自动化平台测试开发

Python测试开发实战

邹辉◎编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书详细介绍了基于Python语言的自动化平台开发以及自动化测试技术知识。本书理论与实践相结合，以自动化平台开发为主线、自动化测试为辅线贯穿所有内容。具体包括自动化平台开发方案、数据库表结构设计、前后端代码编写，同时介绍了自动化测试方案、环境、代码、报告、API接口、AppUI、WebUI自动化测试、性能测试等内容。

本书适合从事自动化测试开发的广大开发和测试人员使用，也可供产品测试开发管理人员参考。

图书在版编目（CIP）数据

自动化平台测试开发：Python测试开发实战 / 邹辉编著. —北京：电子工业出版社，2018.7
ISBN 978-7-121-34290-5

Ⅰ. ①自… Ⅱ. ①邹… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆CIP数据核字(2018)第111029号

策划编辑：安　娜
责任编辑：宋亚东
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：787×980　1/16　印张：17.75　字数：352 千字
版　　次：2018 年 7 月第 1 版
印　　次：2018 年 7 月第 1 次印刷
定　　价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前　言

目前，虽然测试开发在软件测试行业被广泛应用，但其在企业中的投资回报率（ROI）并不是非常理想，在中小企业中尤为突出。究其原因，有自动化脚本维护困难、投入大、自动化用例不直观、自动化框架不适用、不稳定误报率大、前端 UI 变动较大且频繁等。因此，很多中小企业的自动化尚处于演示版本摸索的状态。

一些大企业有资金、人力、技术的支撑，进而自主研发了主流技术，包括自动化测试平台。加之其项目非常多，投入长时间的研发能够产生实际投资回报率，因此持续投入使用的时间也很长。对于中小型的互联网企业，根据企业人员规模、管理观念、公司实力、技术水平等情况，也比较倾向于自主研发和使用自动化测试平台。

自动化平台测试开发是指开发一个平台系统，然后在该系统上进行接口自动化测试、App 自动化测试、WebUI 自动化测试、性能测试等；当然，也需要结合已有的相关开源自动化框架进行集成，包括自动化脚本、Appium、Selenium、Locust 等开源框架。总体而言，开发出来的自动化平台能使自动化测试更高效、更务实、更直观、更可视化、更简洁。

近年来，开发技术发展到了人工智能、大数据、云计算等阶段，测试也将不仅仅停留在功能测试方面，会有开发技术和测试技术融合之势。

为了满足广大自动化测试开发技术人员的需求，笔者特编写本书，希望能给读者提供一个实用的操作指南。本书内容均基于 Python 语言编写，书中的源码在保留版权的情况下可供读者使用，读者使用源码时需要注明出自本书。

读者学习时可能会觉得比较枯燥乏味，但当你掌握了一个技术点，开发出一个功能并解决了某个问题的时候，一定会很有成就感。当遇到解决不了的问题时，多尝试、多思考、多学习、多看书、多提问，相信一定能解决。

另外，近年来开发技术发展到了人工智能，大数据，云计算，等等阶段，而如今测试也不能仅仅停留在相对简单技能的功能测试上，显然测试行业慢慢的有开发技术和测试技术融合的趋势，让重复的工作交给机器去执行，甚至机器学习。后期也有望结合人工智能，大数据，云计算，Devops 装备，进行测试开发技术上的持续探索和进步。

适合读者

绝大多数适用的读者

. 所有软件测试从业人员，包括测试团队 leader。

. 有一定开发语言基础的测试人员。

少部分适用的读者

. 软件测试专业的在校大学生。

. 软件技术、移动互联网相关人员，包括开发人员，研发团队 leader 等。

. 其他任何对自动化平台测试开发感兴趣的人。

作者简介

本书的作者具有丰富的软件测试从业经验，擅长开发和测试技术。

著有《自动化平台测试开发》本书：自动化测试平台开发，基于 Python。

曾著《软件自动化测试开发》一书：自动化测试框架开发，基于 Java。

大纲提要

第 1 章和 2 章：介绍 Python 开发语言、Web 开发、Mysql 数据库。

第 3 章：介绍自动化平台相关功能的开发，是核心内容。

第 4 章和第 5 章：介绍正则表达式和单元测试知识。

第 6 章到第 8 章：介绍自动化测试知识。

第 9 章到第 11 章：介绍性能测试、持续集成，定时任务全自动化测试。

附录 A/B：软件安装和使用。

关于勘误

虽然书中的每个技术点都曾在实际项目中实践和应用过，但也会因为我们个人技术、所测项目和视野的局限，以及本人因时间仓促和能力水平等种种原因，书中难免会有一些错误和纰漏，如果大家在阅读过程中发现了什么问题，恳请反馈给我，读者朋友们可即时在线交流，联系方式如下。

作者微信和 QQ 号：zouhui1003it，7980068

测试博客：http://www.cnblogs.com/finer

读者实战 QQ 互动群：377029807

Autotestplat 官网：http://www.autotestplat.com

微信公众号

测试开发社区，扫一扫即可关注

第 1 章

Python 零基础入门

1.1 Python 介绍

Python 是一门优雅且健壮的面向对象解释型计算机程序编程语言，具有面向对象、可升级、可扩展、可移植语法简洁清晰易学、易读写、易维护、健壮性、通用性、跨平台等特点。目前广泛应用于人工智能、机器学习、科学计算、大数据分析、图像处理、爬虫、区块链、自动化测试、测试开发、自动化运维、Web 开发、接口开发网站搭建等领域。

1.2 环境搭建

本书介绍的是基于 Windows 平台上进行安装，以及选择 Eclipse 4.5.2+Python 3.6.4+ PyDev 的安装环境，因为 Eclipse 功能强大，同时兼容 Windows 和 Mac，支持 Java 和 Python 等。

安装 Python 3.6.4，下载地址：https://pan.baidu.com/s/1diUgNzfMXgLhMQ ZjNNcNKA。官网下载地址：https://www.python.org/downloads/。

如图 1.1 所示，选择版本 Python 3.6.4，单击“Download Python 3.6.4”下载，下载后双击安装包，进入 Python 安装向导，选择默认设置进行安装即可。

设置环境变量“C:\Users\zh\AppData\Local\Programs\Python\Python36”。

▲图 1.1

1.2.1 安装 JDK 1.7

下载解压文件夹进行默认安装，设置 Java 环境变量，右键单击“我的电脑→属性→高级→环境变量”，新建系统变量 JAVA_HOME 和 CLASSPATH。

变量名：JAVA_HOME。

变量值：C:\Program Files\Java\jdk1.7.0_80。

变量名：Path。

变量值：%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;。

变量名：CLASSPATH。

变量值：;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;。

测试环境安装成功：

运行 CMD，输入 java –version，如果成功则出现 Java 信息，如图 1.2 所示。

安装文件路径：https://pan.baidu.com/s/1gf4Ym3L。

▲图 1.2

1.2.2 安装 Eclipse

设置 IDE 集成开发环境，注意确认 Windows 系统是 32 位还是 64 位，Eclipse 版本需要与之对应。

下载地址：https://pan.baidu.com/s/1dF0sBcP。

1.2.3 安装配置 Python 3

Python 2 与 Python 3 在语法上有较大的一些差异，比如输出字符，Python 3 为 print (‘hello python3’)； Python2 为 print‘hello python2’。

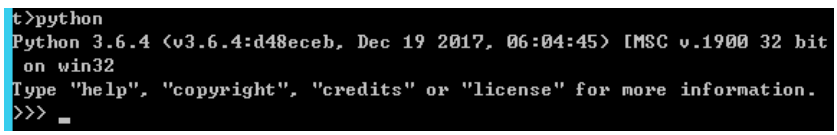
比如异常处理，Python 3 为 exception exc as e； Python 2 为 exception exc,e。

比如操作 MySQL，Python3 为 import pymsql；Python2 为 import mysqldb。

由于官方未来将停止更新和维护 Python 2，所以个人建议用 Python 3。下载地址：https://www.python.org/downloads/。

配置环境变量：C:\Users\zh\AppData\Local\Programs\Python\Python36。

测试环境，运行 CMD，输入 Python，如输出如图 1.3 所示信息则表示配置成功。



▲图 1.3

1.2.4 安装 PyDev

在线安装 PyDev，插件的官方网址：http://www.pydev.org/。在 Eclipse 里，单击“Help→Install New Software”

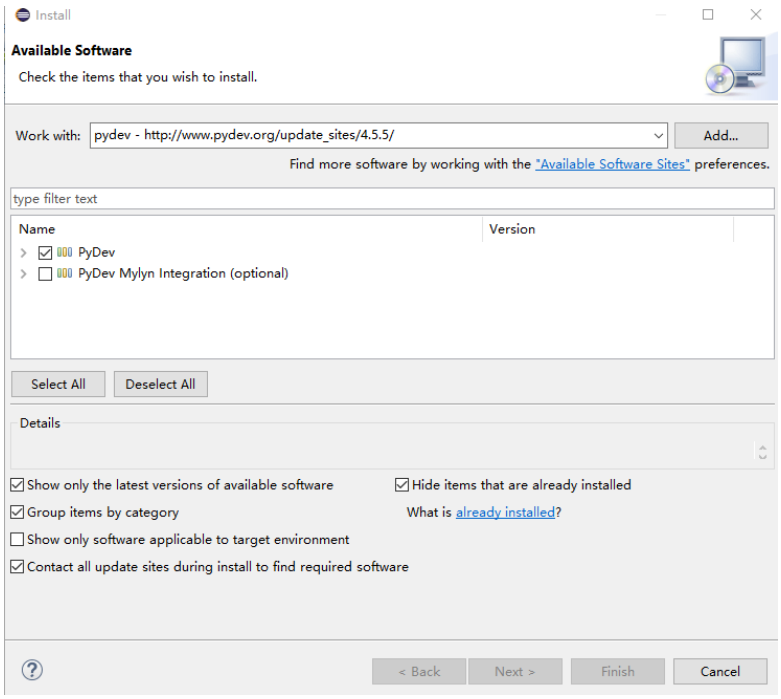
在弹出的对话框中单击“Add”按钮，在 Name 中选择 PyDev，在 Location 中输入 http://www.pydev.org/update_sites/4.5.5/，单击“OK”按钮，如图 1.4 所示。如果是最新版本的 JDK，则输入 http://pydev.org/updates。

单击“Next”按钮，选择默认设置进行安装。

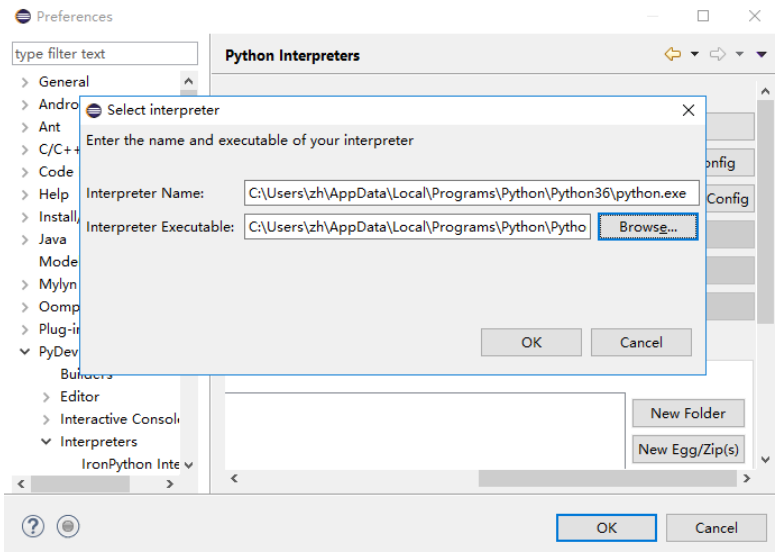
1.2.5 配置 PyDev

接下来配置 Python 解释器路径，在 Eclipse 中，单击“Windows→Preferences”，然后在对话框中单击“PyDev→Interpreters→Python Interpreter”，单击“New”按钮，选择 Python.exe 安装路径，单击“OK”按钮，如图 1.5 所示。

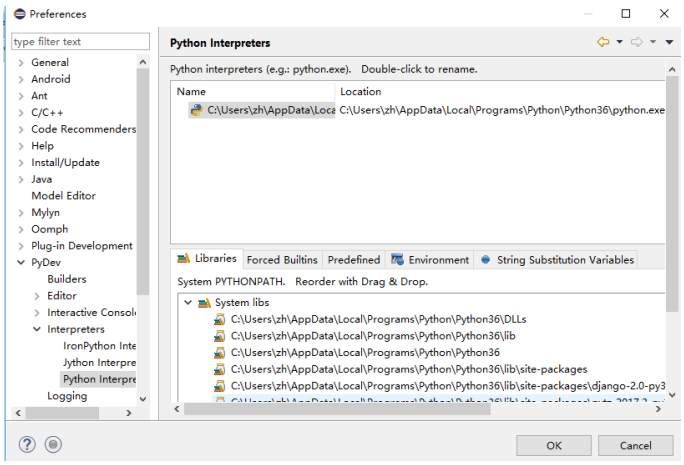
如图 1.6 所示，安装好环境后，就可以创建 Python 工程了。



▲图 1.4



▲图 1.5



▲图 1.6

1.2.6 新建一个项目工程

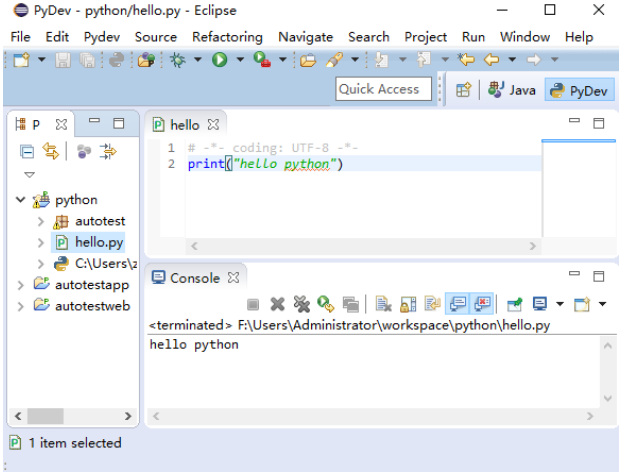
单击“File→New→Projects”，选择 PyDev-PyDev project，输入项目名称:Python。

新建 Python 文件：hello。

```
#include utf-8
print ('hello python')
```

单击“运行”按钮，可以看到控制台输出 hello python，如图 1.7 所示。

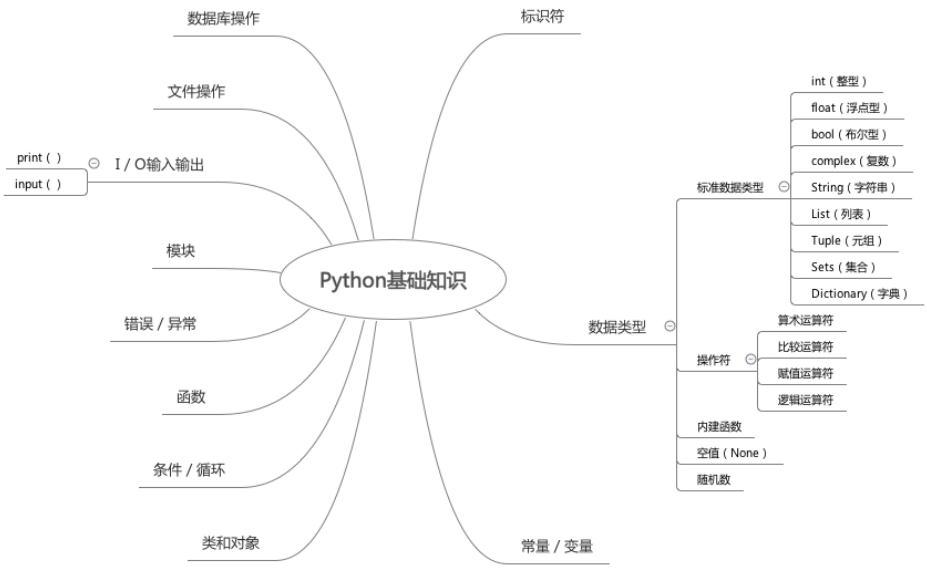
程序可以正常运行，说明 Python 的环境搭建成功了。



▲图 1.7

1.3 基础知识

本章仅简要介绍测试开发会用到的一些 Python 基础知识，如图 1.8 所示。如要了解更详细的内容，请参考 Python 专业书籍和官方资料。



▲图 1.8

1.3.1 语句与语法

1．Print 语句，print(“hello”)

逗号：打印多个表达式时用逗号（,）分开，列表中多个值用逗号分开，参数中多个值用逗号分开。

分号：一行中写多条代码时需要加上分号（;），如果只写一条代码，可以加也可以不加。

冒号：用冒号（:）、缩进及空格来区分代码块， 在 if class def 类、函数、循环、分支等。

示例：

```
if True:
     Print（“hello，python”）
Else:
     Print （“hello，java”）
```

引号：单引号、双引号、多引号都可以表示字符串。

加号：加号可表示连接字符串，也可以表示加法运算符。

缩进：四个空格键，多一个空格、少一个空格或 Tab 键都会报错。

注释：单行注释使用#，多行注释使用"' "'。

2．编码格式

-*- coding: UTF-8 -*-：表示支持 UTF-8 中文。

U‘中文’：表示 unicode 创建实例的格式显示中文。

Encode（“utf-8”）：表示以 UTF-8 编码对对象进行编码，获取 byte 类型对象。

Decode（“utf-8”）：表示以 UTF-8 编码对对象进行解码，获取字符串对象。

3．标识符

大小写敏感，第一个字符必须为字母或下划线，不能使用字符、数字、下画线之外的特殊符号，不要与关键字重名。

常见关键字有：Print，class，def，for，if，else，or，in ，is，import，except，from，try，pass 等。

1.3.2 数据类型

1．基本数据类型

1）数字类型

- 整型 int

Print （100）：输出 100。

- 浮点数 float

Print （100.00）：输出 100.0。

- 布尔型

布尔型数据类型，其值是 True 或 False，它实际上是内置整数类型 int 的子类，其 True 可设置为整数 1，False 可设置为整数 0，True+1=2。

print (True+1) 运行结果为 2。

Python 里的空值用 None 表示。

2）字符串

顾名思义，由一连串的字符组成的字符集合为字符串，Python 语言通常用引号标识，还支持索引。

Strname=“fin is ok”，那么 Strname[0]=f。

如需将多个字符串连接起来，可以用（+）连接，

Strname1=“fin”， strname2=“is”， strname3=“ok”，

Strname== strname1+‘ ’+strname2+‘ ’+strname3。

3）标准型函数

列表：在 Python 中最常用，相当于 Java 里面的数组，用[]来定义空，访问列表等值用 alist [0] ； alist [1:3]。

```
aList = []
alist =  [1, ‘a’,  ’test’,  ’dev’, 1.1]
```

元组：用小括号表示，功能与列表类似，区别在于元组中元素不能修改。

字典：相当于哈希表，即键-值（key-value）组成。

集合：用 set（）表示，常用 add 和 update 增加和修改集合中数据。

数据类型转换：str（x）将 x 转换成字符串类型。

1.3.3 运算符和表达式

1）算术运算符，a=1，b=1，c=a+b，即 c=2。有+、–、*、 / 等算术符。

2）比较运算符，if（a==b），print （“a 等于 b”）。

3）赋值运算符，c=a+b，= 即为简单的赋值运算符。

4）逻辑运算符，if（a or b），其中 and or not 代表布尔值与或非。

1.3.4 变量和赋值

strName = “fin” #定义了字符串变量并赋值。

Number = 100 #定义了整型变量并赋值。

1.3.5 基本控制流程

1）If else elif 条件分支语句

If 经常与 else、elif 一起使用，其中 elif 相当于 else if 的意思。

2）for 语句

遍历一个列表或字符串等

```
Fruits = [“apple”, “banana”,“grape”,“mango”]
For fruit in fruits
   Print (fruit)
```

3）pass 语句，即不做任何事情。

1.3.6 类和对象

面向对象语言都包括类、属性、方法、对象（包含了类成员变量和方法）、实例化和继承等概念。

```
Class Test:   #定义 Test 类
     i=100
     Def num(self):
```

```
          Return ‘hello’
t=Test() #实例化 Test 类
Print(“Test 类的属性是”,t.i)
Print (“Test 类的方法是”,t.num())
Class Test（self）:
……
```

表明 Test 继承类基类 self。

1.3.7 函数

```
Def test（）:
     Print （‘test’）
```

自带函数：如 str（x），int（x），len（x），open（fn‘w’）。

参数：有必选参数、默认参数、可变参数、命名关键字参数和关键字参数。

关键字参数允许传 0 个或多个参数；

```
Def user（name，age，sex）
  Print（'name：'name，'age：'age）
```

参数为 name，age，sex。

```
Def user（name，age=18）
  Name='zouhui'
  While age》17
     Name = name +'is greate'
  Return name
```

必选参数 name，默认值参数 age。

```
Def user（*ages）
  Age=18
  For i in ages
      Age= age + i
  Return age
```

可变参数 ages，返回值为 age。

1.3.8 模块导入和包导入

引用模块或包 import time、sys 是指引用了 time 和 sys 模块。

自定义模块 From X import X 是指从某个模块或某个包导入某个对象。

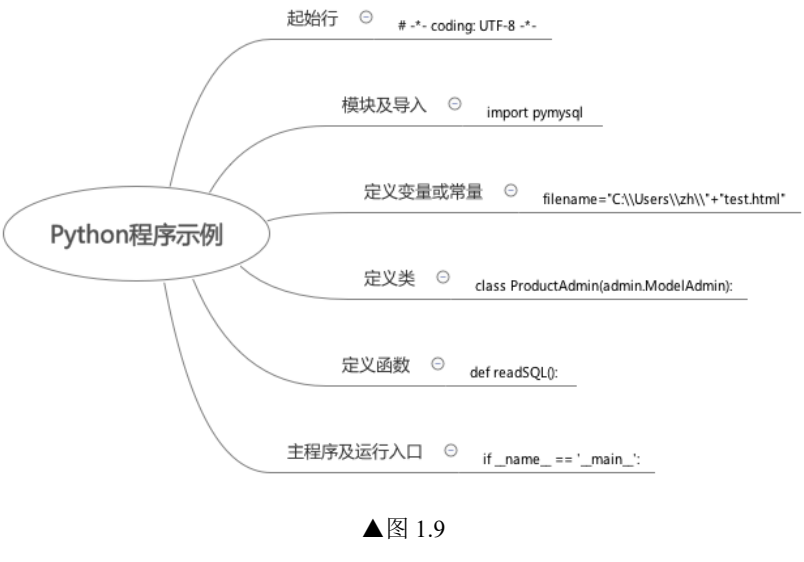
1.3.9 异常处理

格式语法为 try:语句 except as:e 语句。

1.4 实例讲解

1.4.1 文件操作实例

将内容写到相应的文件中，读取并打印到控制台，布局结构示例如图 1.9 所示。



▲图 1.9

程序清单：

```
# -*- coding: UTF-8 -*-

def wfile():    #定义写文件函数
    try:
        filename="C:\\Users\\zh\\"+"test.html"  #定义文件路径和文件名变量
    except IOError:
        print ("file create error")   #文件异常处理
    else:
        fp=open(filename,'wb')   #打开写文件
        fp.write("test".encode('utf_8'))  #将 test 字符串写入到文件中
        fp.close()  #关闭写文件

def rfile():  #定义读文件函数
    try:
        filename="C:\\Users\\zh\\"+"test.html"  #定义读文件路径和文件名变量
        fp=open(filename,'r')   #打开读文件
    except IOError:
        print ("file open error")  #文件异常处理
    else:
        for f in fp:    #循环读取每行数据
    print ("file data is "+ f)  #打印数据
        fp.close()      #关闭读文件

if __name__ == '__main__':   #程序调用 main 主函数
    #readSQL()
    wfile()       #调用写文件函数
    rfile()       #调用读文件函数
    print ('Done!')  #输出‘Done’完毕
```

运行结果：

```
file data is test
Done!
```

1.4.2 数据库操作实例

先安装好 MySQL 数据库环境，详见本书第 2 章 MySQL 数据库使用；写基于 Python 的 MySQL 数据库的查询

语句编程实例，其他增删改与此类似。

程序清单：

```
# -*- coding: UTF-8 -*-
import pymysql

def readSQL():
    #查询 SQL 语句
    sql="SELECT id,`apiname`,apiurl from apitest_apistep where apitest_apistep.Apitest_id=2 "
    #打开 MySQL 数据库连接
coon = pymysql.connect(user='root',passwd='test123456',db='autotest'
,port=3306,host='127.0.0.1',charset='utf8')
    #获取数据库操作游标
    cursor = coon.cursor()
    #执行 MySQL 查询语句
    aa=cursor.execute(sql)
    #获取执行查询语句后的结果数据列表
    info = cursor.fetchmany(aa)
    print(info)
    # 提交
    coon.commit()
    #关闭游标
    cursor.close()
    #关闭连接
    coon.close()

if __name__ == '__main__':
    readSQL()
    print ('Done!')
```

执行后查询结果数据如下：

```
((1, '登录', '127.0.0.1:8000/login'), (2, '购物', '127.0.0.1:8000/login'))
Done!
```

主函数：表示主程序调用 main（）函数即入口，格式固定如下即可：

```
if __name__ == '__main__':
    readSQL()
    print ('Done!')
```

第 2 章

Web 应用框架

2.1 介绍

现在的 SaaS 云平台已经达到不用写代码，只构建一些业务编排、逻辑就能生成相应的文本框、按钮、菜单、表格、数据、页面，以及一套或多套类似 CRM、OA、电子商务等系统。如阿里云、华为、腾讯云等都能在云平台上极速地新建和分配操作系统，已经可以不需要单独的主机和服务器了。另外，有消息称 Google 公司已经能通过机器学习自动化写出前端代码了。因此，随着开发技术的提升，自动化平台的开发以及自动化测试也会变得比以前更便捷。有了完善的自动化平台，可以降低自动化测试的技术门槛。

本书应用的是 Django，它是一个 Python 的高级 Web 框架，功能强大，封装了大量底层，使开发 Web 代码变得高效、快速、简洁。Django 希望不重复自己，具有松耦合与灵活等特性，非常适合快速开发。Django 是一个简洁而强悍的 Web 开框架，基于 Python 语言开发，只需要少量的代码就可以快速实现强大的功能。

掌握 Django 需要理解模型（Model）、视图（Views）和模板（Templates），即 MTV 模式，有点类似 Java 语言开发里面的 MVC 模式。模型：控制数据，如存取，关联关系等；视图：定义显示的方法、函数；模板：业务逻辑。

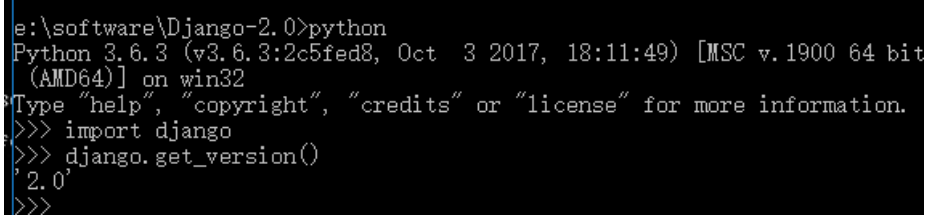
2.2 环境搭建

选择安装 Django 2.0，Django 1.6 及以上版本完全兼容 Python 3.x。下载地址：https://www.djangoproject.com/download/。解压 Django2.0 压缩包，可放到 Python 同一根目录下，运行 CMD，进入 Django 目录，执行 python setup.py install，按 Enter 键，如图 2.1 所示。

```
e:\software\Django-2.0>python setup.py install1
```

▲图 2.1

结果如图 2.2 所示。



▲图 2.2

2.3 Django 开发入门

Django 组成部分如表 2.1 所示。

▼表 2.1

组成部分	含义
django-admin.py	用于管理任务的命令行工具，如创建 Django 项目
manage.py	命令行实用工具，用户 Django 应用进行各种交互
Python manage.py runserver 127.0.0.1:80	启动项目程序和端口
manage.py createsuper	创建超级用户
python manage.py makemigrations	记录 model 同步到 X_initial.py
python manage.py migrate	迁移同步到数据库
settings.py	设置
model.py	模型设计
views.py	视图
urls.py	映射
templates	模板
Form	表单

续表

组成部分	含　义
admin.py	后台模型
django	内置函数使用

2.3.1 创建项目

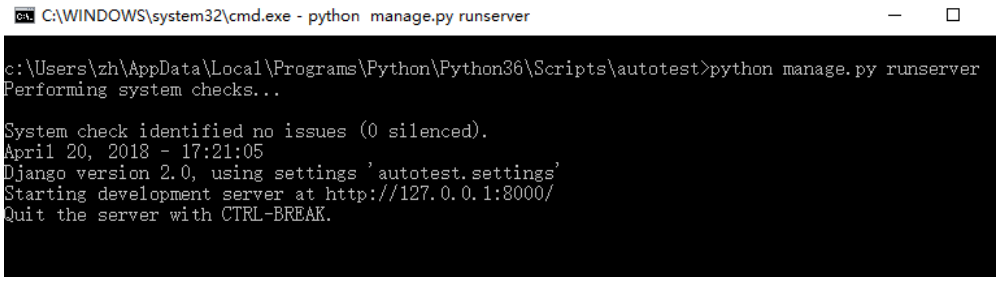
运行 CMD，进入站点目录。C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts

输入命令：django-admin startproject autotest

2.3.2 启动服务

然后输入 cd autoitest 切换到所创建的项目路径，输入命令：python manage.py runserver。

若输出如图 2.3 所示信息，则说明 Django 服务启动成功。



▲图 2.3

默认端口是 8000，如果需要改动端口，可以用 python manage runserver 127.0.0.1:80 或 python manage runserver 0.0.0.0:80 启动。

然后在浏览器中输入：http://127.0.0.1:8000/。

如图 2.4 所示，Web 服务启动成功，前端已经可以正常访问了。

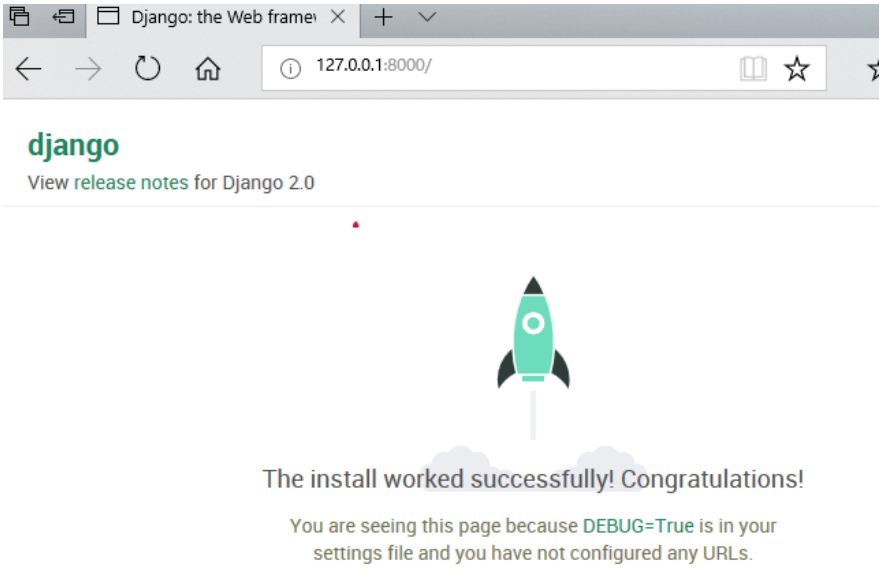
步骤 1 开始构建 Django 后端。

迁移同步数据并创建相应的表，运行 CMD，切换到相应目录，输入指令 `python manage.py makemigrations`，按 Enter 键，如图 2.5 所示。

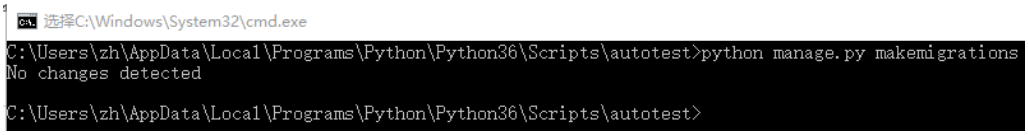
输入指令 `python manage.py migrate`，按 Enter 键，如图 2.6 所示。

步骤 2 创建 admin 超级用户。

输入指令：`python manage.py createsuperuser`。设置账号为 admin，邮箱为 7980068@qq.com，密码为 test123456，如图 2.7 所示。



▲图 2.4



▲图 2.5

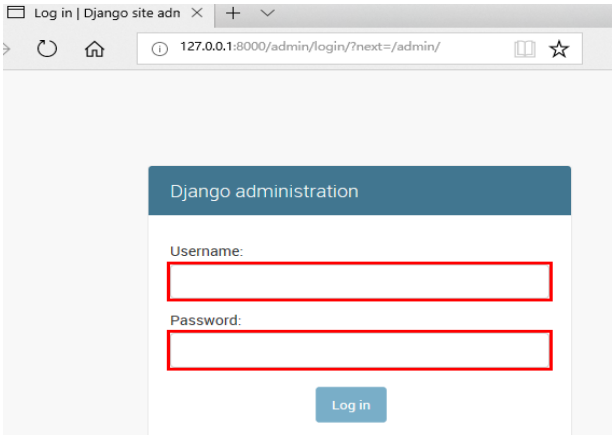
```
C:\Windows\System32\cmd.exe
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
```

▲图 2.6

```
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py createsuperuser
Username (leave blank to use 'zh'): admin
Email address: 7980068@qq.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Password:
Password (again):
Superuser created successfully.
```

▲图 2.7

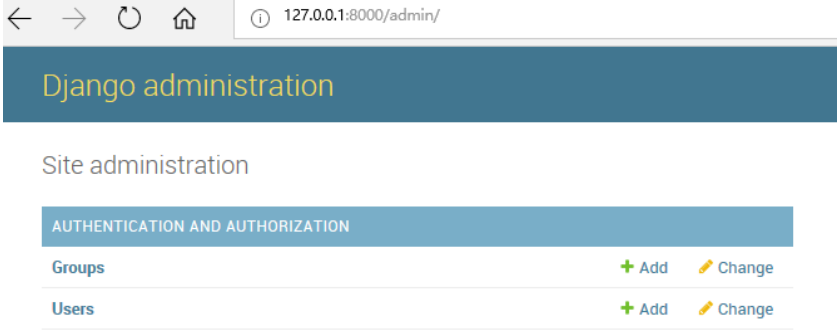
步骤 3 在浏览器中输入 http://127.0.0.1:8000/admin（如图 2.8 所示）。



▲图 2.8

步骤 4 在文本框中分别输入用户名（admin）和密码（test123456）。

单击“登录”（Login）按钮后，如图 2.9 所示。



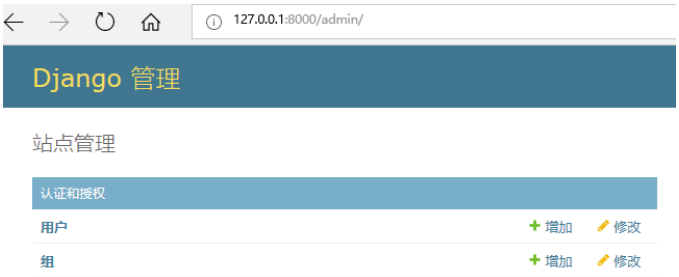
▲图 2.9

步骤 5 汉化为中文界面。

将 Django 设置为中文，默认是英文，注释原有的英文和时区，在 Autotest/Autotest/ Settings.py 中加入中文和时区，如下所示。

```
#LANGUAGE_CODE = 'en-us'
#TIME_ZONE = 'UTC'
LANGUAGE_CODE = 'zh-Hans'
TIME_ZONE = 'Asia/Shanghai'
```

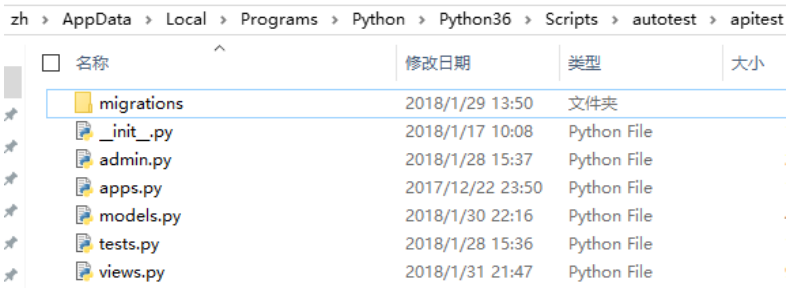
再刷新界面，如图 2.10 所示。



▲图 2.10

2.3.3 创建应用

运行 CMD，切换到 autotest 工程目录，输入指令 `Python manage.py startapp apitest`。创建成功后，再查看 autotest 目录，会发现相应的 apitest 文件夹及文件，如图 2.11 所示。



▲图 2.11

将 apitest 应用添加到 autotest 项目下面，在 settings.py 中加入“apitest”，如图 2.12 所示。

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'apitest',
]
```

▲图 2.12

2.3.4 创建视图

在 views 中加入 test 函数

```
from django.shortcuts import render
from django.http import HttpResponse    #加入引用
# Create your views here.
def test(request):
    return HttpResponse("hello test")   #返回 HttpResponse 响应函数
```

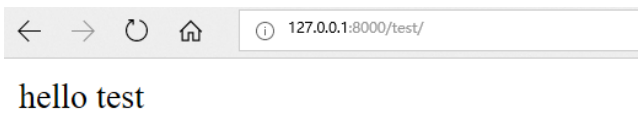
2.3.5 创建映射

Django 2.0 使用的是 path 匹配，支持正则表达式。Django 2.0 以前的版本是 URL 匹配映射，path 匹配可以说是 URL 匹配的增强，匹配功能更强。

现在把视图中的函数映射到浏览器前端页面， 在 autotest/urls.py 中加入如下内容：

```
from django.contrib import admin
from django.urls import path
from apitest import views     #加入引用

urlpatterns = [
    path('admin/', admin.site.urls),
    path('test/', views.test),    #加入关联路径及函数
]
```

在浏览器中输入 127.0.0.1:8000/test，即看到函数返回响应数据，如图 2.13 所示。



▲图 2.13

2.3.6 创建模板

步骤 1 在 apitest 下创建 templates 文件夹，在 templates 文件夹下创建 login.html 文件，在 templates/login.html 文件中，加入如下内容。

```
<!DOCTYPE html>
 <html>
 <head>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
     <title>Login</title>
```

```
</head>
<body>
<h1>login</h1>

<form method="post" action="/login/">
{% csrf_token %}
<br> <input name="username" type="text" placeholder="username" >
<br> <input name="password" type="password" placeholder="password">
{{ error }}<br>
<br> <button id="submit" type="submit">submit</button>

</form>
</body>
</html>
```

步骤 2 在 autotest/urls.py 中创建关联映射。

```
from django.contrib import admin
from django.urls import path
from apitest import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('test/', views.test),
    path('login/', views.login),
   ]
```

步骤 3 在 apitest/views.py 中创建 login 函数，并以 UTF-8 格式保存。

```
from django.shortcuts import render
from django.http import HttpResponse

def login(request):
   return render(request,'login.html')
```

步骤 4 在浏览器中输入 http://127.0.0.1:8000/login，会看到登录页面，如图 2.14 所示。

▲图 2.14

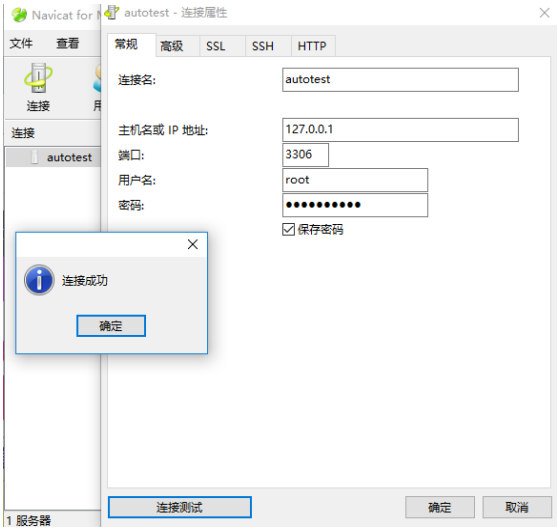
2.4 MySQL 数据库使用

Django 默认使用 SQLite3 数据库，而在稍中大型的项目中大多使用 MySQL，因此我们安装 MySQL 进行后续的开发。

步骤 1 MySQL 安装文件地址为 https://pan.baidu.com/s/1eSiy6Fw，下载后选择默认设置进行安装，设置用户名为 root，密码为 test123456。

然后安装 MySQL 连接的客户端工具 Navicat，安装文件地址为 https://pan.baidu. com/s/1slb8boh。

下载后启动程序，然后连接 MySQL 数据库，如图 2.15 所示。



▲图 2.15

步骤 2 在 Django 中默认连接的是 SQLite 数据库。

```
DATABASES = {
  'default':
   {
      'ENGINE': 'django.db.backends.sqlite3',
      'NAME': 'mydatabase',
   {
}
```

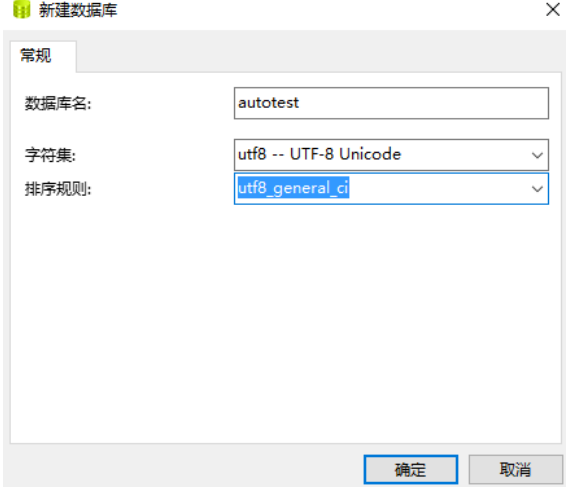
然后把 SQLite 连接改成 MySQL 的连接，在 settings.py 中修改为如下内容。

```
DATABASES = {
   'default':
   {
     'ENGINE': 'django.db.backends.mysql',
     'NAME': 'autotest',
     'USER': 'root',
     'PASSWORD': 'test123456',
     'HOST': '127.0.0.1',
     'PORT': '3306',
   }
}
```

步骤 3 在 autotest 目录的__init__.py 文件中输入如下内容，然后保存该文件。

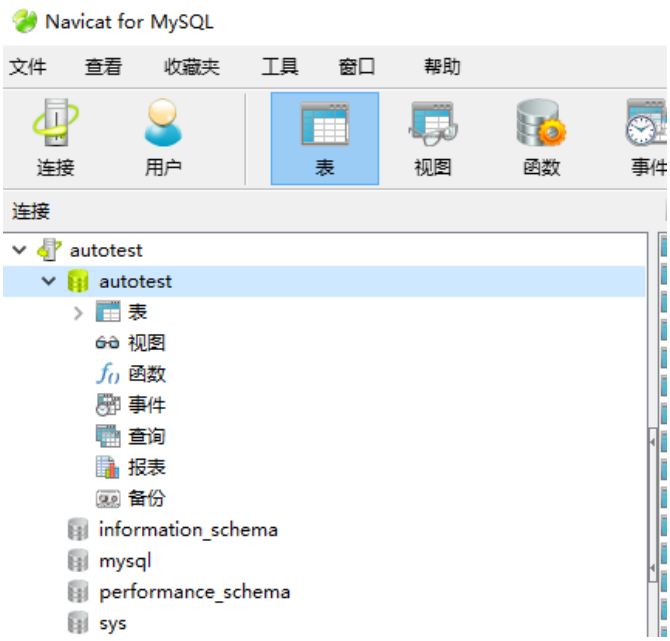
```
import pymysql
pymysql.install_as_MySQLdb()
```

接下来，在 Navicat 客户端中创建 autotest 数据库，如图 2.16 所示。



▲图 2.16

单击“确定”按钮，创建数据库成功，双击 autotest，打开后如图 2.17 所示。



▲图 2.17

步骤 4 安装 PyMySQL，PyMySQL 是通过 Python 3 代码连接和操作 MySQL 的库，安装文件地址为 https://pypi.python.org/pypi/PyMySQL。下载后解压缩文件，运行 CMD，切换到 PyMySQL 所在目录下进行安装，运行命令 python setup.py install。

步骤 5 在 C:\Users\zh\AppData\Local\Programs\Python\Python36\Lib\site-packages\Django-2.0-py3.6.egg\django\db\backends\mysql\base.py 文件中，用#注释如下内容。

```
if version < (1, 3, 3):raise ImproperlyConfigured("mysqlclient 1.3.3 or newer is required; you have %s
" % Database.__version__)
```

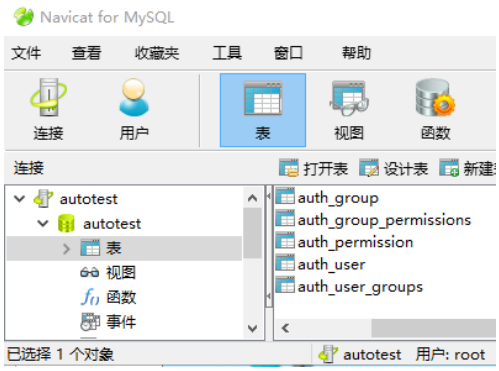
步骤 6 迁移同步数据库和表结构。运行 CMD，切换到相应目录，如图 2.18 所示，输入指令为：

