的多个Web资源组成一个网站。网站存储在运行名为Web服务器的程序的计算机中，该程序响应用户计算机上运行的Web浏览器通过Internet发出的请求。网站内容可以主要由发布者提供，或在用户贡献内容或内容取决于用户或其行为的情况下以交互方式提供。出于各种信息，娱乐，商业，政府或非政府原因，可能会提供网站[21]。

# 系统分析

本章节为测试系统设计章节，第一部分为可行性分析，第二部分为需求分析，第三部分为功能分析。主要描述自动化测试系统的结构分析。

## 需求分析

自动化测试意味着不需要太多的人为干预即可达到测试目的，并减少人为错误。目前，互联网行业的测试能力还远远不能跟上发展进度和技术深度。 在开发工程师完成开发之后，测试工程师通常会开始一个接一个地进行测试，这效率很低。针对此类问题，我们需要开发一种高效便捷的测试方法。

## 功能分析

本系统需要一个登录功能，登录进入的是后台，需要做一个后台管理系统。逻辑层是用Python实现的接口，进行逻辑判断。

登录功能

需要输入后台管理系统的账号密码进行登录，验证通过后进入后台。防止非法人员使用该系统。

测试服务功能

测试服务功能用于剥离测试用例的URL资源，使其IP和端口为独立的。便于测试生产环境和测试环境时不需要重复添加测试用例，只需要使用不同的测试服务即可。简化测试流程，降低依赖。

测试用例功能

测试用例功能记录需要测试的接口的API URL，与其测试服务绑定。需要录入测试需要的数据包和返回的数据包，测试传入的数据包用于保证接口的正常访问。测试返回的数据包用于验证接口是否正常或者数据是否正常。

执行功能

执行功能依赖于测试服务和测试用例，将测试服务和测试用例组合在一起，进行请求。将其结果保存入测试结果记录表里。

## 可行性分析

### 经济可行性

开发本系统需要购买云服务器作为载体，对服务器资源没有要求，购买最低配即可。本系统实现为纯软件实现，无需购买硬件，对服务器资源没有强依赖，只需要极小的资源即可运行。使用过程中也无多余费用的产生，开发所有使用的技术和工具都是免费的，没有版权，都是开源软件。而且本系统实现简单，开发周期短，人员经济支出基本没有，产生的效益远远大于专人软件测试所需的经济支出。所以从经济方面谈开发此系统是可行的。

### 技术可行性

本系统的实现主要使用Python进行开发，还使用一些Python的第三方支持库来实现一些功能。Python有着免费和容易上手的特点，用于自动化方面有着先天优势。数据库使用MySQL数据库，特点也很明显，稳定和免费。综上所述，本系统在技术方面是可行的。

### 运行可行性

本系统是自动化执行，输入相对应的参数和指定结果即可。没有复杂的操作，操作简单容易上手。

# 系统设计

## 业务流程设计

本次测试方案采用前后端配合方式进行测试，前端实现测试后台UI界面，后端对测试用例进行配置和执行测试。

本次设计了四个核心模块：测试服务模块、测试用例模块、测试结果模块、测试用例执行模块。一个非核心模块：用户登录模块。

五个模块组成一套完整的测试方案，对后端接口进行测试，自动化测试、批量测试，测试结果查看。

### 用户登录方案

用户首先登录，进行登录验证成功后进入首页，当用户准备进入测试系统时需要登录，如果账号或密码错误，会提示信息错误并重置账号密码输入框的内容，如果输入数据正确，会提示信息正确并进入系统主页面。

用户登录流程图如下图4-1所示：

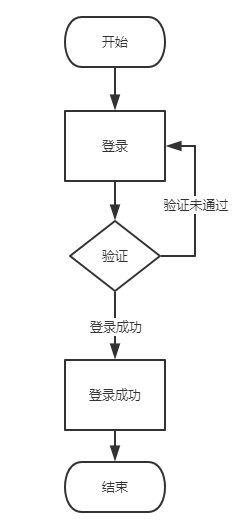


图4-1 用户登录流程

### 测试服务方案

用户登录后，进入测试服务页面。测试服务页面功能有新增测试服务、查看测试服务、修改测试服务、导出测试服务。

测试服务是需要测试的web后端系统，以IP+端口为单位，每一个测试服务都是一个web服务测试。方案流程图如下图4-2所示：

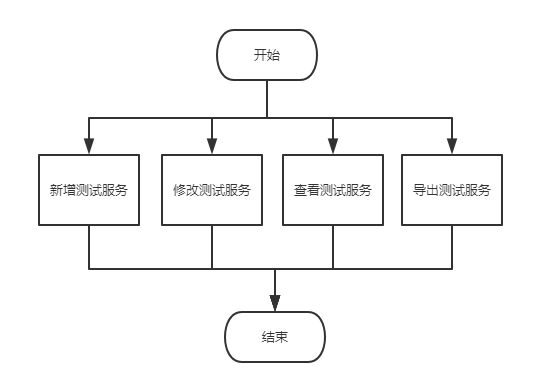


图4-2 测试服务流程

### 测试用例方案

测试用例方案设计，新增测试用例首先需要指定一个测试服务。输入需要测试的API和指定请求的内容。

测试用例新增流程图如下图4-3所示：

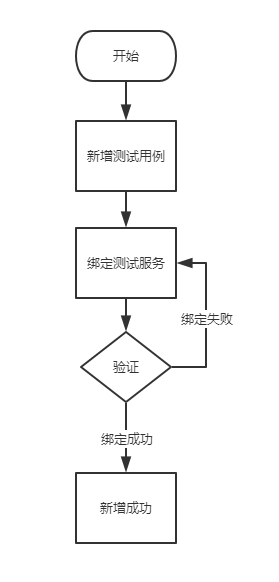


图4-3 新增测试用例

### 测试用例执行方案

为了达到自动化测试目的，本系统将测试执行方案设计为两种执行模式。第一种模式是单次执行，只执行指定测试用例。第二种是批量执行，可以执行多个测试用例。

测试用例执行流程图如下图4-4所示：

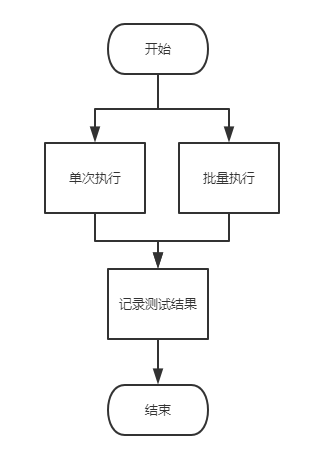


图4-4 执行测试用例流程

## 系统整体设计

后台测试人员先登录，登录后进入主页，有四个核心功能模块：测试服务、测试用例、测试结果、执行测试，架构如下图4-5所示：

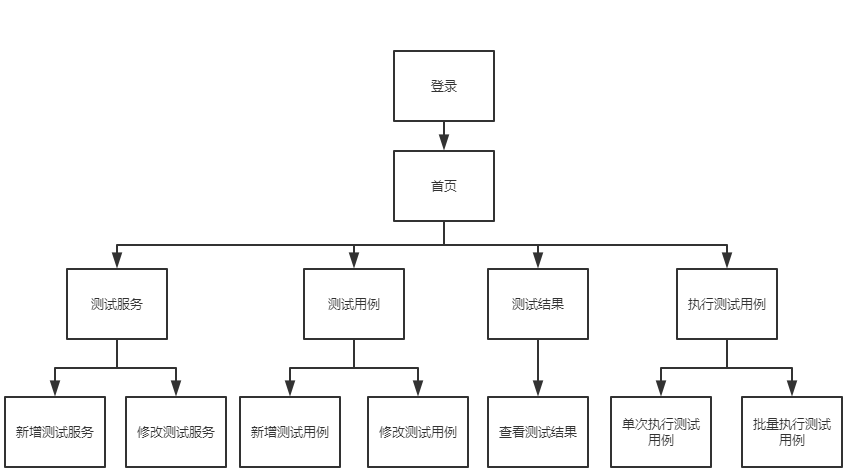


图4-5 测试系统结构

测试服务模块功能有新增测试服务、修改测试服务。测试用例模块功能有新增测试用例、修改测试用例。测试结果模块功能主要是查看测试结果。执行测试用例模块功能有单次执行和批量执行。

## 数据库结构设计

数据库也就是本系统的应用数据和用户数据的存放库，通过对存放数据的增删改查，能够实现本系统中所有对数据的要求。数据库操作中的建表是比较重要的一步，而如何建表则是在建表前所需要考虑的。

本数据库设计主要有以下几类实体：

（1）用户实体，本系统目前的用户群里只有一种，就是测试后台操作人员，测试后台操作人员主要包括用户名密码和用户标识符实体属性，其中标识符为主键，用来鉴别用户登录和用户身份，用户名是用户本系统所展示的，用来在页面上看到用户的登录情况。用户实体属性图如下图4-6所示：

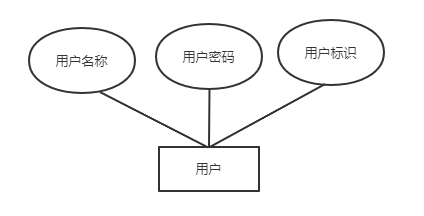


图4-6 用户实体属性图

（2）测试服务实体，测试服务实体是记录测试服务功能的实体，用于区分测试用例所属的服务。主要包括测试服务标识符、测试服务IP，测试服务端口、测试服务说明、测试服务状态。测试服务实体属性图如下图4-7所示：