```
python manage.py makemigrations
python manage.py migrate
```

```
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, apitest, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying apitest.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
```

▲图 2.18

打开数据库客户端查看，默认生成的部分表如图 2.19 所示。

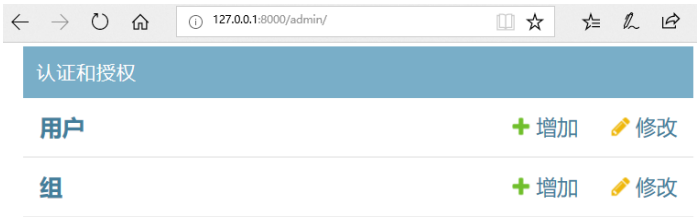


▲图 2.19

步骤 7 创建 MySQL 的超级管理员账号。运行 CMD，切换到相应目录，输入指令 python manage.py createsuperuser，按 Enter 键后，输入用户名和密码等，如图 2.20 所示。

▲图 2.20

步骤 8 在浏览器中输入 http://127.0.0.1:8000/admin，在登录页面中输入用户名（admin）和密码（test123456）登录后如图 2.21 所示。



▲图 2.21

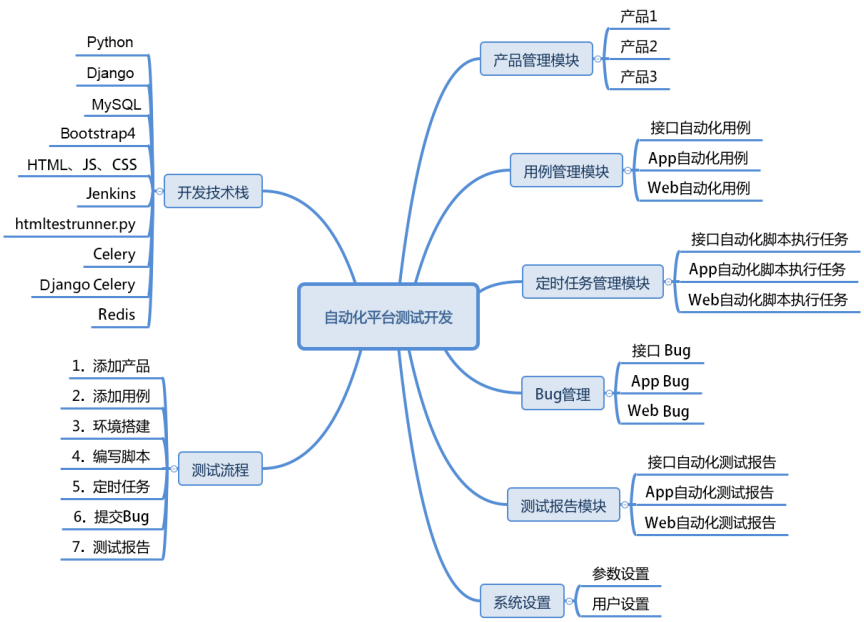
第 3 章

自动化平台开发

3.1 自动化平台开发方案

3.1.1 功能需求

支持 API、AppUI、WebUI 性能等自动化测试，集成实现测试用例管理、产品管理、任务计划、测试报告、定时任务或持续集成等功能模块。使自动化测试的过程达到数据、脚本、任务分离易于维护和管理，成本更低、数据更直观、产出更快等目标，甚至能让不会代码的手工业务测试人员参与后续自动化测试工作等。整体架构如图 3.1 所示。



▲ 图 3.1

▲图 3.2

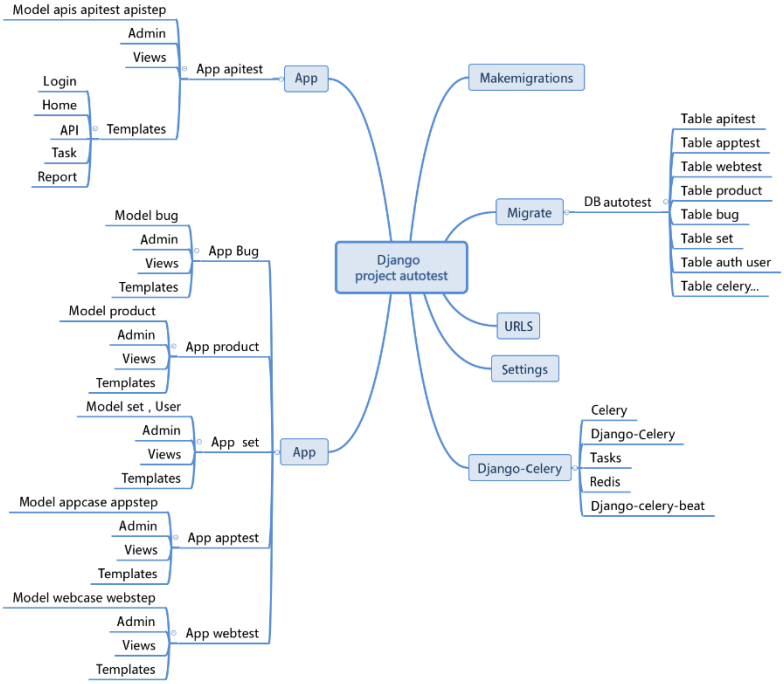
3.1.2 技术知识点

技术知识点如表 3.1 所示。

▼表 3.1

分　类		说　明
平台开发部分	Python	用于开发平台和编写自动化测试脚本
	Django	用于自动化平台后台开发
	MySQL	用于自动化平台开发和测试的数据存储
	HTML，JS，CSS	用于自动化平台前端开发
	Bootstrap4	用于自动化平台前端及优化
自动化测试部分	Request，Unittest	用于接口自动化测试、单元测试脚本
	SDK，ADT，Uiautommuior	用于 App 运行开发环境，元素定位
	Appium	用于 App 自动化测试
	Selenium	用于 Web 自动化测试
	Jenkins	用于定时任务邮件报告等持续集成
	Celery，Django-Celery	用于定时任务管理模块开发

自动化平台开发技术栈如图 3.3 所示。



▲图 3.3

3.1.3 开发时间计划

如果是刚入门、但有一点代码基础的测试人员，大概 3 个月能做出演示版（Demo）进行自动化测试，6 个月内胜任开展工作中项目的自动化测试。

如果是有自动化测试基础的测试人员，大概 1 个月能做出演示版（Demo）进行自动化测试，3 个月内能胜任工作中项目的自动化测试。

3.1.4 投资回报率可视化

投资回报率可视化如表 3.2 所示。

▼表 3.2

计算项	具体内容
自动化测试成本计算	自动化平台开发成本
	用例编写成本
	脚本维护成本
	执行结果分析成本

续表

计算项	具体内容
自动化测试收益计算	自动化测试用例数
	发现有效 Bug 数
	节省的人力和时间，即效率提升
	版本迭代次数，即自动化使用率

3.1.5 后期优化计划

后期有待优化的功能有平台 UI 优化、脚本管理优化、测试报告优化、性能测试模块优化、邮件模块优化等。将来会考虑集成人工智能、机器学习、Devops 装备等技术。

3.2 登录功能实现

步骤 1 创建登录函数，在 autotest/views.py 中，写入如下代码。

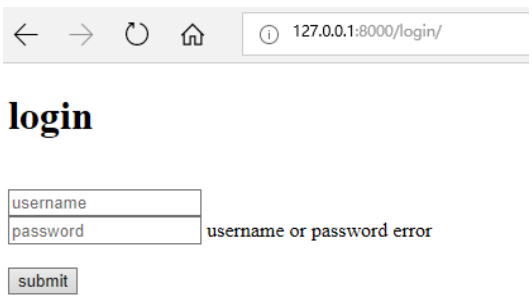
```
from django.shortcuts import render
from django.http import HttpResponse,HttpResponseRedirect
from django.contrib.auth.decorators import login_required
from django.contrib import auth
from django.contrib.auth import authenticate, login

def login(request):
    if request.POST:
        username = password = ''
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = auth.authenticate(username=username, password=password)
        if user is not None and user.is_active:
            auth.login(request, user)
            request.session['user'] = username
            response = HttpResponseRedirect('/home/')
            return response
        else:
            return render(request,'login.html', {'error': 'username or password error'})
    #else:
     #   context = {}
      #  return render(request, 'login.html', context)

    return render(request,'login.html')
```

步骤 2 创建向导。

在 autotest / urls.py 文件中加入如下代码。

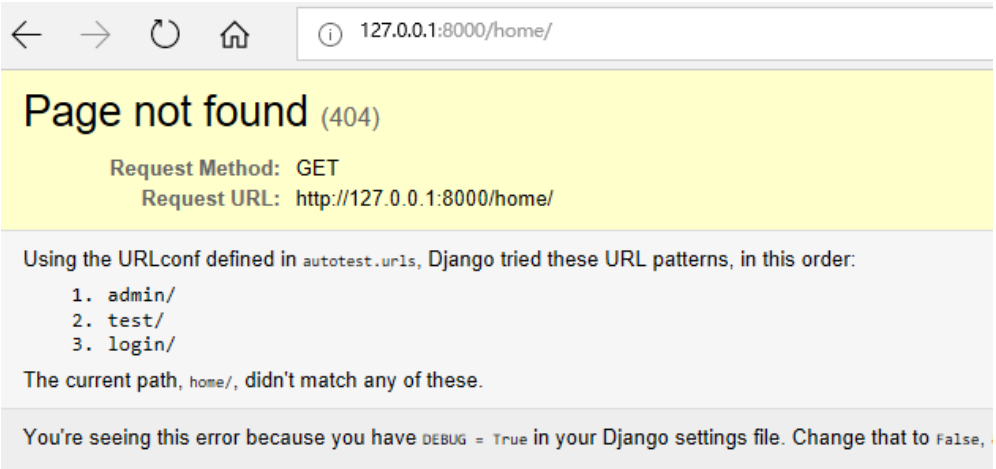
```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('test/', views.test),
    path('login/', views.login),
  # path(r'^home/$', views.home),
]
```

步骤 3 在登录页面 http://127.0.0.1:8000/login，如输入错误的用户名和密码，提示如图 3.4 所示。



▲图 3.4

步骤 4 输入正确的用户名（admin）和密码（test123456），单击“submit”按钮后，会出现错误，如图 3.5 所示。



▲图 3.5

根据关键错误日志信息定位问题，排查错误，得知 home.html 不匹配，在 autotest/apitest/ templates 目录下新建 home.html 文件，添加如下内容：

```
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
     <html>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
     <head>
     <title>自动化测试平台</title>
     </head>
     <body>
    <ul class="nav navbar-nav navbar-right">
     <li>欢迎您，<a href="#">{{user}}</a></li>
     <li><a href="/logout/">退出</a></li>
     </ul>
     </body>
    </html>
```

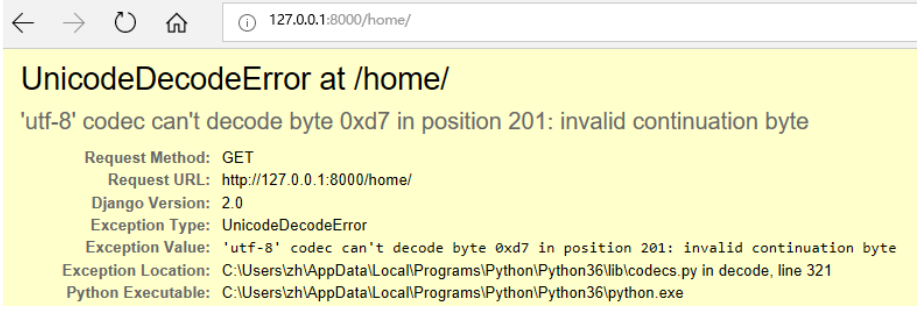
在 apitest/views.py 中加入以下内容：

```
def home(request):
     return render(request,"home.html")
```

在 autotest/urls.py 中加入以下内容：

```
urlpatterns = [
     path('admin/', admin.site.urls),
     path('test/', views.test),
     path('login/', views.login),
     path('home/', views.home),  #加入
 ]
```

在登录页面，再次输入正确的用户名（admin）和密码（test123456），单击“submit”按钮，如出现如图 3.6 所示的错误，可按下述方法解决。

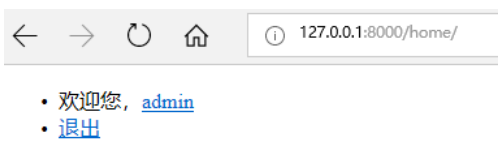


▲图 3.6

步骤 5 根据关键错误日志信息定位问题，排查错误，通过把 home.html 修改另存为 UTF-8 编码格式，即可解决上面的问题。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
 <html>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 <head>
 <title>自动化测试平台</title>
 </head>
 <body>
 </ul>
 <ul class="nav navbar-nav navbar-right">
 <li>欢迎您，<a href="#">{{user}}</a></li>
 <li><a href="/logout/">退出</a></li>
 </ul>
 </body>
 </noframes>
 </html>
以及 apitest/views.py 函数，
def home(request):
     return render(request,"home.html")

以及 autotest/urls.py
urlpatterns = [
     path('admin/', admin.site.urls),
     path('test/', views.test),
     path('login/', views.login),
     path('home/', views.home),  #加入
 ]
```

步骤 6 再次登录后，如图 3.7 所示，登录功能已经成功实现。



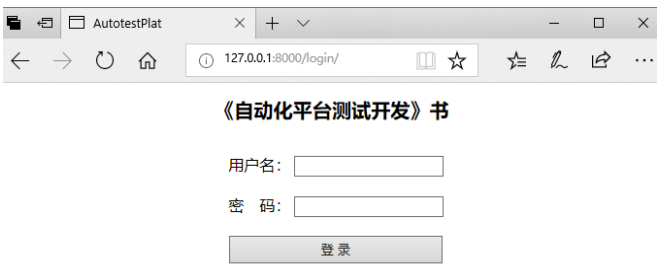
▲图 3.7

步骤 7 优化页面，把 login.html 修改为如下内容。

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<head>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>AutotestPlat</title>
```

```
<style> body{text-align:center} </style>
</head>
<body>
<div>
<font size="1">
<h1>《自动化平台测试开发》书</h1>
</font>
<form method="post" action="/login/">
{% csrf_token %}
<br><a>  用户名: </a> <input name="username" type="text" placeholder="test" >
<br><br>  <a>  密    码: </a> <input name="password" type="password"
placeholder="test123456">
<br>{{ error }}<br>
  <button style="width:220px;height:28px;" id="submit" type="submit">登 录</button>
</form>
</div>
</body>
</html>
```

查看登录页面，如图 3.8 所示。



▲图 3.8

3.3 退出功能实现

使用 Django 内置用户认证退出函数 logout。

步骤 1 在 apitest/views.py 中加入如下代码。

```
def logout(request):
    auth.logout(request)
    return render(request,'login.html')
```

步骤 2 在 autotest/urls.py 中加入如下代码。

```
path('logout/', views.logout),
```

步骤 3 在 apitest/home.html 中加入如下代码。

```
<li><a href="/logout/">退出</a></li>
```

步骤 4 在浏览器中输入地址，输入用户名和密码登录，单击“退出”按钮，可以看到已返回至首页。

3.4 权限功能操作

在创建数据库后，使用命令 python manage.py createsuperuser，创建超级用户。

创建成功后，在 Django admin 后台，用超级用户登录，可以增加用户、组，以及对权限进行控制等。

步骤 1 在浏览器中输入 http://127.0.0.1:8000/admin，输入用户名（admin）和密码（test123456）后登录。单击“用户”→“添加用户”，如图 3.9 所示，输入用户名和密码单击“保存”按钮。

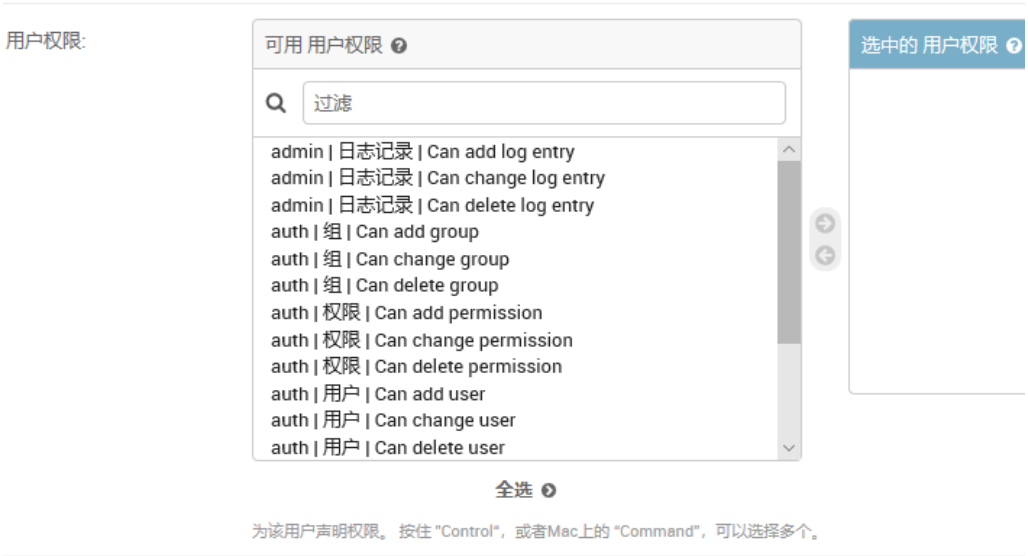
单击刚创建的 test 用户，密码为 test123456。

如图 3.10 所示，往下拉会看到“权限”，勾选“职员状态”复选框，再往下选择用户权限中的模块权限，单击箭头按钮，会显示在图 3.11 右边的文本框中。

▲图 3.9

▲图 3.10

如图 3.11 所示，单击保存。即完成了创建新用户，以及设置新用户的访问操作权限。



▲图 3.11

3.5 产品管理模块开发

3.5.1 产品管理数据库设计

产品管理数据库设计如表 3.3 所示。

▼表 3.3

字段名称	数据类型	长　度	可　空	备　注
id	int	11	False	主键，产品编号
Productname	varchar	64	False	产品名称
Productdescription	varchar	200	False	产品描述
Producter	varchar	200	False	产品负责人
Create_time	datetime	6	False	创建时间

3.5.2 产品管理功能后台开发

步骤 1 创建新的应用 python manage.py startapp product。

步骤 2 根据数据库设计生成 Django admin 后台功能，在 product / admin.py 中加入如下代码。

```
from django. contrib import admin
from product.models import  Product

class ProductAdmin(admin.ModelAdmin):
    list_display = ['productname', 'productdesc', 'producter','create_time','id']

admin.site.register(Product)  # 把产品模块注册到 Django admin 后台并能显示
```

步骤 3 在 product/models.py 中加入如下代码。

```
from django.db import models
class Product(models.Model):
    productname = models.CharField('产品名称',max_length=64)  # 产品名称
```

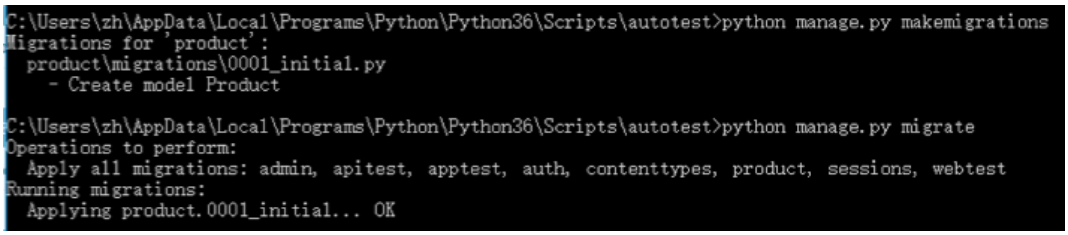
```
    productdesc = models.CharField('产品描述',max_length=200)     # 产品描述
    producter = models.CharField('产品负责人',max_length=200)     # 产品负责人
    create_time = models.DateTimeField('创建时间',auto_now=True)   # 创建时间，自动获取# 当前时间
    class Meta:
        verbose_name = '产品管理'
        verbose_name_plural = '产品管理'
    def __str__(self):
        return self.productname
```

步骤 4 在 autotest/settins.py 中加入应用。

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'product',
]
```

步骤 5 同步数据库，如图 3.12 所示。运行 CMD，输入命令：

```
Python manage.py makemigrations
Python manage.py migrate
```



▲图 3.12

步骤 6 在浏览器中输入 127.0.0.1:8000/admin，登录后，添加产品数据，然后点击产品管理，如图 3.13 所示。

▲图 3.13

3.5.3 产品管理功能前端开发

安装 Bootstrap4 前端框架

步骤 1 安装目前最新版本 Bootstrap4，下载地址为 https://pypi.python.org/pypi/django- bootstrap3，链接到 https://github.com/zostera/django-bootstrap4 进行下载。

单击“clone or download”按钮保存文件。

运行 CMD，切换到所存放目录 C:\Users\zh\AppData\Local\Programs\Python\Python36\ Scripts\django-bootstrap4-

master。运行 python setup.py install，安装成功。

步骤 2 在 autotest/settings.py 文件中加入如下内容。

```
INSTALLED APPS = (
     #..
    'bootstrap4',
    #..
)
```

步骤 3 建立模板。在 product 文件夹下建立 templates 文件夹，然后新建文件 product_manage. html。

```
<html lang="zh-CN">
<head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
<title>产品自动化测试平台</title>

</head>
<body role="document">
<!-- 导航栏-->
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
<div class="container">
<a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">
</ul>
<ul class="nav justify-content-end">
<li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
<li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
</ul>
</div>
</nav>

<!-- 产品列表-->
<div class="row" style="padding-top: 20px;">
<div class="col-md-11">
<table class="table table-striped">
<thead>
<tr><td> </td></tr>
<tr>
<th>ID</th><th>产品名称</th><th>产品描述</th><th>产品负责人</th><th>创建时间</th>
</tr>
</thead>
<tbody>
{% for product in products %}
<tr>
<td>{{ product.id }}</td>
<td>{{ product.productname }}</td>
<td>{{ product.productdesc }}</td>
<td>{{ product.producter }}</td>
<td>{{ product.create_time }}</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>

</body>
</html>
```

步骤 4 视图：在 product/views.py 中将文件名修改为 proviews.py，并加入以下内容。

```
from django. shortcuts import render
from product.models import Product

#产品管理
def product_manage(request):
    username = request.session.get('user', '')
    product_list = Product.objects.all()
    return render(request, "product_manage.html", {"user": username,"products": product_list})
```

步骤 5 映射：在 autotest/urls.py 中加入如下内容。

```
from product import proviews

path('product_manage/', proviews.product_manage),
```

步骤 6 在浏览器中输入地址 http://127.0.0.1:8000/product_manage/，登录后查看前端显示数据页面效果，如图 3.14 所示。

ID	产品名称	产品描述	产品负责人	创建时间
2	商城	图书销售	邹辉	2018年1月29日 13:09
3	app产品	计算器，Csdn	邹辉	2018年1月29日 13:16
10	自动化平台	包括API、WebUI、AppUI自动化测试	邹辉	2018年1月30日 10:33
11	web产品	百度搜索	邹辉	2018年1月29日 13:09

▲图 3.14

3.6 接口自动化模块开发

3.6.1 接口管理数据库设计

业务场景接口用例表如表 3.4 所示，业务场景步骤接口如表 3.5 所示，单一接口表如表 3.6 所示。

▼表 3.4

字段名称	数据类型	长　度	可　空	备　注
id	int	11	False	主键，用例编号
apitestname	varchar	64	False	用例名称
apitestdesc	varchar	64	False	用例描述
apitester	varchar	16	False	测试负责人
apitestresult	tinyint	1	False	测试结果
created_time	datetime	6	False	创建时间
Product_id	int	11	False	关联产品 ID

▼表 3.5

字段名称	数据类型	长　度	可　空	备　注
id	Int	11	False	主键，步骤接口编号
apiname	varchar	100	False	用例名称
apiurl	varchar	200	False	接口 URL
apiparamvalue	varchar	800	False	接口参数和值
apimethod	varchar	200	False	接口方法

apiresult	varchar	200	False	接口预期结果
apiresponse	varchar	5000	True	响应数据
apistatus	tinyint	1	False	测试结果
created_time	datetime	6	False	创建时间
Apitest_id	int	11	False	关联业务流程接口
Apistep	varchar	100	True	步骤

▼表 3.6

字段名称	数据类型	长　度	可　空	备　注
id	int	11	False	主键，单一接口编号
apiname	varchar	100	False	用例名称
apiurl	varchar	200	False	接口 URL
apiparamvalue	varchar	800	False	接口参数和值

续表

字段名称	数据类型	长　度	可　空	备　注
apimethod	varchar	200	False	接口方法
apiresult	varchar	200	False	接口预期结果
apiresponse	varchar	5000	True	响应数据
apistatus	tinyint	1	False	测试结果
created_time	datetime	6	False	创建时间
Product_id	int	11	False	关联产品 ID

3.6.2 流程接口管理功能后台开发

步骤 1 根据数据库表结构设计创建数据模型。

在 apitest/models.py 文件中加入流程接口表和单一接口表。

```
from product.models import Product

from django.db import models

class Apitest(models.Model):
    Product = models.ForeignKey('product.Product',on_delete=models.CASCADE,null=True)
# 关联产品 ID，其中 product 是应用名，Product 是 product 应用的表名
    apitestname = models.CharField('流程接口名称',max_length=64)  # 流程接口测试场景
     apitestdesc = models.CharField('描述',max_length=64,null=True)  # 流程接口描述
apitester = models.CharField('测试负责人',max_length=16)              # 执行人
    apitestresult = models.BooleanField('测试结果')                    # 流程接口测试结果
    create_time = models.DateTimeField('创建时间',auto_now=True)  # 创建时间，自动获取# 当前时间
    class Meta:
        verbose_name = '流程场景接口'
        verbose_name_plural = '流程场景接口'
    def __str__(self):
        return self.apitestname


 class Apistep(models.Model):
     Apitest = models.ForeignKey(Apitest)                                  # 关联接口 ID
     apiname = models.CharField('接口名称',max_length=100)                # 接口标题
     apiurl = models.CharField('url 地址',max_length=200)                  # 地址
     apistep = models.CharField('测试步聚',max_length=100,null=True)          # 测试步聚
    apiparamvalue = models.CharField('请求参数和值',max_length=800)     # 参数和值
```

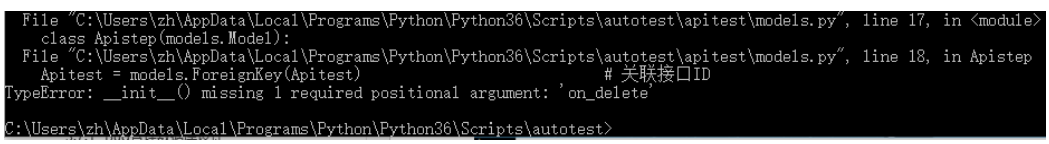
```
REQUEST_METHOD = (('get', 'get'),('post', 'post'),('put', 'put'),('delete', 'delete'),('patch',
'patch'))
    apimethod = models.CharField(verbose_name='请求方法',choices =
REQUEST_METHOD,default='get',max_length=200,null=True)  # 请求方法
     apiresult = models.CharField('预期结果',max_length=200)          # 预期结果
     apistep = models.CharField('测试步骤',max_length=100,null=True)          # 测试步骤
     apistatus = models.BooleanField('是否通过')                          # 测试结果
     create_time = models.DateTimeField('创建时间',auto_now=True)    # 创建时间，自动获# 取当前时间
def __str__(self):
    return self.name
```

步骤 2 在 apitest/admin.py 文件中加入如下内容。

```
from django.contrib import admin
 from apitest.models import Apitest, Apistep

 # Register your models here.

 class ApistepAdmin(admin.TabularInline):
     list_display =
['apiname','apiurl','apiparamvalue','apimethod','apiresult','apistatus','create_time','id','apitest']
     model = Apistep
     extra=1

 class ApitestAdmin(admin.ModelAdmin):
     list_display = ['apitestname', 'apitester','apitestresult','create_time','id']
     inlines = [ApistepAdmin]

 admin.site.register(Apitest,ApitestAdmin)
```

步骤 3 迁移同步数据库，如图 3.15 所示。

```
Python manage.py makemigrations
```



▲图 3.15

出现如上错误时，定位排查问题，将 apitest/models.py 修改为 `Apitest = models.ForeignKey('Apitest',on_delete=models.CASCADE,)`，再次输入 python manage.py makemigrations，如图 3.16 所示。

▲图 3.16

再次输入 `Python manage.py migrate`，如图 3.17 所示。

```
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, apitest, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying apitest.0001_initial... OK
  Applying apitest.0002_auto_20171215_1802... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK

C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>_
```

▲图 3.17

步骤 4 通过 Navcait 客户端工具，查看生成后的数据库和表结构，如图 3.18 和图 3.19 所示。

Table: apitest

apitest_apitest @autotest (autotest) - 表

文件 编辑 窗口 帮助

新建 保存 另存为 添加栏位 插入栏位 删除栏位 主键 上移 下移

栏位 索引 外键 触发器 选项 注释 SQL 预览

名	类型	长度	小数点	允许空值 (	
id	int	11	0	☐	1
apitestname	varchar	64	0	☐	
apitester	varchar	16	0	☐	
apitestresult	tinyint	1	0	☐	
create_time	datetime	6	0	☐	
Product_id	int	11	0	☑	
apitestdesc	varchar	64	0	☑	

▲图 3.18

Table: apistep

apitest_apistep @autotest (autotest) - 表

文件 编辑 窗口 帮助

新建 保存 另存为 添加栏位 插入栏位 删除栏位 主键 上移 下移

栏位 索引 外键 触发器 选项 注释 SQL 预览

名	类型	长度	小数点	允许空值 (	
id	int	11	0	☐	1
apiname	varchar	100	0	☐	
apiurl	varchar	200	0	☐	
apiparamvalue	varchar	800	0	☐	
apimethod	varchar	200	0	☑	
apiresult	varchar	200	0	☐	
apistatus	tinyint	1	0	☐	
create_time	datetime	6	0	☐	
Apitest_id	int	11	0	☐	
apistep	varchar	100	0	☑	
apiresponse	varchar	5000	0	☑	

▲图 3.19

步骤 5 在浏览器中输入 127.0.0.1:8000/admin，登录 admin 账号，如图 3.20 所示。

在 Apitests 中单击“增加”按钮，如图 3.21 所示。

单击“保存”按钮，出现如图 3.22 所示的错误。

▲图 3.20

▲图 3.21

▲图 3.22

根据关键错误日志，定位到问题后，把 apitest/models.py 内容改为如下内容。

```
from django.db import models

 # Create your models here.

 class Apitest(models.Model):
     apitestname = models.CharField('流程接口名称',max_length=64)   # 流程接口，测试场景
     apitester = models.CharField('执行人',max_length=16)              # 执行人
     apitestresult = models.BooleanField('测试结果')                   # 流程接口测试结果
     create_time = models.DateTimeField('创建时间',auto_now=True)   # 创建时间，自动获取# 当前时间

     def __str__(self):
         return self.apitestname

 class Apistep(models.Model):
     Apitest = models.ForeignKey('Apitest',on_delete=models.CASCADE,)                                    # 关联
接口 ID
apistep = models.CharField('测试步骤',max_length=100,null=True)          # 测试步骤
     apiname = models.CharField('接口名称',max_length=100)           # 接口名称描述
     apiurl = models.CharField('url 地址',max_length=200)               # 地址
     apiparamvalue = models.CharField('参数和值',max_length=800)     # 参数和值
     apimethod = models.CharField('方法',max_length=200)               # 方法
     apiresult = models.CharField('预期结果',max_length=200)           # 预期结果
     apistatus = models.BooleanField('是否通过')                       # 测试结果
     create_time = models.DateTimeField('创建时间',auto_now=True)    # 创建时间，自动获# 取当前时间
     def __str__(self):
         return self.apiname
```

再次添加接口，单击保存，如图 3.23 所示。

▲图 3.23

单击“流程接口名称”查看，如图 3.24 所示。

在 admin 后台添加“业务接口流程”以及“单一接口”数据的功能已成功实现。

▲图 3.24

3.6.3 流程接口展示功能前端开发

步骤 1 创建模板：在 templates/目录下创建 apitest_manage.html 和 apistep_manage.html 两个文件。

```
apitest_manage.html:
    <html lang="zh-CN">
     <head>
     <title>自动化测试平台</title>
     </head>
     <body role="document">
     <!-- 导航栏-->
     <nav class="navbar navbar-inverse navbar-fixed-top">
     <div class="container">
     <div class="navbar-header">
     <a class="navbar-brand" href="#">自动化测试平台</a>
     </div>
     <div id="navbar" class="collapse navbar-collapse">
     <ul class="nav navbar-nav">
     <li class="active"><a href="#">流程接口测试</a></li>
     <li><a href="#">流程接口测试步骤</a></li>
     </ul>
     <ul class="nav navbar-nav navbar-right">
     <li><a href="#">{{user}}</a></li>
     <li><a href="/logout/">退出</a></li>
     </ul>
     </div>
     </div>
     </nav>
     <!-- 流程接口列表-->
     <div class="row" style="padding-top: 20px;">
     <div class="col-md-11">
     <table class="table table-striped">
     <thead>

    <tr>
```

```
    <th>ID</th><th>产品</th><th>流程接口测试用例名称</th><th>流程接口描述</th><th>测试负责人</th><th>测试结果
</th><th>测试用例步骤</th>
    </tr>
    </thead>
    <tbody>
    {% for apitest in apitests %}
    <tr>
    <td>{{ apitest.id }}</td>
    <td>{{ apitest.Product.productname }}</td>
    <td>{{ apitest.apitestname }}</td>
    <td>{{ apitest.apitestdesc }}</td>
    <td>{{ apitest.apitester }}</td>
    <td>{{ apitest.apitestresult }}</td>
    <td>{{ apitest.Apistep.apiteststep }}</td>
    </tr>
    {% endfor %}
     </tbody>
     </table>
     </div>
     </div>
     </body>
 </html>
```

步骤 2 apistep_manage.html：

```
     <html lang="zh-CN">
     <head>
     <title>自动化测试平台</title>
     </head>
     <body role="document">
     <!-- 导航栏-->
     <nav class="navbar navbar-inverse navbar-fixed-top">
     <div class="container">
     <div class="navbar-header">
     <a class="navbar-brand" href="#">自动化测试平台</a>
     </div>
     <div id="navbar" class="collapse navbar-collapse">
     <ul class="nav navbar-nav">
     <li><a href="/apitest_manage/">流程接口测试</a></li>
     <li class="active"><a href="/apistep_manage/">流程接口测试步骤</a></li>
     </ul>
     <ul class="nav navbar-nav navbar-right">
     <li><a href="#">{{user}}</a></li>
     <li><a href="/logout/">退出</a></li>
     </ul>
     </div>
     </div>
     </nav>


    <!-- 流程接口测试步骤-->
    <div class="row" style="padding-top: 20px;">
    <div class="col-md-11">
    <table class="table table-striped">
    <thead>

    <tr>
    <th>所属产品</th><th>所属用例</th><th>步骤</th><th>URL 地址</th><th>参数=值</th><th>方法</th><th>预期结果
</th><th>测试结果</th><th>执行时间</th>
    </tr>
    </thead>
```

```
    <tbody>
    {% for apistep in apisteps %}
    <tr>
    <td>{{ apistep.Apitest.Product.productname }}</td>
    <td>case{{ apistep.Apitest.id }}:{{ apistep.Apitest.apitestname }}</td>
    <td>{{ apistep.apistep }}:{{ apistep.apiname }}</td>
    <td>{{ apistep.apiurl }}</td>
    <td>{{ apistep.apiparamvalue }}</td>
    <td>{{ apistep.apimethod }}</td>
    <td>{{ apistep.apiresult }}</td>
    <td>{% if apistep.apistatus == 1 %}
    <a style='color:green'>{{ apistep.apistatus }}</a>
    {% else %}
    <a style='color:red'>{{ apistep.apistatus }}</a>
    {% endif %}
    </td>
    <td>{{ apistep.create time }}</td>
    </tr>
    {% endfor %}
    </tbody>
    </table>
    </div>
    </div>
     </body>
</html>
```

步骤 3 创建视图：在 apitest/views.py 文件中加入如下两个函数。

```
# 接口管理
 @login_required
 def apitest_manage(request):
     apitest_list = Apitest.objects.all() #读取所有流程接口数据
     username = request.session.get('user', '') # 读取浏览器登录 Session
     return render(request,"apitest_manage.html",{"user": username,"apitests":apitest_list})#定义流程接口数据的变量并返回到前端


 # 接口步骤管理
 @login_required
 def apistep_manage(request):
     username = request.session.get('user', '')
     apistep_list = Apistep.objects.all()
     return render(request, "apistep_manage.html", {"user": username,"apisteps": apistep_list})
```

步骤 4 在 autotest/urls.py 中加入如下内容。

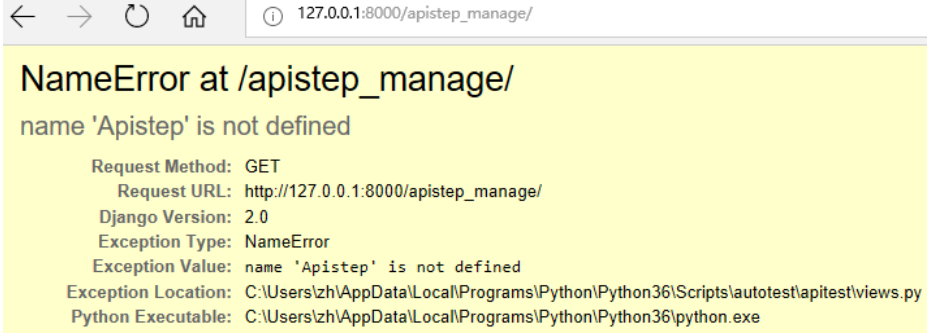
```
    path('apitest_manage/', views.apitest_manage),
     path('apistep_manage/', views.apistep_manage),
```

views 和 template 在上面已经写好了。

步骤 5 登录后在浏览器中输入 127.0.0.1:8000/apistep_manage，如图 3.25 所示。

在 apitest/views.py 中加入：`from apitest.models import Apitest,Apistep`。

在浏览器中再次输入 127.0.0.1:8000/apistep_manage，如图 3.26 所示。



▲图 3.25

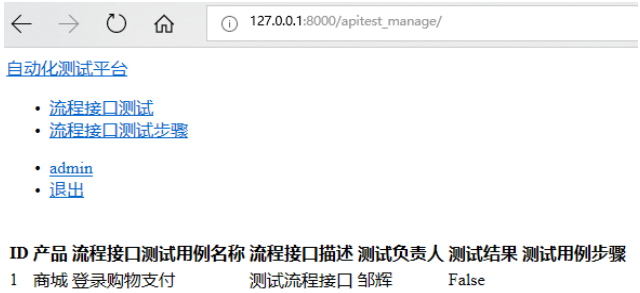
▲图 3.26

发现接口数据并未显示出来，再从 apitest/templates 目录中找到 apistep_manage.html，把{% for apistep in apistep %} 改成{% for apistep in apisteps %}。

再次刷新页面，发现数据显示出来了，如图 3.27 所示。

▲图 3.27

在浏览器中输入 127.0.0.1:8000/apitest_manage，如图 3.28 所示。



▲图 3.28

至此，前端展示“接口流程”和“单一接口步骤”数据的功能已经实现。

3.6.4 单一接口管理功能后台开发

步骤 1 创建数据模型，根据数据库表结构设计。

在 apitest/models.py 文件中加入流程接口表和单一接口表。

```
from django.db import models
from product.models import Product
class Apis(models.Model):
    Product = models.ForeignKey('product.Product',on_delete=models.CASCADE,null=True)        # 关联产品
ID
    apiname = models.CharField('接口名称',max_length=100)          # 接口标题
    apiurl = models.CharField('url 地址',max_length=200)              # 地址
    apiparamvalue = models.CharField('请求参数和值',max_length=800)     # 请求参数和值
    REQUEST_METHOD = (('0', 'get'),('1', 'post'),('2', 'put'),('3', 'delete'),('4', 'patch'))
    apimethod = models.CharField(verbose_name='请求方法',choices =
REQUEST_METHOD,default='0',max_length=200)  # 请求方法
    apiresult = models.CharField('预期结果',max_length=200)          # 预期结果
    apistatus = models.BooleanField('是否通过')                        # 测试结果
    create_time = models.DateTimeField('创建时间',auto_now=True)    # 创建时间，自动获取# 当前时间
    class Meta:
        verbose_name = '单一场景接口'
        verbose_name_plural = '单一场景接口'
    def __str__(self):
        return self.apiname
```

在 apitest/admin.py 文件中加入：

```
from django.contrib import admin
 from apitest.models import Apitest, Apistep , Apis
 from product.models import Product
 # Register your models here.

class ApisAdmin(admin.TabularInline):
    list display =
['apiname','apiurl','apiparamvalue','apimethod','apiresult','apistatus','create_time','id','product']

admin.site.register(Apis)
```

步骤 2 加入产品管理字段。

在 product/admin.py 中加入：

```
from apitest.models import Apitest,Apis

class ApisAdmin(admin.TabularInline):
    list_display =
['apiname','apiurl','apiparamvalue','apimethod','apiresult','apistatus','create time','id','product']
    model = Apis
    extra = 1

class ProductAdmin(admin.ModelAdmin):
    list_display = ['productname', 'productdesc','create_time','id']
    inlines = [ApisAdmin]
```

步骤 3 迁移同步数据库。

运行 CMD，切换到相应目录，输入 Python manage.py makemigrations，按 Enter 键。

再次输入 Python manage.py migrate，如图 3.29 所示。

```
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py makemigrations
Migrations for 'apitest':
  apitest\migrations\0003_apis.py
    - Create model Apis

C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, apitest, auth, contenttypes, sessions
Running migrations:
  Applying apitest.0003_apis... OK
```

▲图 3.29

步骤 4 在浏览器中输入 127.0.0.1:8000/admin，登录 admin 账号，再添加单一接口，单击保存，如图 3.30 所示。单击查看，如图 3.31 所示。

▲图 3.30

▲图 3.31

在 admin 后台添加“业务接口流程”和“单一接口”数据的功能已成功实现。

3.6.5 单一接口展示功能前端开发

步骤 1 创建模板：在 templates/ 目录下创建 apis_manage.html 文件。

```
apis_manage.html:
     <html lang="zh-CN">
     <head>
     <title>产品自动化测试管理平台</title>
     </head>
     <body role="document">
     <!-- 导航栏-->
     <nav class="navbar navbar-inverse navbar-fixed-top">
     <div class="container">
     <div class="navbar-header">
     <a class="navbar-brand" href="#">产品自动化测试管理平台</a>
     </div>
     <div id="navbar" class="collapse navbar-collapse">
     <ul class="nav navbar-nav">
     <li><a href="/apis_manage/">单一接口测试</a></li>
     </ul>
     <ul class="nav navbar-nav navbar-right">
     <li><a href="#">{{user}}</a></li>
     <li><a href="/logout/">退出</a></li>
     </ul>
     </div>
     </div>
     </nav>
     <!—单一接口测试用例-->
     <div class="row" style="padding-top: 20px;">
     <div class="col-md-11">
     <table class="table table-striped">
     <thead>
     <tr>
     <th>ID</th><th>产品</th><th>单一接口测试用例名称</th><th>URL 地址</th><th>参数=值</th><th>方法</th><th>
预期结果</th><th>测试结果</th><th>时间</th>
     </tr>
     </thead>
     <tbody>
     {% for apis in apiss %}
     <tr>
     <td>{{ apis.id }}</td>
    <td>{{ apis.Product.productname }}</td>
     <td>{{ apis.apiname }}</td>
     <td>{{ apis.apiurl }}</td>
     <td>{{ apis.apiparamvalue }}</td>
     <td>{{ apis.apimethod }}</td>
     <td>{{ apis.apiresult }}</td>
     <td>{{ apis.apistatus }}</td>
     <td>{{ apis.create_time }}</td>
     </tr>
     {% endfor %}
     </tbody>
     </table>
     </div>
     </div>
     </body>
 </html>
```

步骤 2 创建视图：在 apitest/views.py 文件中加入如下函数。

```
# 单一接口管理
```

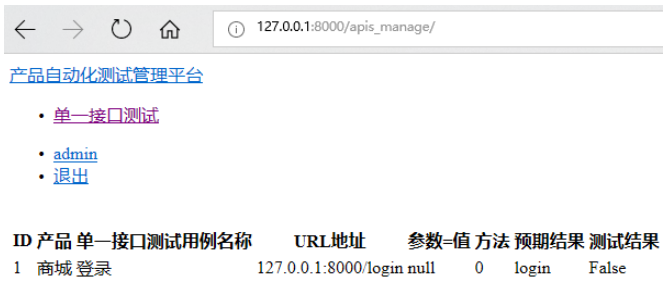
```
from apitest.models import Apitest,Apistep,Apis

 @login_required
 def apis_manage(request):
     username = request.session.get('user', '')
     apis_list = Apis.objects.all()
     return render(request, "apis_manage.html", {"user": username,"apiss": apis_list})
```

步骤 3 在 autotest/urls.py 中加入如下内容。

```
path('apis_manage/', views.apis_manage),
```

views 和 template 在上面已经写好了。

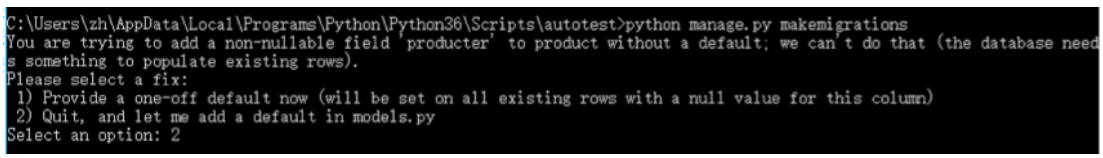
步骤 4 在浏览器中输入 127.0.0.1:8000/apis_manage，如图 3.32 所示。



▲图 3.32

至此，前端展示“单一接口管理”数据的功能已经实现。

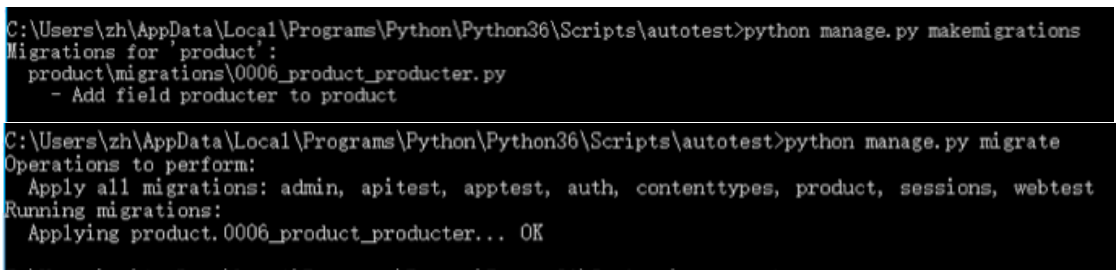
如果碰到如图 3.33 所示的错误提示，则根据关键错误日志定位问题，当新增、修改、删除数据库字段时，要注意允许为空。



▲图 3.33

在 models.py 中新增：producter = models.CharField('产品负责人',max_ length=200,null=True) # 产品负责人

在相应目录下运行 CMD，输入命令 python manage.py makemigrations，python manage.py migrate，如图 3.34 所示。



▲图 3.34

3.7 Bug 管理模块开发

3.7.1 Bug 管理数据库设计

设置表 autotest_bug，如表 3.7 所示。

▼表 3.7

字段名称	数据类型	长　度	可　空	备　注
id	int	11	False	主键，Bug 编号
bugname	varchar	200	False	Bug 名称
bugdetail	varchar	2000	False	详情
bugstatus	varchar	200	False	解决状态
buglevel	varchar	200	False	严重程度
bugcreater	varchar	200	False	创建人
bugassign	varchar	200	False	分配给
create_time	datetime	6	False	创建时间
Product _id	int	11	False	关联产品 ID