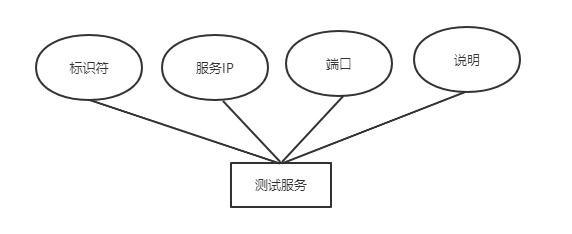


图4-7 测试服务实体属性图

（3）测试用例实体，测试用例是对某个测试服务的具体实现，测试用例指定测试服务和测接口。测试用例实体主要有测试用例标识、URL也就是API、请求数据、返回数据。测试用例实体属性图如下图4-8所示：

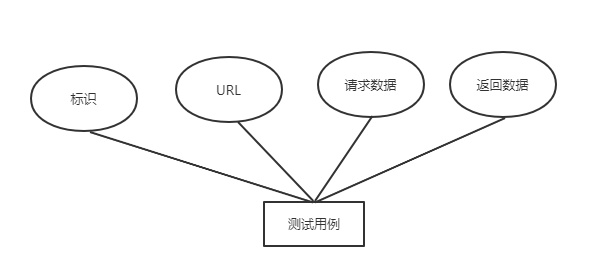


图4-8 测试用例实体属性图

（4）测试结果实体，测试结果是测试用例执行后的数据记录，反应测试的结果。测试结果实体有：测试单号每次测试都有一个唯一的单号、测试用户名、测试用例标识、请求数据、返回数据、返回状态。测试结果实体属性图如下图4-9所示：

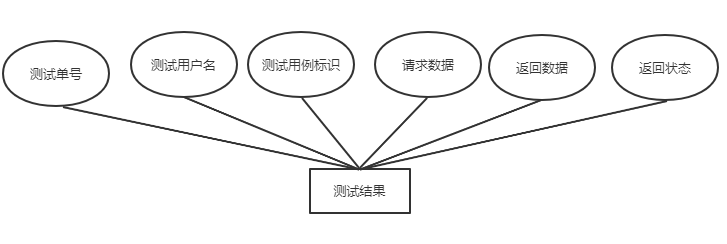


图4-9 测试结果实体属性图

本系统的ER图如下图4-10所示：

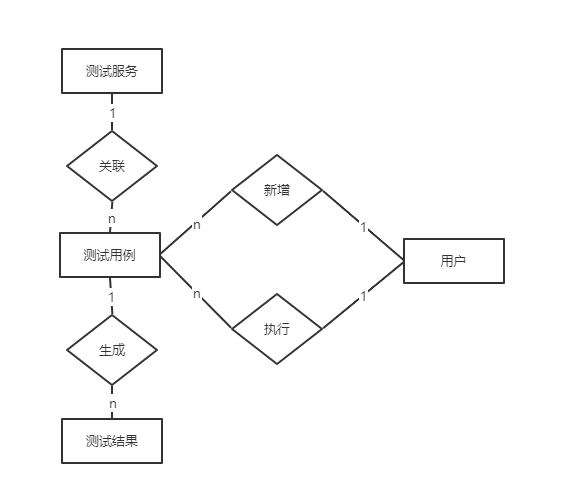


图4-10 系统ER图

本数据库设计主要有以下数据表：

（1）用户数据表，保存用户信息，包括用户名、用户密码、用户账号，如下表4-1所示：

表4-1用户信息表

| 字段名称 | 字段描述 | 数据类型 | 长度 | 允许为空 |
| --- | --- | --- | --- | --- |
| id | 用户ID | int | 10 | 否 |
| name | 用户名 | varchar | 255 | 否 |
| account | 账号 | varchar | 255 | 否 |
| password | 密码 | varchar | 255 | 否 |
| create\_at | 创建时间 | datetime |  | 否 |
| update\_at | 修改时间 | datetime |  | 否 |

（2）测试服务数据表，保存测试服务数据，包括服务主机、端口、说明、状态，如下表4-2所示：

表4-2 测试服务信息表

| 字段名称 | 字段描述 | 数据类型 | 长度 | 允许为空 |
| --- | --- | --- | --- | --- |
| id | 服务ID | int | 10 | 否 |
| host | 服务IP | varchar | 255 | 否 |
| port | 服务端口 | varchar | 255 | 否 |
| explain | 服务说明 | varchar | 255 | 否 |
| state | 服务状态 | varchar | 255 | 否 |

（3）测试用例数据表，保存测试用例数据，包括服务主机ID、接口API、请求数据、返回数据，如下表4-3所示：

表4-3 测试用例信息表

| 字段名称 | 字段描述 | 数据类型 | 长度 | 允许为空 |
| --- | --- | --- | --- | --- |
| id | 测试用例ID | int | 10 | 否 |
| host\_id | 服务IP | int | 10 | 否 |
| requests\_data | 请求数据 | varchar | 255 | 否 |
| result | 返回数据 | varchar | 255 | 否 |
| create\_at | 创建时间 | datetime |  | 否 |
| update\_at | 修改时间 | datetime |  |  |

（4）测试结果数据表，保存测试结果数据，包括测试单号、测试用户名、测试用例ID、测试请求数据、测试返回数据、测试返回码，如下表4-4所示：

表4-4 测试就结果信息表

| 字段名称 | 字段描述 | 数据类型 | 长度 | 允许为空 |
| --- | --- | --- | --- | --- |
| id | 测试就诶过ID | int | 10 | 否 |
| test\_code | 测试单号 | varchar | 20 | 否 |
| test\_username | 测试用户名 | varchar | 100 | 否 |
| testcase\_id | 测试用例ID | int | 100 | 否 |
| testcase\_result | 测试返回数据 | text | 100 | 否 |
| testcase\_recode | 测试返回码 | varchar | 10 | 否 |
| testcase\_count | 测试次数 | int | 10 | 是 |
| create\_at | 创建时间 | datetime |  |  |
| update\_at | 修改时间 | datetime |  |  |

# 系统实现

## 登录模块

输入网址后，跳转到用户登录界面，如下图5-1所示：



图5-1 用户登录



图5-2 登录提示

登录后提示登录成功，即为登录成功，如图4-2所示。

登录成功后跳转到首页，如下图5-3所示：

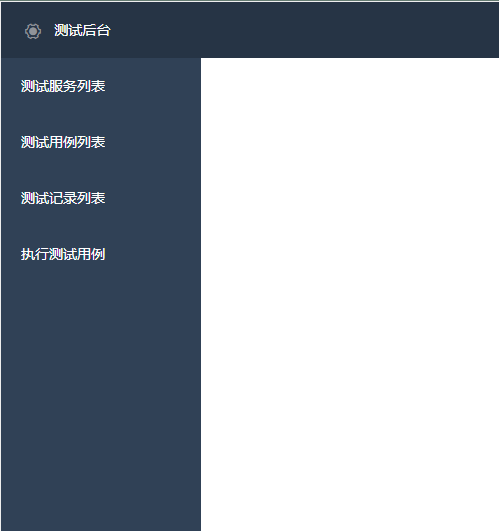


图5-3 测试首页

登录后端API核心代码如代码5-1所示：

代码5-1 登录API核心代码

sql = """

SELECT

id,`name`,account

FROM

`user`

WHERE

account = '{0}'

AND `password` = '{1}'

""".format (request.form['account'], request.form['password'])

\_rdata = mysql.fetchall\_db (sql)

\_logger.info (\_rdata)

if (len(\_rdata) <= 0 ):

return returnData ("404", "失败", None)

登录前端核心代码如下代码5-2所示：

代码5-2 登录前端核心代码

submitForm(formName) {

this.$refs[formName].validate((valid) => {

if (valid) {

let \_self=this;

let formData = new FormData();

formData.append('account', \_self.loginForm.username);

formData.append('password',\_self.loginForm.password);

this.axios.post(\_self.ApiUrlData+'/login',formData).then(response=>{

if(response.data.errcode=='0'){

\_self.loading=true;

\_self.$router.push('/')

\_self.$store.commit('SAVE\_USERINFO',"test")

\_self.$message({type: 'success',message: '登录成功'

});

} else{\_self.$message({type: 'error',message: '用户名密码错误'});}

})

} else {console.log('error submit!!');

return false;}

});

}

## 测试服务模块

测试服务模块，前端展示如下图5-4所示：



图4-4 测试服务模块前端展示

测试服务模块新增功能前端展示，如下图5-5所示：



图5-5 新增测试服务功能

修改测试服务点击对应的测试服务的操作列里面的修改，如图5-6所示：



图5-6 修改测试服务

导出测试服务数据列表功能，点击导出，即可将所有测试服务导出成Excel表格文件。如下图5-7所示：

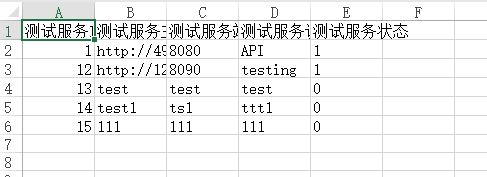


图5-7 导出功能

测试服务API实现，查询测试服务代码如下代码5-3所示：

代码5-3 查询测试服务

@app.route ("/testing/showAllHostService", methods=["POST"])

def showAllHostService():

try:

sql = """

SELECT

\*

FROM

t\_host

-- WHERE

-- state = 1

"""