3.7.2 Bug 管理后端开发

步骤 1 创建新的应用，在 CMD 运行，切换到相应目录，输入命令 python manage.py startapp bug。

步骤 2 根据数据库设计生成 Django admin 后台功能。

1）在 bugs/models.py 中加入如下内容。

```
from django. db import models
from product.models import Product
class Bug(models.Model):
    Product = models.ForeignKey('product.Product',on_delete=models.CASCADE, null=True)          # 关联产品
ID
    bugname = models.CharField('bug 名称',max_length=64)  # Bug 名称
    bugdetail = models.CharField('详情',max_length=200)# 详情
    BUG_STATUS = (('激活', '激活'),('已解决', '已解决'),('已关闭', '已关闭'))
    bugstatus = models.CharField(verbose_name='解决状态',choices = BUG_STATUS,default='激活
',max_length=200,null=True)# 解决状态
    BUG_LEVEL = (('1', '1'),('2', '2'),('3', '3'))
    buglevel = models.CharField(verbose_name='严重程度',choices =
BUG_LEVEL,default='3',max_length=200,null=True)# 严重程度
    bugcreater = models.CharField('创建人',max_length=200)# 创建人
    bugassign = models.CharField('分配给',max_length=200)# 分配给
    created_time = models.DateTimeField('创建时间',auto_now=True)# 更新时间值
    class Meta:
        verbose_name = 'bug 管理'
        verbose_name_plural = 'bug 管理'
    def __str__(self):
        return self.bugname
```

2）在 bug / admin.py 中加入如下内容。

```
from django.contrib import admin
from bug.models import Bug

class BugAdmin(admin.ModelAdmin):
```

```
    list_display = ['bugname ', 'bugdetail ', ' bugstatus', ' buglevel', ' bugcreater', ' bugassign',
'create_time','id']

admin.site.register(Bug)  # 把 Bug 管理模块注册到 Django admin 后台并能显示
```

步骤 3 在 autotest/settins.py 中加入应用。

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bug',
]
```

步骤 4 同步数据库。

```
Python manage.py makemigrations
Python manage.py migrate
```

3.7.3 Bug 管理前端开发

步骤 1 模板：在 Bug 文件夹下建立 templates 文件夹，然后新建文件 bug_manage.html，加入如下内容。

```
<html lang="zh-CN">
<head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
<title>产品自动化测试平台</title>

</head>
<body role="document">
<!-- 导航栏-->
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
<div class="container">
<a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">
</ul>
<ul class="nav justify-content-end">
<li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
<li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
</ul>
</div>
</nav>

<!—bug 列表-->
<div class="row" style="padding-top: 70px;">
<div class="col-md-11">
<table class="table table-striped">
<thead>

<tr>
<th>ID</th><th>产品</th><th>bug 名称</th><th>bug 详情</th><th>解决状态</th><th>严重程度</th><th>创建人
</th><th>分配给</th><th>创建时间</th>
</tr>
</thead>
<tbody>
{% for bug in bugs %}
```

```
<tr>
<td>{{ bug.id }}</td>
<td>{{ bug.Product.productname }}</td>
<td>{{ bug.bugname }}</td>
<td title="{{ bug.bugdetail }}">{{ bug.bugdetail }}</td>
<td>{{ bug.bugstatus }}</td>
<td>{{ bug.buglevel }}</td>
<td>{{ bug.bugcreater }}</td>
<td>{{ bug.bugassign  }}</td>
<td>{{ bug.created_time }}</td>
{% endfor %}
</tbody>
</table>
</div>
</div>

</body>
</html>
```

步骤 2 视图：在 bug / views.py 中将文件名修改为 bugviews.py，并加入以下内容。

```
from django.shortcuts import render
from bug.models import Bug

#bug 管理
def bug_manage(request):
    username = request.session.get('user', '')
    bug_list = Bug.objects.all()
    return render(request, "bug_manage.html", {"user": username,"bugs": bug_list})
```

步骤 3 映射：在 autotest/urls.py 中加入如下内容。

```
from bug import bugviews

path('bug_manage/', bugviews.bug_manage),
```

步骤 4 在浏览器中输入地址 127.0.0.1:8000/bug_manage，登录后查看前端显示数据页面效果，如图 3.35 所示。

▲图 3.35

3.8 系统设置模块开发

很多时候需要对一些公共的、重复的变量进行设置，开发一个模块；这样当这个变量的值变化时，仅需修改设置变量的值，其他则全部引用到这个公共变量，即可大大降低维护成本。

比如 URL 中 IP 地址或域名地址，设置的环境在 Request URL 中总是存在的，而这个环境可能是本地环境（Local）、开发环境（Develop）或测试环境（Test），所以需要设置该地址为公共参数，并设置对应值。

这里的实现过程，需结合第 6 章代码和第 4 章正则表达式的相关内容。

3.8.1 系统设置数据库设计

设置管理如表 3.8 所示。

▼表 3.8

字段名称	数据类型	长　度	可　空	备　注
id	int	11	False	主键，设置编号
setname	varchar	200	False	主机或 URL 地址等
setvalue	varchar	200	False	设置值

3.8.2 系统设置后台开发

步骤 1 创建新的应用 python manage.py startapp set。

步骤 2 根据数据库设计生成 Django admin 后台功能。

在 set/models.py 中加入：

```
class Set(models.Model):
    setname = models.CharField('系统名称',max_length=64)  # 设置名称
   setvalue = models.CharField('系统设置',max_length=200)     # 设置值
    class Meta:
        verbose_name = '系统设置'
        verbose_name_plural = '系统设置'
    def __str__(self):
        return self.setname
```

在 set / admin.py 中加入：

```
from set.models import  Set

class SetAdmin(admin.ModelAdmin):

    list_display = ['setname', 'setvalue','id']

admin.site.register(Set)  # 把系统设置模块注册到 Django admin 后台并显示
```

步骤 3 在 autotest/settins.py 中加入应用。

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'set',
]
```

步骤 4 同步数据库，如图 3.36 所示。

```
Python manage.py makemigrations
Python manage.py migrate
```

```
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py makemigrations
Migrations for 'product':
  product\migrations\0001_initial.py
    - Create model Product

C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, apitest, apptest, auth, contenttypes, product, sessions, webtest
Running migrations:
  Applying product.0001_initial... OK
```

▲图 3.36

步骤 5 在浏览器中输入 127.0.0.1:8000/admin，登录后添加数据并查看，如图 3.37 所示。

▲图 3.37

3.8.3 系统设置前端开发

步骤 1 模板：在 set 文件夹下建立 templates 文件夹，然后新建文件 set_manage.html，并加入以下内容。

```
<html lang="zh-CN">
<head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
<title>产品自动化测试平台</title>

</head>
<body role="document">
<!-- 导航栏-->
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
<div class="container">
<a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">
</ul>
<ul class="nav justify-content-end">
<li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
<li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
</ul>
</div>
</nav>

<!—设置列表-->
<div class="row" style="padding-top: 70px;">
<div class="col-md-11">
<table class="table table-striped">
<thead>
```

```
<tr>
<th>ID</th><th>设置名称</th><th>设置的值</th>
</tr>
</thead>
<tbody>
{% for set in sets %}
<tr>
<td>{{ set.id }}</td>
<td>{{ set.setname }}</td>
<td>{{ set.setvalue }}</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>

</body>
</html>
```

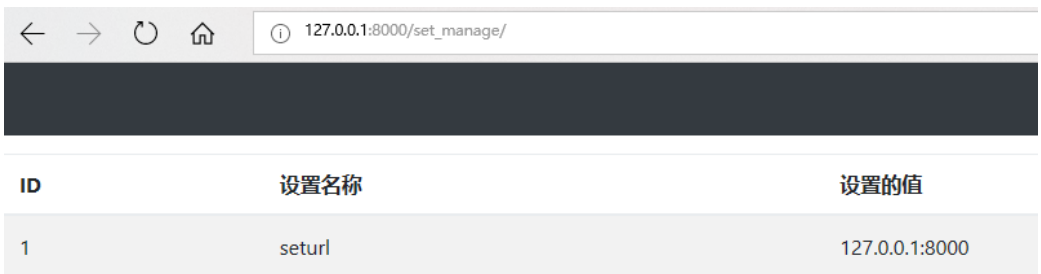
步骤 2 视图：在 set / views.py 中将文件名修改为 setviews.py，并加入以下内容。

```
from set.models import Set

#设置管理
def set_manage(request):
    username = request.session.get('user', '')
    set_list = Set.objects.all()
    return render(request, "set_manage.html", {"user": username,"sets": set_list})
```

步骤 3 映射：在 autotest/urls.py 中加入以下内容。

```
from set import setviews

path('set_manage/', setviews.set_manage),
```

步骤 4 在浏览器中输入地址 127.0.0.1:8000/set_manage，登录后查看前端显示数据，页面效果如图 3.38 所示。



▲图 3.38

步骤 5 为了方便，把用户管理的功能也加到系统设置中，Django 中自带模型 auth.models，可直接引用 User，在 set/setviews.py 中加入如下内容。

```
from django.contrib.auth.models import User

#用户管理
def set_user(request):
    user_list = User.objects.all()
    username = request.session.get('user','')
   return render(request, "set_user.html", {"user": username,"users": user_list})
```

步骤 6 在 set/templates 中新建 set_user.html 文件，加入如下内容。

```
<html lang="zh-CN">
<head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
<title>产品自动化测试管理平台</title>
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />

<script type="text/javascript" src="/admin/jsi18n/"></script>

<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>

<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>

<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>

<script type="text/javascript" src="/static/admin/js/urlify.js"></script>

<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>

<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>

<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">

<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />

<meta name="robots" content="NONE,NOARCHIVE" />

</head>
<body role="document">
<!-- 导航栏-->
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
<div class="container">
<a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">

</ul>

<ul class="nav justify-content-end">
<li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
<li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
</ul>
</div>
</nav>
<div class="row" style="padding-top: 55px;">
<div class="col-md-11">

<!-- 用户管理-->
<table class="table table-striped" style="table-layout: fixed;">
<style>
td {
      white-space:nowrap;overflow:hidden;text-overflow: ellipsis;
}
</style>
<thead>

<tr>
```

```
<th>ID</th><th>用户名</th><th>密码</th><th>邮箱</th><th>最近登录日期</th><th>注册日期</th>
</tr>
</thead>
<tbody>
{% for user in users %}
<tr>

<td>{{user.id}}</td>
<td>{{user.username}}</td>
<td>{{user.password}}</td>
<td>{{user.email}}</td>
<td>{{user.last_login}}</td>
<td>{{user.date_joined}}</td>

</tr>
{% endfor %}
</tbody>
</table>

</div>
</div>

</body>
</html>
```

步骤 7 在 autotest/urls.py 中加入 path('user/', setviews.set_user)。在前端浏览器中输入 127.0.0.1:8000/user，查看页面，如图 3.39 所示。

ID	用户名	密码	邮箱	最近登录日期	注册日期
1	admin	pbkdf2_sha...	7980068@...	2018年2月...	2017年12...
2	test	pbkdf2_sha...	7980068@...	2018年1月...	2017年12...

▲图 3.39

3.9 App 自动化模块开发

在接口自动化测试平台页面加入此功能，细节上大同小异，思路与接口管理功能开发完全相同，后期在进行 App 自动化测试的时候要结合 Appium 自动化工具使用。

3.9.1 App 用例管理数据库设计

用例管理如表 3.9 所示，App 用例步骤表如表 3.10 所示。

▼表 3.9

字段名称	数据类型	长　度	可　空	备　注
id	int	11	False	主键，App 用例编号
appcasename	varchar	200	False	用例名称
apptestresult	tinyint	1	False	测试结果

apptester	varchar	16	False	消息内容
create_time	datetime	6	False	创建时间
Product_id	Int	11	False	关联产品 ID

▼表 3.10

字段名称	数据类型	长　度	可　空	备　注
id	int	11	False	主键，App 用例步骤编号
appteststep	varchar	200	False	用例步骤
apptestobjname	varchar	200	False	操作对象名称
appfindmethod	varchar	200	False	定位方式
appevelement	varchar	800	True	元素控件
apptestdata	varchar	200	True	测试数据
appassertdata	varchar	200	True	校验数据
apptestresult		1	True	测试结果
create_time	datetime	6	False	创建时间
appcase_id	int	11	False	关联 App 用例 ID

3.9.2 App 用例管理功能后台开发

步骤 1 创建新的应用 python manage.py startapp apptest。

步骤 2 根据数据库表设计，在 apptest/models.py 中加入如下内容。

```
class Appcase(models.Model):
      Product = models.ForeignKey('product.Product',on_delete=models.CASCADE,null=True)
# 关联产品 ID
    appcasename = models.CharField('用例名称',max_length=200)        # 测试用例名称
     apptestresult = models.BooleanField('测试结果')                    # 测试结果
     apptester = models.CharField('测试负责人',max_length=16)          # 执行人
     create_time = models.DateTimeField('创建时间',auto_now=True)     # 创建时间，自动获# 取当前时间
    class Meta:
        verbose_name = 'app 测试用例'
        verbose_name_plural = 'app 测试用例'
     def __str__(self):
         return self.appcasename

class Appcasestep(models.Model):
     Appcase = models.ForeignKey(Appcase,on_delete=models.CASCADE)     # 关联接口 ID
     appteststep = models.CharField('测试步骤',max_length=200)         # 测试步骤
     apptestobjname = models.CharField('测试对象名称描述',max_length=200) # 测试对象名# 称描述
     appfindmethod = models.CharField('定位方式',max_length=200)       # 定位方式
     appevelement = models.CharField('控件元素',max_length=800)        # 控件元素
     appoptmethod = models.CharField('操作方法',max_length=200)        # 操作方法
     apptestdata = models.CharField('测试数据',max_length=200,null=True)       # 测试
# 数据，临时增加字段时要设置可为空
     appassertdata = models.CharField('验证数据',max_length=200)       # 验证数据
     apptestresult = models.BooleanField('测试结果')                   # 测试结果
     create_time = models.DateTimeField('创建时间',auto_now=True)      # 创建时间，自动# 获取当前时间
     def __str__(self):
         return self.apptestresult
```

步骤 3 在 apptest/admin.py 文件中加入如下内容。

```
from django.contrib import admin
from apptest.models import Appcase,Appcasestep
```

```
# Register your models here.
class AppcasestepAdmin(admin.TabularInline):
     list_display=['appteststep','apptestobjname','appfindmethod','appevelement','appoptmethod','appas
     sertdata',
'apptestresult','create_time','id','appcase']
     model = Appcasestep
     extra=1

 class AppcaseAdmin(admin.ModelAdmin):
     list_display = ['appcasename', 'apptestresult','create_time','id']
     inlines = [AppcasestepAdmin]
admin.site.register(Appcase,AppcaseAdmin)
```

步骤 4 在 autotest/settins.py 文件中加入如下内容。

```
INSTALLED_APPS = [
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'apitest',
        'bootstrap4',
        'apptest',    #加入
 ]
```

步骤 5 数据库迁移同步，分别输入 python manage.py makemigrations 和 python manage.py migrate，分别按“Enter”键，如图 3.40 所示。

```
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py makemigrations
Migrations for 'apptest':
  apptest\migrations\0001_initial.py
    - Create model Appcase
    - Create model Appcasestep

C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>Python manage.py migrate
Operations to perform:
  Apply all migrations: admin, apitest, apptest, auth, bug, contenttypes, product, sessions, set
Running migrations:
  Applying apptest.0001_initial... OK
```

▲图 3.40

步骤 6 在浏览器中输入 127.0.0.1:8000/admin，登录后查看后台界面，如图 3.41 所示。

步骤 7 增加 App 用例，填入 App 自动化用例内容，单击“保存”按钮，后台界面如图 3.42 所示。

▲图 3.41

▲图 3.42

至此，App 用例管理功能后台开发已经成功实现。

3.9.3 App 用例管理功能前端开发

步骤 1 在 product/admin.py 中加入如下内容。

```
from apptest.models import Appcase

class AppcaseAdmin(admin.TabularInline):
    list_display = ['appcasename', 'apptestresult','create_time','id','product']
    model = Appcase
    extra = 1

class ProductAdmin(admin.ModelAdmin):
    list_display = ['productname', 'productdesc','create_time','id']
    inlines = [AppcaseAdmin]
```

步骤 2 在 apptest/下新建 templates 文件夹，新建 appcase_manage.html 和 appcasestep_ manage.html 文件。

appcase_manage.html 内容如下。

```
<html lang="zh-CN">
 <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
 <title>产品自动化测试管理平台</title>
 </head>
 <body role="document">
 <!-- 导航栏-->
 <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
 <div class="container">
 <a class="navbar-brand" href="#"> </a>
 <ul class="nav justify-content-center">
 </ul>
 <ul class="nav justify-content-end">
 <li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
 <li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
 </ul>
 </div>
```

```
 </nav>
 <!-- app 测试用例列表-->
 <div class="row" style="padding-top: 10px;">
 <div class="col-md-11">
 <table class="table table-striped">
 <thead>
 <tr>
 <th>ID</th><th>产品</th><th>测试用例名称</th><th>测试结果</th><th>测试负责人</th><th>时间</th><th>测试用例步骤
</th>
 </tr>
 </thead>
 <tbody>
 {% for appcase in appcases %}
 <tr>
 <td>{{ appcase.id }}</td>
<td>{{ appcase.Product.productname }}</td>
 <td>{{ appcase.appcasename }}</td>
 <td>{{ appcase.apptestresult }}</td>
 <td>{{ appcase.apptester }}</td>
<td>{{ appcase.create_time }}</td>
 <td><a href="#">{{ appcase.appcasestep }}</a></td>
 <td>{{ appcase.appcasestep }}</td>
 </tr>
 {% endfor %}
 </tbody>
 </table>
 </div>
 </div>
 </body>
 </html>
```

步骤 3 appcasestep_manage.html 内容如下。

```
<html lang="zh-CN">
 <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
 <title>产品自动化测试平台</title>
 </head>
 <body role="document">
 <!-- 导航栏-->
 <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
 <div class="container">
 <a class="navbar-brand" href="#"> </a>
 <ul class="nav justify-content-center">
 </ul>
 <ul class="nav justify-content-end">
 <li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
 <li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
 </ul>
 </div>
 </nav>

<!-- 流程测试步骤-->
<div class="row" style="padding-top: 70px;">
<div class="col-md-11">
<table class="table table-striped">
<thead>
<tr>
```

```
<th>所属产品</th><th>所属用例</th><th>步骤</th><th>定位方式</th><th>控件元素</th><th>操作方法</th><th>测试数据
</th><th>验证数据</th><th>测试结果</th><th>时间</th>
</tr>
</thead>
<tbody>
{% for appcasestep in appcasesteps %}
<tr>
<td>{{ appcasestep.Appcase.Product.productname }}</td>
<td>case{{ appcasestep.Appcase.id }}:{{ appcasestep.Appcase.appcasename }}</td>
<td>{{ appcasestep.appteststep }}:{{ appcasestep.apptestobjname }}</td>
<td>{{ appcasestep.appfindmethod }}</td>
<td>{{ appcasestep.appevelement }}</td>
<td>{{ appcasestep.appoptmethod }}</td>
<td>{{ appcasestep.apptestdata }}</td>
<td>{{ appcasestep.appassertdata }}</td>

<td>{% if appcasestep.apptestresult == 1 %}
<a style='color:green'>{{ appcasestep.apptestresult }}</a>
{% else %}
<a style='color:red'>{{ appcasestep.apptestresult }}</a>
{% endif %}
</td>

<td>{{ appcasestep.create_time }}</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
 </body>
 </html>
```

步骤 4 在 apptest/views.py 中加入如下内容。

```
from django.shortcuts import render
 from django.http import HttpResponse,HttpResponseRedirect
 from django.contrib.auth.decorators import login_required
 from django.contrib import auth
 from django.contrib.auth import authenticate, login
 from apptest.models import Appcase,Appcasestep
 # Create your views here.

 # app 用例管理
 @login_required
 def appcase_manage(request):
     appcase_list = Appcase.objects.all()
     username = request.session.get('user', '') # 读取浏览器登录 Session
     return render(request,"appcase_manage.html",{"user": username,"appcases":appcase_list})

 # App 用例测试步骤
 @login_required
 def appcasestep_manage(request):
     username = request.session.get('user', '')
     appcasestep_list = Appcasestep.objects.all()
     return render(request, "appcasestep_manage.html", {"user": username,"appcasesteps":
appcasestep_list})
```

步骤 5 在 autotest/urls.py 文件中加入如下内容。

```
 from django.contrib import admin
 from django.urls import path
```

```
from apitest import views
from apptest import views

urlpatterns = [
    path('appcase_manage/', views.appcase_manage),
    path('appcasestep_manage/', views.appcasestep_manage),
]
```

这时发现与 API 中的 views.py 冲突了，于是修改 apptest/views.py 文件名为 apptest/appviews.py，把 autotest/urls.py 中的内容改为如下内容：

```
from apptest import appviews
from django.contrib import admin
from django.urls import path
from apitest import views
from apptest import appviews

urlpatterns = [
    path('appcase_manage/', appviews.appcase_manage),
    path('appcasestep_manage/', appviews.appcasestep_manage),
]
```

步骤 6 在浏览器中输入 127.0.0.1:8000/appcasestep_manage，界面显示如图 3.43 所示。

▲图 3.43

步骤 7 在后台添加一个 App 自动化测试用例，前端登录后，单击 App 测试用例步骤，如图 3.44 所示。

产品	所属用例	步骤	定位方式	控件元素	操作方法	测试数据	验证数据	测试结果	时间
app产品	case_1_计算器计算1+1=2	第一步:输入1	find_element_by_name	1	click	null	null	True	2018年1月
app产品	case_1_计算器计算1+1=2	第二步:输入+	find_element_by_name	+	click	null	null	True	2018年1月
app产品	case_1_计算器计算1+1=2	第三步:输入1	find_element_by_name	1	click	null	null	True	2018年1月
app产品	case_1_计算器计算1+1=2	第四步:输入=	find_element_by_name	=	click	null	null	True	2018年1月

▲图 3.44

至此，App 自动化用例管理功能已经实现。

3.10 Web 自动化模块开发

为了便于开发，在接口自动化测试平台页面加入此功能，细节和思路上与 App 自动化几乎相同。由于自动化测试框架开源工具的使用有很大差别，以及 PC 端、手机端的使用也有很大不同，因此我们使用独立的模块，区别于接口自动化和 App 自动化。后期在进行 Web 自动化测试时要结合 PC 端的 Selenium 自动化框架一起使用。

3.10.1 Web 用例管理数据库设计

数据库设计可以和 App 自动化用例功能一样，把字段改成 Web 标识即可，表 3.11 和表 3.12 分别为 Web 用例步骤表。

▼表 3.11

字段名称	数据类型	长　度	可　空	备　注
id	int	11	False	主键，Web 用例编号
webcasename	varchar	64	False	用例名称
webtestresult	tinyint	1	False	测试结果
webtester	varchar	16	False	测试负责人
create_time	datetime	6	False	创建时间
Product_id	int	11	False	关联产品 ID

▼表 3.12

字段名称	数据类型	长　度	可　空	备　注
id	int	11	False	主键，Web 用例步骤编号
webcasename	varchar	200	False	用例名称

续表

字段名称	数据类型	长　度	可　空	备　注
webteststep	varchar	200	False	用例步骤
webtestobjname	varchar	200	False	操作对象名称
webfindmethod	varchar	200	True	定位方式
webevelement	varchar	800	True	元素控件
weboptmethod	varchar	200	True	操作方法
webtestdata	varchar	200	varchar	测试数据
webassertdata	varchar	200	True	校验数据
webtestresult	varchar	1	True	测试结果
create_time	datetime	6	False	创建时间
Webcase_id	int	11	False	关联 Web 用例

3.10.2 Web 用例管理功能后台开发

步骤 1 创建新的应用 python manage.py startapp webtest。

步骤 2 在 product/admin.py 中加入如下内容。

```
from webtest.models import Webcase

class WebcaseAdmin(admin.TabularInline):
    list_display = ['webcasename', 'webtestresult','create time','id','product']
    model = Webcase
    extra = 1

class ProductAdmin(admin.ModelAdmin):
    list_display = ['productname', 'productdesc','create_time','id']
    inlines = [WebcaseAdmin]
```

步骤 3 根据数据库表设计，在 webtest/models.py 中加入如下内容。

```
from product.models import Product

class Webcase(models.Model):
     Product = models.ForeignKey('product.Product',on_delete=models.CASCADE, null=True) # 关联产品 ID
    webcasename = models.CharField('用例名称',max_length=200)        # 测试用例名称
     webtestresult = models.BooleanField('测试结果')                    # 测试结果
     webtester = models.CharField('测试负责人',max_length=16)          # 执行人
```

```
    create_time = models.DateTimeField('创建时间',auto_now=True)     # 创建时间-自动获# 取当前时间
class Meta:
        verbose_name = 'web 测试用例'
        verbose_name_plural = 'web 测试用例'
    def __str__(self):
        return self.webcasename

class Webcasestep(models.Model):
    Webcase = models.ForeignKey(Webcase,on_delete=models.CASCADE)     # 关联接口 ID
    webcasename = models.CharField('测试用例标题',max_length=200)     # 测试用例标题
    webteststep = models.CharField('测试步骤',max_length=200)         # 测试步骤
    webtestobjname = models.CharField('测试对象名称描述',max_length=200) # 测试对象名# 称描述
    webfindmethod = models.CharField('定位方式',max_length=200)       # 定位方式
    webevelement = models.CharField('控件元素',max_length=800)        # 控件元素
    weboptmethod = models.CharField('操作方法',max_length=200)        # 操作方法
    webtestdata = models.CharField('测试数据',max_length=200,null=True)       # 测试
# 数据，临时增加字段时要设置可为空
    webassertdata = models.CharField('验证数据',max_length=200)       # 验证数据
    webtestresult = models.BooleanField('测试结果')                   # 测试结果
    create_time = models.DateTimeField('创建时间',auto_now=True)      # 创建时间-自动
# 获取当前时间
    def __str__(self):
            return self. webcasename
```

步骤 4 在 webtest/admin.py 文件中加入如下内容。

```
from django.contrib import admin
from webtest.models import Webcase,Webcasestep
 # Register your models here.
 class WebcasestepAdmin(admin.TabularInline):

    list_display=['webcasename','webteststep','webtestobjname','webfindmethod','webevelement','weboptm
    ethod','webassertdata',
   'webtestresult','create_time','id','webcase']
    model = Webcasestep
    extra=1

 classWebcaseAdmin(admin.ModelAdmin):
    list_display = ['webcasename', 'webtestresult','create_time','id']
    inlines = [WebcasestepAdmin]
 admin.site.register(Webcase,WebcaseAdmin)
```

步骤 5 在 autotest/settins.py 文件中加入如下内容。

```
INSTALLED_APPS = [
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'apitest',
        'bootstrap4',
        'apptest',    #加入
        'webtest',    #加入
 ]
```

步骤 6 数据库迁移同步，分别使用 python manage.py makemigrations 和 python manage.py migrate，如图 3.45 所示。

```
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py makemigrations
Migrations for 'webtest':
  webtest\migrations\0001_initial.py
    - Create model Webcase
    - Create model Webcasestep

C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>Python manage.py migrate
Operations to perform:
  Apply all migrations: admin, apitest, apptest, auth, bug, contenttypes, product, sessions, set, webtest
Running migrations:
  Applying webtest.0001_initial... OK
```

▲图 3.45

步骤 7 输入 127.0.0.1:8000/admin，登录后查看后台界面，如图 3.46 所示。

▲图 3.46

单击“添加”按钮，后台即可增加 Web 自动化用例，如图 3.47 所示。

▲图 3.47

至此，Web 用例管理功能后台开发已实现。

3.10.3 Web 用例管理功能前端开发

步骤 1 在 webtest/下新建 templates 文件夹，新建 webcase_manage.html 和 webcasestep_ manage.html 文件。webcase_manage.html 内容如下。

```
<html lang="zh-CN">
 <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
 <title>产品自动化测试管理平台</title>
 </head>
 <body role="document">
 <!-- 导航栏-->
 <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
```

```
<div class="container">
<a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">
</ul>
<ul class="nav justify-content-end">
<li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
<li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
</ul>
</div>
</nav>
<!-- web 测试用例列表-->
<div class="row" style="padding-top: 20px;">
<div class="col-md-11">
<table class="table table-striped">
<thead>
<tr>
<th>ID</th><th>产品</th><th>测试用例名称</th><th>测试结果</th><th>测试负责人</th><th>时间</th><th>测试用例步骤
</th>
</tr>
</thead>
<tbody>
{% for webcase in webcases %}
<tr>
<td>{{ webcase.id }}</td>
<td>{{ webcase.Product.productname }}</td>
<td>{{ webcase.webcasename }}</td>
<td>{{ webcase.webtestresult }}</td>
<td>{{ webcase.webtester }}</td>
<td>{{ webcase.create time }}</td>
<td><a href="#">{{ webcase.webcasestep }}</a></td>
<td>{{ webcase.webcasestep }}</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</body>
</html>
```

步骤 2 webcasestep_manage.html 内容如下。

```
<html lang="zh-CN">
<head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
<title>产品自动化测试平台</title>
</head>
<body role="document">
<!-- 导航栏-->
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
<div class="container">
<a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">
</ul>
<ul class="nav justify-content-end">
<li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
<li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
</ul>
</div>
```

```
 </nav>

 <!-- 用例步骤列表-->
 <div class="row" style="padding-top: 20px;">
 <div class="col-md-11">
 <table class="table table-striped">
 <thead>
 <tr>
 <th>所属产品</th><th>所属用例</th><th>步骤</th><th>定位方式</th><th>控件元素</th><th>操作方法</th><th>测试数据
</th><th>验证数据</th><th>测试结果</th><th>时间</th>
 </tr>
 </thead>
 <tbody>
 {% for webcasestep in webcasesteps %}
 <tr>
 <td>{{ webcasestep.Webcase.Product.productname }}</td>
 <td>case{{ webcasestep.Webcase.id }}:{{ webcasestep.Webcase.webcasename }}</td>
 <td>{{ webcasestep.webteststep }}:{{ webcasestep.webtestobjname }}</td>
 <td>{{ webcasestep.webfindmethod }}</td>
 <td>{{ webcasestep.webevelement }}</td>
 <td>{{ webcasestep.weboptmethod }}</td>
 <td>{{ webcasestep.webtestdata }}</td>
 <td>{{ webcasestep.webassertdata }}</td>
 <td>{{ webcasestep.webtestresult }}</td>
 <td>{{ webcasestep.create_time }}</td>
 </tr>
 {% endfor %}
 </tbody>
 </table>
 </div>
 </div>
 </body>
 </html>
```

步骤 3 在 webtest/views.py 中加入如下内容。

```
from django.shortcuts import render
from django.http import HttpResponse,HttpResponseRedirect
from django.contrib.auth.decorators import login_required
from django.contrib import auth
from django.contrib.auth import authenticate, login
from webtest.models import Webcase,Webcasestep
 # Create your views here.

 # Web 用例管理
@login_required
def webcase_manage(request):
     webcase_list = Webcase.objects.all()
     username = request.session.get('user', '') # 读取浏览器登录 Session
     return render(request,"webcase_manage.html",{"user": username,"webcases":webcase_list})

 # Web 用例测试步骤
@login_required
def webcasestep_manage(request):
   username = request.session.get('user', '')
   webcasestep_list = Webcasestep.objects.all()
   return render(request, "webcasestep_manage.html", {"user": username,"webcasesteps":
webcasestep_list})
```

步骤 4 在 autotest/urls.py 中加入如下内容。

```
 from django.contrib import admin
```

```
from django.urls import path
from apitest import views
from webtest import views

urlpatterns = [
    path('webcase_manage/', views.webcase_manage),
    path('webcasestep_manage/', views.webcasestep_manage),
]
```

这时发现与 API 中的 views.py 重名冲突了，于是把 webtest/views.py 文件名改为 webviews.py，把 autotest/urls.py 中的内容改为如下内容。

```
  from webtest import webviews
from django.contrib import admin
from django.urls import path
from apitest import views
from webtest import webviews

urlpatterns = [
    path('webcase_manage/', webviews.webcase_manage),
    path('webcasestep_manage/', webviews.webcasestep_manage),
]
```

步骤 5 在浏览器中输入 127.0.0.1:8000/webcasestep_manage/，如图 3.48 所示。

品	所属用例	步骤	定位方式	素	操作方法	测试数据	据	果
web产品	case1:百度搜索《自动化平台测试开发》	第一步:输入 自动化平台测试开发	find_element_by_id	kw	sendkeys	自动化平台测试开发	null	False
web产品	case1:百度搜索《自动化平台测试开发》	第二步:点击 搜索 按钮	find_element_by_id	su	click	null	null	False

▲图 3.48

至此，Web 自动化用例管理功能已经实现。

3.11 测试报告模块前端开发

3.11.1 单一接口测试报告

功能描述：显示统计接口总数、成功数、失败数和失败接口的详细信息。

步骤 1 在 report.html 文件中加入如下内容。

```
<div class="row" style="padding-top: 55px;">
<div class="col-md-11">

<table class="table table-striped">
<div style="position:absolute;  right:10px; width:10px; ">
<tr><th></th><th></th><th></th><th></th><th></th><th>单一接口扫描-自动化测试汇总报告</th></tr>
<tr><th>用例总数</th>
<td>{{ apiscounts }}</td><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th><
/th></tr> {# 前端读取定义的变量#}
```

```
<tr><th>通过
</th><td>{{ apis_pass_counts }}</td><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th
></th><th></th></tr>
<tr><th>不通过</th><td><a style='color:red'>{{ apis_fail_counts }}</a>
</td><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></tr>

</div>

</table>



<!-- 单一接口测试用例-->
<table class="table table-striped">
<thead>

<tr><th>失败接口列表</th><td>
<tr>
<th>ID</th><th>产品</th><th>测试用例名称</th><th>URL 地址</th><th>参数=值</th><th>方法</th><th>预期</th><th>
测试结果</th><th>执行时间</th>
</tr>
</thead>
<tbody>
{% for apis in apiss %}
<tr>
{% if apis.apistatus == 0 %}
<td>{{apis.id}}</td>
<td>{{ apis.Product.productname }}</td>
<td>{{ apis.apiname }}</td>
<td>{{ apis.apiurl }}</td>
<td>{{ apis.apiparamvalue }}</td>
<td>{{ apis.apimethod }}</td>
<td>{{ apis.apiresult }}</td>
<td>{% if apis.apistatus == 1 %}
<a style='color:green'>{{ apis.apistatus }}</a>
{% else %}
<a style='color:red'>{{ apis.apistatus }}</a>
{% endif %}
</td>
<td>{{ apis.create_time }}</td>

{% else %}

{% endif %}

</tr>
{% endfor %}
</tbody>
</table>

</div>
</div>
```

步骤 2 在 apitest/views.py 中加入函数。

```
# 测试报告
@login_required
def test_report(request):
    username = request.session.get('user', '')
    apis_list = Apis.objects.all()
    apis_count = Apis.objects.all().count()  #统计接口数
    db = pymysql.connect(user='root', db='autotest', passwd='test123456', host='127.0.0.1')
```

```
    cursor = db.cursor()
    sql1 = 'SELECT count(id) FROM apitest_apis WHERE apitest_apis.apistatus=1'
    aa=cursor.execute(sql1)
    apis_pass_count = [row[0] for row in cursor.fetchmany(aa)][0]
    sql2 = 'SELECT count(id) FROM apitest_apis WHERE apitest_apis.apistatus=0'
    bb=cursor.execute(sql2)
    apis_fail_count = [row[0] for row in cursor.fetchmany(bb)][0]
    db.close()
    return render(request, "report.html", {"user": username,"apiss": apis_list,"apiscounts":
apis_count,"apis_pass_counts": apis_pass_count,"apis_fail_counts": apis_fail_count}) #把值赋给
apiscounts 变量
```

加入 import HTMLTestRunner 组件，生成 HTML 式测试报告。

步骤 3 在 urls.py 中加入如下内容。

```
path ('test_report/', views.test_report),
```

3.11.2 流程接口测试报告

1．流程接口测试用例执行

流程接口指的是一个业务流程涉及多个接口，以及多个接口之间数据是有动态衔接关联的，比如登录后会有一个动态 token 关联到下一步购物和提交订单接口，提交后会产生一个动态 orderno，关联到下一个支付接口进行支付。

这时要用到正则表达式和参数化等知识。

功能描述：根据流程接口测试用例的 ID 即编号，查出要执行的用例，然后依次执行该用例的每一个接口步骤如果每个步骤都通过，则每个接口步骤记录通过，且该流程接口测试用例记录为通过。如果有一个或多个接口步骤失败，则相应记录失败，有步骤成功则相应记录成功，且该流程接口测试用例记录为失败。前端通过的以绿色显示 True，失败的以红色显示 False。

在 apitest_manage.html 和 apistep_mange.html 中加入如下内容。

```
<td>{% if apitest.apitestresult == 1 %}
<a style='color:green'>{{ apitest.apitestresult }}</a>
{% else %}
<a style='color:red'>{{ apitest.apitestresult }}</a>
{% endif %}
</td>


<td>{% if apistep.apistatus == 1 %}
<a style='color:green'>{{ apistep.apistatus }}</a>
{% else %}
<a style='color:red'>{{ apistep.apistatus }}</a>
{% endif %}
</td>
```

程序清单 1：apiauto_testcase3.py

```
# -*- coding:utf-8 -*-
import requests, time, sys, re
import urllib, zlib
import pymysql
from trace import CoverageResults
import json
```

```
from idlelib.rpc import response_queue
from time import sleep

HOSTNAME = '127.0.0.1'

def readSQLcase():              #读取数据库中相应的接口用例数据
    sql="SELECT id,`apiname`,apiurl,apimethod,apiparamvalue,apiresult,`apistatus` from apitest_apistep
where apitest_apistep.Apitest_id=3 "
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    for ii in info:
        case_list = []
        case_list.append(ii)
_____#   CredentialId()
        interfaceTest(case_list)
    coon.commit()
    cursor.close()
    coon.close()

def interfaceTest(case_list):
    res_flags = []
    request_urls = []
    responses = []
    strinfo = re.compile('{TaskId}')
    strinfo1 = re.compile('{AssetId}')
    strinfo2 = re.compile('{PointId}')
    assetinfo = re.compile('{assetno}')
    tasknoinfo = re.compile('{taskno}')
    schemainfo = re.compile('{schema}')
    for case in case_list:
        try:
            case_id = case[0]
            interface_name = case[1]
            method = case[3]
            url = case[2]
            param = case[4]
            res_check = case[5]
        except Exception as e:

            return '测试用例格式不正确%s'%e

        if param== '':
            new_url = 'http://'+'api.test.com.cn'+url
        elif param== 'null':
            new_url = 'http://'+url
        else:
            url = strinfo.sub(TaskId,url)
            param = strinfo.sub(TaskId,param)
            param = tasknoinfo.sub(taskno,param)
            new_url = 'http://'+'127.0.0.1'+url
            request_urls.append(new_url)
        if method.upper() == 'GET':
            headers = {'Authorization':'', 'Content-Type': 'application/json' }
            if "=" in urlParam(param):
                data = None
```

```
                print (str(case_id)+' request  is get' +new_url.encode('utf-
8')+'?'+urlParam(param).encode('utf-8'))
                results = requests.get(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is get'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (' request  is get ' +new url+'   body is '+urlParam(param))
                data = None
                req = urllib.request.Request(url=new_url,data=data,headers=headers,method="GET")
                results = urllib.request.urlopen(req).read()
                print ('   response is get ')
                print(results)
                res = readRes(results,res check)
            #print results
            if 'pass' == res:
                writeResult(case id,'1')
                res_flags.append('pass')
            else:
                res flags.append('fail')
                writeResult(case_id,'0')
        if  method.upper()=='PUT':
            headers = {'Host':HOSTNAME, 'Connection':'keep-alive',  'CredentialId':id, 'Content-Type':
'application/json'}
            body_data=param
            results = requests.put(url=url,data=body data,headers=headers)
            responses.append(results)
            res = readRes(results,res_check)
            if 'pass' == res:
                writeResult(case_id,'pass')
                res_flags.append('pass')
            else:
                res_flags.append('fail')
                writeResult(case_id,'fail')

writeBug(case_id,interface_name,new_url,results,res_check)

            try:
                preOrderSN(results)
            except:
                print ('ok')
        if method.upper()=="PATCH":
            headers = {'Authorization':'Credential '+id, 'Content-Type': 'application/json' }
            data = None
            results = requests.patch(new_url+'?'+urlParam(param),data,headers=headers).text
            responses.append(results)
            res = readRes(results,res_check)
            if 'pass' == res:
                writeResult(case_id,'pass')
                res_flags.append('pass')
            else:
                res_flags.append('fail')
                writeResult(case_id,'fail')
                writeBug(case_id,interface_name,new_url,results,res_check)
            try:
                preOrderSN(results)
            except:
                print ('ok')
        if method.upper()=="POST":
            headers = {'Authorization':'Credential '+id, 'Content-Type': 'application/json' }
```

```
            if "=" in urlParam(param):
                data = None
                results = requests.patch(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is post'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (str(case id)+' request  is ' +new url.encode('utf-8')+'   body is
'+urlParam(param).encode('utf-8'))
                results = requests.post(new_url,data=urlParam(param).encode('utf-
8'),headers=headers).text
                print ('   response is post'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,res check)
            if 'pass' == res:
                writeResult(case_id,'1')
                res flags.append('pass')
            else:
                res_flags.append('fail')
                writeResult(case id,'0')

writeBug(case_id,interface_name,new_url,results,res_check)

            try:
                TaskId(results)
            except:
                print ('ok1')



def readRes(res,res_check):
    res = res.decode().replace('":"',"=").replace('":',"=")
    res_check = res_check.split(';')
    for s in res_check:
        if s in res:
            pass
        else:
            return  '错误，返回参数和预期结果不一致'+s
    return 'pass'

def urlParam(param):
    param1=param.replace('&quot;','"')
    return param1

def CredentialId():
    global id
    url = 'http://'+'api.test.com.cn'+'/api/Security/Authentication/Signin/web'
    body_data= json.dumps({"Identity":'test',"Password":'test'})
    headers = { 'Connection':'keep-alive','Content-Type': 'application/json'}
    response = requests.post(url=url,data=body_data,headers=headers)
    data=response.text
    regx = '.*"CredentialId":"(.*)","Scene"'
    pm = re.search(regx, data)
    id = pm.group(1)


def preOrderSN(results):
    global preOrderSN
    regx = '.*"preOrderSN":"(.*)","toHome"'
    pm = re.search(regx, results)
```

```
    if pm:
        preOrderSN = pm.group(1).encode('utf-8')
        return preOrderSN
    return False

def TaskId(results):
    global TaskId
    regx = '.*"TaskId":(.*),"PlanId"'
    pm = re.search(regx, results)
    if pm:
        TaskId = pm.group(1).encode('utf-8')
        return TaskId
    return False

def taskno(param):
    global taskno
    a = int(time.time())
    taskno='task_'+str(a)
    return taskno

def writeResult(case_id,result):
    result = result.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apitest_apistep set apitest_apistep.apistatus=%s where
apitest_apistep.Apitest_id=%s;"
    param = (result,case id)
    print ('api autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
    coon.commit()
    cursor.close()
    coon.close()

 def writeBug(bug_id,interface_name,request,response,res_check):
    interface name = interface name.encode('utf-8')
    res_check = res_check.encode('utf-8')
    request = request.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    bugname = str(bug_id)+ '_' + interface_name.decode() + '_出错了'
    bugdetail = '[请求数据]<br />'+request.decode()+'<br/>'+'[预期结果]<br/>'+res_
check.decode()+'<br/>'+'<br/>'+'[响应数据]<br />'+'<br/>'+response.decode()
    print (bugdetail)
    sql = "INSERT INTO `bug_bug` ("\
    "`bugname`,`bugdetail`,`bugstatus`,`buglevel`, `bugcreater`,
`bugassign`,`created_time`,`Product_id`)  "\
    "VALUES ('%s','%s','1','1','邹辉', '邹辉', '%s',
'2');"%(bugname,pymysql.escape_string(bugdetail),now)
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql)
    coon.commit()
    cursor.close()
coon.close()

if __name__ == '__main__':
```

```
    readSQLcase()
    print ('Done!')
    time.sleep(1)
```

2．配置文件操作

文件清单 2：settings.conf。

```
[project]
schema=apitest

[user]
username=admin
password=test123456

[database]
host=127.0.0.1
db=autotest
user=root
passwd=test123456
```

程序清单 3: config.py

```
#encoding:utf-8
#name:mod_config.py

import ConfigParser
import os

def getConfig(section, key):
    config = ConfigParser.ConfigParser()
    path = os.path.split(os.path.realpath(__file__))[0] + '/settins.conf'
    config.read(path)
return config.get(section, key)
```

3．运行结果

控制台打印请求接口数据、响应数据、测试结果等，如图 3.49 所示。

▲图 3.49

加入 UnitTest 和 HTMLTestRunner，自动出测试报告文件。

程序清单：

```
# -*- coding:utf-8 -*-
import requests, time, sys, re
import urllib, zlib#,
import pymysql

import HTMLTestRunner
import unittest
from trace import CoverageResults
import json
from idlelib.rpc import response_queue
from time import sleep
```

```
#import fconfig

HOSTNAME = '127.0.0.1'

class ApiFlow(unittest.TestCase):
    """登录支付购物接口流程"""
    def setUp(self):
        time.sleep(1)

    def test_readSQLcase(self):
        sql="SELECT id,`apiname`,apiurl,apimethod,apiparamvalue,apiresult,`apistatus` from
apitest_apistep where apitest_apistep.Apitest_id=2 "
        coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
        cursor = coon.cursor()
        aa=cursor.execute(sql)
        info = cursor.fetchmany(aa)
        print (info)
        for ii in info:
            case_list = []
            case_list.append(ii)
_____#   CredentialId()
            interfaceTest(case_list)
        coon.commit()
        cursor.close()
        coon.close()

    def tearDown(self):
_________time.sleep(1)

def interfaceTest(case_list):
    res_flags = []
    request_urls = []
    responses = []
    strinfo = re.compile('{TaskId}')
    strinfo1 = re.compile('{AssetId}')
    strinfo2 = re.compile('{PointId}')
    assetinfo = re.compile('{assetno}')
    tasknoinfo = re.compile('{taskno}')
    schemainfo = re.compile('{schema}')
    for case in case_list:
        try:
            case_id = case[0]
            interface_name = case[1]
            method = case[3]
            url = case[2]
            param = case[4]
            res_check = case[5]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        if param== '':
            new_url = 'http://'+'api.test.com.cn'+url
        elif param== 'null':
            new_url = 'http://'+url
        else:
            url = strinfo.sub(TaskId,url)
            param = strinfo2.sub(PointId,param)
            param = strinfo.sub(TaskId,param)
```

```
            param = tasknoinfo.sub(taskno,param)
            new_url = 'http://'+'127.0.0.1'+url
            request_urls.append(new_url)
        if method.upper() == 'GET':
            headers = {'Authorization':'', 'Content-Type': 'application/json' }
            if "=" in urlParam(param):
                data = None
                print (str(case id)+' request  is get' +new url.encode('utf-
8')+'?'+urlParam(param).encode('utf-8'))
                results = requests.get(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is get'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (' request  is get ' +new_url+'   body is '+urlParam(param))
                data = None
                req = urllib.request.Request(url=new url,data=data,headers=headers,method="GET")
                results = urllib.request.urlopen(req).read()
                print ('   response is get ')
                print(results)
                res = readRes(results,res_check)

                if 'pass' == res:
                    res_flags.append('pass')
                    writeResult(case_id,'1')
                    caseWriteResult(case id,'1')
                else:
                    res_flags.append('fail')
                    writeResult(case id,'0')
                    caseWriteResult(case_id,'0')
        if  method.upper()=='PUT':
            headers = {'Host':HOSTNAME, 'Connection':'keep-alive',  'CredentialId':id, 'Content-Type':
'application/json'}
            body_data=param
            results = requests.put(url=url,data=body data,headers=headers)
            responses.append(results)
            res = readRes(results,res_check)
            if 'pass' == res:
                writeResult(case_id,'pass')
                res_flags.append('pass')
            else:
                res_flags.append('fail')
                writeResult(case_id,'fail')

writeBug(case_id,interface_name,new_url,results,res_check)

            try:
                preOrderSN(results)
            except:
                print ('ok')
        if method.upper()=="PATCH":
            headers = {'Authorization':'Credential '+id, 'Content-Type': 'application/json' }
            data = None
            results = requests.patch(new_url+'?'+urlParam(param),data,headers=headers).text
            responses.append(results)
            res = readRes(results,res_check)
            if 'pass' == res:
                writeResult(case_id,'pass')
                res_flags.append('pass')
            else:
```

```
                res_flags.append('fail')
                writeResult(case_id,'fail')

writeBug(case_id,interface_name,new_url,results,res_check)
            try:
                preOrderSN(results)
            except:
                print ('ok')
        if method.upper()=="POST":
            headers = {'Authorization':'Credential '+id, 'Content-Type': 'application/json' }
            if "=" in urlParam(param):
                data = None
                results = requests.patch(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is post'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (str(case_id)+' request  is ' +new_url.encode('utf-8')+'   body is
'+urlParam(param).encode('utf-8'))
                results = requests.post(new url,data=urlParam(param).encode('utf-
8'),headers=headers).text
                print ('   response is post'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,res_check)
            if 'pass' == res:
                writeResult(case id,'1')
                res_flags.append('pass')
            else:
                res flags.append('fail')
                writeResult(case_id,'0')

writeBug(case id,interface name,new url,results,res check)
            try:
                TaskId(results)
            except:
                print ('ok1')
            try:
                PointId(results)
            except:
                print ('ok2')


def readRes(res,res_check):
    res = res.decode().replace('":"',"=").replace('":',"=")
    res_check = res_check.split(';')
    for s in res_check:
        if s in res:
            pass
        else:
            return  '错误，返回参数和预期结果不一致'+s
    return 'pass'

def urlParam(param):
    param1=param.replace('&quot;','"')
    return param1

def CredentialId():
    global id
    url = 'http://'+'api.test.com.cn'+'/api/Security/Authentication/Signin/web'
    body_data= json.dumps({"Identity":'test',"Password":'test'})
```

```
    headers = { 'Connection':'keep-alive','Content-Type': 'application/json'}
    response = requests.post(url=url,data=body_data,headers=headers)
    data=response.text
    regx = '.*"CredentialId":"(.*)","Scene"'
    pm = re.search(regx, data)
    id = pm.group(1)


def preOrderSN(results):
    global preOrderSN
    regx = '.*"preOrderSN":"(.*)","toHome"'
    pm = re.search(regx, results)
    if pm:
        preOrderSN = pm.group(1).encode('utf-8')
        return preOrderSN
    return False

def TaskId(results):
    global TaskId
    regx = '.*"TaskId":(.*),"PlanId"'
    pm = re.search(regx, results)
    if pm:
        TaskId = pm.group(1).encode('utf-8')
        return TaskId
    return False

def taskno(param):
    global taskno
    a = int(time.time())
    taskno='task_'+str(a)
    return taskno

def writeResult(case_id,result):
    result = result.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apitest_apistep set apitest_apistep.apistatus=%s,apitest_apistep.create_time=%s
where apitest_apistep.id=%s;"
    param = (result,now,case id)
    print ('api autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
    coon.commit()
    cursor.close()
    coon.close()

def caseWriteResult(case_id,result):
    result = result.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apitest_apitest set apitest_apitest.apitestresult=%s,apitest_apitest.create_time=%s
where apitest_apitest.id=%s;"
    param = (result,now,case_id)
    print ('api autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
```

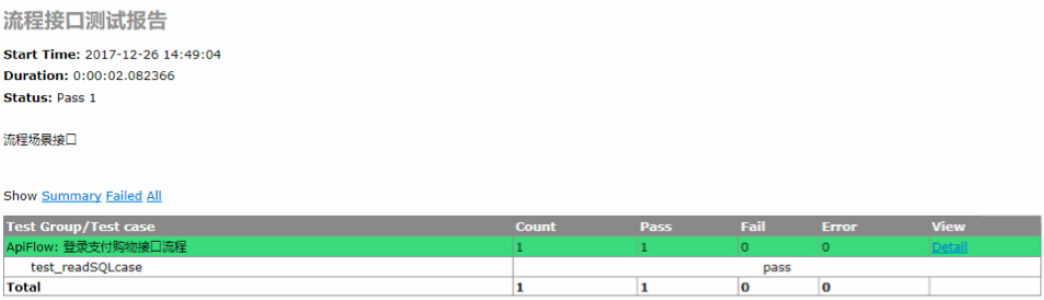
```
    coon.commit()
    cursor.close()
    coon.close()

def writeBug(bug_id,interface_name,request,response,res_check):
    interface_name = interface_name.encode('utf-8')
    res_check = res_check.encode('utf-8')
    request = request.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    bugname = str(bug_id)+ '_' + interface_name.decode() + '_出错了'
    bugdetail = '[请求数据]<br />'+request.decode()+'<br/>'+'[预期结
果]<br/>'+res_check.decode()+'<br/>'+'<br/>'+'[响应数据]<br />'+'<br/>'+response.decode()
    print (bugdetail)
    sql = "INSERT INTO `bug_bug` ("\
    "`bugname`,`bugdetail`,`bugstatus`,`buglevel`, `bugcreater`,
`bugassign`,`created_time`,`Product_id`)  "\
    "VALUES ('%s','%s','1','1','邹辉', '邹辉', '%s',
'2');"%(bugname,pymysql.escape_string(bugdetail),now)
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql)
    coon.commit()
    cursor.close()
    coon.close()

if __name__ == '__main__':
    now = time.strftime("%Y-%m-%d-%H_%M_%S", time.localtime(time.time()))
    testunit = unittest.TestSuite()
    testunit.addTest(ApiFlow("test readSQLcase"))

filename="C:\\Users\\zh\\AppData\\Local\\Programs\\Python\\Python36\\Scripts\\autotest\\apitest\\templ
ates\\"+"apitest report.html"
    fp=open(filename,'wb')
    runner = HTMLTestRunner.HTMLTestRunner(stream=fp, title=u"流程接口测试报告", description=u"流程场景接口
")
    runner.run(testunit)

___# readSQLcase()
    print ('Done!')
    time.sleep(1)
```

4．接口自动化测试汇总报告

每日定时执行，本书第 11 章中将会详细介绍。根据实例 2 运行单一接口自动化测试，单一接口测试报告如图 3.50 所示。

▲图 3.50

每日流程接口类测试执行，运行一个流程测试用例的报告，如图 3.51 所示。



▲图 3.51

3.11.3 AppUI 测试报告

1．App 自动化测试实例

步骤 1 准备。准备好待测的 App 包，如 csdn.apk，进行自动化测试模拟手机登录 CSDN。

csdn.apk 下载路径链接：https://pan.baibu.com/s/1MlAT4be64P_C5GUSxy_CMw。

步骤 2 安装 apk，运行 CMD，切换到相应目录，输入 adb intsll csdn.apk。

```
C:\Users\zh\Desktop\software\android-sdk-windows\platform-tools>adb install csdn.apk
csdn.apk: 1 file pushed. 1.0 MB/s (12642464 bytes in 12.653s)
        pkg: /data/local/tmp/csdn.apk
Success
```

要抓取到 apppackage 和 appActivity。

步骤 3 程序清单：appauto_testcase3.py。

```
#-*- coding: UTF-8 -*-
import os
import time
import unittest
from selenium import webdriver

PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)
global driver
```

```
class Login(unittest.TestCase):
    def setUp(self):
        desired_caps={}
        desired_caps['device'] = 'android'
        desired_caps['platformName']='Android'
        desired_caps['browserName']=''
        desired_caps['version']='4.4'
        desired caps['deviceName']='emulator-5554'
        desired_caps['app'] = PATH('c:\\Users\\zh\\Desktop\\software\\android-sdk-windows\\platform-
tools\\csdn.apk')
        self.driver=webdriver.Remote('http://127.0.0.1:4723/wd/hub',desired caps)

    def test_login(self):
        time.sleep(10)
        time.sleep(10)
        identity = self.driver.find_element_by_name('输入 CSDN 账号').send_keys("test")
        usn = self.driver.find_element_by_name('输入密码')
        usn.click()
        usn.send_keys("test2")
        self.driver.find_element_by_name('登录').click()

        time.sleep(10)

    def tearDown(self):
        self.driver.quit()

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

步骤 4 然后即可运行程序，查看测试效果。

步骤 5 修改到 Autotestplat，即把测试用例信息录入到自动化平台上，抓取元素，用例录入到 Autotestplat 平台录入相关测试用例、定位方式、元素控件 ID、操作方法、测试数据、预期结果、测试结果等。

程序清单：

```
#coding=utf-8
from appium import webdriver
import time
import os
import unittest

PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)
global driver

# -*- coding:utf-8 -*-
import requests,  pymysql, time, sys, re
import urllib,  zlib
from trace import CoverageResults
import json
from idlelib.rpc import response queue
from time import sleep
import mod_config

HOSTNAME = '127.0.0.1'

def readSQLcase():              #流程的相关接口
    sql="SELECT id,`appcasename`,appfindmethod,appevelement,appoptmethod,appassertdata,`apptestresult`
from apptest_appcasestep where apptest_appcasestep.Appcase_id=1 ORDER BY id ASC "
```

```
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host=mod_config.getConfig("dat
abase", "host"),charset='utf8')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    for ii in info:
        case_list = []
        case_list.append(ii)
        apptestcase(case_list)
    coon.commit()
    cursor.close()
    coon.close()

def apptestcase(case_list):
    for case in case_list:
        try:
            case_id = case[0]
            findmethod = case[2]
            evelement = case[3]
            optmethod = case[4]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        print (evelement)
        time.sleep(10)
        if optmethod== 'click' and findmethod=='find_element_by_id':
            driver.find_element_by_id(evelement).send_keys('wayto')
        elif optmethod== 'click' and findmethod=='find_element_by_name':
            driver.find_element_by_name(evelement).click()
        elif optmethod=='sendkey':
            driver.find_element_by_name(evelement).send_keys()

if __name__ == '__main__':
    desired_caps = {}
    desired_caps['platformName'] = 'Android'
    desired_caps['platformVersion'] = '4.4'
    desired_caps['deviceName'] = 'emulator-5554'
    desired_caps['appPackage'] = 'com.android.test
    desired_caps['appActivity'] = '.Test'
    desired_caps['app'] = PATH('E:\\release\\csdn.apk')

    time.sleep(1)
    driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired_caps)
    time.sleep(1)
    readSQLcase()
    driver.quit()
    print ('Done!')
    time.sleep(1)
```

步骤 6 运行程序，查看测试效果。

2．App 自动化测试汇总报告

步骤 1 在源码中加入 HTMLTestRunner 组件及应用。

```
#coding=utf-8
from appium import webdriver
import time
import os
import unittest
```

```
import HTMLTestRunner

PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)
global driver

# -*- coding:utf-8 -*-
import requests,  pymysql, time, sys, re
import urllib,  zlib
#import  mysql #import result
#from httplib import ResponseNotReady
from trace import CoverageResults
import json
#from test.test_ast import eval_results
from idlelib.rpc import response_queue
#from cPickle import dumps
from time import sleep
import mod_config

HOSTNAME = '127.0.0.1'

class Calculator(unittest.TestCase):
    def setUp(self):
        time.sleep(1)

    def test_readSQLcase(self):             #流程的相关接口
        sql="SELECT id,appfindmethod,appevelement,appoptmethod,appassertdata,`apptestresult` from
apptest_appcasestep where apptest_appcasestep.Appcase_id=1 ORDER BY id ASC "
        coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host=mod_config.getConfig("dat
abase", "host"),charset='utf8')
        cursor = coon.cursor()
        aa=cursor.execute(sql)
        info = cursor.fetchmany(aa)
        for ii in info:
            case_list = []
            case_list.append(ii)
            apptestcase(case_list)
        coon.commit()
        cursor.close()
        coon.close()

        def tearDown(self):
            self.driver.quit()

def apptestcase(case_list):
    for case in case_list:
        try:
            case_id = case[0]
            findmethod = case[1]
            evelement = case[2]
            optmethod = case[3]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        print (evelement)
        time.sleep(10)
        if optmethod== 'click' and findmethod=='find_element_by_id':
            driver.find_element_by_id(evelement).click()
            writeResult(case_id,'1')
```

```
        elif optmethod== 'click' and findmethod=='find_element_by_name':
            driver.find_element_by_name(evelement).click()
            writeResult(case_id,'1')
        elif optmethod=='sendkey':
            driver.find_element_by_id(evelement).send_keys('testdata')
            writeResult(case_id,'1')

def writeResult(case_id,result):
    result = result.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apptest appcasestep set
apptest_appcasestep.apptestresult=%s,apptest_appcasestep.create_time=%s where
apptest_appcasestep.id=%s;"
    param = (result,now,case id)
    print ('app autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
    coon.commit()
    cursor.close()
    coon.close()

def caseWriteResult(case_id,result):
    result = result.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apptest_appcase set apptest_apptest.apptestresult=%s,apptest_apptest.create_time=%s
where apptest apptest.id=%s;"
    param = (result,now,case_id)
    print ('app autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
    coon.commit()
    cursor.close()
    coon.close()

if   name   == '  main  ':
    desired_caps = {}
    desired_caps['platformName'] = 'Android'
    desired_caps['platformVersion'] = '4.4'
    desired_caps['deviceName'] = 'emulator-5554'
    desired_caps['appPackage'] = 'com.android.calculator2'
    desired_caps['appActivity'] = '.Calculator'
    time.sleep(1)
    driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired_caps)
    time.sleep(1)

    now = time.strftime("%Y-%m-%d-%H_%M_%S", time.localtime(time.time()))
    testunit = unittest.TestSuite()
    testunit.addTest(Calculator("test_readSQLcase"))

filename="C:\\Users\\zh\\AppData\\Local\\Programs\\Python\\Python36\\Scripts\\autotest\\apptest\\templ
ates\\"+"apptest_report.html"
    fp=open(filename,'wb')
    runner = HTMLTestRunner.HTMLTestRunner(stream=fp, title=u"app 自动化测试汇总报告", description=u"app 自
动化测试")
```

```
    runner.run(testunit)

    driver.quit()

    print ('Done!')
time.sleep(1)
```

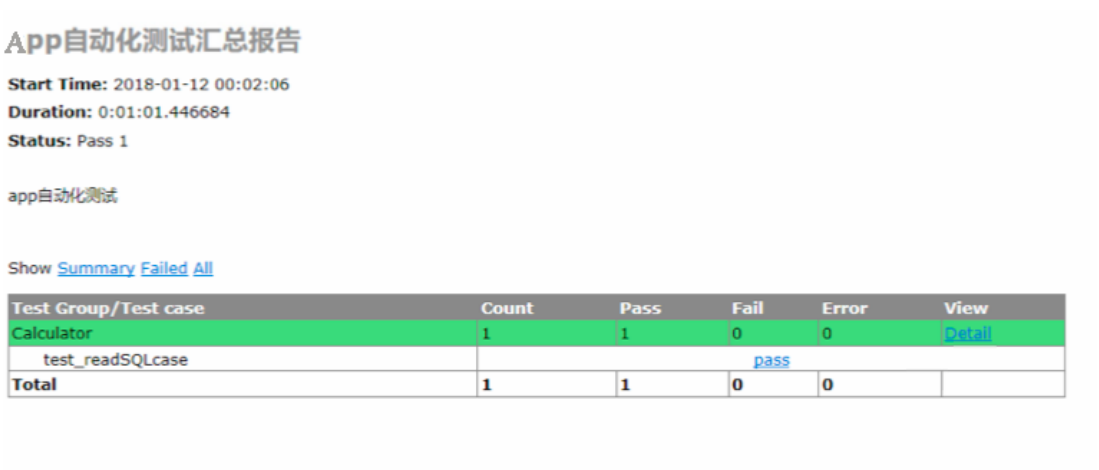
步骤 2 在 apptest/appviews.py 中加入如下内容。

```
# App 测试报告
@login_required
def apptest_report(request):
    username = request.session.get('user', '')
    return render(request, "apptest_report.html")
```

步骤 3 在 autotest/urls.py 中加入如下内容。

```
  path('apptest_report/', appviews.apptest_report),
```

步骤 4 运行程序清单，计算器 1+1= 自动化测试用例，查看 App 测试报告，如图 3.52 所示。



App自动化测试汇总报告

Start Time: 2018-01-12 00:02:06

Duration: 0:01:01.446684

Status: Pass 1

app自动化测试

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
Calculator	1	1	0	0	Detail
test_readSQLcase			pass		
Total	**1**	**1**	**0**	**0**	

▲图 3.52

把 apptest/templates/appcasestep_manage.html 中的

```
<td>{{ appcasestep.apptestresult }}</td>
```

修改成：

```
<td>{% if appcasestep.apptestresult == 1 %}
<a style='color:green'>{{ appcasestep.apptestresult }}</a>
{% else %}
<a style='color:red'>{{ appcasestep.apptestresult }}</a>
{% endif %}
</td>
```

查看运行的测试用例，如图 3.53 所示。

对于在源码中加入多个测试用例执行后的测试报告，和执行一个测试用例的逻辑类似，增加即可。

▲图 3.53

3.11.4 WebUI 测试报告

1．实例

使用 Autotestplat 平台，录入两个或批量的用例，在脚本中关联用例。

1）执行第 1 个用例，启动火狐浏览器，打开 www.baidu.com，输入：“自动化平台测试开发”，单击“搜索”按钮，退出火狐浏览器。

2）执行第 2 个用例，启动火狐浏览器，打开 www.baidu.com，输入：“软件自动化测试开发”，单击“搜索”按钮，退出火狐浏览器。

3）输出两个测试用例执行的测试报告。

程序清单：webauto_testcase3.py。

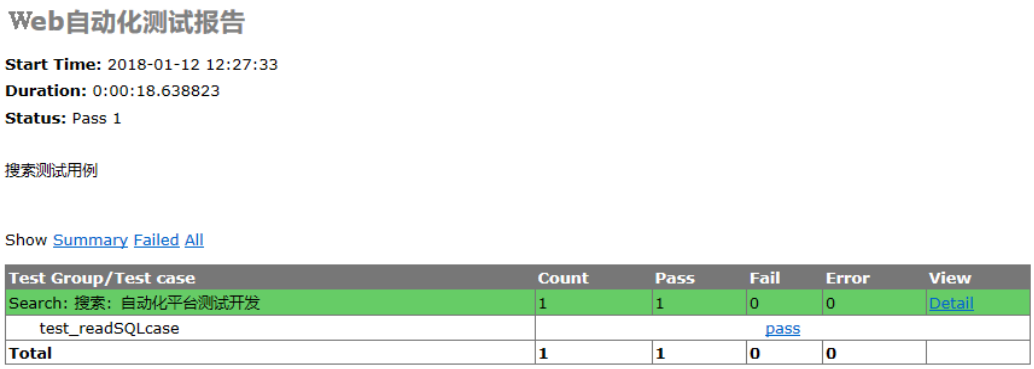
```
#-*- coding: UTF-8 -*-
import os
import time
import unittest
import pymysql
import fconfig
from selenium import webdriver
import HTMLTestRunner

PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)
global driver

HOSTNAME = '127.0.0.1'

class Search(unittest.TestCase):
    """搜索：自动化平台测试开发，软件自动化测试开发"""
    def setUp(self):
```

```
        self.driver =webdriver.Firefox()
        self.driver.get("http://www.baidu.com")
        time.sleep(1)

    def test_readSQLcase1(self):            #流程的相关接口
        sql="SELECT
id,webfindmethod,webevelement,weboptmethod,webtestdata,webassertdata,`webtestresult` from
webtest_webcasestep where webtest_webcasestep.Webcase_id=1 ORDER BY id ASC "
        coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host=fconfig.getConfig("databa
se", "host"),charset='utf8')
        cursor = coon.cursor()
        aa=cursor.execute(sql)
        info = cursor.fetchmany(aa)
        for ii in info:
            case_list = []
            case_list.append(ii)
            webtestcase(case_list,self)
        coon.commit()
        cursor.close()
        coon.close()

    def test_readSQLcase2(self):            #流程的相关接口
        sql="SELECT
id,webfindmethod,webevelement,weboptmethod,webtestdata,webassertdata,`webtestresult` from
webtest_webcasestep where webtest_webcasestep.Webcase_id=2 ORDER BY id ASC "
        coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host=fconfig.getConfig("databa
se", "host"),charset='utf8')
        cursor = coon.cursor()
        aa=cursor.execute(sql)
        info = cursor.fetchmany(aa)
        for ii in info:
            case_list = []
            case_list.append(ii)
            webtestcase(case_list,self)
        coon.commit()
        cursor.close()
        coon.close()

    def tearDown(self):
        self.driver.quit()

def webtestcase(case_list,self):
    for case in case_list:
        try:
            case_id = case[0]
            findmethod = case[1]
            evelement = case[2]
            optmethod = case[3]
            testdata = case[4]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        print (case)
        time.sleep(5)
        if optmethod=='sendkeys' and findmethod=='find_element_by_id':
            self.driver.find_element_by_id(evelement).send_keys(testdata)
        elif optmethod=='click' and findmethod=='find_element_by_name':
            print (evelement)
            self.driver.find_element_by_name(evelement).click()
```

```
        elif optmethod=='click' and findmethod=='find_element_by_id':
            print (evelement)
            self.driver.find_element_by_id(evelement).click()

if __name__ == '__main__':
    time.sleep(1)
    now = time.strftime("%Y-%m-%d-%H_%M_%S", time.localtime(time.time()))
    testunit = unittest.TestSuite()
    testunit.addTest(Search("test_readSQLcase1"))
    testunit.addTest(Search("test_readSQLcase2"))

filename="C:\\Users\\zh\\AppData\\Local\\Programs\\Python\\Python36\\Scripts\\autotest\\webtest\\templ
ates\\"+"webtest_report.html"
    fp=open(filename,'wb')
    runner = HTMLTestRunner.HTMLTestRunner(stream=fp, title=u"web 自动化测试报告", description=u"搜索测试用
例")
    runner.run(testunit)
    print ('Done!')
    time.sleep(1)
```

2．Web 自动化测试报告

运行查看 Web 自动化测试报告-1 个用例，如图 3.54 所示。



Web自动化测试报告

Start Time: 2018-01-12 12:27:33
Duration: 0:00:18.638823
Status: Pass 1

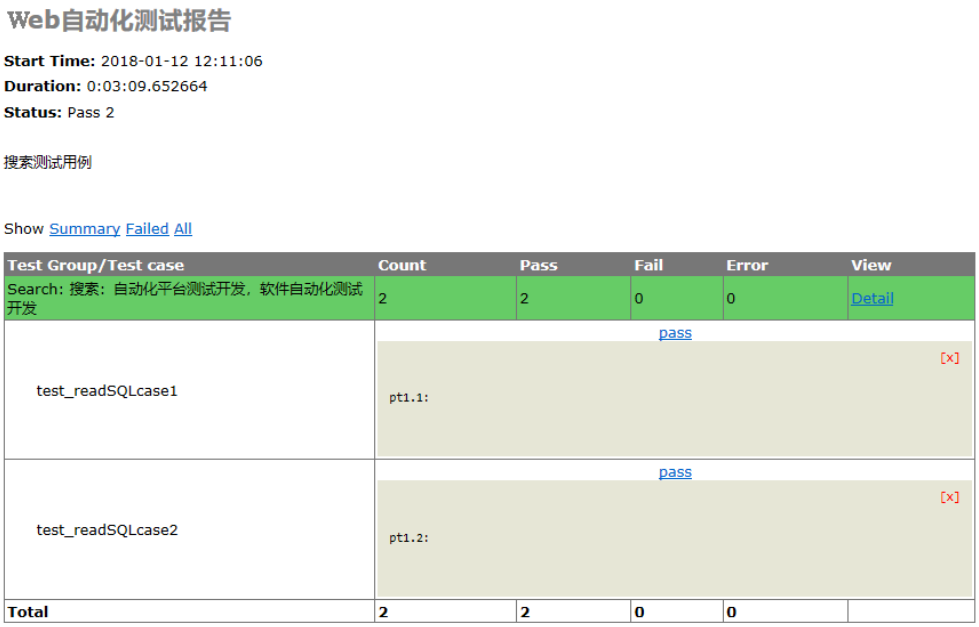
搜索测试用例

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
Search: 搜索：自动化平台测试开发	1	1	0	0	Detail
test_readSQLcase	pass				
Total	**1**	**1**	**0**	**0**	

▲图 3.54

运行查看 Web 自动化测试报告-2 个或批量，如图 3.55 所示。



Web自动化测试报告

Start Time: 2018-01-12 12:11:06
Duration: 0:03:09.652664
Status: Pass 2

搜索测试用例

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
Search：搜索：自动化平台测试开发，软件自动化测试开发	2	2	0	0	Detail
test_readSQLcase1	pass [x] pt1.1:				
test_readSQLcase2	pass [x] pt1.2:				
Total	**2**	**2**	**0**	**0**	

▲图 3.55

链接在 AutotestPlat 平台的测试报告，如图 3.56 所示。

《自动化平台测试开发》书

产品中心
bug管理
用例管理
计划任务
测试报告
系统设置

Web自动化测试报告

Start Time: 2018-01-12 12:27:33
Duration: 0:00:18.638823
Status: Pass 1

搜索测试用例

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
Search：搜索：自动化平台测试开发	1	1	0	0	Detail
test_readSQLcase	pass				
Total	**1**	**1**	**0**	**0**	

▲图 3.56

3.12 自动化平台前端优化

前端是直接与用户交互的地方，因此一个优雅美观的界面显得尤为重要。在软件开发过程中，前端界面往往变动比较大，也比较频繁，因此需要定期优化，以提升用户体验满意度。

3.12.1 HTML 简要知识

HTML 是超文本标记语言，用来描述 Web 网页。其中，HTML 标签由尖括号组成，成对地出现在 HTML 网页中，常用的约有 10 多个标签属性，如表 3.13 所示。

▼表 3.13

<html></html>	HTML 标签，用来标记网页
<head></head>	标记网页头
<title></title>	标记网页标题
<nav></nav>	标记导航
<body></body>	标记可见内容和网页主体
<table></table>	标记表格
<tr></tr>	标记多行
<td></td>	标记多列

续表

<th></th>	标记字体
<td>{{ apitest }}</td>	标记变量值
	标记列表项目的标识
<hr></hr>	标记水平线
<img></img>	标记图片
	标记链接
	标记标题
	标记段落
<form></form>	标记表单，包含输入框<input/> ，按钮，action，label 等
	标记空格
<!-- -->	标记注释

3.12.2 主页面优化 1

1．HTML

使用主框架（home.html）、frameset、左边功能菜单（left.html），形成左右两框的界面效果，右边内容是 apitest_manage.html 等。

步骤 1 将 apitest/home.html 修改为如下内容。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dhtd">
 <html>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 <head>
 <title>autotestplat-自动化平台测试开发</title>
 </head>
   <frameset id="frame1" rows="*" cols="265,*" framespacing="0" frameborder="yes" border="0">
     <frame src="../left" name="leftFrame" scrolling="auto" noresize>
     <frame src="../apitest_manage" name="mainFrame" scrolling="NO"  noresize>
   </frameset>
 <noframes>
 <body>
 </body>
 </noframes>
 </html>
```

步骤 2 在 apitest/templates 目录下新建文件 left.html，内容如下。

```
<html>
     <head>
         <title>test</title>
     </head>
     <body>
    <nav class="navbar navbar-inverse navbar-fixed-top"></nav>
         <table border="0">
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="../apitest_manage" target="mainFrame">  项目管理</a></td>
             </tr>
                <tr><td> </td></tr>
                <tr>
                 <td>-----------------------------------------------</a></td>
             </tr>
                 <tr><td> </td></tr>
                <tr>
                 <td><a href="../apitest_manage" target="mainFrame">  用例管理</a></td>
             </tr>
                <tr><td> </td></tr>

             <tr>
                 <td><a href="../apitest_manage" target="mainFrame">  api 接口测试用例列表
</a></td>
             </tr>
             <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  新增 api 接口测试用例</a></td>
             </tr>
                <tr><td> </td></tr>

                <tr>
                 <td>-----------------------------------------------</a></td>
             </tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  任务管理</a></td>
             </tr>
                <tr><td> </td></tr>
                <tr>
                 <td>-----------------------------------------------</a></td>
             </tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  测试报告</a></td>
             </tr>

         </table>
     </body>
 </html>
```

步骤 3 在浏览器中输入 127.0.0.1:8000/login，登录后出现如图 3.57 所示的错误页面。

▲图 3.57

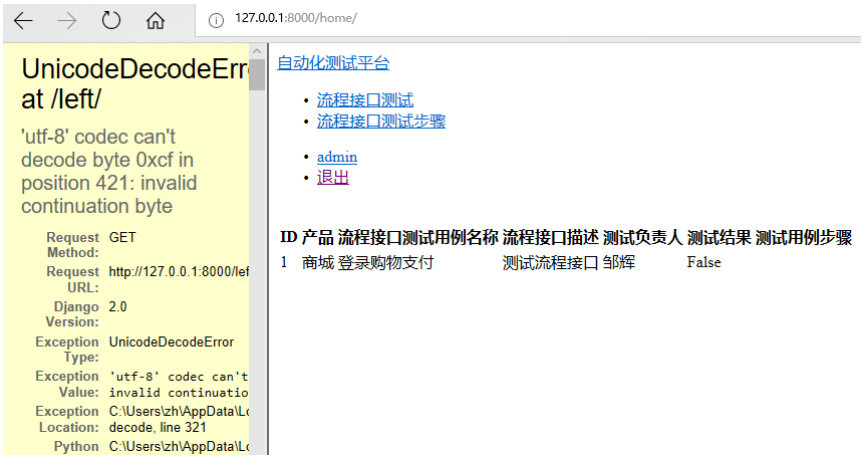
步骤 4 在 apitest/views.py 文件中加入函数。

```
def left(request):
    return render(request,"left.html")
```

在 autotest/urls.py 文件中加入

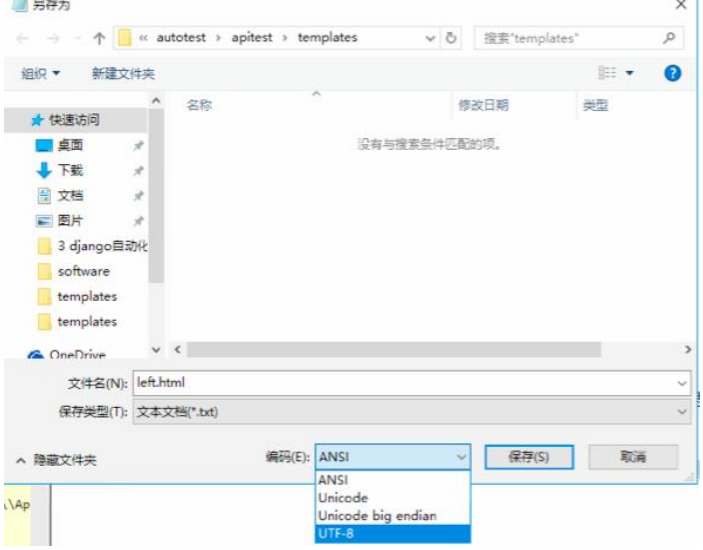
```
path('left/', views.left)
```

再次登录出现错误，如图 3.58 所示。



▲图 3.58

重新打开 apitest/templates/left.html，选择“另存为”按钮，如图 3.59 所示。



▲图 3.59

“编码”选择“UTF-8”，单击“保存”按钮，并覆盖原 left.html 文件。再次登录或刷新页面，如图 3.60 所示。

▲图 3.60

至此，左侧菜单栏和内容栏已经关联显示和实现。

2．加入 Bootstrap 前端框架

Django 虽然简洁强悍，但前端界面尚有提升空间，因此加入 Bootstrap。

步骤 1 安装当前最新版本 Bootstrp4，下载地址为 https://pypi.python.org/pypi/ django-bootstrap3，链接到 https://github.com/zostera/django-bootstrap4 进行下载。

单击“clone or download”按钮保存文件。

运行 CMD，切换到所存放目录 C:\Users\zh\AppData\Local\Programs\Python\Python36\ Scripts\django-bootstrap4-master，运行 python setup.py install。

如图 3.61 所示信息，表明 Bootstrap4 已安装成功。

```
Adding django-bootstrap4 0.0.4 to easy-install.pth file

Installed c:\users\zh\appdata\local\programs\python\python36\lib\site-packages\django_bootstrap4-0.0.4-py3.6.egg
Processing dependencies for django-bootstrap4==0.0.4
Finished processing dependencies for django-bootstrap4==0.0.4

C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\django-bootstrap4-master>
```

▲图 3.61

步骤 2 在 autotest/settings.py 文件中加入如下内容。

```
INSTALLED_APPS = (
     #..
    'bootstrap4',
    #..
)
```

步骤 3 在 apitest/templates 目录下，将 apitest_manage.html 修改为如下内容。

```
<html lang="zh-CN">
 <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
{% bootstrap_javascript %}
 <title>自动化测试平台</title>
 </head>
 <body role="document">
 <!-- 导航栏-->
 <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
 <div class="container">
 <a class="navbar-brand" href="#">自动化测试平台</a>
 <ul class="nav justify-content-center">
 <li class="nav-link"><a href="#">流程接口测试</a></li>
 <li class="nav-link"><a href="/apistep_manage/">流程接口测试步骤</a></li>
 </ul>
 <ul class="nav justify-content-end">
 <li class="nav-link"><a href="#">{{user}}</a></li>
 <li class="nav-link"><a href="/logout/">退出</a></li>
 </ul>
 </div>
 </nav>
 <!-- 流程接口列表-->
 <div class="row" style="padding-top: 70px;">
 <div class="col-md-11">
 <table class="table table-striped">
 <thead>

 <tr>
 <th>ID</th><th>业务接口流程名称</th><th>执行人</th><th>结果</th><th>接口列表</th>
 </tr>
 </thead>
 <tbody>
 {% for apitest in apitests %}
 <tr>
 <td>{{ apitest.id }}</td>
 <td>{{ apitest.apitestname }}</td>
 <td>{{ apitest.apitester }}</td>
 <td>{{ apitest.apitestresult }}</td>
 </tr>
 {% endfor %}
 </tbody>
```

```
</table>
</div>
</div>
</body>
</html>
```

步骤 4 再次登录或刷新页面，效果如图 3.62 所示。

▲图 3.62

步骤 5 在 apitest/templates 目录下，将 apistep_manage.html 修改为如下内容。

```
<html lang="zh-CN">
 <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
 <title>自动化测试平台</title>
 </head>
 <body role="document">
 <!-- 导航栏-->
 <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
 <div class="container">
 <a class="navbar-brand" href="#">自动化测试平台</a>
 <ul class="nav justify-content-center">
 <a class="nav-link" style='color:white' href="/apitest_manage/">流程接口测试</a>
                   
 <a class="nav-link" style='color:white' href="/apistep_manage/">流程接口测试步骤</a>
 </ul>
 <ul class="nav justify-content-end">
 <li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
 <li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
 </ul>
 </div>
 </nav>

 <!-- 流程接口测试步骤-->
 <div class="row" style="padding-top: 70px;">
 <div class="col-md-11">
 <table class="table table-striped">
 <thead>

 <tr>
 <th>ID</th><th>接口名称</th><th>接口名称</th><th>参数=值</th><th>方法</th><th>预期结果</th><th>测试结果
</th><th>时间</th><th>业务流名称</th>
 </tr>
 </thead>
 <tbody>
```

```
{% for apistep in apisteps %}
<tr>
<td>{{ apistep.id }}</td>
<td>{{ apistep.apiname }}</td>
<td>{{ apistep.apiurl }}</td>
<td>{{ apistep.apiparamvalue }}</td>
<td>{{ apistep.apimethod }}</td>
<td>{{ apistep.apiresult }}</td>
<td>{{ apistep.apistatus }}</td>
<td>{{ apistep.create_time }}</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</body>
```

打开链接到接口测试步骤，如图 3.63 所示。

▲图 3.63

步骤 6 在 apitest/templates 目录下，将 apis_manage.html 修改为如下内容。

```
<html lang="zh-CN">
 <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
 <title>产品自动化测试平台</title>
 </head>
 <body role="document">
 <!-- 导航栏-->
 <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
 <div class="container">
 <a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">
<li><a style='color:white' href="/apitest_manage/">流程接口测试用例</a></li>
             
             
<li class="active"><a style='color:white' href="/apistep_manage/">流程接口测试步骤</a></li>
             
             
<li class="active"><a style='color:white' href="/apis_manage/">单一接口测试用例</a></li>
</ul>

 <ul class="nav justify-content-end">
 <li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
 <li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
```

```
 </ul>
 </div>
 </nav>
     <!—单一接口测试用例-->
     <div class="row" style="padding-top: 70px;">
     <div class="col-md-11">
     <table class="table table-striped">
     <thead>
     <tr>
     <th>ID</th><th>产品</th><th>单一接口测试用例名称</th><th>URL 地址</th><th>参数=值</th><th>方法</th><th>
预期结果</th><th>测试结果</th><th>时间</th>
     </tr>
     </thead>
     <tbody>
     {% for apis in apiss %}
     <tr>
     <td>{{ apis.id }}</td>
    <td>{{ apis.Product.productname }}</td>
     <td>{{ apis.apiname }}</td>
     <td>{{ apis.apiurl }}</td>
     <td>{{ apis.apiparamvalue }}</td>
     <td>{{ apis.apimethod }}</td>
     <td>{{ apis.apiresult }}</td>
     <td>{{ apis.apistatus }}</td>
     <td>{{ apis.create_time }}</td>
     </tr>
     {% endfor %}
     </tbody>
     </table>
     </div>
     </div>
     </body>
</html>
```

步骤 7 将 apitest/templates 目录下的 left.html 文件修改为如下内容。

```
<html>
     <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
         <title>test</title>
     </head>
     <body>
       <nav class="navbar navbar-inverse navbar-fixed-top"></nav>
         <table border="0">
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="../apitest_manage" target="mainFrame">  项目管理</a></td>
            </tr>
                <tr><td> </td></tr>
                <tr>
                 <td>-----------------------------------------------</a></td>
            </tr>
                  <tr><td> </td></tr>
                 <tr>
                 <td><a href="../apitest_manage" target="mainFrame">  用例管理</a></td>
```

```
            </tr>
               <tr><td> </td></tr>

            <tr>
                <td><a href="../apitest_manage" target="mainFrame">  api 接口测试用例列表
</a></td>
            </tr>
            <tr><td> </td></tr>
               <tr><td> </td></tr>
               <tr>
                <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  新增 api 接口测试用例</a></td>
            </tr>
               <tr><td> </td></tr>

               <tr>
                <td>-----------------------------------------------</a></td>
            </tr>
               <tr><td> </td></tr>
               <tr>
                <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  任务管理</a></td>
            </tr>
               <tr><td> </td></tr>
               <tr>
                <td>-----------------------------------------------</a></td>
            </tr>
               <tr><td> </td></tr>
               <tr>
                <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  测试报告</a></td>
            </tr>


        </table>
    </body>
</html>
```

再次刷新当前页面，前端页面优化效果如图 3.64 所示。

步骤 8 左侧栏顶部黑底白字。将 apitest/templates 目录下的 left.html 文件修改为如下内容。

```
<html>
    <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
        <title>test</title>
    </head>
    <body>
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
        <ul><li></li></ul><ul><li></li></ul>
        </nav>
```

▲图 3.64

刷新查看前端页面，效果如图 3.65 所示。

▲图 3.65

步骤 9 重新调整优化页面，把顶部的菜单栏放到左侧菜单栏中，修改 apitest/left.html、apitest/apitest_manage.html 和 apitest/apistep_manage.html。

left.html 内容如下。

```
<html>
    <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
        <title>test</title>
    </head>
    <body>
      <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
 <div class="container">
        <a class="navbar-brand" href="../apistep_manage" target="mainFrame">《自动化平台测试开发》书</a>
</div>
        </nav>
        <table border="0">
               <tr><td> </td></tr>
               <tr><td> </td></tr>
               <tr><td> </td></tr>
               <tr><td> </td></tr>
               <tr>
                <td><a href="../apitest_manage" target="mainFrame">  项目管理</a></td>
            </tr>
               <tr><td> </td></tr>
               <tr>
                <td>-----------------------------------------------</a></td>
            </tr>
               <tr><td> </td></tr>
               <tr>
```

```
                <td><a href="../apitest_manage" target="mainFrame">  用例管理</a></td>
            </tr>
                <tr><td> </td></tr>

          <tr>
                <td><a href="../apitest_manage" target="mainFrame">  流程接口测试用例</a></td>
            </tr>
           <tr><td> </td></tr>
           <tr>
                <td><a href="../apistep_manage" target="mainFrame">  流程接口测试步骤</a></td>
            </tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  新增接口测试用例</a></td>
            </tr>
                <tr><td> </td></tr>

                <tr>
                 <td>-----------------------------------------------</a></td>
            </tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  任务管理</a></td>
            </tr>
                <tr><td> </td></tr>
                <tr>
                 <td>-----------------------------------------------</a></td>
            </tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  测试报告</a></td>
            </tr>
        </table>
    </body>
</html>
```

步骤 10 apitest_manage.html 内容如下。

```
<html lang="zh-CN">
 <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
 <title>产品自动化测试平台</title>
 </head>
 <body role="document">
 <!-- 导航栏-->
 <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
 <div class="container">
 <a class="navbar-brand" href="#"> </a>
 <ul class="nav justify-content-center">
 </ul>
 <ul class="nav justify-content-end">
 <li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
 <li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
 </ul>
 </div>
 </nav>
```

```
<!-- 流程接口列表-->
<div class="row" style="padding-top: 20px;">
<div class="col-md-11">
<table class="table table-striped">
<thead>
<tr>
<th>ID</th><th>业务接口流程名称</th><th>执行人</th><th>结果</th><th>接口列表</th>
</tr>
</thead>
<tbody>
{% for apitest in apitests %}
<tr>
<td>{{ apitest.id }}</td>
<td>{{ apitest.apitestname }}</td>
<td>{{ apitest.apitester }}</td>
<td>{{ apitest.apitestresult }}</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</body>
</html>
```

步骤 11 apistep_manage.html 内容如下。

```
<html lang="zh-CN">
<head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap javascript %}
<title>产品自动化测试平台</title>
</head>
<body role="document">
<!-- 导航栏-->
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
<div class="container">
<a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">
</ul>
<ul class="nav justify-content-end">
<li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
<li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
</ul>
</div>
</nav>

<!-- 流程接口测试步骤-->
<div class="row" style="padding-top: 70px;">
<div class="col-md-11">
<table class="table table-striped">
<thead>
<tr>
<th>ID</th><th>接口名称</th><th>接口名称</th><th>参数=值</th><th>方法</th><th>预期结果</th><th>测试结果
</th><th>时间</th><th>业务流名称</th>
</tr>
</thead>
<tbody>
{% for apistep in apisteps %}
```

```
<tr>
<td>{{ apistep.id }}</td>
<td>{{ apistep.apiname }}</td>
<td>{{ apistep.apiurl }}</td>
<td>{{ apistep.apiparamvalue }}</td>
<td>{{ apistep.apimethod }}</td>
<td>{{ apistep.apiresult }}</td>
<td>{{ apistep.apistatus }}</td>
<td>{{ apistep.create_time }}</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</body>
</html>
```

刷新查看前端页面，效果如图 3.66 所示。

▲图 3.66

3．将用例管理左侧的四个菜单改成两个菜单。

步骤 1 分别在 apitest/templates/apitest_manage.html、apistep_manage.html 和 apis_ manage. html 的导航栏中间分别加入如下内容。

```
<ul class="nav justify-content-center">
<li><a style='color:white' href="/apitest_manage/">流程接口测试用例</a></li>
             
             
<li class="active"><a style='color:white' href="/apistep_manage/">流程接口测试步骤</a></li>
             
             
<li class="active"><a style='color:white' href="/apis_manage/">单一接口测试用例</a></li>
</ul>
```

步骤 2 在 apptest/templates/appcase_manage.html 和 appcasestep_manage.html 的导航栏中间分别加入如下内容。

```
<ul class="nav justify-content-center">
<li><a style='color:white' href="/appcase_manage/">app 测试用例</a></li>
             
             
<li class="active"><a style='color:white' href="/appcasestep_manage/">app 测试用例步骤</a></li>
             
             
</ul>
```

步骤 3 在 webtest/templates/webcase_manage.html 和 webcasestep_manage.html 的导航栏中间分别加入如下内容。

```
<ul class="nav justify-content-center">
<li><a style='color:white' href="/webcase_manage/">web 测试用例</a></li>
             
             
<li class="active"><a style='color:white' href="/webcasestep_manage/">web 测试用例步骤</a></li>
             
             
 </ul>
```