\_rdata = mysql.fetchall\_db (sql)

\_logger.info (\_rdata)

except Exception as e:

\_logger.error ("error: {}".format (e))

return returnData ("404", "失败", None)

return returnData ("0", "成功", \_rdata)

新增测试服务代码如下代码5-4所示：

代码5-4 新增测试服务

@app.route ("/testing/addHostService", methods=["POST"])

def addHostService():

try:

sql = """INSERT INTO `t\_host` (`host`, `port`, `explain`, `state`)

VALUE ('{0}', '{1}', '{2}', '1');

""".format (request.form['host'], request.form['port'], request.form['explain'])

\_rdata = mysql.exe (sql)

\_logger.info (\_rdata)

except Exception as e:

\_logger.error ("error: {}".format (e))

return returnData ("404", "失败", None)

return returnData ("0", "成功", \_rdata)

修改测试服务代码如下代码5-5所示：

代码5-5 修改测试服务

@app.route ("/testing/updateHostService", methods=["POST"])

def updateHostService():

try:

sql = """

UPDATE t\_host

SET `host` = "{1}",`port` = "{2}",`explain` = "{3}",state = "{4}"

WHERE

id = {0}

""".format (request.form['id'], request.form['host'], request.form['port'], request.form['explain'], request.form['state'])

\_rdata = mysql.exe (sql)

\_logger.info (\_rdata)

except Exception as e:

\_logger.error ("error: {}".format (e))

return returnData ("404", "失败", None)

return returnData ("0", "成功", \_rdata)

前端导出测试服务列表代码，如下代码5-6所示：

代码5-6 前端导出测试服务列表

exportExcel() {

require.ensure([], () => {

const { export\_json\_to\_excel } = require("../../vendor/Export2Excel");

const tHeader = [

"测试服务ID","测试服务主机","测试服务端口","测试服务说明","测试服务状态"];

const filterVal = ["id","host","port","explain","state"];

const list = this.tableData; //把data里的tableData存到list

const data = this.formatJson(filterVal, list);

var dayTemp = new Date();

dayTemp.setTime(dayTemp.getTime());

var excelFlieName =

dayTemp.getFullYear() +"-" +(dayTemp.getMonth() + 1) +"-" +dayTemp.getDate() +

"测试用例";

export\_json\_to\_excel(tHeader, data, excelFlieName);

});}

## 测试用例模块

测试用例前端实现效果，测试用例列表展示，如下图5-8所示：



图5-8 测试用例列表

新增测试用例功能实现图，如下图5-9所示：

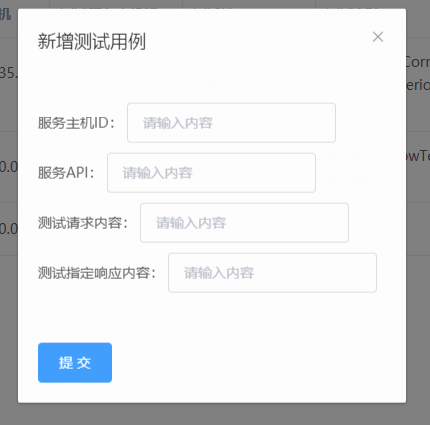


图5-9 新增测试用例

修改测试用例功能实现，如下图5-10所示：

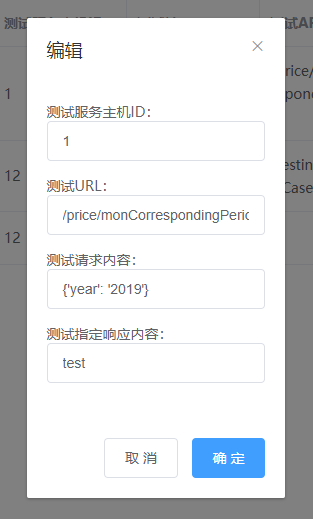


图5-10 修改测试用例

查看测试用例核心接口代码如下代码5-7所示：

代码5-7 查看测试用例

@app.route ("/testing/showTestCase", methods=["POST"])

def showTestCase():

try:

sql = """

SELECT

tc.id,h.id as host\_id,h.host,h.port,tc.url,tc.requests\_data,tc.result

FROM

t\_testcase tc

LEFT JOIN t\_host h on h.id = tc.host\_id

WHERE

h.state = 1

"""

\_rdata = mysql.fetchall\_db (sql)

\_logger.info (\_rdata)

except Exception as e:

\_logger.error ("error: {}".format (e))

return returnData ("404", "失败", None)

return returnData ("0", "成功", \_rdata)

新增测试用例核心接口代码如下代码5-8所示：

代码5-8 新增测试用例

@app.route ("/testing/addTestCase", methods=["POST"])

def addTestCase():

try:

sql = """