步骤 4 在 apitest/templates/left.html 中分别修改为以下内容。

```
<html>
     <head>
 {% load bootstrap4 %}
 {% bootstrap_css %}
 {% bootstrap_javascript %}
         <title>test</title>
     </head>
     <body>
       <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
 <div class="container">
         <a class="navbar-brand" href="../apistep_manage" target="mainFrame">《自动化平台测试开发》书</a>
</div>
         </nav>
         <table border="0">
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="../apitest_manage" target="mainFrame">  项目管理</a></td>
             </tr>
                <tr><td> </td></tr>
                <tr>
                 <td>-----------------------------------------------</a></td>
             </tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="../apitest_manage" target="mainFrame">  用例管理</a></td>
             </tr>
                <tr><td> </td></tr>

           <tr>
                <td><a href="../apitest_manage" target="mainFrame">  流程接口测试用例</a></td>
             </tr>
            <tr><td> </td></tr>

  <tr><td>-----------------------------------------------</a></td></tr>
             <tr>
                 <td><a href="../appcase_manage" target="mainFrame">  app 用例管理</a></td>
             </tr>
            <tr><td> </td></tr>
             <tr>
                 <td><a href="../appcase_manage" target="mainFrame">  app 测试用例</a></td>
             </tr>
             <tr><td> </td></tr>

             <tr><td>----------------------</a></td></tr>

                <tr><td> </td></tr>
                       <tr>
                 <td><a href="../webcase_manage" target="mainFrame">  web 用例管理</a></td>
             </tr>
```

```
                <tr><td> </td></tr>
            <tr>
                <td><a href="../webcase_manage" target="mainFrame">  web 测试用例</a></td>
            </tr>
            <tr><td> </td></tr>

            <tr><td>

                <tr>
                 <td>-----------------------------------------------</a></td>
            </tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  任务管理</a></td>
            </tr>
                <tr><td> </td></tr>
                <tr>
                 <td>-----------------------------------------------</a></td>
            </tr>
                <tr><td> </td></tr>
                <tr>
                 <td><a href="http://127.0.0.1:8000/admin/apitest/apitest/add/"
target="mainFrame">  测试报告</a></td>
            </tr>
        </table>
    </body>
</html>
```

步骤 5 添加 Admin 链接到 Djanog 后台。把 HTML 页面的导航栏修改为以下内容。

```
<li>欢迎您，<a href="../admin/">{{user}}</a></li>
```

主页面优化结束，查看前端首页的页面效果，如图 3.67 所示。

▲图 3.67

3.12.3 前端搜索功能实现

Django 框架由内置的方法实现，在 Model 中分别提供了 filter 方法和 icontains 方法来实现简单的搜索功能。

步骤 1 模板前端页面：apitest/templates 的 apitest_manage.html 文件内容。

```
<!-- 搜索栏-->
```

```
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/apisearch/">
{% csrf_token %}
<input type="search" name="apitestname" placeholder="名称" required>
<button type="submit">搜索</button>
</form>
</div>
```

步骤 2 在视图 apitest/views.py 中增加 search 函数，并赋值给返回的变量。

```
# 搜索功能
 @login required
 def apisearch(request):
     username = request.session.get('user', '') # 读取浏览器登录 session
     search_apitestname = request.GET.get("apitestname", "")
     apitest_list = Apitest.objects.filter(apitestname__icontains=search_apitestname)
     return render(request,'apitest_manage.html', {"user": username,"apitests":apitest_list})
```

步骤 3 在 autotest/urls.py 中增加连接映射。

```
   path ('apisearch/', views.apisearch),
```

步骤 4 再次刷新页面，如图 3.68 所示。

▲图 3.68

输入名称“登录”，单击“搜索”按钮，如图 3.69 所示。

在单一接口测试用例页面、流程接口测试步骤页面、产品管理页面、系统设置页面分别加入搜索功能，全部与流程接口测试用例页面类似。

▲图 3.69

1．系统设置页面

步骤 1 在 set/templates/set_manage.html 页面加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/setsearch/">
{% csrf_token %}
<input type="search" name="setname" placeholder="名称" required>
<button type="submit">搜索</button>
```

步骤 2 在 set/setviews.py 中加入如下内容。

```
@login_required
 def setsearch(request):
     username = request.session.get('user', '') # 读取浏览器登录 Session
     search_setname = request.GET.get("setname", "")
     set_list = Set.objects.filter(setname__icontains=search_setname)
     return render(request,'set_manage.html', {"user": username,"sets":set_list})
```

步骤 3 在 autotest/urls.py 中增加函数映射。

```
   path ('setsearch/', setviews.setsearch),
```

2．产品管理页面

步骤 1 在 product/templates/product_manage.html 页面中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/productsearch/">
{% csrf_token %}
<input type="search" name="productname" placeholder="名称" required>
<button type="submit">搜索</button>
```

步骤 2 在 product/proviews.py 中加入如下内容。

```
# 搜索功能
@login_required
def productsearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_productname = request.GET.get("productname", "")
    product_list = Product.objects.filter(productname__icontains=search_productname)
return render(request,'product_manage.html', {"user": username,"products":product_list})
```

步骤 3 在 autotest/urls.py 中增加函数映射。

```
   path ('productsearch/', proviews.productsearch),
```

3．单一接口测试用例页面

步骤 1 在 apitest/templates/apis_manage.html 页面中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/apissearch/">
{% csrf_token %}
<input type="search" name="apiname" placeholder="名称" required>
<button type="submit">搜索</button>
```

步骤 2 在 apitest/views.py 中加入如下内容。

```
# 搜索功能
@login_required
def apissearch(request):
```

```
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_apiname = request.GET.get("apiname", "")
    apis_list = Apis.objects.filter(apiname__icontains=search_apiname)
    return render(request,'apis_manage.html', {"user": username,"apiss":apis_list})
```

步骤 3 在 autotest/urls.py 中增加函数映射。

```
   path ('apissearch/', views.apissearch),
```

4．Bug 管理页面

步骤 1 在 bug/templates/bug_manage.html 页面中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/bugsearch/">
{% csrf_token %}
<input type="search" name="bugname" placeholder="名称" required>
<button type="submit">搜索</button>
```

步骤 2 在 bug/bugviews.py 中加入如下内容。

```
from django.contrib.auth.decorators import login_required
from django.contrib import auth
from django.contrib.auth import authenticate, login
# 搜索功能
@login_required
def bugsearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_bugname = request.GET.get("bugname", "")
    bug_list = Bug.objects.filter(bugname__icontains=search_bugname)
return render(request,'bug_manage.html', {"user": username,"bugs":bug_list})
```

步骤 3 在 autotest/urls.py 中增加函数映射。

```
   path ('bugsearch/', bugviews.bugsearch),
```

5．appcase 和 appstep

步骤 1 在 Apptest/appviews.py 中加入如下内容。

```
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
# 搜索功能
@login_required
def appsearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_appcasename = request.GET.get("appcasename", "")
    appcase list = Appcase.objects.filter(appcasename  icontains=search appcasename)
    return render(request,'appcase_manage.html', {"user": username,"appcases":appcase_list})


# 搜索功能
@login required
def appstepsearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_appcasename = request.GET.get("appcasename", "")
    appcasestep_list = Appcasestep.objects.filter(appcasename__icontains=search_appcasename)
    return render(request,'appcasestep_manage.html', {"user":
username,"appcasesteps":appcasestep_list})
```

步骤 2 在 apptest/templates/appcase_manage.html 中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
```

```
<form class="navbar-form" method="get" action="/appsearch/">
{% csrf_token %}
<input type="search" name="appcasename" placeholder="名称" required>
<button type="submit">搜索</button>
</form>
</div>
```

步骤 3 在 apptest/templates/appcasestep_manage.html 中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/appstepsearch/">
{% csrf_token %}
<input type="search" name="appcasename" placeholder="名称" required>
<button type="submit">搜索</button>
</form>
</div>
```

步骤 4 在 / autotest/urls.py 中加入如下内容。

```
   path('appsearch/', appviews.appsearch),
   path('appstepsearch/', appviews.appstepsearch),
```

6．Web 用例

步骤 1 在 webtest/webviews.py 中加入如下内容。

```
# Web 用例管理

@login_required
# 搜索功能
@login_required
def websearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_webcasename = request.GET.get("webcasename", "")
    webcase_list = Webcase.objects.filter(webcasename__icontains=search_webcasename)
    return render(request,'webcase_manage.html', {"user": username,"webcases":webcase_list})

# 搜索功能
@login_required
def webstepsearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_webcasename = request.GET.get("webcasename", "")
    webcasestep_list = Webcasestep.objects.filter(webcasename__icontains=search_webcasename)
    return render(request,'webcasestep_manage.html', {"user":
username,"webcasesteps":webcasestep_list})
```

步骤 2 在 webtest/templates/webcase_manage.html 中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/websearch/">

{% csrf_token %}
<input type="search" name="webcasename" placeholder="名称" required>

<button type="submit">搜索</button>

</form>
</div>
```

步骤 3 在 webtest/templates/webcasestep_manage.html 中加入如下内容。

```
<!-- 搜索栏-->
```

```
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/webstepsearch/">

{% csrf_token %}
<input type="search" name="webcasename" placeholder="名称" required>

<button type="submit">搜索</button>

</form>
</div>
```

步骤 4 在 autotest/urls.py 中加入如下内容。

```
path('websearch/', webviews.websearch),
path('webstepsearch/', webviews.webstepsearch),
```

至此，Web 用例管理前端页面开发已成功实现。也可跳转到本书第 8 章，进行 Web 自动化测试执行的相关内容。

7．用户设置加入搜索功能

步骤 1 在 set/setviews.py 中加入如下内容。

```
# 搜索功能
@login_required
def usersearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_username = request.GET.get("username", "")
    user_list = User.objects.filter(username__icontains=search_username)
return render(request,'set_user.html', {"user": username,"users":user_list})
```

步骤 2 在 set/templates/Set_manage.html 中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/usersearch/">

{% csrf_token %}
<input type="search" name="username" placeholder="名称" required>

<button type="submit">搜索</button>

</form>
</div>
```

步骤 3 在 autotest/urls.py 中加入如下内容。

```
path ('usersearch/', setviews.usersearch),
```

至此，搜索功能已经实现。

3.12.4 前端翻页功能实现

1）Django 提供了翻页器，常用 Django 中的 Paginator 类来实现。

步骤 1 在前端 template 中加入代码。在 body 与 body 中间加入如下内容。

```
<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if apitests.has_previous %}
            {#  如果有上一页，则正常显示上一页链接#}
```

```
            <li class="previous"><a href="/apitest_manage/?page={{ apitests.previous_page_number }}">上一页</a></li>    {# 上一页标签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页，则上一页的链接不可单击#}
        {% endif %}
        {# 上一页链接结束#}

        {% for num in apitests.paginator.page_range %}

           {% if num == currentPage %}
               <li class="item active"><a href="/apitest_manage/?page={{ num }}">{{ num }}</a></li> {# 显示当页数链接#}
            {% else %}
                <li class="item"><a href="/apitest_manage/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if apitests.has_next %} {#  如果有下一页，则正常显示下一页链接#}
            <li class="next"><a href="/apitest_manage/?page={{ apitests.next_page_number }}">下一页</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
</div>
```

步骤 2 在 views 中加入如下代码。

```
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

# 流程接口管理
@login_required
def apitest_manage(request):
    apitest_list = Apitest.objects.all()       #获取所有接口测试用例
    username = request.session.get('user', '') # 读取浏览器登录 Session
    paginator = Paginator(apitest_list, 8)  #生成 paginator 对象，设置每页显示 8 条记录
    page = request.GET.get('page',1)  #获取当前的页码数，默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        apitest_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        apitest_list = paginator.page(1)#如果输入的页数不是整数，则显示第 1 页内容
    except EmptyPage:
        apitest_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统的页数# 中，则显示最后一页内容
    return render(request,"apitest_manage.html",{"user": username,"apitests":apitest_list})
```

步骤 3 在浏览器中查看流程接口测试用例。

第 1 页如图 3.70 所示。

ID	产品	流程接口测试用例名称	测试负责人	测试结果
2	卖家	登录购物支付test	邹辉	False
3	卖家	登录购物支付	邹辉	False
4	卖家	登录购物支付aa	邹辉	False
5	卖家	登录购物支付bb	邹辉	False
7	卖家	登录购物支付cc	邹辉	False

上一页12下一页

▲图 3.70

第 2 页如图 3.71 所示。

ID	产品	流程接口测试用例名称	测试负责人	测试结果
8	卖家	登录购物支付dd	邹辉	False

上一页12下一页

▲图 3.71

步骤 4 然后把页码显示放到右下角，在翻页代码前一行加入如下内容。

```
<span style="position:absolute; right:100px; bottom:30px;">    {# 把翻页功能固定显示在右下角#}
```

再次刷新前端，如图 3.72 所示。

ID	产品	流程接口测试用例名称	测试负责人	测试结果
2	卖家	登录购物支付test	邹辉	False
3	卖家	登录购物支付	邹辉	False
4	卖家	登录购物支付aa	邹辉	False
5	卖家	登录购物支付bb	邹辉	False
7	卖家	登录购物支付cc	邹辉	False

上一页12下一页

▲图 3.72

2）在 product、apistep、apis 中加入搜索和翻页功能。

步骤 1 在 Product_manage.html 中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/productsearch/"
{% csrf_token %}
<input type="search" name="productname" placeholder="名称" required>
<button type="submit">搜索</button>
</form>
</div>

<!-- 翻页功能-->
```

```
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}
<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if products.has_previous %}
            {#  如果有上一页则正常显示上一页的链接#}
            <li class="previous"><a href="/product_manage/?page={{ products.previous_page_number }}">上
一页</a></li>    {#  上一页标签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页则上一页的链接不
可单击#}
        {% endif %}
        {# 上一页链接开始#}

        {% for num in products.paginator.page_range %}

            {% if num == currentPage %}
                <li class="item active"><a href="/product_manage/?page={{ num }}">{{ num }}</a></li>
{#显示当页数链接#}
            {% else %}
                <li class="item"><a href="/product_manage/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if products.has_next %} {#  如果有下一页，则正常显示下一页的链接#}
            <li class="next"><a href="/product_manage/?page={{ products.next_page_number }}">下一页
</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
</div>
```

步骤 2 在 product/proviews.py 中加入如下内容。

```
from django.shortcuts import render
from product.models import Product
from django.http import HttpResponse,HttpResponseRedirect
from django.contrib.auth.decorators import login_required
from django.contrib import auth
from django.contrib.auth import authenticate, login
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

# Create your views here.

#产品管理
@login_required
def product_manage(request):
    username = request.session.get('user', '')
    product_list = Product.objects.all()
    paginator = Paginator(product_list, 8)  #生成 paginator 对象，设置每页显示 8 条记录
    page = request.GET.get('page',1)  #获取当前的页码数,默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        product_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        product_list = paginator.page(1)#如果输入的页数不是整数，则显示第 1 页内容
    except EmptyPage:
        product_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统的页数# 中，则显示最后一页内容
```

```
    return render(request, "product_manage.html", {"user": username,"products": product_list})

# 搜索功能
@login_required
def productsearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_productname = request.GET.get("productname", "")
    product_list = Product.objects.filter(productname__icontains=search_productname)
    return render(request,'product_manage.html', {"user": username,"products":product_list})

在 autotest/urls.py.py 中加入：
   path('productsearch/', proviews.productsearch),
```

步骤 3 Apistep_manage.html 中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/apistepsearch/">

{% csrf_token %}
<input type="search" name="apiname" placeholder="名称" required>

<button type="submit">搜索</button>

</form>
</div>

<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}
<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if apisteps.has_previous %}
            {#  如果有上一页则正常显示上一页的链接#}
            <li class="previous"><a href="/apistep_manage/?page={{ apisteps.previous_page_number }}">上
一页</a></li>    {#  上一页标签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页则上一页的链接不
可单击#}
        {% endif %}
        {# 上一页链接开始#}

        {% for num in apisteps.paginator.page_range %}

            {% if num == currentPage %}
                <li class="item active"><a href="/apistep_manage/?page={{ num }}">{{ num }}</a></li>
{#显示当前页数链接#}
            {% else %}
                <li class="item"><a href="/apistep_manage/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if apisteps.has_next %} {#  如果有下一页，则正常显示下一页的链接#}
            <li class="next"><a href="/apistep_manage/?page={{ apisteps.next_page_number }}">下一页
</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
</div>
```

步骤 4 在 Apis_mange.html 中加入如下内容。

```
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/apissearch/">

{% csrf_token %}
<input type="search" name="apiname" placeholder="名称" required>

<button type="submit">搜索</button>

</form>
</div>


<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}
<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if apiss.has_previous %}
            {#  如果有上一页则正常显示上一页的链接#}
            <li class="previous"><a href="/apis_manage/?page={{ apiss.previous_page_number }}">上一页
</a></li>    {#  上一页标签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页则上一页的链接不
可单击#}
        {% endif %}
        {# 上一页链接开始#}

        {% for num in apiss.paginator.page_range %}

            {% if num == currentPage %}
                <li class="item active"><a href="/apis_manage/?page={{ num }}">{{ num }}</a></li> {#显
示当前页数链接#}
            {% else %}
                <li class="item"><a href="/apis_manage/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if apiss.has_next %} {#  如果有下一页则正常显示下一页的链接#}
            <li class="next"><a href="/apis_manage/?page={{ apiss.next_page_number }}">下一页</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
</div>
```

步骤 5 在 Views.py 中加入如下内容。

```
# 搜索功能
@login_required
def apistepsearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_apiname = request.GET.get("apiname", "")
    apistep_list = Apistep.objects.filter(apiname__icontains=search_apiname)
    return render(request,'apistep_manage.html', {"user": username,"apisteps":apistep_list})

# 搜索功能
@login_required
```

```
def apissearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_apiname = request.GET.get("apiname", "")
    apis_list = Apis.objects.filter(apiname__icontains=search_apiname)
    return render(request,'apis_manage.html', {"user": username,"apiss":apis_list})


# 接口步骤管理
@login_required
def apistep_manage(request):
    username = request.session.get('user', '')
    apistep_list = Apistep.objects.all()
    paginator = Paginator(apistep_list, 8)  #生成 paginator 对象，设置每页显示 8 条记录
    page = request.GET.get('page',1)  #获取当前的页码数，默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
       apistep_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
       apistep_list = paginator.page(1)#如果输入的页数不是整数，则显示第 1 页内容
    except EmptyPage:
        apistep_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统的页数中，则显示最后一页内容
    return render(request, "apistep_manage.html", {"user": username,"apisteps": apistep_list})

# 单一接口管理
@login_required
def apis_manage(request):
    username = request.session.get('user', '')
    apis_list = Apis.objects.all()
    paginator = Paginator(apis_list, 8)  #生成 paginator 对象，设置每页显示 8 条记录
    page = request.GET.get('page',1)  #获取当前的页码数，默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        apis_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        apis_list = paginator.page(1)#如果输入的页数不是整数，则显示第 1 页内容
    except EmptyPage:
        apis_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统的页数中，# 则显示最后一页内容
    return render(request, "apis_manage.html", {"user": username,"apiss": apis_list})
```

步骤 6 在 autotest/urls.py 中加入如下内容。

```
    path('apistepsearch/', views.apistepsearch),
    path('apis_manage/', views.apis_manage),
```

3）系统设置页面。

步骤 1 在 set/setviews.py 的 set_manage 函数中加入如下内容。

```
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

paginator = Paginator(set_list, 8)  #生成 paginator 对象，设置每页显示 8 条记录
    page = request.GET.get('page',1)  #获取当前的页码数，默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        set_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        set_list = paginator.page(1)#如果输入的页数不是整数，则显示第 1 页内容
    except EmptyPage:
        set_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统的页数中，
# 则显示最后一页的内容
```

步骤 2 在 set/templates/set_manage.html 中加入如下内容。

```
<!--统计和翻页功能-->
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}

<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if sets.has_previous %}
            {#  如果有上一页，则正常显示上一页的链接#}
            <li class="previous"><a href="/set_manage/?page={{ sets.previous_page_number }}">上一页
</a></li>    {#  上一页标签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页，则上一页的链接
不可单击#}
        {% endif %}
        {# 上一页链接开始#}
        {% for num in sets.paginator.page_range %}
            {% if num == currentPage %}
                <li class="item active"><a href="/set_manage/?page={{ num }}">{{ num }}</a></li> {#显示
当前页数链接#}
            {% else %}
                <li class="item"><a href="/set_manage/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if sets.has_next %} {#  如果有下一页则正常显示下一页的链接#}
            <li class="next"><a href="/set_manage/?page={{ sets.next_page_number }}">下一页</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
```

4）在 Bug 管理页面加入翻页功能。

步骤 1 在 bug/bugviews.py 的 bug_manage 函数中加入如下内容。

```
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

paginator = Paginator(bug_list, 8)  #生成 paginator 对象，设置每页显示 8 条记录
    page = request.GET.get('page',1)  #获取当前的页码数，默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        bug_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        bug_list = paginator.page(1)#如果输入的页数不是整数，则显示第 1 页内容
    except EmptyPage:
        bug_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统的页数中，
# 则显示最后一页的内容
```

步骤 2 在 bug/templates/bug_manage.html 中加入如下内容。

```
<!--统计和翻页功能-->
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}

<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if bugs.has_previous %}
            {#  如果有上一页，则正常显示上一页的链接#}
            <li class="previous"><a href="/bug_manage/?page={{ bugs.previous_page_number }}">上一页
</a></li>    {#  上一页标签 #}
```

```
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页，则上一页的链接
不可单击#}
        {% endif %}
        {# 上一页链接开始#}
        {% for num in bugs.paginator.page_range %}
            {% if num == currentPage %}
                <li class="item active"><a href="/bug_manage/?page={{ num }}">{{ num }}</a></li> {#显示
当前页数链接#}
            {% else %}
                <li class="item"><a href="/bug_manage/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if bugs.has_next %} {#  如果有下一页，则正常显示下一页的链接#}
            <li class="next"><a href="/bug_manage/?page={{ bugs.next_page_number }}">下一页</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
```

5）在 Apptest 和 appcasestep 加入翻页功能。

步骤 1 在 apptest/appviews.py 的 appcase_manage 和 appcasestep()函数中加入以下内容。

```
# App 测试用例管理
@login_required
def appcase_manage(request):
    appcase_list = Appcase.objects.all()
    username = request.session.get('user', '') # 读取浏览器登录 Session
    paginator = Paginator(appcase_list, 8)  #生成 paginator 对象,设置每页显示 8 条记录
    page = request.GET.get('page',1)  #获取当前的页码数,默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        appcase_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        appcase_list = paginator.page(1)#如果输入的页数不是整数则显示第 1 页的内容
    except EmptyPage:
        appcase_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统的页数# 中则显示最后一页的内容
    return render(request,"appcase_manage.html",{"user": username,"appcases":appcase_list})


# App 测试步骤管理
@login_required
def appcasestep_manage(request):
    username = request.session.get('user', '')
    appcasestep_list = Appcasestep.objects.all()
    paginator = Paginator(appcasestep_list, 8)  #生成 paginator 对象,设置每页显示 8 条记# 录
    page = request.GET.get('page',1)  #获取当前的页码数,默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        appcasestep_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        appcasestep_list = paginator.page(1)#如果输入的页数不是整数则显示第 1 页的内容
    except EmptyPage:
        appcasestep_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统
# 的页数中则显示最后一页
    return render(request, "appcasestep_manage.html", {"user": username,"appcasesteps":
appcasestep_list})
```

步骤 2 在 apptest/templates/appcase_manage.html 中加入如下内容。

```
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}
<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if appcases.has_previous %}
            {#  如果有上一页，则正常显示上一页的链接#}
            <li class="previous"><a href="/appcase_manage/?page={{ appcases.previous_page_number }}">上
一页</a></li>    {#  上一页标签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页，则上一页的链接
不可单击#}
        {% endif %}
        {# 上一页链接开始#}

        {% for num in appcases.paginator.page range %}

            {% if num == currentPage %}
                <li class="item active"><a href="/appcase_manage/?page={{ num }}">{{ num }}</a></li>
{#显示当前页数链接#}
            {% else %}
                <li class="item"><a href="/appcase_manage/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if appcases.has_next %} {#如果有下一页，则正常显示下一页的链接#}
            <li class="next"><a href="/appcase_manage/?page={{ appcases.next_page_number }}">下一页
</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
</div>
```

步骤 3 在 apptest/templates/appcasestep_manage.html 中加入如下内容。

```
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}
<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if appcasesteps.has_previous %}
            {#  如果有上一页，则正常显示上一页的链接#}
            <li class="previous"><a
href="/appcasestep_manage/?page={{ appcasesteps.previous_page_number }}">上一页</a></li>    {#  上一页标
签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页，则上一页的链接
不可单击#}
        {% endif %}
        {# 上一页链接开始#}

        {% for num in appcasesteps.paginator.page_range %}

            {% if num == currentPage %}
                <li class="item active"><a
href="/appcasestep_manage/?page={{ num }}">{{ num }}</a></li> {#显示当前页数链接#}
            {% else %}
                <li class="item"><a href="/appcasestep_manage/?page={{ num }}">{{ num }}</a></li>
```

```
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if appcasesteps.has_next %} {#  如果有下一页，则正常显示下一页的链接#}
            <li class="next"><a href="/appcasestep_manage/?page={{ appcasesteps.next_page_number }}">下
一页</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
</div>
```

6）Web 用例和 Web 用例步骤页面翻页。

在 webcase 和 webstep 的 webtest/webviews.py 中加入如下内容。

```
# Web 用例管理

@login required

def webcase_manage(request):

    webcase_list = Webcase.objects.all()

    username = request.session.get('user', '') # 读取浏览器登录 Session
    paginator = Paginator(webcase_list, 8)  #生成 paginator 对象，设置每页显示 8 条记录
    page = request.GET.get('page',1)  #获取当前的页码数，默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        webcase_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        webcase_list = paginator.page(1)#如果输入的页数不是整数，则显示第 1 页的内容
    except EmptyPage:
        webcase_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统的页数# 中，则显示最后一页

    return render(request,"webcase manage.html",{"user": username,"webcases":webcase list})


# Web 用例测试步骤

@login_required

def webcasestep_manage(request):

    username = request.session.get('user', '')

    webcasestep_list = Webcasestep.objects.all()
   paginator = Paginator(webcasestep_list, 8)  #生成 paginator 对象，设置每页显示 8 条
# 记录
    page = request.GET.get('page',1)  #获取当前的页码数，默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        webcasestep_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        webcasestep_list = paginator.page(1)#如果输入的页数不是整数，则显示第 1 页内容
    except EmptyPage:
        webcasestep_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统
# 的页数中，则显示最后一页内容
```

```
    return render(request, "webcasestep_manage.html", {"user": username,"webcasesteps":
webcasestep_list})
```

至此，Web 用例管理前端页面开发已成功实现。也可跳转到本书第 8 章，进行 Web 自动化测试执行的相关内容。

在 webtest/templates/webcase_manage.html 分别加入如下内容。

```
<span style="position:absolute; right:100px; bottom:30px;">    {# 把翻页功能固定显示在右下角#}
<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if webcases.has_previous %}
            {#  如果有上一页，则正常显示上一页链接#}
            <li class="previous"><a href="/webcase_manage/?page={{ webcases.previous_page_number }}">上
一页</a></li>    {#  上一页标签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页，则上一页的链接
不可单击#}
        {% endif %}
        {# 上一页链接开始#}

        {% for num in webcases.paginator.page_range %}

            {% if num == currentPage %}
                <li class="item active"><a href="/webcase_manage/?page={{ num }}">{{ num }}</a></li>
{#显示当前页数链接#}
            {% else %}
                <li class="item"><a href="/webcase_manage/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if webcases.has_next %} {#  如果有下一页，则正常显示下一页链接#}
            <li class="next"><a href="/webcase_manage/?page={{ webcases.next_page_number }}">下一页
</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
</div>
```

在 webtest/templates/webcasestep_manage.html 中加入如下内容。

```
<span style="position:absolute; right:100px; bottom:30px;">    {# 把翻页功能固定显示在右下角#}
<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if webcasesteps.has_previous %}
            {#  如果有上一页则正常显示上一页的链接#}
            <li class="previous"><a
href="/webcasestep_manage/?page={{ webcasesteps.previous_page_number }}">上一页</a></li>    {#  上一页标
签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页，则上一页的链接
不可单击#}
        {% endif %}
        {# 上一页链接开始#}

        {% for num in webcasesteps.paginator.page_range %}
```

```
            {% if num == currentPage %}
                <li class="item active"><a
href="/webcasestep_manage/?page={{ num }}">{{ num }}</a></li> {#显示当前页数链接#}
            {% else %}
                <li class="item"><a href="/webcasestep_manage/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if webcasesteps.has_next %} {#  如果有下一页，则正常显示下一页链接#}
            <li class="next"><a href="/webcasestep_manage/?page={{ webcasesteps.next_page_number }}">下
一页</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
</div>
```

翻页功能到此实现。

3.12.5 数据统计功能实现

1．产品管理页面

在 product/proviews.py 中加入如下内容。

```
def product_manage(request):
  product_count = Product.objects.all().count()  #统计产品数
  return render(request, "product_manage.html", {"user": username,"products":
product_list,"productcounts": product_count}) #把值赋给 productcounts 变量
```

在 product/templates/product_manage.html 中加入如下内容。

```
<!--统计和翻页功能-->
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}

<div style="position:absolute;  right:100px; width:100px; ">
<tr><th>总数</th><td>{{ productcounts }}</td></tr> {# 前端读取定义的变量#}
</div>
```

2．查看前端（如图 3.73 所示）

ID	产品名称	产品描述	产品负责人	创建时间
2	商城	图书	邹辉	2018年1月30日
3	app产品	计算器，Csdn	邹辉	2018年1月29日
10	自动化平台	包括API、WebUI、AppUI自动化测试	邹辉	2018年1月30日
11	web产品	百度搜索	邹辉	2018年1月29日

共:4条记录

▲图 3.73

同理，在用例管理等模块中操作类似。

3．单一接口测试用例页面

步骤 1 在 apitest/views.py 中加入如下内容。

```
# 统计功能
@login_required
def apis_manage(request):
  apis_count = Apis.objects.all().count()  #统计产品数
  return render(request, "apis_manage.html", {"user": username,"apiss": apis_list,"apiscounts":
apis_count}) #把值赋给 apiscounts 变量
```

步骤 2 在 apitest/templates/apis_manage.html 中加入如下内容。

```
<!--统计和翻页功能-->
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}

<div style="position:absolute;  right:100px; width:100px; ">
<tr><th>总数</th><td>{{ apiscounts }}</td></tr> {# 前端读取定义的变量#}
</div>
```

4．流程接口测试用例页面

步骤 1 在 apitest/views.py 中加入如下内容。

```
# 统计功能
@login_required
def apitest_manage(request):
  apitest_count = Apitest.objects.all().count()  #统计产品数
  return render(request, "apitest_manage.html", {"user": username,"apitests":
apitest_list,"apitestcounts": apitest_count}) #把值赋给 apitestcounts 变量
```

步骤 2 在 apitest/templates/apitest_manage.html 中加入如下内容。

```
<!--统计和翻页功能-->
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}

<div style="position:absolute;  right:100px; width:100px; ">
<tr><th>总数</th><td>{{ apitestcounts }}</td></tr> {# 前端读取定义的变量#}
</div>
```

5．App 用例管理页面

步骤 1 在 apptest/appviews.py 中加入如下内容。

```
# 统计功能
@login_required
def appcase_manage(request):
  appcase_count = Appcase.objects.all().count()  #统计产品数
  return render(request, "appcase_manage.html", {"user": username,"appcases":
appcase_list,"appcasecounts": appcase_count}) #把值赋给 appcasecounts 变量
```

步骤 2 在 apptest/templates/appcase_manage.html 中加入如下内容。

```
<!--统计和翻页功能-->
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}

<div style="position:absolute;  right:100px; width:100px; ">
<tr><th>总数</th><td>{{ appcasecounts }}</td></tr> {# 前端读取定义的变量#}
```

```
</div>
```

6．Web 用例管理页面

步骤 1 在 webtest/webviews.py 中加入如下内容。

```
# 统计功能
@login_required
def webcase_manage(request):
  webcase_count = Webcase.objects.all().count()  #统计产品数
  return render(request, "webcase_manage.html", {"user": username,"webcases":
webcase_list,"webcasecounts": webcase_count}) #把值赋给 webcasecounts 变量
```

步骤 2 在 webtest/templates/webcase_manage.html 中加入如下内容。

```
<!--统计和翻页功能-->
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}

<div style="position:absolute;  right:100px; width:100px; ">
<tr><th>总数</th><td>{{ webcasecounts }}</td></tr> {# 前端读取定义的变量#}
</div>
```

7．Bug 管理页面统计功能

步骤 1 在 bug/bugviews.py 中加入如下内容。

```
# 统计功能
@login_required
def bug_manage(request):
  bug_count = Bug.objects.all().count()  #统计 Bug 数
  return render(request, "bug_manage.html", {"user": username,"bugs": bug_list,"bugcounts":
bug_count}) #把值赋给 bugcounts 变量
```

步骤 2 在 bug/templates/bug_manage.html 中加入如下内容。

```
<!--统计和翻页功能-->
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}

<div style="position:absolute;  right:100px; width:100px; ">
<tr><th>总数</th><td>{{ bugcounts }}</td></tr> {# 前端读取定义的变量#}
</div>
```

3.12.6 添加数据功能实现

如图 3.74 所示。

▲图 3.74

步骤 1 在 apitest/templates/apitest_manage.html 中加入如下内容。

```
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />
<script type="text/javascript" src="/admin/jsi18n/"></script>
<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>
<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>
<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>
<script type="text/javascript" src="/static/admin/js/urlify.js"></script>
<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>
<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">
<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />
<meta name="robots" content="NONE,NOARCHIVE" />

<!-- 增加 api 流程测试用例-->
<div style="float:right;width:73%">

<select name="Apitest" id="id_Apitest">
<option value="" selected>----api 流程测试用例----</option>
</select>
<a class="related-widget-wrapper-link change-related" id="change_id_Apitest" data-href-
template="/admin/apitest/apitest/__fk__/change/?_to_field=id&amp;_popup=1" title="更改选中的测试用例">
<img src="/static/admin/img/icon-changelink.svg" alt="修改"/>
</a>
<a class="related-widget-wrapper-link add-related" id="add_id_Apitest"
href="/admin/apitest/apitest/add/?_to_field=id&amp;_popup=1" title="增加另一个测试用例">
<img src="/static/admin/img/icon-addlink.svg" alt="增加"/>增加
</a>
```

在 apitest/templates/apis_manage.html 中加入如下内容。

```
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />
<script type="text/javascript" src="/admin/jsi18n/"></script>
<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>
<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>
<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>
<script type="text/javascript" src="/static/admin/js/urlify.js"></script>
<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>
<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">
<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />
<meta name="robots" content="NONE,NOARCHIVE" />

<!-- 增加 api 单一接口测试用例-->
<div style="float:right;width:73%">
<select name="Apis" id="id_Apis">
<option value="" selected>----api 单一接口测试用例----</option>
</select>
<a class="related-widget-wrapper-link change-related" id="change_id_Apis" data-href-
template="/admin/apitest/apis/__fk__/change/?_to_field=id&amp;_popup=1" title="更改选中的测试用例">
<img src="/static/admin/img/icon-changelink.svg" alt="修改"/>
</a>
<a class="related-widget-wrapper-link add-related" id="add_id_Apis"
href="/admin/apitest/apis/add/?_to_field=id&amp;_popup=1" title="增加另一个测试用例">
<img src="/static/admin/img/icon-addlink.svg" alt="增加"/>增加
```

```
</a>
```

在 apptest/templates/appcase_manage.html 中加入如下内容。

```
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />
<script type="text/javascript" src="/admin/jsi18n/"></script>
<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>
<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>
<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>
<script type="text/javascript" src="/static/admin/js/urlify.js"></script>
<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>
<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">
<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />
<meta name="robots" content="NONE,NOARCHIVE" />


<!-- 增加 app 测试用例-->
<div style="float:right;width:73%">

<select name="Appcase" id="id_Appcase">
<option value="" selected>----app 测试用例----</option>
</select>
<a class="related-widget-wrapper-link change-related" id="change_id_Appcase" data-href-
template="/admin/apptest/appcase/__fk__/change/?_to_field=id&amp;_popup=1" title="更改选中的测试用例">
<img src="/static/admin/img/icon-changelink.svg" alt="修改"/>
</a>
<a class="related-widget-wrapper-link add-related" id="add_id_Appcase"
href="/admin/apptest/appcase/add/?_to_field=id&amp;_popup=1" title="增加另一个 测试用例">
<img src="/static/admin/img/icon-addlink.svg" alt="增加"/>增加
</a>
```

在 webtest/templates/webcase_manage.html 中加入如下内容。

```
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />
<script type="text/javascript" src="/admin/jsi18n/"></script>
<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>
<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>
<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>
<script type="text/javascript" src="/static/admin/js/urlify.js"></script>
<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>
<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">
<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />
<meta name="robots" content="NONE,NOARCHIVE" />

<!-- 增加 web 测试用例-->
<div style="float:right;width:73%">

<select name="Webcase" id="id_Webcase">
<option value="" selected>----web 测试用例----</option>
</select>
<a class="related-widget-wrapper-link change-related" id="change_id_Webcase" data-href-
template="/admin/webtest/webcase/__fk__/change/?_to_field=id&amp;_popup=1" title="更改选中的测试用例">
<img src="/static/admin/img/icon-changelink.svg" alt="修改"/>
</a>
```

```
<a class="related-widget-wrapper-link add-related" id="add_id_Webcase"
href="/admin/webtest/webcase/add/?_to_field=id&amp;_popup=1" title="增加另一个测试用例">
<img src="/static/admin/img/icon-addlink.svg" alt="增加"/>增加
</a>
```

在 product/template/product_manage.html 中加入如下内容。

```
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />
<script type="text/javascript" src="/admin/jsi18n/"></script>
<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>
<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>
<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>
<script type="text/javascript" src="/static/admin/js/urlify.js"></script>
<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>
<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">
<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />
<meta name="robots" content="NONE,NOARCHIVE" />

<!-- 增加产品-->
<div style="float:right;width:73%">
<select name="Product" id="id_Product">
<option value="" selected>----产品----</option>
</select>
<a class="related-widget-wrapper-link change-related" id="change_id_Product" data-href-
template="/admin/product/product/__fk__/change/?_to_field=id&amp;_popup=1" title="更改选中的产品">
<img src="/static/admin/img/icon-changelink.svg" alt="修改"/>
</a>
<a class="related-widget-wrapper-link add-related" id="add_id_Product"
href="/admin/product/product/add/?_to_field=id&amp;_popup=1" title="增加另一个 产品">
<img src="/static/admin/img/icon-addlink.svg" alt="增加"/>增加
</a>
```

在 set/templates/set_manage.html 中加入如下内容。

```
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />
<script type="text/javascript" src="/admin/jsi18n/"></script>
<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>
<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>
<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>
<script type="text/javascript" src="/static/admin/js/urlify.js"></script>
<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>
<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">
<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />
<meta name="robots" content="NONE,NOARCHIVE" />

<!-- 增加系统设置-->
<div style="float:right;width:73%">
<select name="Set" id="id_Set">
<option value="" selected>----设置名称----</option>
</select>
<a class="related-widget-wrapper-link change-related" id="change_id_Set" data-href-
template="/admin/set/set/__fk__/change/?_to_field=id&amp;_popup=1" title="更改选中的系统设置">
<img src="/static/admin/img/icon-changelink.svg" alt="修改"/>
</a>
```

```
<a class="related-widget-wrapper-link add-related" id="add_id_Set"
href="/admin/set/set/add/?_to_field=id&amp;_popup=1" title="增加另一个系统设置">
<img src="/static/admin/img/icon-addlink.svg" alt="增加"/>增加
</a>
```

在 bug/templates/bug_manage.html 中加入如下内容。

```
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />
<script type="text/javascript" src="/admin/jsi18n/"></script>
<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>
<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>
<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>
<script type="text/javascript" src="/static/admin/js/urlify.js"></script>
<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>
<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">
<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />
<meta name="robots" content="NONE,NOARCHIVE" />

<!-- 增加 bug-->
<div style="float:right;width:73%">
<select name="Bug" id="id_Bug">
<option value="" selected>----bug 名称----</option>
</select>
<a class="related-widget-wrapper-link change-related" id="change_id_Set" data-href-
template="/admin/bug/bug/__fk__/change/?_to_field=id&amp;_popup=1" title="更改选中的 bug">
<img src="/static/admin/img/icon-changelink.svg" alt="修改"/>
</a>
<a class="related-widget-wrapper-link add-related" id="add_id_Set"
href="/admin/bug/bug/add/?_to_field=id&amp;_popup=1" title="增加另一个系统设置">
<img src="/static/admin/img/icon-addlink.svg" alt="增加"/>增加
</a>
</form>
</div>

对于用户管理页面，在 set/set_user.html 中相应位置加入如下内容。
<!-- 增加用户-->
<div style="float:right;width:73%">
<select name="User" id="id_User">
<option value="" selected>----用户列表----</option>
</select>
<a class="related-widget-wrapper-link change-related" id="change_id_User" data-href-
template="/admin/auth/user/__fk__/change/?_to_field=id&amp;_popup=1" title="更改选中的用户">
<img src="/static/admin/img/icon-changelink.svg" alt="修改"/>
</a>
<a class="related-widget-wrapper-link add-related" id="add_id_User"
href="/admin/auth/user/add/?_to_field=id&amp;_popup=1" title="增加另一个用户">
<img src="/static/admin/img/icon-addlink.svg" alt="增加"/>增加
</a>
```

到此，添加数据功能基本完成。

3.12.7 编辑数据功能实现

如图 3.75 所示。

ID	产品名称	产品描述	产品负责人	创建时间	编辑
2	商城	图书	邹辉	2018年1月30日 20:00	
3	app产品	计算器，Csdn	邹辉	2018年1月29日 13:16	
10	自动化平台	包括API、WebUI、AppUI自动化测试	邹辉	2018年1月30日 10:33	
11	web产品	百度搜索	邹辉	2018年1月29日 13:09	

▲图 3.75

1．单一接口测试用例页面

在 apitest/templates/apis_manage.html 中加入如下内容。

```
<th>编辑</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Apis"
href="../admin/apitest/apis/{{ apis.id  }}/change/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-changelink.svg"/></a></td>
```

2．流程接口测试用例页面

在 apitest/templates/apitest_manage.html 中加入如下内容。

```
<th>编辑</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add id Apitest"
href="../admin/apitest/apitest/{{ apitest.id  }}/change/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-changelink.svg"/></a></td>
```

3．产品管理页面

在 product/templates/product_manage.html 中加入如下内容。

```
<th>编辑</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Product"
href="../admin/product/product/{{ product.id  }}/change/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-changelink.svg"/></a></td>
```

4．App 用例管理页面

在 apptest/templates/appcase_manage.html 中加入如下内容。

```
<th>编辑</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Appcase"
href="../admin/apptest/appcase/{{ appcase.id  }}/change/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-changelink.svg"/></a></td>
```

5．Web 用例管理页面

在 webtest/templates/webcase_manage.html 中加入如下内容。

```
<th>编辑</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Webcase"
href="../admin/webtest/webcase/{{ webcase.id  }}/change/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-changelink.svg"/></a></td>
```

6．系统设置页面

在 set/templates/set_manage.html 中加入如下内容。

```
<th>编辑</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Set"
href="../admin/set/set/{{ set.id  }}/change/?_to_field=id&_popup=1"><img src="/static/admin/img/icon-
changelink.svg"/></a></td>
```

7．Bug 管理页面

在 bug/templates/bug_manage.html 中加入如下内容。

```
<th>编辑</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Bug"
href="../admin/bug/bug/{{ bug.id  }}/change/?_to_field=id&_popup=1"><img src="/static/admin/img/icon-
changelink.svg"/></a></td>
```

8．用户管理页面

在 set/templates/set_user.html 中相应位置加入如下内容。

```
<th>编辑</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add id User"
href="../admin/auth/user/{{ user.id  }}/change/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-changelink.svg"/></a></td>
```

3.12.8 删除数据功能实现

如图 3.76 所示。

ID	产品名称	产品描述	产品负责人	创建时间	编辑	删除
2	商城	图书	邹辉	2018年1月30日 20:00		
3	app产品	计算器，Csdn	邹辉	2018年1月29日 13:16		
10	自动化平台	包括API、WebUI、AppUI自动化测试	邹辉	2018年1月30日 10:33		
11	web产品	百度搜索	邹辉	2018年1月29日 13:09		

▲图 3.76

1．单一接口测试用例页面

在 apitest/templates/apis_manage.html 中分别加入如下内容。

```
<th>删除</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Apis"
href="../admin/apitest/apis/{{ apis.id  }}/delete/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-deletelink.svg"/></a></td>
```

2．流程接口测试用例页面

在 apitest/templates/apitest_manage.html 中加入如下内容。

```
<th>删除</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Apitest"
href="../admin/apitest/apitest/{{ apitest.id  }}/delete/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-deletelink.svg"/></a></td>
```

在 product/templates/product_manage.html 中加入如下内容。

```
<th>删除</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Product"
href="../admin/product/product/{{ apitest.id  }}/delete/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-deletelink.svg"/></a></td>
```

3．App 用例管理页面

在 apptest/templates/appcase_manage.html 中加入如下内容。

```
<th>删除</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Appcase"
href="../admin/apptest/appcase/{{ appcase.id  }}/delete/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-deletelink.svg"/></a></td>
```

4．Web 用例管理页面

在 webtest/templates/webcase_manage.html 中加入如下内容。

```
<th>删除</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Webcase"
href="../admin/webtest/webcase/{{ webcase.id  }}/delete/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-deletelink.svg"/></a></td>
```

5．系统设置页面

在 set/templates/set_manage.html 中加入如下内容。

```
<th>删除</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Set"
href="../admin/set/set/{{ set.id  }}/delete/? to field=id& popup=1"><img src="/static/admin/img/icon-
deletelink.svg"/></a></td>
```

6．Bug 管理页面

在 bug/templates/bug_manage.html 中加入如下内容。

```
<th>删除</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Bug"
href="../admin/bug/bug/{{ bug.id  }}/delete/?_to_field=id&_popup=1"><img src="/static/admin/img/icon-
deletelink.svg"/></a></td>
```

7．用户管理页面

在 set/templates/set_user.html 中加入如下内容。

```
<th>删除</th>
```

```
<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_User"
href="../admin/auth/user/{{ user.id  }}/delete/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-deletelink.svg"/></a></td>
```

3.12.9 主页面优化 2

继续进行优化，开发一个登录成功后的默认首页。

在 apitest/templates/新建 welcome.html 中加入如下内容。

```
<html lang="zh-CN">
<head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
<title>autotestplat-自动化平台测试开发</title>
<link href="../static/css/bootstrap.min.css" rel="stylesheet">
<style> body{text-align:center} </style>
</head>
<body role="document">
<!-- 导航栏-->
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
<div class="container">
<a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">
</ul>
<ul class="nav justify-content-end">
<li class="nav-link"><a style='color:white' href="#">{{user}}</a></li>
<li class="nav-link"><a style='color:white' href="/logout/">退出</a></li>
</ul>
</div>
</nav>
<div class="row" style="padding-top: 1px;">
<div class="container">
<font size="8">
<h1>欢迎您，来到 autotestplat V1.0</h1>
</font>
</div></div>
<div class="row" style="padding-top: 23px;">
<div class="container">
<tr>
<td><a href="../test_report" target="mainFrame">单击进入测试报告</a></td>
</tr>
</div></div>
</body>
</html>
```

在 apitest/views.py 中加入如下内容。

```
def welcome(request):
    return render(request,"welcome.html")
```

在 autotest/urls.py 中加入如下内容。

```
  path ('welcome/', views.welcome),
```

把 home.html 中内容修改为如下内容。

```
<frame src="../welcome" name="mainFrame" scrolling="NO"  noresize>
```

把 apitest/templates/left.html 中内容修改为如下内容。

```
<html>
```

```
    <head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
        <title>autotestplat</title>
    </head>
    <body>
    <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
<div class="container">
        <a class="navbar-brand" href="../welcome" target="mainFrame">《自动化平台测试开发》书</a>
</div>
        </nav>
        <table border="0">
               <tr><td> </td></tr>
               <tr><td> </td></tr>
               <tr><td> </td></tr>
<tr>
                <td><a href="../product_manage" target="mainFrame">  产品中心</a></td>
            </tr>
                <tr><td> </td></tr>
          <tr>
                 <td>----------------------------------------</a></td></tr>
          <tr>
                <td><a href="../bug_manage" target="mainFrame">  bug 管理</a></td>
            </tr>
               <tr><td> </td></tr>
        <tr>
                <td>----------------------------------------</a></td></tr>
            <tr>
                <td><a href="../apis_manage" target="mainFrame">  用例管理</a></td>
            </tr>

               <tr><td> </td></tr>
            <tr><td>----------------------------------------</a></td></tr>

            </tr>
               <tr>
                <td><a href="http://127.0.0.1:8080" target="mainFrame">  计划任务</a></td>
            </tr>
               <tr><td> </td></tr>
               <tr>
                <td>----------------------------------------</a></td>
            </tr>
            <tr>
                <td><a href="../test_report" target="mainFrame">  测试报告</a></td>
            </tr>

                 <tr><td> </td></tr>
          <tr>
                <td>----------------------------------------</a></td>
            </tr>
            <tr>
                <td><a href="../set_manage" target="mainFrame">  系统设置</a></td>
            </tr>
        </table>
    </body>
</html>
```

刷新查看前端默认首页，如图 3.77 所示。

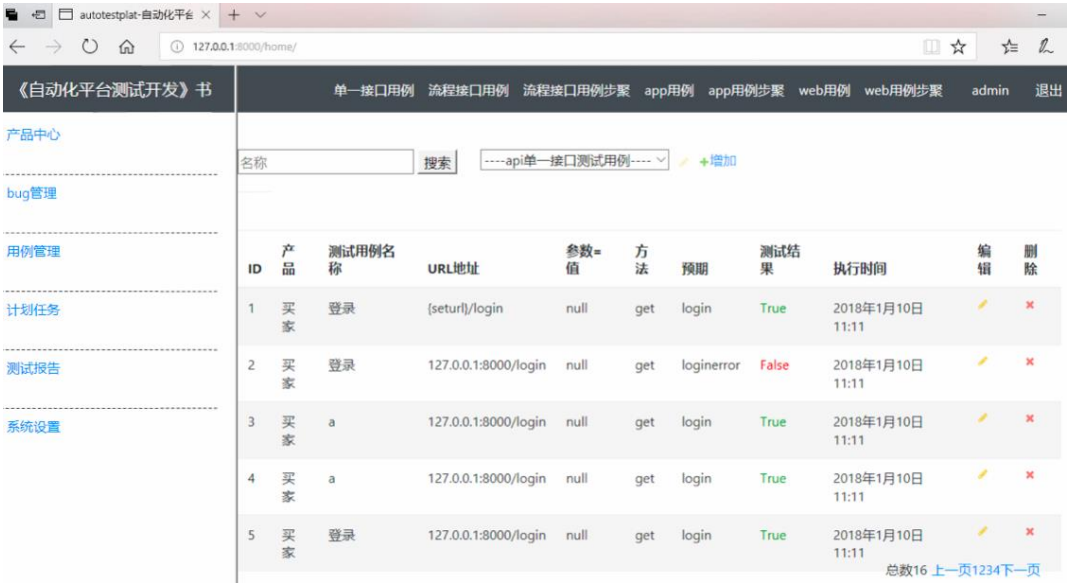
▲图 3.77

优化 Django 后台的标题，把默认的 Django 站点管理员和标题改为 autotestplat。

在 apitest/admin.py 中加入如下内容。

```
admin.site.site_title = 'AutotestPlat'
admin.site.site_header = 'AtotestPlat'
```

前端优化已初步完成，效果如图 3.78 所示。



▲图 3.78

3.12.10 主页面优化 3

现在的页面已经比较简洁、美观、大方了，还可以再加入 Bootstrap 的 CSS 和 JS 继续优化。实现左侧菜单折叠功能，实现对 API、AppUI、WebUI 分开管理以及展示。

步骤 1 从官网上下载相应版本的 bootstrap.min.css、jquery.min.js、bootstrap.min.js 文件，以及 fonts 文件夹。Bootstrap 官网下载地址为 http://v3.bootcss.com/getting-started/# download。在 manage.py 同级目录下新建 static 文件夹，在 static 目录下新建 CSS、JS、fonts 文件夹，把上面的文件分类放入相应类型的文件夹中。

步骤 2 在 settins.py 中加入如下内容。

```
STATIC_URL = '/static'
HERE = os.path.dirname(os.path.abspath(__file__))
HERE = os.path.jion(HERE,../')
STATICFILES_DIRS = (os.path.join(HERE,'static/'),)
```

步骤 3 修改 apitest/left.html 中的内容。

```
<html>
<head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
<link href="../static/css/bootstrap.min.css" rel="stylesheet">
<script src="../static/js/jquery.min.js"></script>
<script src="../static/js/bootstrap.min.js"></script>
   <title>autotestplat</title>
    </head>
    <body>
    <nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
        <a style='color:white' href="../welcome" target="mainFrame">《自动化平台测试开发》书</a>
        </nav>

<!-- <table border="0">glyphicon glyphicon-credit-card-->
 <div class="container-fluid">

<tr> <td>
                    <li>
                        <a href="../product manage" target="mainFrame" style="font-size:18px">
                            <i class="glyphicon glyphicon-gift"></i>
                           产品中心
                        </a>
                    </li>
</tr> </td>

         <tr><td> </td></tr>

      <tr>  <td>
                <li>
                        <a href="#caseManage" class="nav-header collapsed" data-toggle="collapse"
style="font-size:18px">
                            <i class="glyphicon glyphicon-star"></i>
                            用例管理
                         <!--       <span class="pull-right glyphicon glyphicon-chevron-down"></span>-
->
                        </a>
                        <ul id="caseManage" class="nav nav-list collapse secondmenu" style="height:
0px;">
                            <li><a href="../apis_manage"
target="mainFrame">             <i
class="glyphicon glyphicon-star"></i>测试用例-api</a></li>
                            <li><a href="../appcase_manage"
target="mainFrame">             <i
class="glyphicon glyphicon-star"></i>测试用例-app</a></li>
                            <li><a href="../webcase_manage"
target="mainFrame">             <i
class="glyphicon glyphicon-star"></i>测试用例-web</a></li>
                        </ul>
                    </li>
                    </tr> </td>

            <tr><td> </td></tr>
<tr> <td>
                    <li>
                        <a  href="../periodic_task" target="mainFrame" style="font-size:18px">
                            <i class="glyphicon glyphicon-fire"></i>
```

```
                        定时任务
                    </a>
                </li>
</tr> </td>

             <tr><td> </td></tr>

<tr> <td>
                <li>
                    <a  href="../bug_manage" target="mainFrame" style="font-size:18px">
                        <i class="glyphicon glyphicon-eye-open"></i>
                        bug 管理
                    </a>
                </li>
</tr> </td>

            <tr><td> </td></tr>

<tr> <td>
                <li>
                    <a  href="../test_report" target="mainFrame" style="font-size:18px">
                        <i class="glyphicon glyphicon-cog"></i>
                       测试报告
                    </a>
                </li>
</tr> </td>

            <tr><td> </td></tr>

<tr> <td>
                <li>
                    <a  href="../set_manage" target="mainFrame" style="font-size:18px">
                        <i class="glyphicon glyphicon-cog"></i>
                       系统设置
                    </a>
                </li>
</tr> </td>

         <tr><td> </td></tr>


    </div>
   <!--</table>-->

    </body>
</html>
```

刷新查看首页效果，如图 3.79 所示。

▲图 3.79

点开折叠菜单效果，如图 3.80 所示。

▲图 3.80

3.12.11 主页面优化 4

发现页面还是不够简洁，风格也不统一，于是继续进一步优化。

把左侧主菜单用例管理放到一个链接中，即把单一接口、流程接口、App 用例、Web 用例都放到这个菜单展示

步骤 1 在 apitest/left.html 中将原用例管理修改为如下内容。

```
<tr>  <td>
                <li>
                       <a  href="../apis_manage" target="mainFrame" style="font-size:18px">
                           <i class="glyphicon glyphicon-star"></i>
                           用例管理
                       </a>
                </li>
</tr> </td>
```

步骤 2 在 apitest/apis_manage.html、apitest_manage.html、apistep_manage.html、apptest/appcase_manage.html、appcasestep_manage.html、webtest/webcase_manage.html 和 webcasestep_manage.html 中分别修改统一导航栏。

```
<li class="active"><a style='color:white' href="/apis_manage/">单一接口</a></li>
    
<li><a style='color:white' href="/apitest_manage/">流程接口</a></li>
    
<li class="active"><a style='color:white' href="/appcase_manage/">app 用例</a></li>
    
<li class="active"><a style='color:white' href="/webcase_manage/">web 用例</a></li>
    
```

步骤 3 修改 apitest/apistep_manage.html、apptest/appcasestep_manage.html 和 webtest/ webcasestep_manage.html，分别在 header 中加入如下内容。

```
<link href="../static/css/bootstrap.min.css" rel="stylesheet">
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />
<script type="text/javascript" src="/admin/jsi18n/"></script>
<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>
```

```
<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>
<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>
<script type="text/javascript" src="/static/admin/js/urlify.js"></script>
<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>
<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">
<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />
<meta name="robots" content="NONE,NOARCHIVE" />
<style> body{text-align:center} </style>
```

把流程接口子用例放到主用例链接详情中展示。

步骤 1 在 apitest/views.py 中，把原 apistep_mange 函数内容修改为如下内容。

```
# 接口步骤管理
@login_required
def apistep_manage(request):
    username = request.session.get('user', '')
    apitestid = request.GET.get('apitest.id',None)
    apitest = Apitest.objects.get(id=apitestid)
    apistep_list = Apistep.objects.all()
    return render(request, "apistep_manage.html",  {"user": username,"apitest": apitest,"apisteps":
apistep_list})
```

步骤 2 在 urls.py 中增加如下内容。

```
name='apistep_manage'
   path('apistep_manage/', views.apistep_manage,name='apistep_manage'),
```

步骤 3 在 apitest/apitest_manage.html 中增加详情、查看及链接字段。

```
<th>查看</th>

<td><a href="{% url "apistep_manage" %}?apitest.id={{ apitest.id  }} "><img
src="/static/admin/img/icon-yes.svg"/></a></td>
```

步骤 4 在 apitest/apistep_manage.html 中增加 if 语句。

```
{% for apistep in apisteps %}
<tr>
{% if apistep.Apitest.id == apitest.id  %}
<td>{{ apistep.Apitest.Product.productname }}</td>
<td>case_{{ apistep.Apitest.id }}_{{ apistep.Apitest.apitestname }}</td>
<td>{{ apistep.apistep }}:{{ apistep.apiname }}</td>
<td>{{ apistep.apiurl }}</td>
<td>{{ apistep.apiparamvalue }}</td>
<td>{{ apistep.apimethod }}</td>
<td>{{ apistep.apiresult }}</td>
<td>{% if apistep.apistatus == 1 %}
<a style='color:green'>{{ apistep.apistatus }}</a>
{% else %}
<a style='color:red'>{{ apistep.apistatus }}</a>
{% endif %}
</td>
<td>{{ apistep.create_time }}</td>
{% else %}
{% endif %}
```

步骤 5 用例步骤页增加返回和编辑按钮，在 apitest/apistep_manage.html 的 body 中加入如下内容。

```
<td>
<a style='color:light blue'  href="../apitest_manage/" >
```

```
<img src="/static/admin/img/icon-yes.svg" alt="返回"/>返回
</a>
</td>
<td>
<a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Apitest"
href="../admin/apitest/apitest/{{ apitest.id  }}/change/?_to_field=id&_popup=1">
<img src="/static/admin/img/icon-changelink.svg"/>编辑</a>
</td>
```

步骤 6 前端查看流程接口页面，如图 3.81 所示。

▲图 3.81

步骤 7 单击相应用例“查看”列下面的图标，用例详情如图 3.82 所示。

▲图 3.82

把 App 用例步骤放到主用例链接详情中展示。

步骤 1 在 apptest/appviews.py 中，把原 appcasestep_mange 函数修改为如下内容。

```
# 步骤管理
@login_required
def appcasestep_manage(request):
    username = request.session.get('user', '')
    appcasestep_list = Appcasestep.objects.all()
    appcaseid = request.GET.get('appcase.id',None)
    appcase = Appcase.objects.get(id=appcaseid)
    appcasestep_list = Appcasestep.objects.all()
return render(request, "appcasestep_manage.html", {"user": username,"appcase": appcase,"appcasesteps":
appcasestep_list})
```

步骤 2 在 urls.py 中增加如下内容。

```
name='appcasestep_manage'
   path('appcasestep_manage/', appviews.appcasestep_manage,name='appcasestep_manage'),
```

步骤 3 在 apptest\templates\appcase_manage.html 中增加详情，查看及链接字段。

```
<th>查看</th>
```

```
<td><a href="{% url "appcasestep_manage" %}?appcase.id={{ appcase.id  }} "><img
src="/static/admin/img/icon-yes.svg"/></a></td>
```

步骤 4 在 apptest\templates\appcasestep_manage.html 中增加 if 语句。

```
{% for appcasestep in appcasesteps %}
<tr>
{% if appcasestep.Appcase.id == appcase.id  %}
<td>{{ appcasestep.Appcase.Product.productname }}</td>
<td>case_{{ appcasestep.Appcase.id }}_{{ appcasestep.Appcase.appcasename }}</td>
<td>{{ appcasestep.appteststep }}:{{ appcasestep.apptestobjname }}</td>
<td>{{ appcasestep.appfindmethod }}</td>
<td>{{ appcasestep.appevelement }}</td>
<td>{{ appcasestep.appoptmethod }}</td>
<td>{{ appcasestep.apptestdata }}</td>
<td>{{ appcasestep.appassertdata }}</td>
<td>{% if appcasestep.apptestresult == 1 %}
<a style='color:green'>{{ appcasestep.apptestresult }}</a>
{% else %}
<a style='color:red'>{{ appcasestep.apptestresult }}</a>
{% endif %}
</td>
<td>{{ appcasestep.create_time }}</td>
{% else %}
{% endif %}
</tr>
{% endfor %}
```

步骤 5 用例步骤页增加返回和编辑按钮，在 templates\appcasestep_manage 的 body 中加入如下内容。

```
<td>
<a style='color:light blue'  href="../appcase_manage/" >
<img src="/static/admin/img/icon-yes.svg" alt="返回"/>返回
</a>
</td>

<td>
<a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Appcase"
href="../admin/apptest/appcase/{{ appcase.id  }}/change/?_to_field=id&_popup=1">
<img src="/static/admin/img/icon-changelink.svg"/>编辑</a>
</td>
```

步骤 6 前端查看 App 用例页面，如图 3.83 所示。

▲图 3.83

步骤 7 单击相应用例“查看”列下面图标，用例详情如图 3.84 所示。

▲图 3.84

把 Web 用例步骤放到主用例链接详情中展示。

步骤 1 在 webtest/webviews.py 中，把原 webcasestep_mange 函数修改为如下内容。

```
# Web 用例测试步骤
@login_required
def webcasestep_manage(request):
    username = request.session.get('user', '')
    webcaseid = request.GET.get('webcase.id',None)
    webcase = Webcase.objects.get(id=webcaseid)
    webcasestep_list = Webcasestep.objects.all()
    return render(request, "webcasestep_manage.html", {"user": username,"webcase":
webcase,"webcasesteps": webcasestep_list})
```

步骤 2 在 urls.py 中增加如下内容。

```
name='webcasestep_manage'
   path('webcasestep_manage/', webviews.webcasestep_manage,name='webcasestep_manage'),
```

步骤 3 在 webtest\templates\webcase_manage.html 中增加详情、查看及链接字段。

```
<th>查看</th>

<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Webcase"
href="../admin/webtest/webcase/{{ webcase.id  }}/change/? to field=id& popup=1"><img
src="/static/admin/img/icon-changelink.svg"/></a></td>
```

步骤 4 在 webtest\templates\webcasestep_manage.html 中增加 if 语句。

```
{% for webcasestep in webcasesteps %}
 <tr>
{% if webcasestep.Webcase.id == webcase.id  %}
 <td>{{ webcasestep.Webcase.Product.productname }}</td>
 <td>case_{{ webcasestep.Webcase.id }}_{{ webcasestep.Webcase.webcasename }}</td>
 <td>{{ webcasestep.webteststep }}:{{ webcasestep.webtestobjname }}</td>
 <td>{{ webcasestep.webfindmethod }}</td>
 <td>{{ webcasestep.webevelement }}</td>
 <td>{{ webcasestep.weboptmethod }}</td>
 <td>{{ webcasestep.webtestdata }}</td>
 <td>{{ webcasestep.webassertdata }}</td>
 <td>{{ webcasestep.webtestresult }}</td>
 <td>{{ webcasestep.create_time }}</td>
{% else %}
{% endif %}
 </tr>
```

```
{% endfor %}
```

步骤 5 用例步骤页增加返回和编辑按钮，在 webtest\templates\webcasestep_manage.html 的 body 中加入如下内容

```
<td>
<a style='color:light blue'  href="../webcase_manage/" >
<img src="/static/admin/img/icon-yes.svg" alt="返回"/>返回
</a>
</td>

<td>
<a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Webtest"
href="../admin/webtest/webcase/{{ webcase.id  }}/change/?_to_field=id&_popup=1">
<img src="/static/admin/img/icon-changelink.svg"/>编辑</a>
</td>
```

步骤 6 前端查看流程接口页面如图 3.85 所示。

▲图 3.85

步骤 7 单击相应用例“查看”列下面的图标，用例详情如图 3.86 所示。

平台的优化暂时告一段落，本书第 6、7、8 章会介绍在自动化平台的基础上进行接口 API、AppUI、WebUI 自动化测试脚本编写、调试、测试执行、报告等工作内容。

▲图 3.86

第 4 章

正则表达式

学习和掌握正则表达式并不容易，因为它的语法结构很独特，并且不易理解和记忆，但是掌握了原理就很简单。

4.1 为什么要用正则表达式

在软件开发、性能测试、自动化测试、测试开发中都可以看到正则表达式的身影，可以说掌握正则表达式对程序员或自动化测试来说都是必不可少的。

对于静态文本，因为有提供与预期的搜索结果匹配的确切文本内容，所以典型的搜索和替换操作已经足够了但它缺乏灵活性。如果要搜索动态文本，就变得很困难，而用正则表达式能很容易地解决这一问题。

正则表达式的格式，就是在预期的搜索结果中，进行左匹配和右匹配。这两个是已知的，也是固定的。然后中间的是需要的数据，根据这个数据的格式，用相应的正则表达式进行替换，并用参数化对其进行赋值。因而，参数化就代表了需要搜索的结果，动态值可以应用到相关的地方，详见实例中的注释部分。

总而言之，使用正则表达式可以很方便地过滤，筛选出需要的特定数据信息。

4.2 正则表达式元字符及其作用

如表 4.1 所示。

▼表 4.1

常用元字符	作　用
.	代表任何字符
^	匹配的起始位置

续表

常用元字符	作　用
\	开始对下一个字符取非
$	匹配字符串结束
[]	匹配方括号内任意字符
-	定义区间，匹配区间内字符或数字

4.3 正则表达式字符串匹配示例

如表 4.2 所示。

▼表 4.2

正则表达式字符串	说　明
"^\d+$"	用于匹配非负整数（正整数 + 0）
"^[0-9]*[1-9][0-9]*$"	用于匹配正整数
"^((-\d+)|(0+))$"	用于匹配非正整数（负整数 + 0）
"^-[0-9]*[1-9][0-9]*$"	用于匹配负整数
"^-?\d+$"	用于匹配整数
"[x{4e00}-x{9fa5}]+/u"	用于匹配汉字中文
"^d{15}|d{}18$"	用于匹配身份证号（15 位或 18 位数字）
"^((13[0-9])|(15[^4,\\D])|(18[0,0-9]))\\d{8}$"	用于匹配手机号
"^(-?\d+)(\.\d+)?$"	用于匹配浮点数
"^[A-Za-z]+$"	用于匹配由 26 个英文字母组成的字符串
"^[A-Z]+$"	用于匹配由 26 个英文字母的大写组成的字符串
"^[a-z]+$"	用于匹配由 26 个英文字母的小写组成的字符串
"^[A-Za-z0-9]+$"	用于匹配由数字和 26 个英文字母组成的字符串
"^\w+$"	用于匹配由数字、26 个英文字母或者下画线组成的字符串
"^[a-zA-z]+://(\w+(-\w+)*)(\.(\w+(-\w+)*))*(\?\S*)?$"	用于匹配 URL
/^(d{2}|d{4})-((0([1-9]{1}))|(1[1|2]))-(([0-2]([1-9]{1}))|(3[0|1]))$/	用于匹配年月日

续表

正则表达式字符串	说　明
"^([w-.]+)@((([[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.)|(([w-]+.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(]?)$"	用于匹配 E-mail
"(d+-)?(d{4}-?d{7}|d{3}-?d{8}|^d{7,8})(-d+)?"	电话号码
"^(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5])$"	IP 地址

4.4 Python 正则表达式使用介绍

Python 通过 re 模块对正则表达式进行操作，使用时需引入 import re。

首先，使用函数 complie 将正则表达式编译成 pattern 对象，然后，使用 pattern 对象匹配文本，获得需要的结果。

常用的实例方法如表 4.3 所示。

▼表 4.3

方　法	说　明
Search	匹配所有字符串并返回找到一个匹配
Match	只在字符串的开始位置尝试匹配
Split	将字符串匹配正则表达式的部分分割开，并返回一个列表
Findall	查找所有匹配
Sub	查找并替换

4.5 正则表达式源码详解

4.5.1 正则表达式实例 1

定义 UTF-8 支持中文，导入引用 re 模块，定义 TaskId 函数，在里面定义 TaskId 全局变量。定义正则表达式语句，使用 re 模块的 search 方法与参数 results 和正则表达式进行全字符串的匹配，返回一个 pattern 变量为 pm。如果 pm 为 true，则把值分组赋值给全局变量 TaskId，并返回 TaskId；否则返回 False 定义主函数，主函数中定义参数变量并将一段字符串赋值给变量 results。调用 TaskId 方法，并把正则表达式所返回的值赋给变量 value，打印 value。把 byte 类型转换成 UTF-8 输出，得到数据 000123456。

```
# -*- coding: UTF-8 -*-
import re

def TaskId(results):
    global TaskId
    regx = '.*"TaskId":(.*),"PlanId"'
    pm = re.search(regx, results)
    if pm:
        TaskId = pm.group(1).encode('utf-8')
______#  print (TaskId)
        return TaskId
    return False

if __name__ == '__main__':
    results = '"TaskId":000123456,"PlanId"'
    value=TaskId(results)
    print(value.decode('utf-8'))
    print ('Done!')
```

运行输出结果：

```
000123456
Done
```

4.5.2 正则表达式实例 2

token 在 App 登录后，相当于一个凭条，代表了该用户的唯一通行证，放到 header 里面传递至另外的接口。

```
----------接口以及返回的响应数据如下：----------
http://192.168.215.55/user/login.do?phone=13798359580&pwd=123456
{"msg":"成功
","data":{"birthday":"","sex":"0","cityId":"4524157","userLogo":"","provinceId":"4524130","token":"p:s
id:e3f9ff89eaf74a3ba208aa6ba74d00a44043","niceName":"test1234567","provinceName":"广东"},"state":0}
调用如下包含正则表达式的函数，则可以得到 token 为
p:sid:e3f9ff89eaf74a3ba208aa6ba74d00a44043
```

```
————————————基于 Python 的函数如下————————————
def GetToken():                                    #取用户登录的 token 值
    global token                                     #定义 token 全局变量
    url ='http://'+HOSTNAME+'/buyer/user/login.do' #接口的 URL
    params ={
       'phone': '13798359580',
        'pwd': '123456',
    }                                          #参数为登录手机号和密码
    request =urllib2.Request(url = url, data = urllib.urlencode(params))   #发送接口# 请求 URL 和参数
    response= urllib2.urlopen(request)                #返回响应数据
    data =response.read()
    regx ='.*"token":"(.*)","ud"'                      #正则表达式 token，左匹配 "token":"    右匹配","ud"'
    pm =re.search(regx, data)                      #取 token 匹配值
    token =pm.group(1)                     #如果匹配到，则返回 token 值
    returnFalse
```

4.5.3 正则表达式实例 3

购物提交订单时会生成一个订单编号，当进行接口自动化测试时，要根据这个订单编号进行支付。这时要动态取值，即利用正则表达式动态匹配需要的订单编号。

```
————————————接口以及返回的响应数据如下：————————————
 http://192.168.215.55/buyer/cart/submit.do?goodsids=20394
{"msg":"成功
","data":{"goodsStatusResult":1,"receiverInfo":{"id":"661","name":"test123","province":"4524130","city
":"4524157","district":"4524163","defaultDeliverySeq":"1","preOrderSN":"1000160_240_1","toHome":"0"}],
"totalFee":"359.00","totalCount":"1"}],"receipt":{"type":"2","typeName":"公司
","title":"test","receiptId":"70","receiptContent":""},"activityList":[{"activityId":"0","activityName
":"默认"}]},"state":0}
```

调用如下包含正则表达式的函数，则可以得到 preOrderSN 为：1000160_240_1。

```
————————————基于 Python 的函数如下————————————
#预提交订单参数取动态值，订单编号 preOrderSN，引用时比如 strinfo = re.compile('{preOrderSN}')
#————————————————————————————————。
def preOrderSN(results):
    global preOrderSN
# 预提交订单取值的正则表达式，左匹配"preOrderSN":" 右匹配"toHome"
    regx ='.*"preOrderSN":"(.*)","toHome"'
    pm =re.search(regx, results)
#————————————————————————————————。
    if pm:
 #———————如果匹配到，转换为中文并返回值————————————
       preOrderSN = pm.group(1).encode('utf-8')
       return preOrderSN
 #————————————————————————————————。
    returnFalse
```

第 5 章

单元测试

随着测试方法和测试过程的多样化，以及测试技术水平的提高，分层测试逐渐成为比较常见和易被接受的测试思路。Unit 层即单元测试应在分层测试中占大部分比例，也是最先进行的测试。

什么是单元测试，顾名思义就是对最小的一个单元代码或不受其他代码逻辑影响的函数等进行测试，对输入参数和输出都要求比较明确。单元测试的目的是把程序里每个独立的最小单元隔离开来，保证其正确性，使得整个程序相对也更正确。单元测试所占比例应最大，最有效率，最早被发现问题，最便于维护。单元测试隔离通常有 stub 和 mock 两种方法，单元测试框架 Unittest 也支持测试自动化。目前来看，单元测试通常由开发人员自测完成，甚至很多的企业并未开展，所以本书也不重点讲解，只做简单概述。

5.1 Unittest 单元测试

Unittest 基于 Python 的单元测试框架，同时也支持 Web 自动化测试、App 自动化测试、API 接口自动化测试等各种自动化测试。

因为 Unittest 是本书第 6～8 章的基础，所以在介绍基于 Python 语言的自动化测试之前，先介绍 Unittest 单元测试并作为一个独立的章节。

1．Test Fixture

需要执行的一个或多个测试等。

2．Test Case

Test Case 是单元测试中最小的单位。它校验输入所返回的响应，提供了基类 TestCase，可以用于创建新的测试用例，包括 test/test action、 test data、assert 等。

1）引入 Unittest。

2）继承基类 Unittest.TestCase。

- Teardown：退出清理等善后工作。

3．Test Suite

TestSuite 测试套件类是测试用例的集合，用于使测试用例被有组织、有规则地批量执行，支持添加 addTest()和删除，然后传递给 TestRunner 进行测试执行。

4．Test Runner

TestRunner 是以文本形式返回测试结果的测试用例执行的实例。

5．TestLoader

测试加载器，是测试流程的一个组成部分，包括加载多个测试用例的方法，返回测试套件，用于执行测试并给出测试结果。

单元测试的框架，第一步是 Test Loader 根据传入参数得到相应的测试用例和测试方法；第二步是通过 MakeSuite 把测试用例组装成 TestSuite，第三步是传递给 TestRunner 执行具体的测试。

6．程序实例

被测试模块及函数

```
#encoding: utf-8
Class unitclass:
   Def sum （x, y）
     Return x+y

#encoding: utf-8
Import unittest
Import unitclass
Class unittestsum（unittest.TestCase）:
   Def setUp（self）:

   Def testcase1（def）:
       Self.assertEqual（self.class.sum（1, 1）, 2, 'test sum success'）
   Def testcase2（def）:
       Self.assertEqual（self.class.sum（1, 1）, 2, 'test sum success'）
@unittest.skip('暂时跳过测试用例 3 的测试')
    def testcase3(self):
            Self.assertEqual（self.class.sum（1, 1）, 2, 'test sum success'）


If ——name——=='——main——':
    Unites.main（）
```

查看控制台运行结果。

5.2 Django 单元测试

Django 支持基于 Python 的单元测试，Django 继承 Python 的 unittest.TestCase，实现了 django.test.TestCase。编写测试用例通常从这里开始，其中测试代码放 tests.py 文件里。

1）测试页面的标题是否正确。在首页的 HTML 页面 title 中包含 login，在 apitest/test.py 中加入如下内容。

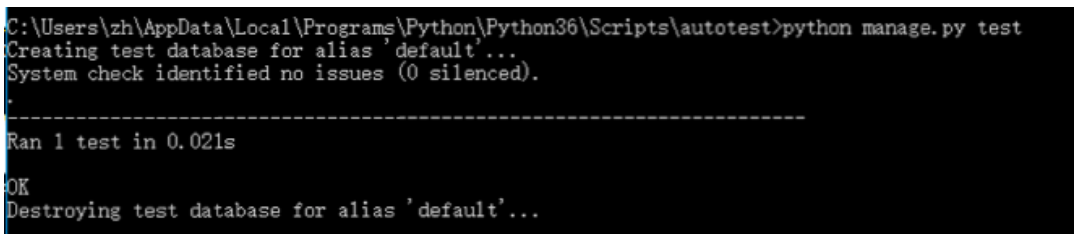
```
from apitest.views import test,home,login

from django.urls import reverse
```

```
from django.http import HttpRequest

class titlePageTest(TestCase):
    def test_loginpage_returns_title_html(self):
        request = HttpRequest()
        response = login(request)
        self.assertIn(b'<title>AutotestPlat</title>', response.content)
```

2）运行 python manage.py test。

```
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.021s

OK
Destroying test database for alias 'default'...
```

如图 5.1 所示，说明单元测试通过了。



▲图 5.1

把 apitest/test.py 中的测试函数改为如下内容。

```
class titlePageTest(TestCase):
    def test_loginpage_returns_title_html(self):
        request = HttpRequest()
        response = login(request)
        self.assertIn(b'<title>Logout</title>', response.content)
```

运行 python manage.py test。

```
C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest>python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
======================================================================
FAIL: test_page_returns_title_html (apitest.tests.titlePageTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts\autotest\apitest\tests.py", line
20, in test_page_returns_title_html
    self.assertIn(b'<title>Logout</title>', response.content)
AssertionError: b'<title>Logout</title>' not found in b'\xef\xbb\xbf<!DOCTYPE html>\n<html>\n<head>\n
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/> \n
<title>AutotestPlat</title>\n</head>\n<body>\n<h1>login</h1>\n\n<form method="post"
action="/login/">\n<input type=\'hidden\' name=\'csrfmiddlewaretoken\'
value=\'ZBUuCRbHhaCfiy9m94TH3KRCr7IIFMRX2CA0Fi2oZlzgU6VYispDfivKPAmg8rSU\' />\n<br> <input
name="username" type="text" placeholder="username" >\n<br> <input name="password" type="password"
placeholder="password">\n<br>\n<br> <button id="submit" type="submit">submit</button>
\n\n</form>\n</body>\n</html>'
```

```
----------------------------------------------------------------------
Ran 1 test in 0.024s

FAILED (failures=1)
Destroying test database for alias 'default'...
```

▲图 5.2

如图 5.2 所示，由此可见，断言的标题中不存在 logout，所以该用例测试失败。

测试页面内容如下。

```
class contentTest(TestCase):
    def test_content_url_resolves_to_view(self):
        request = HttpRequest()
        response = login(request)
        self.assertIn(b'test123456', response.content)
```

断言数据返回的响应数据中包含 byte 格式的 test123456。

第 6 章

接口自动化测试

Service 层和接口测试所占比例居中，模块之间的数据通信、联调、项目之间的拉通测试，都涉及接口测试。

首先要知道什么是被测对象接口，才能更好地对它进行测试。

6.1 接口概述

API 接口是一种传输或操作数据的方式，广泛应用于 App、服务端、Web 端等，适用于数据的获取、更新、删除以及其他操作。较常见的是 HTTP 接口和 WebService 接口，用得最多的是 HTTP 协议的 POST 接口和 GET 接口。

在项目开发过程中，很多时候专业的人做专业的事，开发人员往往前、后端分离，后台 API 和前端是专人负责。前端 App 端、Web 端、H5 等可共同调用后台 API 接口，以及前后端经常要进行连调，那么规范接口以及接口文档就显得尤为重要了。这里也包括了接口自动化测试人员，测试开发人员，他们都需要了解接口的概念。接口直接操作和返回数据，而前端开发工程师、 App 开发工程师、后端开发工程师和测试工程师在软件开发中也是对涉及数据大家都是要去清楚明白并应用。接口测试涉及数据的底层，相对于 UI 层，发现问题时更易于定位和修复不受前端频繁变动的影响。

6.1.1 接口示例

发送 HTTP 数据，请求方法为 Request，发送数据格式的参数值可以为 JSON。服务端处理请求后，返回数据，调用方法为 Response，返回数据格式为 JSON。HTTP 请求包括 Headers、URL、Params、Body 值等，例如，参数 1=值 2&参数 2=值 2https://passport.cnblogs.com/user/signin? ReturnUrl=http%3A%2F%2Fwww.cn blogs.com%2F。返回的数据基本都是 JSON 格式，例如：

```
{
     "msg": 系统繁忙，请稍后再试",
    "state":  "3720001"
}
```

接口的方法如表 6.1 所示。

Patch	接口修改数据的方法
Delete	接口删除数据的方法

6.1.2 接口工具

常见的开源接口调试和测试工具有 Postman、SoapUI、JMeter，抓包工具有 Fiddler，在 MacOS 操作系统上使用 Charles 以及浏览器。接口来源：一是开发人员提供 API 文档，二是用 Fiddler 等工具抓包。

6.1.3 JSON 数据

JSON 是一种数据展示方式，接口入参数据、响应数据普遍都采用 JSON 数据格式，另外，数据展示方式还有 XML、HTML、Excel、TXT 等。

JSON 的格式规范如下。

格式 1：

```
{
   'id':111,
   "ss":'abc',
}
```

格式 2

```
{
   'id':111,
   "ss":'abc',
   "data":{ "istrue":1,"length":5}
}
```

6.1.4 接口文档

接口 Wiki 的文档样例如图 6.1 所示，表示调用这个接口，能查询到相应的用户信息数据。

/buyer/user/find.do

URL

/buyer/user/find.do

HTTP请求方式

POST

请求参数

字段名	类型	必选	说明
phone	string	true	手机号码

返回结果

```
{
    "state":0,
    "msg":"ok",
    "data":{
        "isRegist":0,   // 是否注册。当state=0时为必须 ,1=已经注册 0=未注册 2=被禁用
    }
}
```

▲图 6.1

6.2 接口测试用例设计

接口测试也是功能测试，只不过测试的主要是数据，和界面无关。用例设计的思路和功能测试是一样的，比

如等价类、边界值、错误猜测、业务场景分析等。

接口测试用例设计的重点如下。

（1）状态检查：请求是否正确，比如默认请求成功是 200 或 success。如果请求错误，则返回 404 等错误。

（2）检查返回数据的正确性与格式：JSON 是一种常用的格式，也可以是 XML 格式。

（3）边界和异常扩展检查：

- 参数字段默认值。
- 参数字段是否必填、是否为空检查。
- 参数字段携带错误值。
- 接口字段多少的检查：5 个字段变成 8 个字段等。
- 字段类型的检查：Int 型变成 String 型时如何判断。
- 限制条件：如店铺名重复、店铺标签修改重复、短信验证码次数超过 5 次、店铺名长度超过 20 等。

（4）流程接口测试：比如购物流程，要依次调用登录接口、商品加入购物车接口、提交订单接口、支付接口同样，要依照这些接口的逻辑流程进行接口测试，通常前一个接口会动态地产生一个特定的数据关联到下一个接口。比如，登录接口后会有特定的 token，供接下来的购物等接口调用，然后提交订单接口会产生一个特定的 orderid，供下一个支付接口调用。

6.3 环境准备

安装接口测试所需组件：基于 Python 的 Request、MySQLdb。

安装环境：安装接口资源 Requests 包和数据库资源 MySQLdb 包。

1）安装 Requests。文件下载地址为 https://github.com/requests/requests。

- 解压缩 requests-master.zip。
- CMD 切换到相应目录文件下，执行 python setup.py install。

2）安装 PyMySQL。文件下载地址为 https://pypi.python.org/pypi/PyMySQL，解压缩安装 PyMySQL。

- 通过命令 CMD 切换到相应目录文件下，执行 python setup.py install。

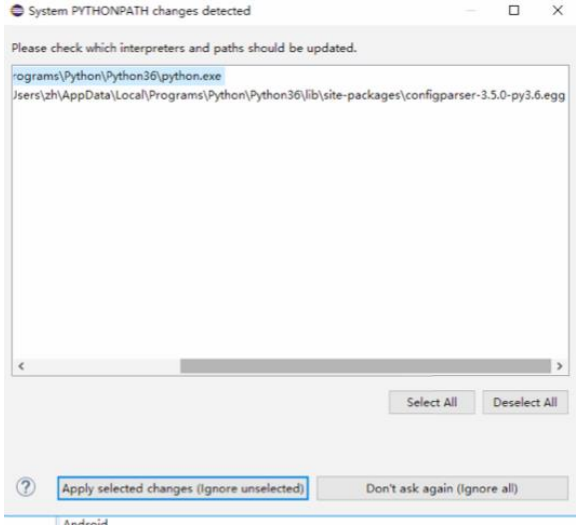
安装成功后，查看源码，发现 import pymysql 正常引用，如图 6.2 所示。

```
# -*- coding:utf-8 -*-
import pymysql
import urllib
import time,requests,re
```

▲图 6.2

3）安装 ConfigParser。文件下载地址为 https://pypi.python.org/pypi/configparser/。解压缩 configparser-3.5.0.tar.gz 通过运行 CMD 命令切换到相应目录文件下，执行 python setup.py install。

安装成功后，Eclipse 提示更新环境，如图 6.3 所示，单击“确定”按钮。



▲图 6.3

开始写 Python 代码，新建 Pydev 工程，写一个 hellowrold 程序并输出，安装环境后进行测试。

6.4 接口自动化测试源码详解

6.4.1 接口自动化测试实例 1

通过接口文档，或者抓包接口录入 Autotestplat 自动化平台。Web 端调用的接口可以通过按 F12 键在网络里抓取，如图 6.4 所示。

▲图 6.4

抓取到登录页面接口 URL 为 http://127.0.0.1:8000/login，接口方法为 Get，参数名和参数值为空。App 端调用的接口可以通过 Fiddler 等工具抓取。

功能描述：

- 引入与接口调用有关的 Request 和 Urllib 包。
- 校验函数并核对结果。
- 定义相关 URL、Headers 变量并赋值。
- 通过 Request 的 Get 等方法，发送接口数据。

- 定义变量 Results 赋值为接口，向服务器返回响应数据。
- 根据校验函数 readRes 的值核对结果并返回 pass 或 failed。

程序清单 1 apiauto_testcase1.py

```
# -*- coding:utf-8 -*-
import requests, time, sys, re
import urllib

def readRes(res,res_check):
    res = res.decode().replace('":"',"=").replace('":',"=")
    res_check = res_check.split(';')
    for s in res_check:
        if s in res:
            pass
        else:
            return  '错误，返回参数和预期结果不一致'+s
    return 'pass'

if __name__ == '__main__':
    url = 'http://'+'127.0.0.1:8000'+'/login'
    headers = {'Authorization':'', 'Content-Type': 'application/json' }
    data = None
    req = urllib.request.Request(url=url,data=data,headers=headers,method="GET")
    print('request is '+ url)
    try:
        results = urllib.request.urlopen(req).read()
        print ('response is get ')
        print(results)
    except Exception as e:
        print(' error ')
    res = readRes(results,'login')
    print(res)
    print ('Done!')
```

控制台打印运行结果。

```
request is http://127.0.0.1:8000/login
response is get
b'\xef\xbb\xbf<!DOCTYPE html>\n<html>\n<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />\n<head>\n     <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
\n    <title>AutotestPlat</title>\n<style> body{text-align:center} </style>
\n</head>\n<body>\n<div>\n<font
size="1">\n<h1>\xe3\x80\x8a\xe8\x87\xaa\xe5\x8a\xa8\xe5\x8c\x96\xe5\xb9\xb3\xe5\x8f\xb0\xe6\xb5\x8b\xe
8\xaf\x95\xe5\xbc\x80\xe5\x8f\x91\xe3\x80\x8b\xe4\xb9\xa6</h1>\n</font>\n<form method="post"
action="/login/">\n<input type=\'hidden\' name=\'csrfmiddlewaretoken\'
value=\'ruqdVhwyRFt6NYToSZDA2MH7yVXXZ8vbkM4QQGtvGiQLTjqDAIr0E3kop6pAGwgg\'
/>\n<br><a>  \xe7\x94\xa8\xe6\x88\xb7\xe5\x90\x8d\xef\xbc\x9a</a> <input name="username"
type="text" placeholder="test" >\n<br><br>  <a>  \xe5\xaf\x86   
\xe7\xa0\x81\xef\xbc\x9a</a> <input name="password" type="password"
placeholder="test123456">\n<br><br>\n  <button style="width:220px;height:28px;" id="submit"
type="submit">\xe7\x99\xbb \xe5\xbd\x95</button> \n</form>\n</div>\n</body>\n</html>'
pass
Done!
```

程序清单 2

```
# -*- coding:utf-8 -*-
import requests, time, sys, re
import urllib

def readRes(res,res_check):
    res = res.decode().replace('":"',"=").replace('":',"=")
```

```
    res_check = res_check.split(';')
    for s in res_check:
        if s in res:
            pass
        else:
            return  '错误，返回参数和预期结果不一致'+s
    return 'pass'

if __name__ == '__main__':
    url = 'http://'+'127.0.0.1:8000'+'/login'
    headers = {'Authorization':'', 'Content-Type': 'application/json' }
    data = None
    req = urllib.request.Request(url=url,data=data,headers=headers,method="GET")
    print('request is '+ url)
    try:
        results = urllib.request.urlopen(req).read()
        print ('response is get ')
        print(results)
    except Exception as e:
        print(' error ')
    res = readRes(results,' nologin')
    print(res)
    print ('Done!')
```

控制台打印运行结果。

```
request is http://127.0.0.1:8000/login
response is get
b'\xef\xbb\xbf<!DOCTYPE html>\n<html>\n<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />\n<head>\n     <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
\n    <title>AutotestPlat</title>\n<style> body{text-align:center} </style>
\n</head>\n<body>\n<div>\n<font
size="1">\n<h1>\xe3\x80\x8a\xe8\x87\xaa\xe5\x8a\xa8\xe5\x8c\x96\xe5\xb9\xb3\xe5\x8f\xb0\xe6\xb5\x8b\xe
8\xaf\x95\xe5\xbc\x80\xe5\x8f\x91\xe3\x80\x8b\xe4\xb9\xa6</h1>\n</font>\n<form method="post"
action="/login/">\n<input type=\'hidden\' name=\'csrfmiddlewaretoken\'
value=\'AjuBGdgtjAqfrlGT1jAmDi6MYVSFo8pZvgjF3EPaHv0rN7AQ6gJ6SPgN6Pg3nL3G\'
/>\n<br><a>  \xe7\x94\xa8\xe6\x88\xb7\xe5\x90\x8d\xef\xbc\x9a</a> <input name="username"
type="text" placeholder="test" >\n<br><br>  <a>  \xe5\xaf\x86   
\xe7\xa0\x81\xef\xbc\x9a</a> <input name="password" type="password"
placeholder="test123456">\n<br><br>\n  <button style="width:220px;height:28px;" id="submit"
type="submit">\xe7\x99\xbb \xe5\xbd\x95</button> \n</form>\n</div>\n</body>\n</html>'
错误，返回参数和预期结果不一致 nologin
Done!
```

6.4.2 接口自动化测试实例 2

1．单一接口测试用例执行

通过接口文档，或者抓包接口录入 AutotestPlat 自动化平台。Web 端调用的接口可以通过按 F12 键在网络里抓取，如图 6.5 所示。

▲图 6.5

抓取到登录页面接口 URL 为 http://127.0.0.1:8000/login，接口为 Get 方法，参数名和参数值为空。App 端调用的接口可以通过 Fiddler 等工具抓取。于是把捕获到的接口录入 AutotestPlat 自动化平台，如图 6.6 所示。

▲图 6.6

功能描述：执行所有单一接口测试用例，响应数据与校验字段值进行匹配，把测试结果记录到数据库中，1 表示通过，0 表示失败，并在前端显示如果 1 通过，则显示绿色 True，如果 0 失败，则显示红色 False。