INSERT INTO `apitesting`.`t\_testcase`(`host\_id`, `url`, `requests\_data`, `result`)

VALUES ( {0}, '{1}', '{2}', "{3}")

""".format (request.form['host\_id'], request.form['url'], request.form.get("requests\_data", type=str, default=None), request.form['result'])

\_rdata = mysql.exe (sql)

\_logger.info (\_rdata)

except Exception as e:

\_logger.error ("error: {}".format (e))

return returnData ("404", "失败", None)

return returnData ("0", "成功", \_rdata)

修改测试用例代码如下代码5-9所示：

代码5-9 修改测试用例

@app.route ("/testing/updateTestCase", methods=["POST"])

def updateTestCase():

try:

sql = """

UPDATE t\_testcase

SET host\_id = {4},url = "{1}",result = "{3}",requests\_data = "{2}"

WHERE

id = {0}

""".format (request.form['testcase\_id'], request.form['url'],

request.form.get ("requests\_data", type=str, default=None), request.form['result'],request.form['host\_id'])

\_rdata = mysql.exe (sql)

\_logger.info (\_rdata)

except Exception as e:

\_logger.error ("error: {}".format (e))

return returnData ("404", "失败", None)

return returnData ("0", "成功", \_rdata)

## 测试结果模块

测试结果查看，前端展示如下图5-11所示：

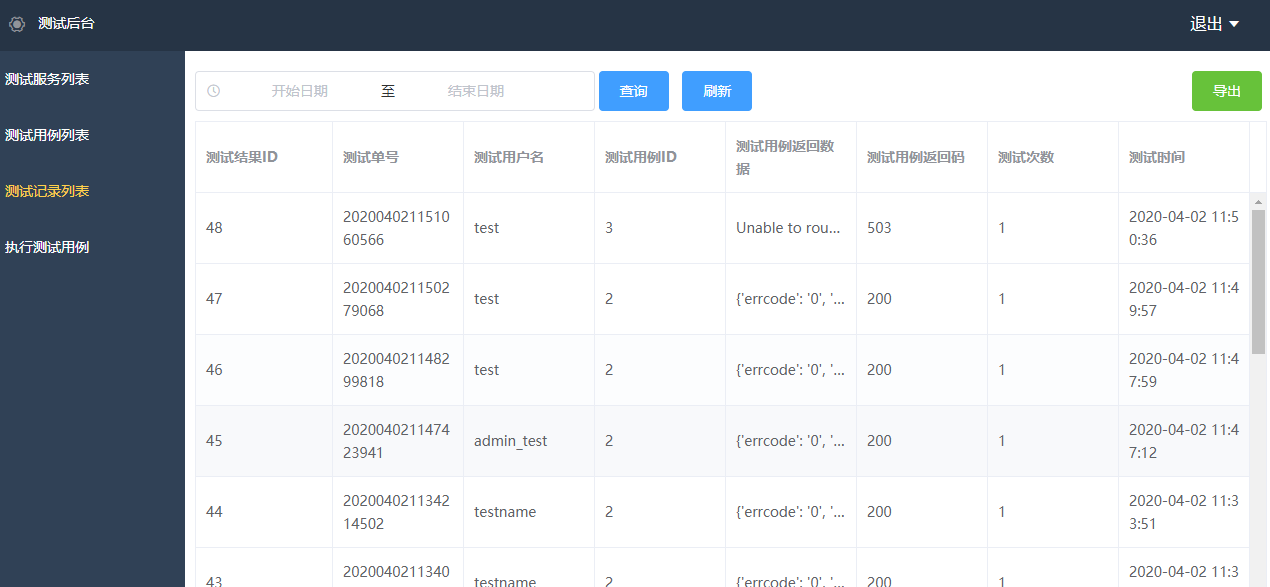


图5-11测试结果列表

测试结果可以使用时间作为索引进行查看，默认查看所有数据。

测试结果查看核心代码，如下代码5-10所示：

代码5-10 测试结果查看

@app.route ("/testing/showTestResult", methods=["POST"])

def showTestResult():

try:

sql = """

SELECT

tl.id,tl.test\_code,tl.test\_username,tl.testcase\_id,tl.testcase\_result,tl.testcase\_recode,tl.testcase\_count,date\_format(tl.create\_at,"%Y-%m-%d %H:%i:%S") as create\_at

FROM

t\_test\_log tl

LEFT JOIN t\_testcase tc on tc.id = tl.testcase\_id

order by

tl.create\_at DESC

"""

\_rdata = mysql.fetchall\_db (sql)

\_logger.info (\_rdata)

except Exception as e:

\_logger.error ("error: {}".format (e))

return returnData ("404", "失败", None)

return returnData ("0", "成功", \_rdata)