代码清单：apiauto_testcase2.py。

```
# -*- coding:utf-8 -*-
import requests, time, re
import urllib
import pymysql

import json

#import fconfig

HOSTNAME = '127.0.0.1'

def test_readSQLcase():
        sql="SELECT id,`apiname`,apiurl,apimethod,apiparamvalue,apiresult,`apistatus` from
apitest_apis  "
        coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
        cursor = coon.cursor()
        aa=cursor.execute(sql)
        info = cursor.fetchmany(aa)
```

```
        print (info)
        for ii in info:
            case_list = []
            case_list.append(ii)
            interfaceTest(case_list)
        coon.commit()
        cursor.close()
        coon.close()

def interfaceTest(case_list):
    res flags = []
    request_urls = []
    responses = []
    strinfo = re.compile('{seturl}')
    for case in case_list:
        try:
            case id = case[0]
            interface_name = case[1]
            method = case[3]
            url = case[2]
            param = case[4]
            res_check = case[5]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        if param== '':
            new_url = 'http://'+'api.test.com.cn'+url
        elif param== 'null':
            url = strinfo.sub(str(seturl('seturl')),url)
            new_url = 'http://'+ url
        else:
            url = strinfo.sub(str(seturl('seturl')),url)
            new_url = 'http://'+'127.0.0.1'+url
            request_urls.append(new_url)
        if method.upper() == 'GET':
            headers = {'Authorization':'', 'Content-Type': 'application/json' }
            if "=" in urlParam(param):
                data = None
                print (str(case_id)+' request  is get' +new_url.encode('utf-
8')+'?'+urlParam(param).encode('utf-8'))
                results = requests.get(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is get'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (' request  is get ' +new_url+'   body is '+urlParam(param))
                data = None
                req = urllib.request.Request(url=new_url,data=data,headers=headers,method="GET")
                try:
                    results = urllib.request.urlopen(req).read()
                    print ('   response is get ')
                    print(results)
                except Exception as e:
                    return caseWriteResult(case_id,'0')
                res = readRes(results,res_check)
                if 'pass' == res:
                    res_flags.append('pass')
                    writeResult(case_id,'1')
                    caseWriteResult(case_id,'1')
                else:
                    res_flags.append('fail')
```

```
                    writeResult(case_id,'0')
                    caseWriteResult(case_id,'0')

writeBug(case_id,interface_name,new_url,results,res_check)
        if  method.upper()=='PUT':
            headers = {'Host':HOSTNAME, 'Connection':'keep-alive',  'CredentialId':id, 'Content-Type':
'application/json'}
            body data=param
            results = requests.put(url=url,data=body_data,headers=headers)
            responses.append(results)
            res = readRes(results,res check)
            if 'pass' == res:
                writeResult(case_id,'pass')
                res flags.append('pass')
            else:
                res_flags.append('fail')
                writeResult(case id,'fail')

writeBug(case_id,interface_name,new_url,results,res_check)
        if method.upper()=="PATCH":
            headers = {'Authorization':'Credential '+id, 'Content-Type': 'application/json' }
            data = None
            results = requests.patch(new url+'?'+urlParam(param),data,headers=headers).text
            responses.append(results)
            res = readRes(results,res_check)
            if 'pass' == res:
                writeResult(case_id,'pass')
                res_flags.append('pass')
            else:
                res_flags.append('fail')
                writeResult(case_id,'fail')

writeBug(case_id,interface_name,new_url,results,res_check)

        if method.upper()=="POST":
            headers = {'Authorization':'Credential '+id, 'Content-Type': 'application/json' }
            if "=" in urlParam(param):
                data = None
                results = requests.patch(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is post'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (str(case_id)+' request  is ' +new_url.encode('utf-8')+'   body is
'+urlParam(param).encode('utf-8'))
                results = requests.post(new_url,data=urlParam(param).encode('utf-
8'),headers=headers).text
                print ('   response is post'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,res_check)
            if 'pass' == res:
                writeResult(case_id,'1')
                res_flags.append('pass')
            else:
                res_flags.append('fail')
                writeResult(case_id,'0')

writeBug(case_id,interface_name,new_url,results,res_check)

def readRes(res,res_check):
```

```
    res = res.decode().replace('":"',"=").replace('":',"=")
    res_check = res_check.split(';')
    for s in res_check:
        if s in res:
            pass
        else:
            return  '错误，返回参数和预期结果不一致'+s
    return 'pass'

def urlParam(param):
    param1=param.replace('&quot;','"')
    return param1

def CredentialId():
    global id
    url = 'http://'+'api.test.com.cn'+'/api/Security/Authentication/Signin/web'
    body_data= json.dumps({"Identity":'test',"Password":'test'})
    headers = { 'Connection':'keep-alive','Content-Type': 'application/json'}
    response = requests.post(url=url,data=body_data,headers=headers)
    data=response.text
    regx = '.*"CredentialId":"(.*)","Scene"'
    pm = re.search(regx, data)
    id = pm.group(1)

def seturl(set):
    global setvalue
    sql = "SELECT `setname`,`setvalue` from set_set"
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    print  (info)
    coon.commit()
    cursor.close()
    coon.close()
    if info[0][0] == set:
        setvalue = info[0][1]
        print (setvalue)
    return setvalue

def writeResult(case_id,result):
    result = result.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apitest_apistep set apitest_apistep.apistatus=%s,apitest_apistep.create_time=%s
where apitest_apistep.id=%s;"
    param = (result,now,case_id)
    print ('api autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
    coon.commit()
    cursor.close()
    coon.close()

def caseWriteResult(case_id,result):
    result = result.encode('utf-8')
```

```
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apitest_apis set apitest_apis.apistatus=%s,apitest_apis.create_time=%s where
apitest_apis.id=%s;"
    param = (result,now,case_id)
    print ('api autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
    coon.commit()
    cursor.close()
    coon.close()

def writeBug(bug_id,interface_name,request,response,res_check):
    interface_name = interface_name.encode('utf-8')
    res check = res check.encode('utf-8')
    request = request.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    bugname = str(bug_id)+ '_' + interface_name.decode() + '_出错了'
    bugdetail = '[请求数据]<br />'+request.decode()+'<br/>'+'[预期结
果]<br/>'+res_check.decode()+'<br/>'+'<br/>'+'[响应数据]<br />'+'<br/>'+response.decode()
    print (bugdetail)
    sql = "INSERT INTO `bug_bug` ("\
    "`bugname`,`bugdetail`,`bugstatus`,`buglevel`, `bugcreater`,
`bugassign`,`created_time`,`Product_id`)  "\
    "VALUES ('%s','%s','1','1','邹辉', '邹辉', '%s',
'2');"%(bugname,pymysql.escape_string(bugdetail),now)
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql)
    coon.commit()
    cursor.close()
    coon.close()

if __name__ == '__main__':
    test_readSQLcase()
    print ('Done!')
    time.sleep(1)
```

加入实现系统设置 seturl 参数化的脚本。

```
1）strinfo = re.compile('{seturl}')

2） elif param== 'null':
            url = strinfo.sub(str(seturl('seturl')),url)
            new_url = 'http://'+ url

3）def seturl(set):
    global setvalue
    sql = "SELECT `setname`,`setvalue` from set_set"
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    print  (info)
    coon.commit()
```

```
    cursor.close()
    coon.close()
    if info[0][0] == set:
        setvalue = info[0][1]
        print (setvalue)
    return setvalue
```

若前端发现执行时间显示与当前系统时间相差了 8 小时，在 autotest/settins.py 中注释 USE_TZ = True 这一行内容即可。

在前端 apis_manage.html 页面中加入如下内容。

```
<td>{% if apis.apistatus == 1 %}
<a style='color:green'>{{ apis.apistatus }}</a>
{% else %}
<a style='color:red'>{{ apis.apistatus }}</a>
{% endif %}
</td>
```

提交 Bug 到 Bug 管理数据库和页面。

- 字段长度到设置：bugdetail 字段长度设置为 2000。
- 字符编码到转换：encode，decode。
- 字符编码转义：pymysql.escape_string。

```
def writeBug(bug_id,interface_name,request,response,res_check):
    interface_name = interface_name.encode('utf-8')
    res_check = res_check.encode('utf-8')
__#  response = response.encode('utf-8')
    request = request.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    bugname = str(bug_id)+ '_' + interface_name.decode() + '_出错了'
    bugdetail = '[请求数据]<br />'+request.decode()+'<br/>'+'[预期结
果]<br/>'+res_check.decode()+'<br/>'+'<br/>'+'[响应数据]<br />'+'<br/>'+response.decode()
    print (bugdetail)
    sql = "INSERT INTO `bug_bug` ("\
    "`bugname`,`bugdetail`,`bugstatus`,`buglevel`, `bugcreater`,
`bugassign`,`created_time`,`Product_id`)  "\
    "VALUES ('%s','%s','1','1','邹辉', '邹辉', '%s',
'2');"%(bugname,pymysql.escape string(bugdetail),now)
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql)
    coon.commit()
    cursor.close()
    coon.close()
```

会发现 Bug 详情页太长，影响前端显示效果，于是在 bug_mange.html 中加入如下内容。

```
<table class="table table-striped" style="table-layout: fixed;">
<style>
td {
      white-space:nowrap;overflow:hidden;text-overflow: ellipsis;
}
</style>
```

固定表格显示长度，超过部分用…代替。鼠标移动到表格字段时显示全部内容到 tips，使用 td 和 title 标签：

```
<td title="{{ bug.bugdetail }}">{{ bug.bugdetail }}</td>
```

前端显示效果如图 6.7 所示。


产品中心
用例管理
定时任务
bug 管理
测试报告
名称
搜索
----bug名称----
增加
ID
产品
bug名称
bug详情
解决状态
严重等级
创建人
分配给
创建时间
2
商城
2登录_出...
请求数据...
激活
1
邹辉
邹辉
2018年1...
3
商城
2登录_出...
[请求数...
激活
1
邹辉
邹辉
2018年1...
4
商城
2_登录_...
[请求数...
激活
1
邹辉
邹辉
2018年1...

▲图 6.7

第 7 章

App 自动化测试

App 自动化测试属于 UI 层，在分层测试中所占比重最轻，在项目中相对滞后，发现问题后维护成本较大。手机端的是 App 等，PC 端的是 Web。

7.1 概述

App 自动化是指给 Android 或 iOS 上的软件应用程序做自动化测试。手工测试和自动化测试的对比如下。

手工测试优势：不可替代、能发现更多 Bug、包含了人的想象力与理解力。

自动化测试优势：可重复、效率高，能增加对软件质量的信任度。

注意，不是所有功能都需要自动化，只需把重复执行的以及主要的工作交给自动化即可。

App 自动化测试的特点如下。

（1）执行自动化测试只能发现一小部分 Bug。

（2）执行自动化冒烟测试或回归测试来验证系统状态，而不是找出更多 Bug。

（3）执行自动化测试可以让测试同学有更多的精力来关注复杂场景，做更多、更深层次的测试。

（4）编写自动化测试过程中会发现一部分 Bug，发现后要及时记录。

7.2 风险分析

自动化测试的主要风险分析如下。

（1）测试用例覆盖率。覆盖率决定了测试效率，因此要选择合适的用例，如主流程用例，常用的需重复执行的用例，应约占功能用例集的 20%~50%。

（2）测试结果准确度。准确度决定了测试有效性，因此应尽可能减少误报。

能优化，

改变动量较小时，测试人员可根据提供的元素提前介入，开发自动化脚本。

7.3 软硬件需求

自动化测试的软硬件需求如下。

（1）硬件：Windows 计算机、Android 手机。

（2）软件

- Appium 测试框架：运行 App 驱动的自动化平台，通过识别的控件元素，模拟用户的手工操作，支持 iOS 和 Android 系统。
- Jenkins：持续集成自动构建和执行任务。
- JDK、Eclipse、Python 语言开发编写环境。

7.4 测试计划

对于有良好代码基础的熟手，可用一周时间做出演示 Demo。如果是从零开始的初级工程师，则可用 3～6 个月的时间做出演示 Demo。

对于有良好代码基础的熟手，可用一个月时间试运行冒烟测试用例。如果是从零开始的初级工程师，则可用半年到一年的时间试运行冒烟测试用例。

目前已经做到了把用例放在 AutotestPlat 平台上，把脚本放在 Jenkins 平台并集成到自动化平台，测试报告也是根据 Python 代码自动化生成并展示到自动化平台上，非常的方便和强大。目前，增删改用例时，仍需到相应脚本中进行少量配置。

7.5 Appium 移动自动化框架

1．需要掌握的技能

（1）Appium API、WebDriver 基础知识和环境搭建。

（2）Nunit 等测试框架。

（3）Android、iOS 开发测试基础以及环境搭建。

（4）开发移动自动化项目的 Python 语言等。

2．Appium 框架的功能

使用 Python 和 UnitTest 编写测试脚本。Appium 自动化测试框架的功能概括如下。

（1）支持 iOS、Android，可在多台机器上并行 App 自动化，测试机型适配。

（2）代码实现关键字驱动。

- 测试集：关联自动化平台上的测试用例和脚本注册到定时任务中。
- 测试数据：MySQL 数据库存储输入数据、控件元素、测试结果。
- 测试脚本：由 Python 和 Nunit 编写。

（3）自动测试用例执行：从功能测试用例中抽取需要重复执行的、主要的功能进行用例覆盖。

（4）持续集成环境 Jenkins、Djcelery 定时自动构建和执行测试任务。测试结果报告展示，自动邮件展示。

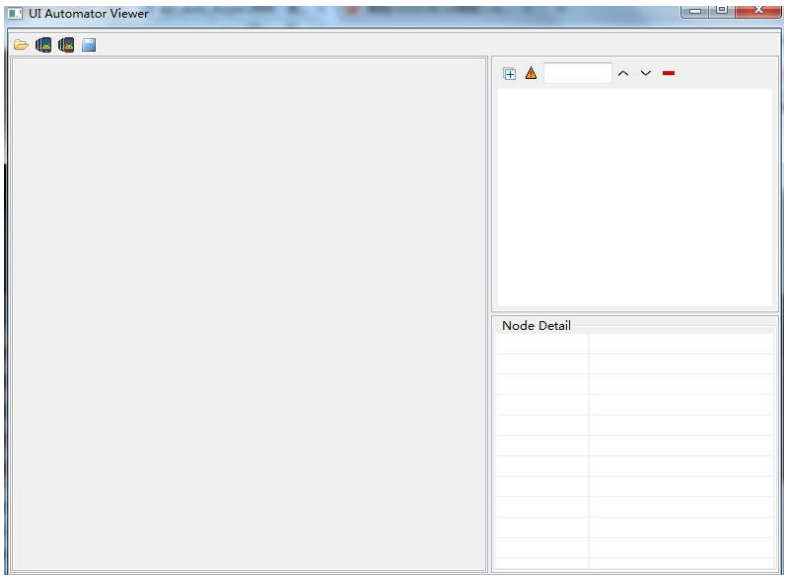
3．测试 App 的基本过程

Appium 自动化测试一个 App 的基本过程如下。

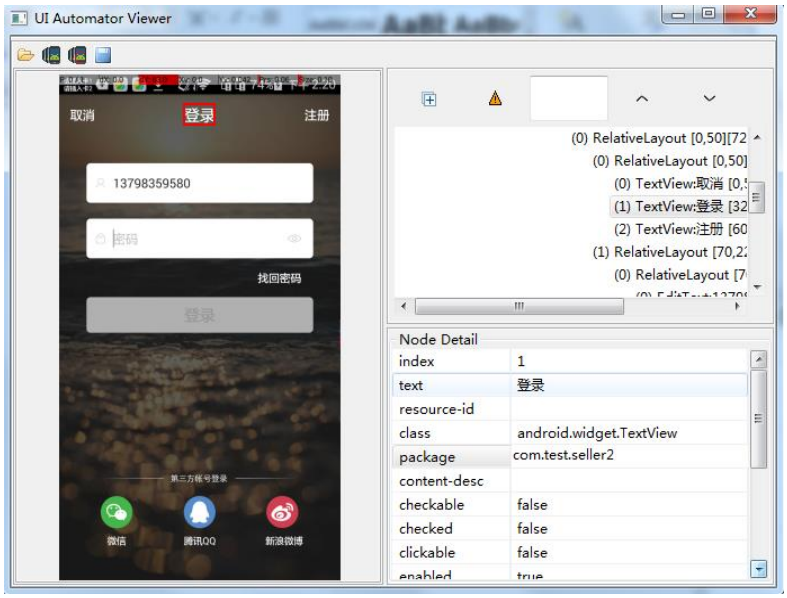
基于 Appium 自动化测试框架，测试人员要进行的是连接计算机、连接手机、解锁、安装 App、卸载 App、启动 App、元素定位、元素的操作、屏幕的操作、页面等待、异常处理截图、数据校验、日志、报告等一系列自动化测试执行的详细过程。

Appium 自动化框架元素控件的捕获，根据捕获到的元素控件进行相应的操作。

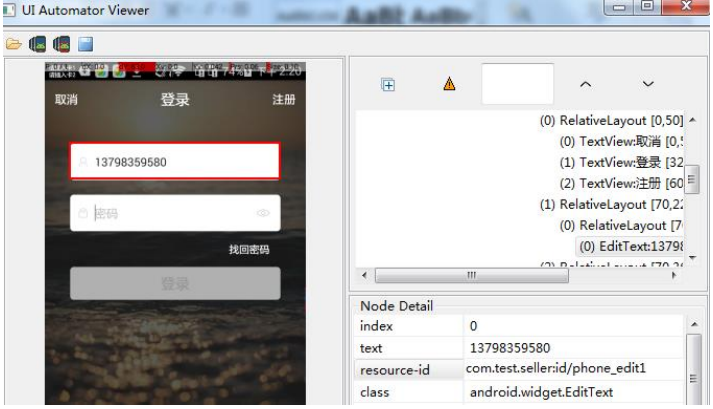
Appium 元素控件有多种定位方法，最常用的是元素的 ID（即 By.id）和元素的值（即 By.name）。还可以通过元素类型 TagName、元素的位置 XPath、手机设备的坐标等进行定位操作。安卓的元素控件可以通过 SDK 中的 uiautomatorviewer.bat 文件进行录制和捕获定位，如图 7.1 至图 7.3 所示。



▲图 7.1



▲图 7.2



▲图 7.3

如图 7.3 所示，Node Detail 下面的 resource-id com.test.seller:id/phone_edit1 对应 Excel 和代码中的定位方法 By.id 控件元素数据 text 13798359580 对应 Excel 和代码中的操作方法 sendkeys()，控件元素赋值数据为 13798359580。

可以这样理解：首先找到文本框，接着输入数据。即通过 ID 属性值 com.test.seller:id/phone_edit1 找到此用户名文本框的控件元素，然后通过 sendkeys()方法输入用户名数据 13798359580 找到此用户名文本。其他自动化测试步骤的定位方法、控件元素以及操作方法也都与此类似。实际上，自动化测试就是通过程序代码来实现模拟手动测试操作的过程。

上面介绍了用户名文本框输入用 sendkeys()方法，其他元素的操作方法有单击（click）、输入（sendkeys）、元素滑动、页面滑动、长按、下拉、弹出、屏幕放大缩小等，最常用的就是单击和输入。

数据校验。其实元素本身就是数据校验，当程序找不到元素时，用例会失败。另外，测试用例时可以加入一个或多个断言进行验证数据，还可设置步骤等待延迟时间。

测试结果。测试用例中记录了运行后的测试结果，如 pass、failed 或是 skip。

4．Appium 介绍（参考 Appium 官方资料）

Appium 是一个移动端自动化测试开源工具，支持 iOS 和 Android 平台，支持 Python、Java 等语言，即同一套 Java 或 Python 脚本可以同时运行在 iOS 和 Android 平台上。

Appium 是跨平台的，可以针对不同的平台用一套 API 来编写测试用例。

Appium 是一个 C/S 架构，核心是一个 Web 服务器，它提供了一套 REST 的接口。当收到客户端的连接后，就会监听到命令，然后在移动设备上执行这些命令，最后将执行结果放在 HTTP 响应中返还给客户端。

5．Session

自动化始终围绕一个 Session（会话）进行。客户端初始化一个 Session 来与服务端交互，不同的语言有不同的实现方式，但它们最终都是发送一个 POST 请求给服务端，请求中包含一个 JSON 对象，其被称作“Desired Capabilities”。此时，服务端就会开启一个自动化的 Session，然后返回一个 Session ID，Session ID 将会被用户发送后续的命令。

6．Desired Capabilities

Desired Capabilities 是一些键值对的集合（比如一个 map 或者 hash）。客户端将这些键值对发送给服务端，告诉服务端想要怎样测试。比如，测试人员可以把 platformName capability 设置为 iOS，告诉 Appium 服务端，测试人

员想要一个 iOS 的 Session，而不是一个 Android 的 Session。

7．Appium Server 服务端

Appium Server 是用 Node.js 编写的，既可以用源码编译，也可以从 NPM 直接安装。

Appium 服务端有很多语言库，如 Java、Ruby、Python、PHP、JavaScript 和 C#等，这些库都实现了 Appium 对 WebDriver 协议的扩展。当使用 Appium 的时候，只需使用这些库代替常规的 WebDriver 库就可以了。

8．Appium Clients 客户端

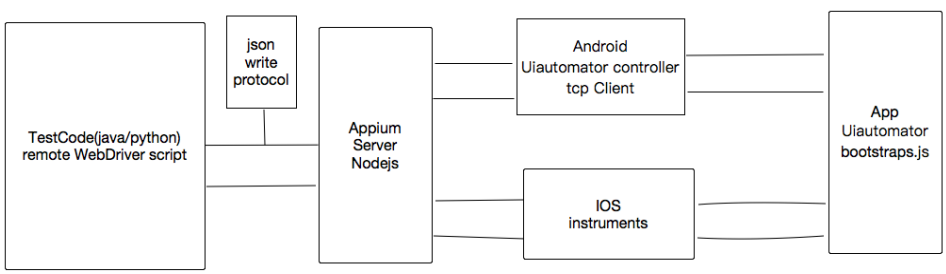
此客户端的概念不是传统意义上的客户端，它更像是一个扩展的 WebDriver 协议库，当用自己喜欢的语言写 case 时，会将该语言扩展的 WebDriver 库添加到自己的环境中，这时你可以把它理解为这就是一个客户端。

对于 Appium Clients 客户端的安装包，Mac 系统上直接运行 Appium.dmg，Windows 系统上运行 Appium.exe。

9．Appium Android/iOS 工作原理

API 接口调用 Selenium 的接口，Appium Server 接收 WebDriver 标准请求，解析请求内容，调用对应的框架响应操作。代码将 Desired Capabilities 中的键值对组合成一个 JSON，然后通过 HTTP 协议发送到 Appium 服务器创建一个 Session。代码与 Appium 的所有交互都围绕这个 Session 进行。Session 创建成功后，Appium 再通过 USB 接口与手机之间创建 TCP 连接，先安装一些服务端 App，比如 Android API 4.2+是 Uiautomator，Android 2.3+是 Instrumentation；如果是 iOS，则是 Uiautomation。手机的操作都是由 Appium 发送指令到 Uiautomator，然后再由 Uiautomator 进行控制的。

Appium 原理图如图 7.4 所示。



▲图 7.4

Appium 的核心是一个遵守 REST 设计风格的 Web 服务器，它接收客户端的连接和命令，在手机设备上执行命令，然后通过 HTTP 的响应收集命令执行的结果。这种架构提供了很好的开放特性：只要某种语言有 HTTP 客户端的 API，就可以通过该语言编写自己的测试代码。

7.6 环境搭建

安装 Android 环境变量、SDK、Android API、ADT 等。

步骤 1 安装 SDK：下载地址为 https://pan.baidu.com/s/1mi6PT9m。如提示错误：'xcopy' 不是内部或外部命令，也不是可运行的程序，则在环境变量 path 中加入 C:\Windows\System32。

步骤 2 安装 Android API19 等，如图 7.5 所示。

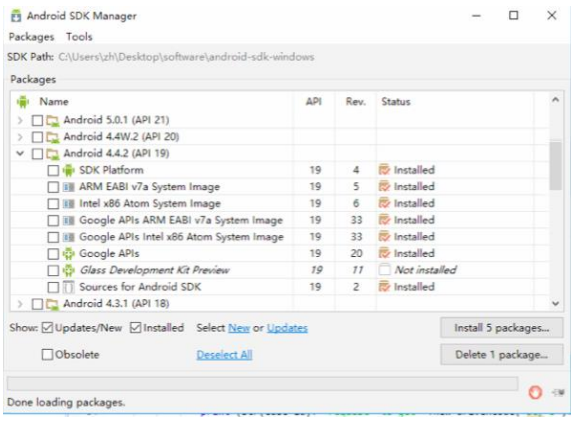
步骤 3 安装 ADT，下载离线安装包 https://pan.baidu.com/s/1sl2BZit，在 Eclipse 中单击“help→install new

software”，在弹出框中单击“Add”按钮，单击“Archive”，选择离线安装包，如图 7.6 所示。

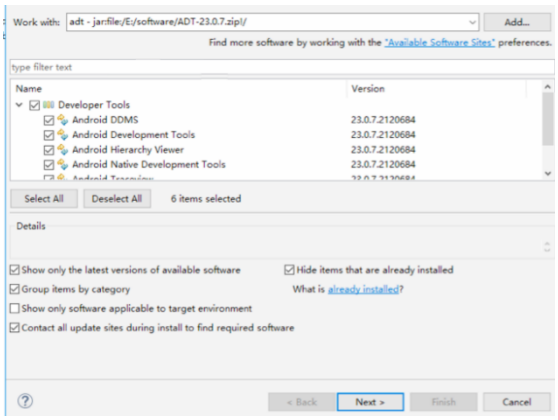
步骤 4 配置环境变量 Adb 和 SDK。

在 path 中添加如下内容。

```
ANDROID_HOME
C:\Users\zh\Desktop\software\android-sdk-windows
Path
%ANDROID_HOME%\platform-tools
%ANDROID_HOME%\tools
```

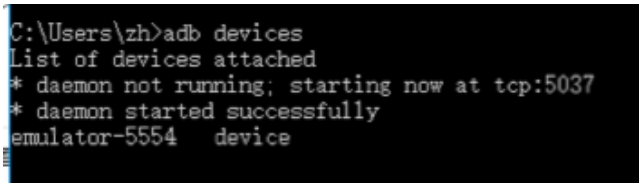


▲图 7.5



▲图 7.6

步骤 5 运行 CMD，输入命令 `Adb devices`，如图 7.7 所示。

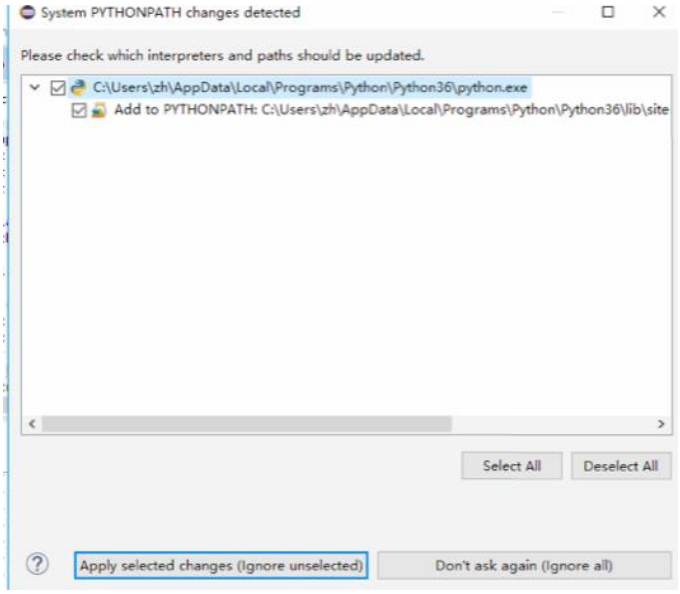


▲图 7.7

步骤 6 安装 appium_python_client。

可以用命令安装 pip install Appium_Python_Client，也可以在 https://pypi.python.org/pypi/ Appium-Python-Client 地址下载安装包，进行安装。

解压缩，运行 CMD 进入到相应目录，输入命令：python setup.py install，在 Eclipse 中弹出如图 7.8 所示窗口，单击“Apply Selected changes（Ignore unselected）”。

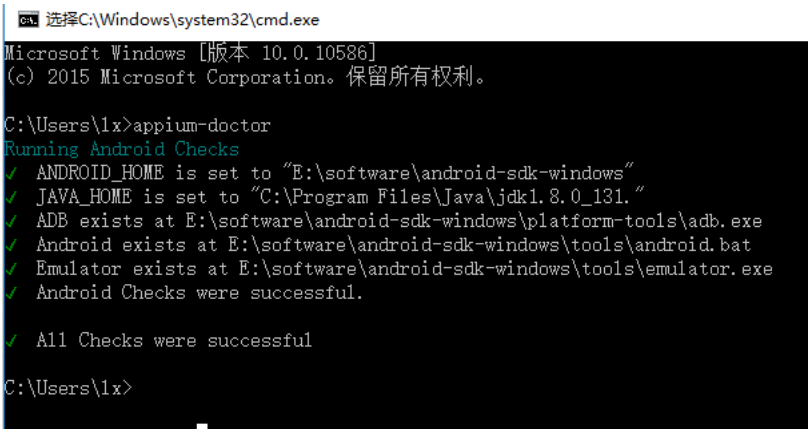


▲图 7.8

步骤 7 安装 Appium 环境。

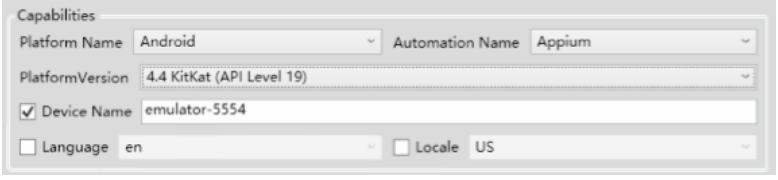
这里依然用早期的老版本（appium1.4.13），进行讲解。

安装包下载地址为 https://pan.baidu.com/s/1jIJfruA，安装后检查环境，如图 7.9 所示。



▲图 7.9

步骤 8 默认安装成功后，打开 appium.exe，然后配置 Capabilities，如图 7.10 所示。



▲图 7.10

步骤 9 单击 Appium 右上角按钮启动服务，如图 7.11 所示。

▲图 7.11

7.7 App 自动化测试源码详解

搭建好环境后，开始写脚本并进行 AppUI 自动化测试。

7.7.1 App 自动化测试实例 1

1．功能描述

Android 模拟器上的计算器可以说是 Android 内置的 App，实现自动化模拟手工操作，单击 1，单击+，单击 1，单击=，于是计算器上自动得出结果 2。实例 1 采用纯代码方式实现，实例 2 为自动平台实现。

2．程序清单 1：appauto_testcase1.py

```
#coding=utf-8
from appium import webdriver
import time

desired caps = {}
desired_caps['platformName'] = 'Android'
dersired_caps['platformVersion'] = '4.4'
desired caps['deviceName'] = 'emulator-5554'
desired_caps['appPackage'] = 'com.android.calculator2'
desired_caps['appActivity'] = '.Calculator'

driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired_caps)

time.sleep(5)
driver.find_element_by_name("1").click()
driver.find_element_by_name("+").click()
driver.find_element_by_name("1").click()
driver.find_element_by_name("=").click()
time.sleep(5)
```

```
driver.find_element_by_name("清除").click()
time.sleep(5)

driver.quit()
```

7.7.2 App 自动化测试实例 2

首先编写 App 自动化测试用例，在 Autotestplat 自动化平台新增测试，输入相应的字段内容，计算器 1+1=2 的相应测试步骤、定位方式、测试控件名称、操作方法等。其中，控件元素通过 SDK 自带的 Uiautomatorviewer 工具抓取，在 E:\software\android-sdk- windows\tools 目录下的 uiautomatorviewer.bat，抓取到计算器的元素 name，默认英文时依次为“1”，“+”，“1”，“=”，“CLR” 设置为中文时依次为“1”，“+”，“1”，“=”，“清除”，如图 7.12 所示。

▲图 7.12

自动化平台-实现自动化测试用例-模拟器上操作计算器 1+1=2。

功能详情描述：Android 模拟器上的计算器，可以说是 Android 内置的 App；实现自动化模拟手工操作单击 1，单击+，单击 1，单击=；于是计算器上自动得出结果 2。

在代码里引入 Appium 库、Request 库、PyMySQL 库等。

调用 SQL 查询到需要执行的测试用例，根据测试用例对每个字段进行判断，并执行相应的操作。App 测试用例的几个核心字段，如用例 ID 和名称、元素定位方式、元素控件属性、操作方法，测试数据。

程序清单 2：appauto_testcase2.py。

```
#coding=utf-8
from appium import webdriver
import time
import os
import unittest

PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)
global driver

# -*- coding:utf-8 -*-
import requests,  pymysql, time, sys, re
```

```
import urllib,  zlib
#import _mysql #import result
#from httplib import ResponseNotReady
from trace import CoverageResults
import json
#from test.test_ast import eval_results
from idlelib.rpc import response_queue
#from cPickle import dumps
from time import sleep
import mod_config

HOSTNAME = '127.0.0.1'

def readSQLcase():              #流程的相关接口
    sql="SELECT id,`appcasename`,appfindmethod,appevelement,appoptmethod,appassertdata,`apptestresult`
from apptest_appcasestep where apptest_appcasestep.Appcase_id=1 ORDER BY id ASC "
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host=mod_config.getConfig("dat
abase", "host"),charset='utf8')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    for ii in info:
        case_list = []
        case_list.append(ii)
        apptestcase(case_list)
    coon.commit()
    cursor.close()
    coon.close()

def apptestcase(case_list):
    for case in case_list:
        try:
            case_id = case[0]
            findmethod = case[2]
            evelement = case[3]
            optmethod = case[4]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        print (evelement)
        time.sleep(10)
        if optmethod== 'click' and findmethod=='find_element_by_id':
driver.find_element_by_id(evelement).send_keys('wayto')
        elif optmethod== 'click' and findmethod=='find_element_by_name':
            driver.find_element_by_name(evelement).click()
        elif optmethod=='sendkey':
            driver.find_element_by_name(evelement).send_keys()

if __name__ == '__main__':
    desired_caps = {}
    desired_caps['platformName'] = 'Android'
    desired_caps['platformVersion'] = '4.4'
    desired_caps['deviceName'] = 'emulator-5554'
    desired_caps['appPackage'] = 'com.android.calculator2'
    desired_caps['appActivity'] = '.Calculator'
    time.sleep(1)
    driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired_caps)
    time.sleep(1)
    readSQLcase()
    driver.quit()
```

```
print ('Done!')
time.sleep(1)
'''
```

第 8 章

Web 自动化测试

8.1 Selenium 介绍

Selenium 是一个 Web 开源自动化测试框架，具有页面级操作、模拟用户真实操作、API 从系统层面触发事件等特点。

1．版本

Selenium 1.0

Sever/Client 工作方式，可在本地或远程机器上运行基于 JS 注入的 Case 底层。

为什么一定要用代理服务器的模式？答案是同源策略，它是由 Netscape 提出的一个著名的安全策略，现在所有可支持 JavaScript 的浏览器都在使用这个策略。

Selenium 2.0

Selenium 2.0 基于 Selenium 1.0（即 JavaScript ），并结合其 WebDriver 模拟用户的真实操作。WebDriver 原生绑定浏览器，绕过浏览器安全模型。它有很好的处理 Ajax 的能力，并且支持多种浏览器（如 Safari、IE、Firefox、Chrome 等），可以运行在多种操作系统上。目前，很多人在使用 Selenium 2.0。

Selenium 3.0

Selenium3.0 在 Selenium 2.0 的基础上做了更多的改进，JDK 的版本必须在 1.8 以上，geckodriver 驱动必须在 Firefox 48 版本以上。该版本支持苹果公司的 Safari 浏览器，去掉了 Selenium RC，全部采用了 WebDriver 等。总之 Selenium 3.0 支持的原生驱动更加丰富。

2．元素定位及用例编写

（1）多种方式定位并控制页面元素。掌握 Web 元素定位方式 id name linktext xpath tag css。

（2）自动化测试用例。通过元素定位和步骤依次封装成自动化测试用例，并写到 Autotestplat 自动化测试平台

注意，版本需要兼容 JDK 1.7 以及 IE 8～IE 10，如是 IE 11 及以上，需要更新驱动才能支持，请大家自行研究更新版本驱动或采用旧版本浏览器。

8.2 环境搭建

从 https://pypi.python.org/pypi/selenium/#downloads selenium-3.8.0.tar.gz 解压缩，切换到相应目录 python setup.py install。

从 https://pypi.python.org/pypi/configparser/configparser-3.5.0.tar.gz 解压缩，切换到相应目录 pyton setup.py install

安装 Firefox 浏览器，选择默认安装。

安装火狐驱动：Geckodriver.exe，将 gson-1.6.jar 导入到相应的正确的代码目录下。

设置环境变量 C:\Users\zh\Desktop\software\autotestweb\src\geckodriver.exe。

若不设置，可能在运行时出现如下错误：

```
    os.path.basename(self.path), self.start_error_message)
selenium.common.exceptions.WebDriverException: Message: 'geckodriver' executable needs to be in PATH.
```

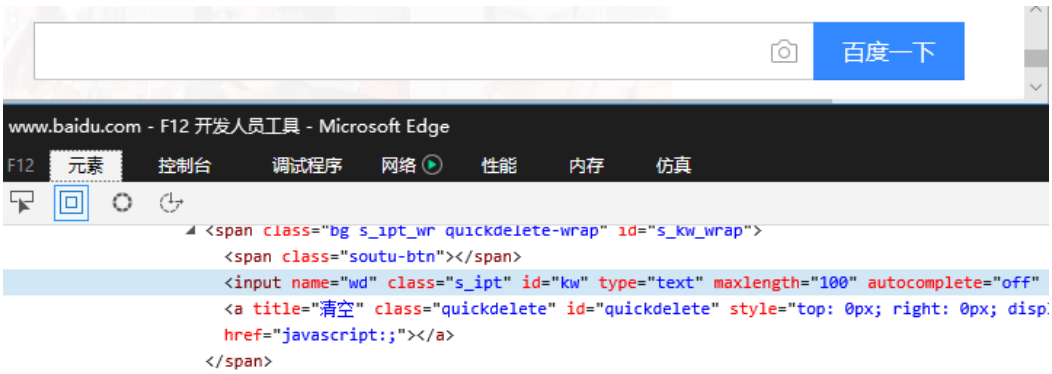
8.3 Web 自动化测试源码详解

8.3.1 Web 自动化测试实例 1

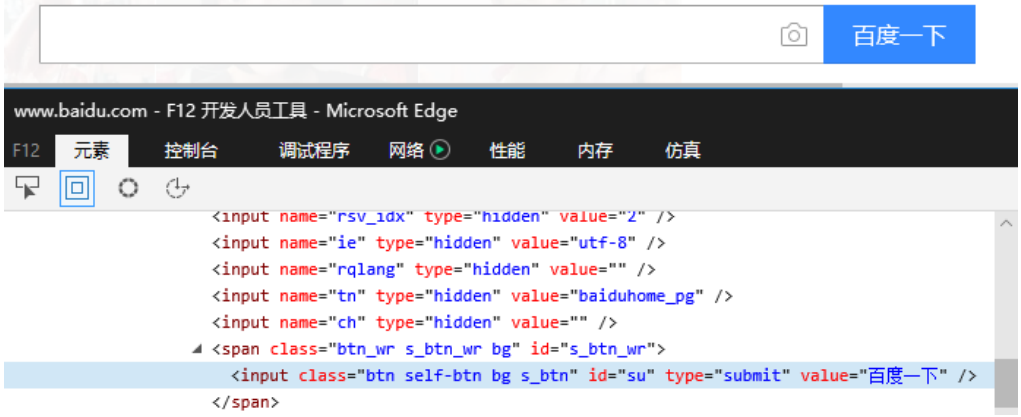
通过按 F12 键，以及左上角的箭头，单击要捕获的元素，就能定位到源码并查看该元素的信息，比如 id、type name、class 等。搜索文本捕获到文本框 id 为 kw，百度一下按钮捕获到 id 为 su，如图 8.1 和图 8.2 所示。

1．功能描述

未使用 Autotestplat 平台，引入 Selenium 的 Webdriver 驱动包，进入主函数，定义火狐浏览器驱动，取百度网页，查找控件元素 id 为 kw 的文本框，Sendkeys 输入软件自动化测试开发，查找控件元素 id 为 su 的按钮，单击“搜索”按钮，结束。



▲图 8.1



▲图 8.2

2．程序清单：webauto_testcase1.py

```
#-*- coding: UTF-8 -*-
import time
from selenium import webdriver

if __name__ == '__main__':
    global driver
    driver =webdriver.Firefox()
    driver.get("http://www.baidu.com")
    time.sleep(1)
    driver.find_element_by_id('kw').send_keys('软件自动化测试开发')
    time.sleep(1)
    driver.find_element_by_id('su').click()
    driver.quit()
    print ('Done!')
    time.sleep(1)
```

8.3.2 Web 自动化测试实例 2

1．功能描述

使用 Autotestplat 平台，在平台上录入测试用例，在脚本中关联用例。

默认选择 Firefox 浏览器，打开浏览器，输入百度网站，输入搜索“自动化平台测试开发”，单击搜索按钮，输出测试报告。

在 Autotestplat 平台上输入自动化测试用例相关步骤，如定位方式、元素 id、操作方式、测试数据、预期结果测试结果等相关信息。

2．程序清单 1：webauto_testcase2.py

```
#-*- coding: UTF-8 -*-
import os
import time
import unittest
import pymysql
import fconfig
from selenium import webdriver
```

```
PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)
global driver

HOSTNAME = '127.0.0.1'

def readSQLcase():              #流程的相关接口
    sql="SELECT
id,`webcasename`,webfindmethod,webevelement,weboptmethod,webtestdata,webassertdata,`webtestresult`
from webtest_webcasestep where webtest_webcasestep.Webcase_id=1 ORDER BY id ASC "
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host=mod_config.getConfig("dat
abase", "host"),charset='utf8')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    for ii in info:
        case_list = []
        case_list.append(ii)
        webtestcase(case_list)
    coon.commit()
    cursor.close()
    coon.close()

def webtestcase(case_list):
    for case in case_list:
        try:
            case_id = case[0]
            findmethod = case[2]
            evelement = case[3]
            optmethod = case[4]
            testdata = case[5]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        print (case)
        time.sleep(5)
        if optmethod=='sendkeys' and findmethod=='find_element_by_id':
            print (evelement)
            driver.find_element_by_id(evelement).send_keys(testdata)
        elif optmethod=='click' and findmethod=='find_element_by_name':
            print (evelement)
            driver.find_element_by_name(evelement).click()
        elif optmethod=='click' and findmethod=='find_element_by_id':
            print (evelement)
            driver.find_element_by_id(evelement).click()

if __name__ == '__main__':
    time.sleep(1)
___# unittest.main()
    driver =webdriver.Firefox()
__#  driver =webdriver.Ie(iedriver)
    driver.get("http://www.baidu.com")
    time.sleep(1)
    readSQLcase()
    time.sleep(1)
    driver.quit()

    print ('Done!')
```

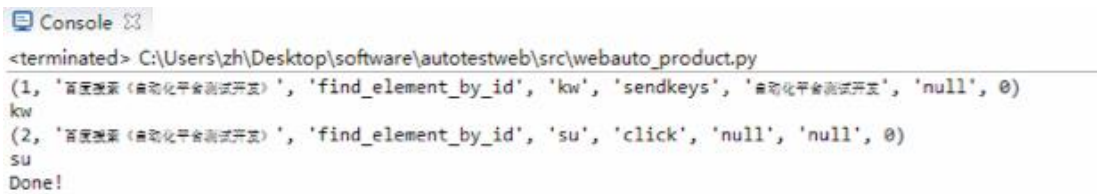
配置文件设置公共信息，如数据库等。

```
Settins.ini
[project]
schema=tests

[user]
Username=admin
Password=test123456

[database]
host=127.0.0.1
db=autotest
user=root
passwd=test123456
```

3．程序清单 2：读取调用配置文件。

```
fconfig.py
#encoding:utf-8

import configparser
import os

def getConfig(section, key):
    config = configparser.ConfigParser()
    path = os.path.split(os.path.realpath(  file  ))[0] + '/settings.ini'
    config.read(path)
    return config.get(section, key)
```

Web 自动化测试运行日志如图 8.3 所示。



▲图 8.3

功能描述：加入 Unittest 和 HTMLTestRunner 框架，从而生成测试报告。

程序清单：

```
#-*- coding: UTF-8 -*-
import os
import time
import unittest
import pymysql
import fconfig
from selenium import webdriver
import HTMLTestRunner

PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)
global driver

HOSTNAME = '127.0.0.1'
```

```
class Search(unittest.TestCase):
    """搜索：自动化平台测试开发"""
    def setUp(self):
        time.sleep(1)

    def test_readSQLcase(self):             #流程的相关接口
        sql="SELECT
id,webfindmethod,webevelement,weboptmethod,webtestdata,webassertdata,`webtestresult` from
webtest_webcasestep where webtest_webcasestep.Webcase_id=1 ORDER BY id ASC "
        coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host=fconfig.getConfig("databa
se", "host"),charset='utf8')
        cursor = coon.cursor()
        aa=cursor.execute(sql)
        info = cursor.fetchmany(aa)
        for ii in info:
            case_list = []
            case_list.append(ii)
            webtestcase(case_list)
        coon.commit()
        cursor.close()
        coon.close()

        def tearDown(self):
            self.driver.quit()

def webtestcase(case_list):
    for case in case_list:
        try:
            case_id = case[0]
            findmethod = case[1]
            evelement = case[2]
            optmethod = case[3]
            testdata = case[4]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        print (case)
        time.sleep(5)
        if optmethod=='sendkeys' and findmethod=='find_element_by_id':
            print (evelement)
            driver.find element by id(evelement).send keys(testdata)
        elif optmethod=='click' and findmethod=='find_element_by_name':
            print (evelement)
            driver.find_element_by_name(evelement).click()
        elif optmethod=='click' and findmethod=='find_element_by_id':
            print (evelement)
            driver.find_element_by_id(evelement).click()

if __name__ == '__main__':
    driver =webdriver.Firefox()
    driver.get("http://www.baidu.com")
    time.sleep(1)
    now = time.strftime("%Y-%m-%d-%H %M %S", time.localtime(time.time()))
    testunit = unittest.TestSuite()
    testunit.addTest(Search("test_readSQLcase"))

filename="C:\\Users\\zh\\AppData\\Local\\Programs\\Python\\Python36\\Scripts\\autotest\\webtest\\templ
ates\\"+"webtest_report.html"
    fp=open(filename,'wb')
```

```
    runner = HTMLTestRunner.HTMLTestRunner(stream=fp, title=u"web 自动化测试报告", description=u"搜索测试用例")
    runner.run(testunit)
    driver.quit()
    print ('Done!')
time.sleep(1)
```

Web 自动化测试运行时的测试过程：在运行过程中，会发现计算机桌面自动启动浏览器，自动打开了www.baidu.com 网站，并输入了自动化平台测试开发，单击了搜索，然后自动显示结果，最后自动退出了浏览器。