## 测试执行模块

执行测试用例有两种执行方式，一种是单个执行，另一种是批量执行。单个执行效果如下图5-12所示：



图5-12 单次执行测试用例

批量执行测试用例，首先在测试用例前面打勾，最后点击批量执行即可，如图4-13所示：

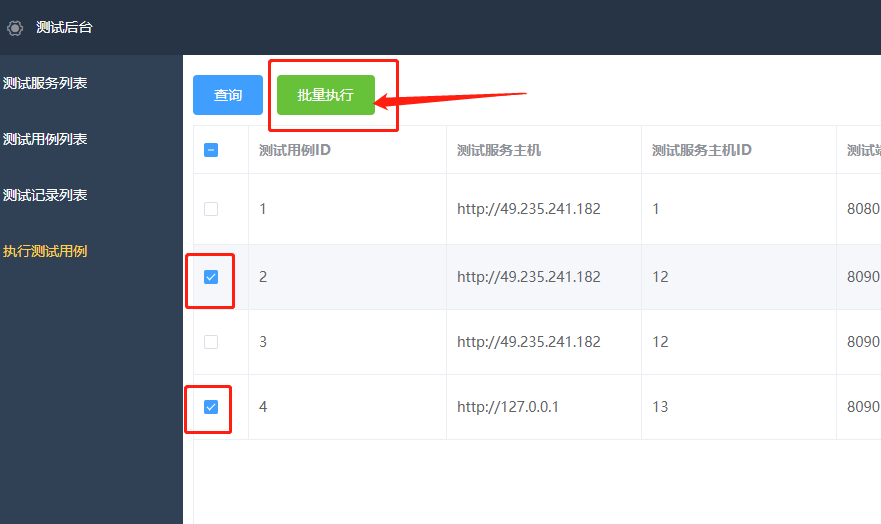


图5-13 批量执行测试用例

执行测试用例后台API代码如下代码5-11所示：

代码5-11 执行测试用例API

@app.route ("/testing/exeTestCase", methods=["POST"])

def exeTestCase():

try:

sql = """

SELECT

tc.id,h.host,h.port,tc.url,tc.requests\_data,tc.result

FROM

t\_testcase tc

LEFT JOIN t\_host h on h.id = tc.host\_id

WHERE

h.state = 1

and tc.id = {0}

""".format(request.form['id'])

\_rdata = mysql.fetchall\_db (sql)

\_requestsTool = requestsTool ()

if len(\_rdata[0]['requests\_data']) < 2:

data = ""

else:

data = ast.literal\_eval (\_rdata[0]['requests\_data'])

url = get\_url(\_rdata[0]['host'],\_rdata[0]['port'],\_rdata[0]['url'])

code,rdata = \_requestsTool.send\_post(url,data)

addHostService(getTestCode(),request.form['username'],\_rdata[0]['id'],(str(rdata).replace("\"","\'").replace("'","\'")),code,1)

except Exception as e:

\_logger.error ("error: {}".format (e))

return returnData ("404", "失败", None)

return returnData ("0", "成功", None)

# 系统测试

本次系统总共有五大模块，其中四个是核心模块，一个登录模块。五个模块的设计和实现都已经在上文详细描述。登录模块主要是用于后台系统权限认证，防止非测试人员使用。测试服务模块主要是记录测试服务IP加端口的方式，用于与测试用例进行关联，降低测试用例的耦合性。测试用例模块记录了需要测试的具体API和具体的请求方式，测试用例模块也是执行测试模块的基础。执行测试模块将测试用例模块关联测试服务模块得到测试目的数据，执行测试目的数据得到测试结果数据。

本次系统测试模块功能测试如下表6-1所示：

表6-1 模块功能测试

| 测试模块 | 测试目标 | 测试结果 | 存在问题 | 优化方向 |
| --- | --- | --- | --- | --- |
| 登录模块 | 是否能对登录的账号密码进行校验 | 测试通过 | 无 | 加入手机号验证登录，提高系统安全性。 |
| 测试服务模块 | 是否能新增/修改测试服务 | 测试通过 | 无 | 无 |
| 测试用例模块 | 是否能新增/修改测试用例 | 测试通过 | 无 | 新增不同的测试用例，如前端测试用例。 |
| 测试执行模块 | 是否能正确执行指定测试用例 | 测试通过 | 无 | 新增不同的请求方式 |
| 测试结果模块 | 是否能查看和筛选需要的测试结果 | 测试通过 | 无 | 通过用户名和订单号检索测试结果 |

由表6-1所得，本次系统测试通过，各个模块虽然存在问题，但是总体功能是可用的，根据后续优化问题，在下文已经提出。

本次自动化测试系统开发完成，本文提供一套完整的系统开发方案，经过测试，可行性高。后续优化可配置，系统耦合度低，模块可单独使用。

结束语

本次自动测试系统设计与实现，其中最大的难点在于设计。如何设计一套可行性高可视化程度高的测试系统是一直困扰的问题，本次测试系统创新的把测试服务和测试用例拆分开，这样测试用例是独立的。这样测试用例的变化并不影响测试服务，测试服务只是简单的IP+端口。而测试用例相对复杂很多，测试用例有API路径、请求内容、响应内容。

本次开发的自动化测试系统，能用批量执行测试用例，对测试结果进行保存，方便问题排查。前端实现使用VUE轻量级前端框架，系统有着开发成本低，代码量少的优点。后端采用Python进行开发，使用Flask作为后端API框架。Flask是目前Python最为轻量级的Web开发框架之一，能够大大降低开发陈本。

自动化测试系统的测试结果每次执行测试都会有唯一的测试单号，用于识别批量测试分批查看。

本系统的缺点是目前无法做到邮件反馈，也是后续需要改进的地方。本系统目前还有定时任务功能模块没有做，后续优化可以加入考虑。

本文提出的自动化测试系统尚有许多不足，后期需要完善的功能点在于后端智能化测试，进一步减少人员干预。

参考文献

1. Testing APIs protects applications and reputations, by Amy Reichert, SearchSoftwareQuality March 2015
2. All About API Testing: An Interview with Jonathan Cooper, by Cameron Philipp-Edmonds, Stickyminds August 19, 2014
3. The Forrester Wave™ Evaluation Of Functional Test Automation (FTA) Is Out And It's All About Going Beyond GUI Testing Archived 2015-05-28 at the Wayback Machine, by Diego Lo Giudice, Forrester April 23, 2015
4. Produce Better Software by Using a Layered Testing Strategy, by SEAN Kenefick, Gartner January 7, 2014
5. Onus for third-party APIs is on enterprise developers, by Amy Reichert, SearchSoftwareQuality July 2014
6. Accelerate Development with Automated Testing, by Nathan Wilson, Gartner December 30, 2013
7. A Guidance Framework for Designing a Great Web API, by Eric Knipp and Gary Olliffe , Gartner August 20, 2014
8. The Fight Against Brittle Scripts and Software Defects, by Adrian Bridgwater, Dr. Dobb's Journal October 26, 2011
9. How Do We Learn Composite App Testing-Speak?, by Adrian Bridgwater, Dr. Dobb's Journal February 14, 2012
10. Cohn, Mike (2009). Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional. p. 312. ISBN 978-0321579362.
11. Kuhlman, Dave. "A Python Book: Beginning Python, Advanced Python, and Python Exercises". Section 1.1. Archived from the original (PDF) on 23 June 2012.
12. Oliphant, Travis E. Python for Scientific Computing[J]. Computing in Science & Engineering, 9(3):10-20.
13. Rossum G V. The Python Language Reference Manual[M]. 2011.
14. Summerfield M. Programming in Python 3[M]. 2010.
15. Flanagan B D. JavaScript: The Definitive Guide[M]. 2011.
16. Musch J, Reips U D. A brief history of Web experimenting.[M]// false. 2000.
17. Widenius M , Axmark D , Dubois P . Mysql Reference Manual[M]. O'Reilly & Associates, Inc. 2002.
18. Vaswani V. MySQL Database Usage and Administration[M]// MySQL Database Usage & Administration. 2009.
19. Tobin, James (12 June 2012). Great Projects: The Epic Story of the Building of America, from the Taming of the Mississippi to the Invention of the Internet. Simon and Schuster. ISBN 978-0-7432-1476-6.
20. McPherson, Stephanie Sammartino (2009). Tim Berners-Lee: Inventor of the World Wide Web. Twenty-First Century Books. ISBN 978-0-8225-7273-2.
21. Dewey, Caitlin (12 March 2014). "36 Ways the Web Has Changed Us". The Washington Post. Archived from the original on 9 September 2015. Retrieved 1 August 2015.

致谢

白驹过隙，往事竟成秋风，光阴似箭，转瞬即逝。到这里论文的撰写工作也预示着完成了，也同样代表的我的大学生涯的结束，即将步入社会。脑海中回想大学这四年的时光，感受颇多，也或多或少的有些许遗憾，但更多的是对母校的留恋和敬意，我为此生可以有这样一段经历感到开心，回想四年的人和事，会让人觉得像家一般得温暖，对在此期间帮助过以及批评过我、关心过我的老师、同学表示衷心的敬意和感谢。

在此，我要感谢我的导师惠征老师。老师教学严谨，学识渊博，从设计之初到结束，老师尽心尽力，在遇到问题是总能给出相应的指导，在此，再一次向老师表示感谢。

其次，我要感谢我的父母，这四年以来他们总是给予了我很大的帮助，在此，衷心的感谢我的父母对我的付出，您们辛苦了。

最后，要感谢在场的各位老师，谢谢大家。