第 9 章

性能测试

介绍使用 Locust 的使用方法并集成到 Autotestplat 进行展示。

9.1 环境搭建

步骤 1 安装相关版本到 vc_redist_x64.exe，即 Locust 的第三方库依赖环境 https://www.microsoft.com/en-us/download/details.aspx?id=53840。

步骤 2 安装 pyzmp，即 Locust 的第三方库依赖环境 https://pypi.python.org/pypi/ pyzmq/。

解压缩下载包进入所在目录，运行 CMD，输入 python setup.py install，选择默认安装。

步骤 3 安装 Locust，http://locust.io。解压缩下载包进入所在目录，运行 CMD，输入 python setup.py install，选择默认安装。

步骤 4 配置 Locust 环境变量。在 path 中加入 script 路径，即 C:\Users\zh\AppData\Local\Programs\Python\Python36\Scripts。

步骤 5 运行 CMD，输入 locust –help 或 locust--v，测试安装是否成功，如图 9.1 所示。

▲图 9.1

9.2 使用入门

1．功能描述

实现一个链接地址 Get 接口、登录 Post 接口的性能测试

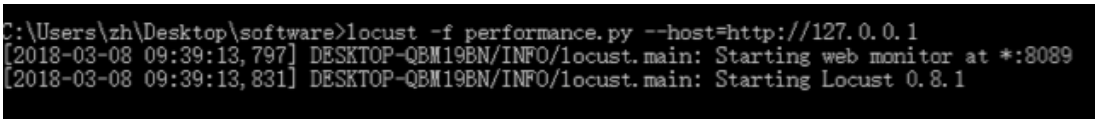
```
class WebsiteTasks(TaskSet):
      @task
      def index(self):
```

```
            self.client.get("/")

class WebsiteUser(HttpLocust):
      task_set = WebsiteTasks
      min_wait = 100
      max_wait = 1000
```

在 autotest\apitest\目录下新建 performance2.py 文件，加入如下内容。

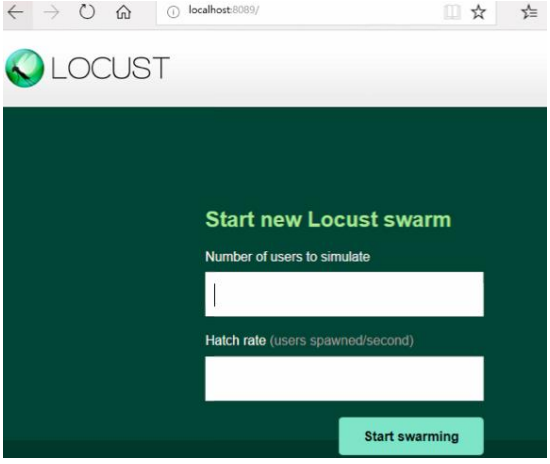
```
from locust import HttpLocust, TaskSet, task

class WebsiteTasks(TaskSet):
      @task
      def login(self):
            self.client.post("/test", {
                  "username": "admin",
                  "password": "test123456"
            })

class WebsiteUser(HttpLocust):
      task_set = WebsiteTasks
      min_wait = 100
      max_wait = 1000
```

步骤 1 进入文件目录，运行 CMD，输入 Locust 启动命令 locust -f performance.py --host=http://127.0.0.1，如图 9.2 所示。



▲图 9.2

步骤 2 在浏览器中输入 localhost:8089，如图 9.3 所示。



▲图 9.3

步骤 3 集成到 Autotestplat，在 apitest/templates 的 left.html 文件中加入如下内容。

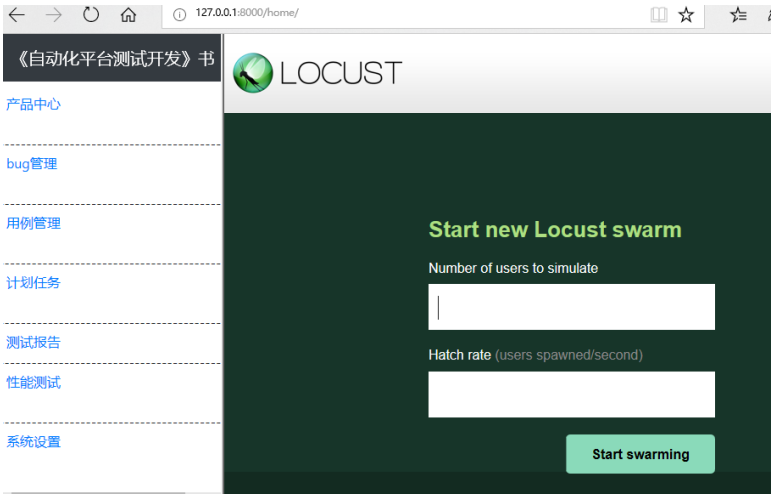
```
<tr> <td>
          <li>
          <a  href="http://localhost:8089" target="mainFrame" style="font-size:18px">
          <i class="glyphicon glyphicon-calendar"></i>
                                    性能测试
          </a>
```

```
        </li>
</tr> </td>
        <tr><td> </td></tr>
```

在 autotest/urls.py 中加入以下内容：

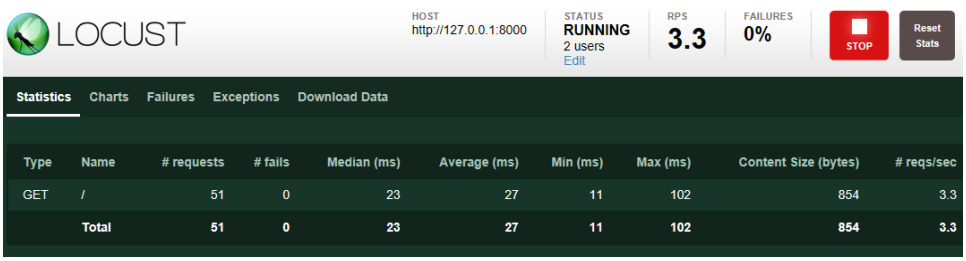
```
path('', views.login),
```

步骤 4 登录 Autotestplat 平台，如图 9.4 所示。

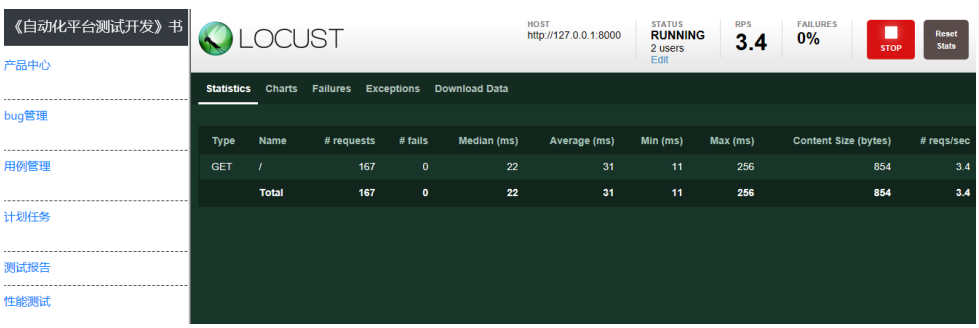


▲图 9.4

输入虚拟并发用户数，比如 2，每秒增加用户数为 1，单击“Start swarming”按钮，如图 9.5 和图 9.6 所示。



▲图 9.5



▲图 9.6

表明性能测试正常运行，页面展示了相关性能测试的实时数据。

第 10 章

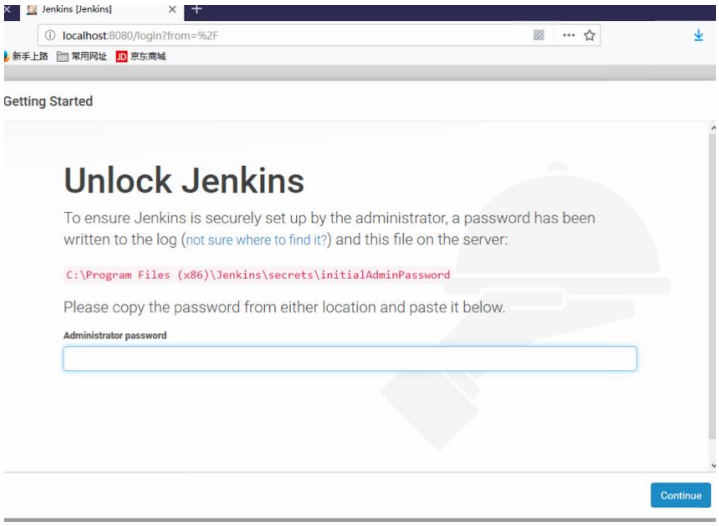
持续集成

10.1 介绍

Jenkins 持续集成，包括自动化部署、编译、运行、测试结果自动邮件通知等，从而实现无人值守全程自动化如果没有用到 Jenkins，也可以直接阅读第 11 章，并进行相应功能的开发实践。

Tomcat 启动 startup.bat 后，把 jenkins.war 包放在 WebApp 目录下，Jenkins 即可运行。或用 Jenkins 1.6 安装包和最新安装包，直接安装后即可运行。

安装文件路径为 https://pan.baidu.com/s/1b0OzXs 或 https://jenkins.io/。下载并安装后，在火狐浏览器中输入 localhost：8080 即可启动，如图 10.1 所示。

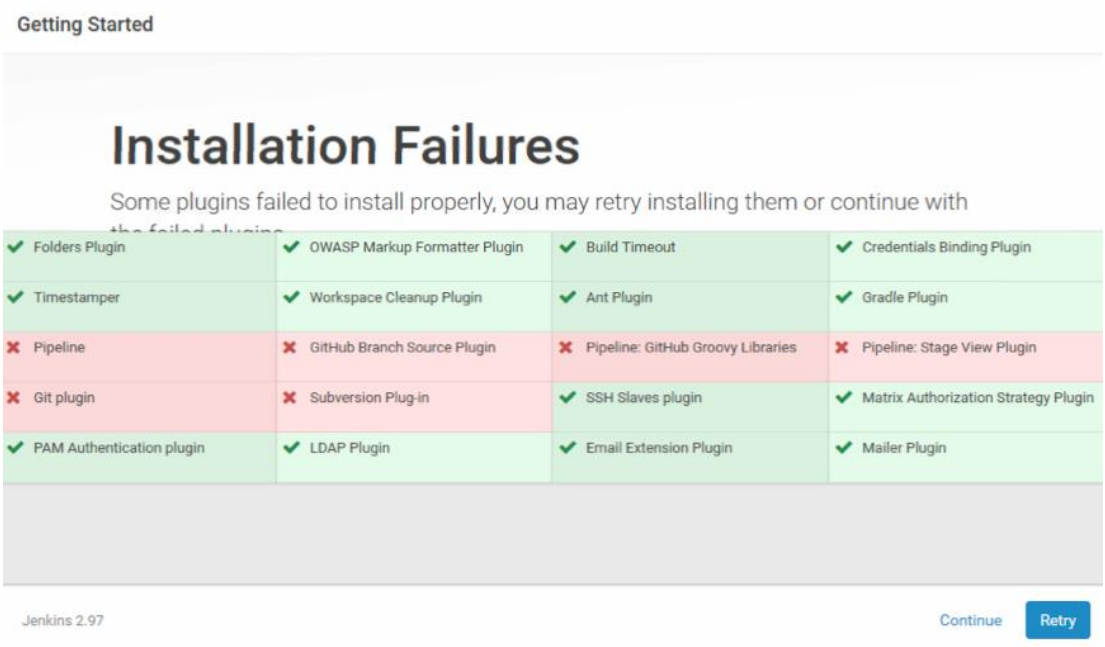


▲图 10.1

然后打开红色显示的目录文件，复制密码并输入到空白密码栏，如图 10.2 所示

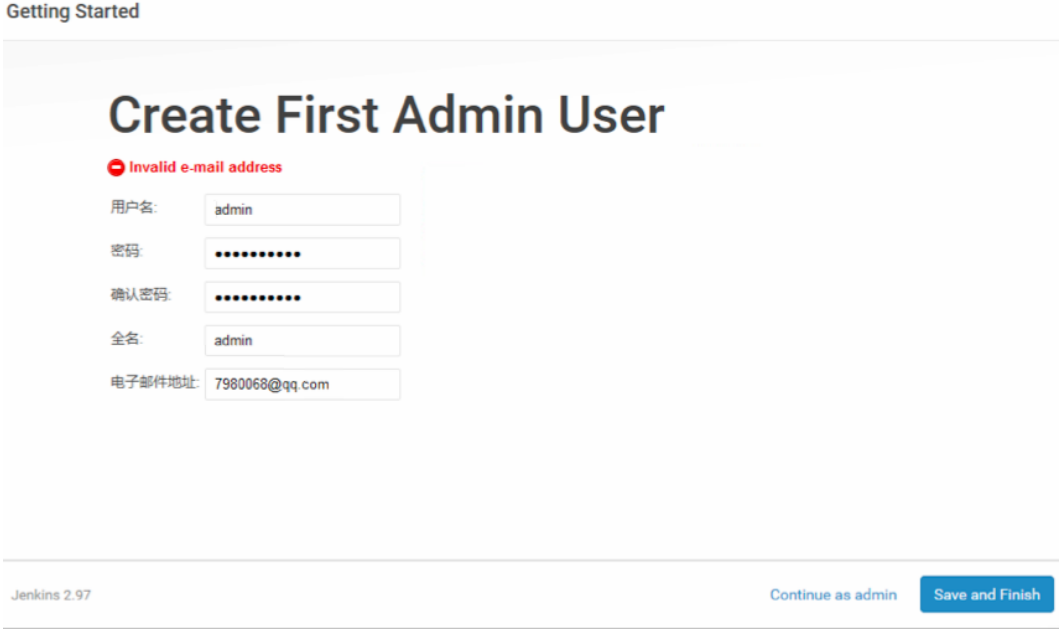
▲图 10.2

选择建议到默认第三方插件，如图 10.3 所示。



▲图 10.3

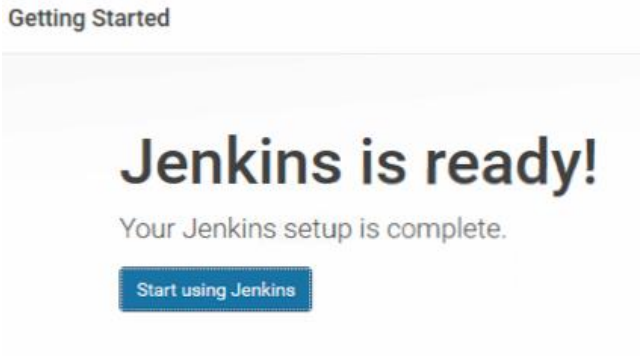
单击“Continue”按钮，如图 10.4 所示。



▲图 10.4

设置用户名（admin）和密码（test123456），单击“登录”按钮，如图 10.5 和图 10.6 所示。

▲图 10.5



▲图 10.6

10.2 系统配置

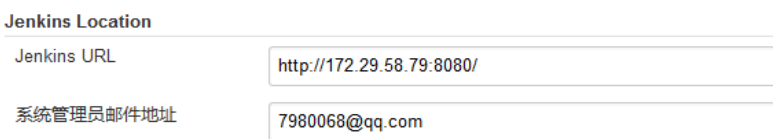
步骤 1 在系统管理（Configure Global Security）界面设置全局变量，勾选“允许用户注册”复选框，如图 10.7 所示。

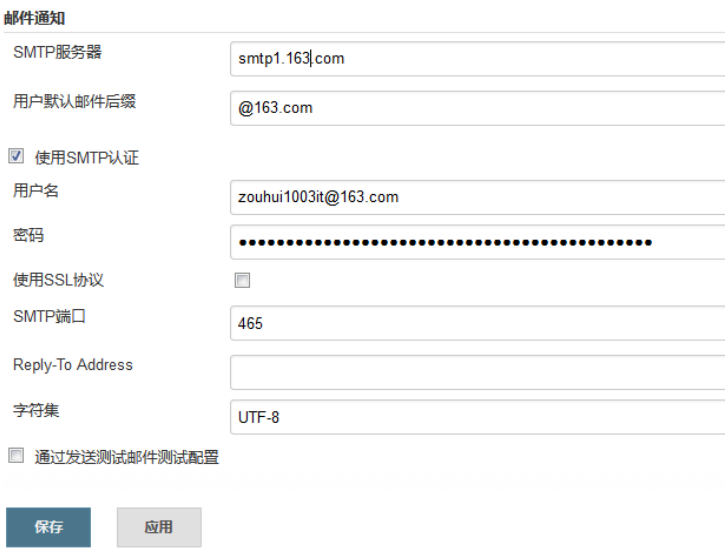
▲图 10.7

步骤 2 注册账号并登录，如图 10.8 所示。

▲图 10.8

步骤 3 设置邮件，如图 10.9 所示。

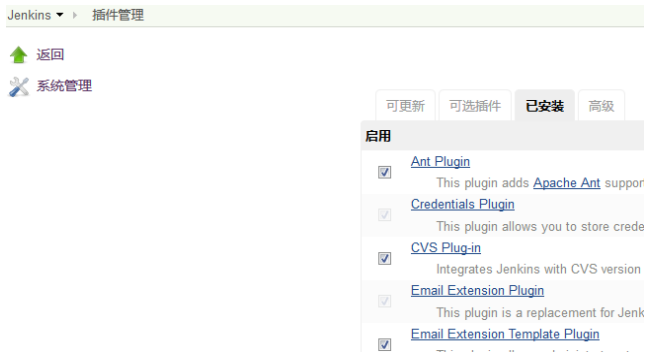




▲图 10.9

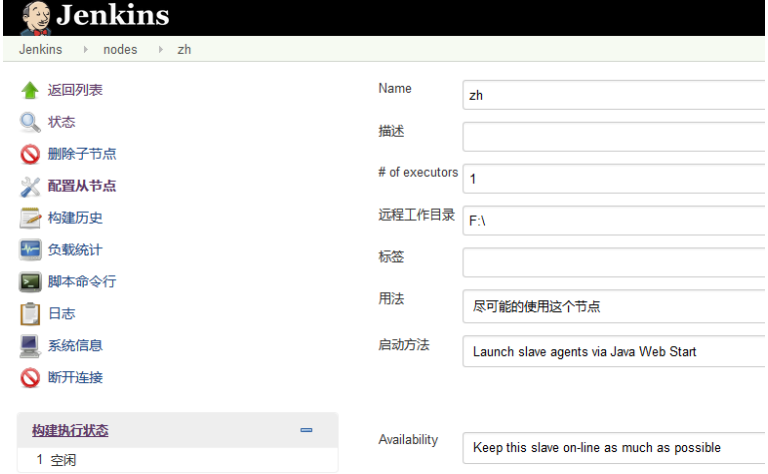
步骤 4 管理插件。可先安装如下要用到的插件，如图 10.10 所示。

- 邮件插件：Regex Email Plugin。
- SVN 插件：Subversion Plug-in。
- 报表插件：ExtentReports。



▲图 10.10

步骤 5 管理节点，新建本地节点并连接，如图 10.11 所示。

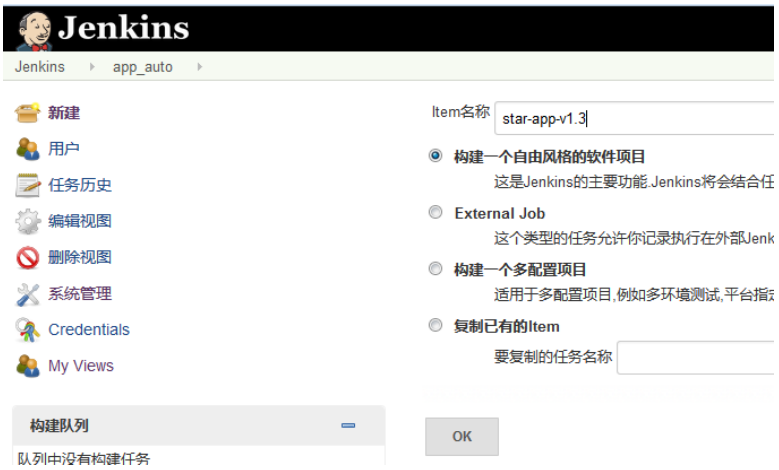


▲图 10.11

10.3 项目配置

增加 project 构建项目，包括上述的 Android 与 iOS 自动化、API 自动化、Web 自动化、JMeter 自动化、LoadRunner 性能自动化项目。

步骤 1 在视图中新建 job 项目，api_autotest、app_autotest、web_autotest，如图 10.12 所示。

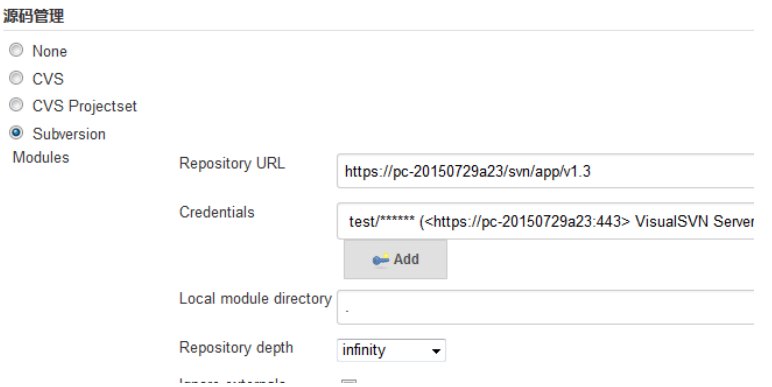


▲图 10.12

步骤 2 所运行的客户端机器为 zh，如图 10.13 所示。

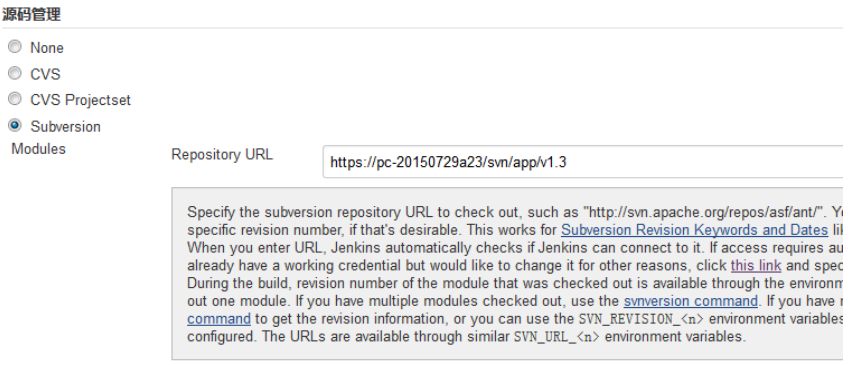
▲图 10.13

步骤 3 SVN 设置如图 10.14 所示。

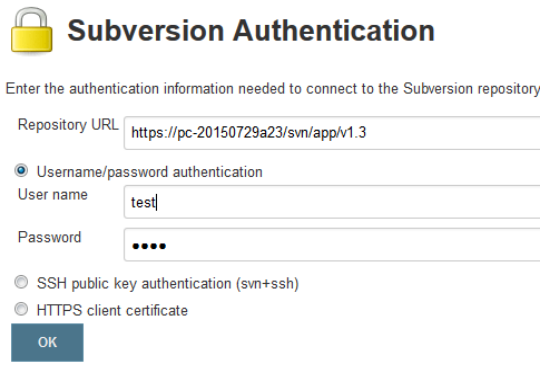


▲图 10.14

密码验证如图 10.15 和图 10.16 所示。



▲图 10.15



▲图 10.16

步骤 4 设置触发器，选择“Build Periodically”（自动周期执行），时间为周一到周五早上 10 点。也可以选择“Build after other projects are built”，即在某一项目执行后再触发此项目，如图 10.17 所示。

▲图 10.17

步骤 5 配置 E-mail 自动邮件，邮件发送的内容设置如图 10.18 所示。

▲图 10.18

步骤 6 API 自动化项目中的 Python 脚本如图 10.19 所示。

步骤 7 在 App 自动化项目中的 Python 脚本如图 10.20 所示。

步骤 8 在 Web 自动化项目中的 Python 脚本如图 10.21 所示。

▲图 10.19

▲图 10.20

▲图 10.21

10.4 多机器节点配置

步骤 1 节点 Slave 配置及连接，可以设置多台机器节点：http://192.168.115.146:8080/ jenkins/computer/。

增加子节点，单击“系统管理→管理节点→新建节点”，输入节点名称，单击“OK”按钮，保存该节点。配置节点如图 10.22 所示。

▲图 10.22

远程工作目录，即节点计算机从 SVN 传自动化脚本的目录，启动方法如图 10.23 所示。

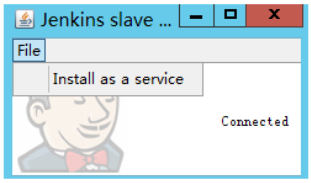
▲图 10.23

步骤 2 连接子节点，单击“Launch”按钮，或新建.bat 文件，如图 10.24 所示。

▲图 10.24

如果出现启动程序失败，则需在 Firefox 下直接单击“Launch”按钮，选择 Java 运行。

连接后不要单击“install as a service”，否则会在后台运行测试用例，如图 10.25 所示。



▲图 10.25

步骤 3 job 和节点机器关联，勾选“Restrict where this project can be run”，如图 10.26 所示。

▲图 10.26

步骤 4 启动远程节点自动化，为多台机器配置远程节点 JDK、Ant 环境、XP 系统，如果构建出现 test flied，则需要查看启动的浏览器安装路径是否和程序中的一致。

10.5 结果展示视图

Jenkins 展示项目视图（各个自动化测试项目），如图 10.27 所示。

▲图 10.27

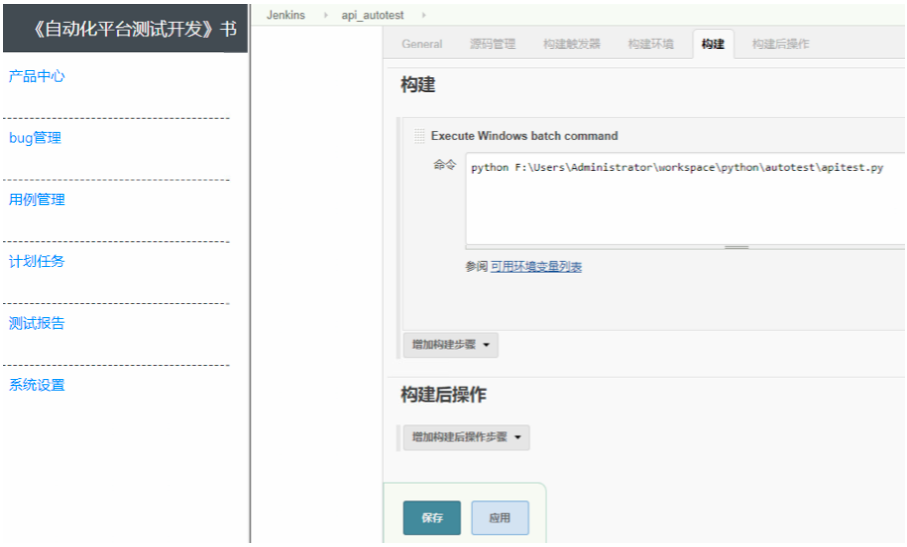
最后，把 Jenkins 集成到 Autotestplat 自动化平台上展示和操作，不需要 Jenkins 进行集成。目前 Jenkins 上的功能是脚本源码管理、定时或即时构建任务、自动化邮件发送等。在后期的版本中会单独在 Autotestplat 平台上加入运行模块实现这三个功能。

Jenkins 集成显示到 Autotestplat 平台很简单，只需在 apitest/templates/left.html 中加入如下链接：<tr><td> 计划任务</td></tr>。

同时保存 Jenkins 服务启动状态，如图 10.28 至 10.30 所示。

▲图 10.28

▲图 10.29



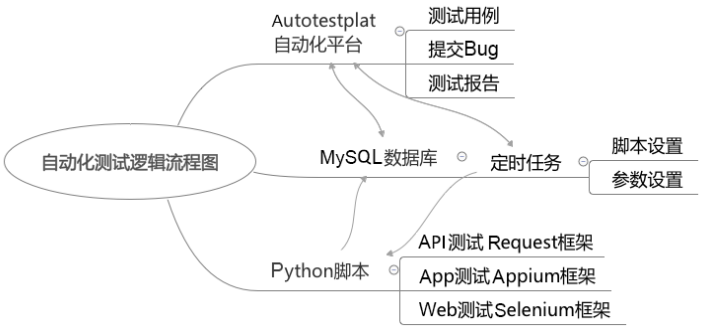
▲图 10.30

第 11 章

定时任务开发

本章介绍使用 DjCelery 即 Django+Celery 框架开发定时任务功能，在 Autotestplat 平台上实现单一接口自动化测试脚本、业务场景接口自动化测试脚本、App 自动化测试脚本、Web 自动化测试脚本等任务的定时执行、调度、管理等，从而取代 Jenkins 上的定时执行脚本和发送邮件等功能。

自动化测试逻辑流程图 11.1 所示。



▲图 11.1

11.1 环境搭建

1．安装

步骤 1 安装 Celery。pyramid_celery-3.0.0，配置 https://pypi.python.org/pypi/pyramid_ celery/。

步骤 2 安装 django-clery。django-celery-3.2.2，配置 https://pypi.python.org/pypi/django- celery。

INSTALLED_APPS= []

加入 2：

```
'djcelery'
```

步骤 5 下载 Redis-x64-3.2.100，Redis-x64-3.2.100.zip https://github.com/MicrosoftArchive/ redis/releases。

2．配置

步骤 1 在 Settings.py 中增加如下内容。

加入 1：

```
import djcelery
djcelery.setup_loader()     #加载 djcelery
```

加入 2:

```
#数据库调度
CELERYBEAT_SCHEDULER = 'djcelery.schedulers.DatabaseScheduler'
```

加入 3:

```
BROKER_URL = 'redis://127.0.0.1:6379/0'
BROKER_TRANSPORT = 'redis'
```

步骤 2 在应用 Apitest 目录下新建 celery.py 文件 1，加入如下内容。

```
from __future__ import absolute_import
import os
import django
from celery import Celery
from django.conf import settings

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'autotest.settings')
django.setup()

app = Celery('autotest')

app.config_from_object('django.conf:settings')
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)
```

步骤 3 新建 tasks.py 文件，加入如下内容。

```
# -*- coding:utf-8 -*-
import requests, time, sys, re
import urllib, zlib#,
import pymysql

import unittest
from trace import CoverageResults
import json
from idlelib.rpc import response_queue
from apitest.celery import app

from time import sleep

@app.task
def hello_world():
    print('已运行')
```

步骤 4 启动服务 python manage.py runserver。

步骤 5 解压缩后，运行 CMD，切换到相应目录，输入启动 Redis 指令 redis-server redis. windows.conf，成功后出现如图 11.2 所示界面。

▲图 11.2

步骤 6 启动指令 python manage.py celery worker -l info。

步骤 7 启动指令 python manage.py celery beat。

11.2 前端功能实现

1．功能描述

完成实现单一接口测试用例、业务场景接口 API 测试用例、AppUI 测试用例、WebUI 测试用例的自动化定时任务。

2．程序清单

在 autotest\apitest\templates 目录下新建 periodic_task.html 文件，加入如下内容。

```
<html lang="zh-CN">
<head>
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
<title>产品自动化测试平台</title>
<link rel="stylesheet" type="text/css" href="/static/admin/css/forms.css" />


<script type="text/javascript" src="/admin/jsi18n/"></script>

<script type="text/javascript" src="/static/admin/js/vendor/jquery/jquery.js"></script>

<script type="text/javascript" src="/static/admin/js/jquery.init.js"></script>
<script type="text/javascript" src="/static/admin/js/core.js"></script>

<script type="text/javascript" src="/static/admin/js/admin/RelatedObjectLookups.js"></script>
<script type="text/javascript" src="/static/admin/js/actions.js"></script>

<script type="text/javascript" src="/static/admin/js/urlify.js"></script>

<script type="text/javascript" src="/static/admin/js/prepopulate.js"></script>
```

```
<script type="text/javascript" src="/static/admin/js/vendor/xregexp/xregexp.js"></script>


<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1.0, maximum-
scale=1.0">

<link rel="stylesheet" type="text/css" href="/static/admin/css/responsive.css" />


<meta name="robots" content="NONE,NOARCHIVE" />


</head>
<body role="document">
<!-- 导航栏-->
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-top">
<div class="container">
<a class="navbar-brand" href="#"> </a>
<ul class="nav justify-content-center">
</ul>
<ul class="nav justify-content-end">
<li class="nav-link"><a style='color:white' href="#"></a></li>
<li class="nav-link"><a style='color:white' href="/logout/"></a></li>
</ul>
</div>
</nav>
<!-- 搜索栏-->
<div class="page-header" style="padding-top: 70px;">
<form class="navbar-form" method="get" action="/tasksearch/">

{% csrf_token %}
<input type="search" name="task" placeholder="名称" required>

<button type="submit">搜索</button>

<!-- 增加定时任务-->
<div style="float:right;width:73%">

<select name="PeriodicTask" id="PeriodicTask">
<option value="" selected>----定时任务----</option>
</select>
<a class="related-widget-wrapper-link change-related" id="change_id_PeriodicTask" data-href-
template="/admin/djcelery/periodictask/__fk__/change/?_to_field=id&amp;_popup=1" title="更改选中的定时任
务">
<img src="/static/admin/img/icon-changelink.svg" alt="修改"/>
</a>
<a  style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_PeriodicTask"
href="/admin/djcelery/periodictask/add/?_to_field=id&amp;_popup=1" title="增加另一个测试用例">
<img src="/static/admin/img/icon-addlink.svg" alt="增加"/>增加
</a>
</form>
</div>


<!-- 任务计划列表-->
<div class="row" style="padding-top: 20px;">
<div class="col-md-11">
<table class="table table-striped">
<thead>
```

```
<tr>
<th>ID</th><th>任务名称</th><th>任务模块</th><th>时间计划</th><th>修改时间</th><th>开启</th><th>立即</th><th>
编辑</th><th>删除</th>
</tr>
</thead>
<tbody>
{% for task in tasks %}{% for periodic in periodics %}
<tr>

{% if task.interval_id != null and task.interval_id == periodic.id %}
<td>{{ task.id }}</td>
<td>{{ task.name }}</td>
<td>{{ task.task }}</td>
<td><a style='color:green'>每{{ periodic.period }} {{ periodic.every }}次</a> </td>
<td>{{ task.date_changed }}</td>
<td>{{ task.enabled }}</td>
<td>{% if task.id == 1 %}
<a href="../task_apis" target="mainFrame">运行</a>
{% elif task.id == 2 %}
<a href="../task_apitest" target="mainFrame">运行</a>
{% else %}
{% endif %}
</td>
<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Apitest"
href="../admin/djcelery/periodictask/{{ task.id }}/change/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-changelink.svg"/></a></td>
<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Apitest"
href="../admin/djcelery/periodictask/{{ task.id }}/delete/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-deletelink.svg"/></a></td>

{% else %}

{% endif %}


{% for crontab in crontabs %}
{% if task.crontab_id != null and task.crontab_id == crontab.id and task.interval_id == 1 %}
<td>{{ task.id }}</td>
<td>{{ task.name }}</td>
<td>{{ task.task }}</td>
<td><a style='color:green'>{{ crontab.month_of_year }}年{{ crontab.day_of_month }}月
{{ crontab.day_of_week }}日{{ crontab.hour }}时{{ crontab.minute }}分</a> </td>
<td>{{ task.date_changed }}</td>
<td>{{ task.enabled }}</td>
<td><a href="../task_apis" target="mainFrame">运行</a></td>
<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Apitest"
href="../admin/djcelery/periodictask/{{ task.id  }}/change/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-changelink.svg"/></a></td>
<td><a style='color:light blue' class="related-widget-wrapper-link add-related" id="add_id_Apitest"
href="../admin/djcelery/periodictask/{{ task.id  }}/delete/?_to_field=id&_popup=1"><img
src="/static/admin/img/icon-deletelink.svg"/></a></td>
{% else %}
{% endif %}
{% endfor %}{% endfor %}{% endfor %}

</tbody>
</table>
</div>
</div>
```

```
<span style="position:absolute; right:100px; bottom:30px;">   {# 把翻页功能固定显示在右下角#}

<div style="position:absolute;  right:100px; width:100px; ">
<tr><th>总数</th><td>{{ taskcounts }}</td></tr> {# 前端读取定义的变量#}
</div>

<div class="container">
    <ul class="pagination" id="pager">
          {# 上一页链接开始#}
        {% if tasks.has_previous %}
            {#  如果有上一页，则正常显示上一页链接#}
            <li class="previous"><a href="/periodic_task/?page={{ tasks.previous_page_number }}">上一页
</a></li>    {#  上一页标签 #}
        {% else %}
            <li class="previous disabled"><a href="#">上一页</a></li> {# 如果当前不存在上一页，则上一页的链接
不可单击#}
        {% endif %}
        {# 上一页链接开始#}

        {% for num in tasks.paginator.page_range %}

            {% if num == currentPage %}
                <li class="item active"><a href="/periodic_task/?page={{ num }}">{{ num }}</a></li> {#
显示当前页数链接#}
            {% else %}
                <li class="item"><a href="/periodic_task/?page={{ num }}">{{ num }}</a></li>
            {% endif %}
        {% endfor %}

        {# 下一页链接开始#}
        {% if tasks.has_next %} {#  如果有下一页，则正常显示下一页链接#}
            <li class="next"><a href="/periodic_task/?page={{ tasks.next_page_number }}">下一页
</a></li>
        {% else %}
            <li class="next disabled"><a href="#">下一页</a></li>
        {% endif %}
        {# 下一页链接结束#}
    </ul>
</div>

</body>
</html>
```

功能描述：实现自动化测试任务调度执行，包括单一接口、扫描、流程接口、业务场景、Web 搜索、自动化平台测试开发、App 登录，CSDN 定时任务注册，定时任务执行等功能。

程序清单：在 apitest/views.py 中加入如下内容。

```
from .tasks import hello_world
from .tasks import test_readSQLcase
from djcelery.models import PeriodicTask,CrontabSchedule,IntervalSchedule


# 任务计划
@login_required
def periodic_task(request):
    username = request.session.get('user', '')
    task_list = PeriodicTask.objects.all()
    task_count = PeriodicTask.objects.all().count()  #统计数
    periodic_list = IntervalSchedule.objects.all()  # 周期任务（如每隔 1 小时执行 1 次）
    crontab_list = CrontabSchedule.objects.all()    # 定时任务（如某年某月某日的某时，每# 天的某时）
```

```
    paginator = Paginator(task_list, 5)  #生成 paginator 对象,设置每页显示 5 条记录
    page = request.GET.get('page',1)  #获取当前的页码数,默认为第 1 页
    currentPage=int(page)  #把获取的当前页码数转换成整数类型
    try:
        task_list = paginator.page(page)#获取当前页码数的记录列表
    except PageNotAnInteger:
        task_list = paginator.page(1)#如果输入的页数不是整数，则显示第 1 页内容
    except EmptyPage:
        task_list = paginator.page(paginator.num_pages)#如果输入的页数不在系统的页数中，# 则显示最后一页内容
    return render(request, "periodic_task.html", {"user": username,"tasks": task_list,"taskcounts":
task_count, "periodics": periodic_list,"crontabs": crontab_list })

# 搜索功能
@login_required
def tasksearch(request):
    username = request.session.get('user', '') # 读取浏览器登录 Session
    search_name = request.GET.get("task", "")
    task_list = PeriodicTask.objects.filter(task__icontains=search_name)
    periodic_list = IntervalSchedule.objects.all()  # 周期任务（如每隔 1 小时执行 1 次）
    crontab_list = CrontabSchedule.objects.all()    # 定时任务（如某年某月某日的某时，每# 天的某时）
    return render(request,'periodic_task.html', {"user": username,"tasks":task_list,"periodics":
periodic_list,"crontabs": crontab_list })
```

在 autotest/urls.py 中加入：

```
path('periodic_task/', views.periodic_task),
    path('tasksearch/', views.tasksearch),
```

在 apitest/left.html 中加入：

```
<tr> <td>
                    <li>
                        <a  href="../periodic_task" target="mainFrame">
                            <i class="glyphicon glyphicon-fire"></i>
                           任务计划
                        </a>
                    </li>
</tr> </td>
            <tr><td> </td></tr>
```

查看前端页面效果，如图 11.3 所示。

ID	任务名称	任务模块	时间计划	修改时间	开启	立即
1	单一接口扫描测试	apitest.tasks.apisauto_testcase	每hours 1次	2018年1月22日 20:02	True	运行
2	业务流程接口测试	apitest.tasks.apitest_testcase	每hours 1次	2018年1月22日 20:01	True	运行
3	web搜索：自动化平台测试开发	apitest.tasks.webauto_testcase	每hours 1次	2018年1月22日 20:03	True	运行
5	web搜索：软件自动化测试开发	apitest.tasks.webauto_testcase2	每hours 1次	2018年1月22日 20:03	True	运行
6	app登录用例	apitest.tasks.appauto_testcase	每hours 1次	2018年1月22日 11:00	True	运行

▲图 11.3

11.3 定时任务测试源码

修改本书第 6～8 章的自动化测试源码，加到定时任务中去执行。

11.3.1 接口扫描自动化测试源码

1．功能描述

注册到定时任务，运行单一接口测试用例，单一接口是指接口之间没有太多关联，比如仅仅是未登录用户的查询操作或登录后进行的某一步操作。

2．程序清单

在新建的 tasks.py 文件中，加入如下内容。

```
import requests, time, sys, re
import urllib, zlib
import pymysql

import unittest
from trace import CoverageResults
import json
from idlelib.rpc import response_queue
from apitest.celery import app

from time import sleep

import os
from selenium import webdriver
#from appium import webdriver

PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)

global driver

@app.task
def hello_world():
    print('已运行')

@app.task
def apisauto_testcase():
    sql="SELECT id,`apiname`,apiurl,apimethod,apiparamvalue,apiresult,`apistatus` from apitest_apis "
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    print (info)
    for ii in info:
        case_list = []
        case_list.append(ii)
        interfaceTest1(case_list)
    coon.commit()
    cursor.close()
    coon.close()

def interfaceTest1(case_list):
    res_flags = []
    request_urls = []
```

```
    responses = []
    strinfo = re.compile('{seturl}')
    for case in case_list:
        try:
            case_id = case[0]
            interface_name = case[1]
            method = case[3]
            url = case[2]
            param = case[4]
            res_check = case[5]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        if param== '':
            new_url = 'http://'+url
        elif param== 'null':
            url = strinfo.sub(str(seturl('seturl')),url)
            new_url = 'http://'+ url
        else:
            url = strinfo.sub(str(seturl('seturl')),url)
            new_url = 'http://'+url
            request_urls.append(new_url)
        if method.upper() == 'GET':
            headers = {'Authorization':'', 'Content-Type': 'application/json' }
            if "=" in urlParam(param):
                data = None
                print (str(case_id)+' request  is get' +new_url.encode('utf-
8')+'?'+urlParam(param).encode('utf-8'))
                results = requests.get(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is get'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (' request  is get ' +new_url+'   body is '+urlParam(param))
                data = None
                req = urllib.request.Request(url=new_url,data=data,headers=headers,method="GET")
                try:
                    results = urllib.request.urlopen(req).read()
                    print ('   response is get1 ')
                    print(results)
                except Exception as e:
                    return caseWriteResult1(case_id,results,'0')
                res = readRes(results,res_check)
                print(res)
                if 'pass' == res:
                    res_flags.append('pass')
                    caseWriteResult1(case_id,results,'1')
                else:
                    res_flags.append('fail')
                    caseWriteResult1(case_id,results,'0')

writeBug(case_id,interface_name,new_url,results,res_check)
        if  method.upper()=='PUT':
            headers = {'Host':HOSTNAME, 'Connection':'keep-alive',  'CredentialId':id, 'Content-Type':
'application/json'}
            body_data=param
            results = requests.put(url=url,data=body_data,headers=headers)
            responses.append(results)
            res = readRes(results,res_check)
            if 'pass' == res:
          #      writeResult(case_id,'pass')
```

```
                res_flags.append('pass')
            else:
                res_flags.append('fail')
          #     writeResult(case_id,'fail')

writeBug(case_id,interface_name,new_url,results,res_check)
        if method.upper()=="PATCH":
            headers = {'Authorization':'Credential '+id, 'Content-Type': 'application/json' }
            data = None
            results = requests.patch(new_url+'?'+urlParam(param),data,headers=headers).text
            responses.append(results)
            res = readRes(results,res_check)
            if 'pass' == res:
           #     writeResult(case id,'pass')
                res_flags.append('pass')
            else:
                res flags.append('fail')
          #      writeResult(case_id,'fail')
                writeBug(case_id,interface_name,new_url,results,res_check)

        if method.upper()=="POST":
            headers = {'Authorization':'', 'Content-Type': 'application/json' }
            if "=" in urlParam(param):
                data = None
                print (str(case_id)+' request  is get' +new_url.encode('utf-
8')+'?'+urlParam(param).encode('utf-8'))
                results = requests.get(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is get'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (' request  is get ' +new url+'   body is '+urlParam(param))
                data = None
                req = urllib.request.Request(url=new_url,data=data,headers=headers,method="POST")
                try:
                    results = urllib.request.urlopen(req).read()
                    print ('   response is get1 ')
                    print(results)
                except Exception as e:
                    return caseWriteResult1(case_id,results,'0')
                res = readRes(results,res check)
                print(res)
                if 'pass' == res:
                    res_flags.append('pass')
                    caseWriteResult1(case_id,results,'1')
                else:
                    res_flags.append('fail')
                    caseWriteResult1(case_id,results,'0')

writeBug(case_id,interface_name,new_url,results,res_check)

def readRes(res,res_check):
    res = res.decode().replace('":"',"=").replace('":',"=")
    res_check = res_check.split(';')
    for s in res_check:
        if s in res:
            pass
        else:
            return  '错误，返回参数和预期结果不一致'+s
    return 'pass'
```

```
def urlParam(param):
    param1=param.replace('&quot;','"')
    return param1

def CredentialId():
    global id
    url = 'http://'+'api.test.com.cn'+'/api/Security/Authentication/Signin/web'
    body_data= json.dumps({"Identity":'test',"Password":'test'})
    headers = { 'Connection':'keep-alive','Content-Type': 'application/json'}
    response = requests.post(url=url,data=body data,headers=headers)
    data=response.text
    regx = '.*"CredentialId":"(.*)","Scene"'
    pm = re.search(regx, data)
    id = pm.group(1)

def seturl(set):
    global setvalue
    sql = "SELECT `setname`,`setvalue` from set_set"
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    print  (info)
    coon.commit()
    cursor.close()
    coon.close()
    if info[0][0] == set:
        setvalue = info[0][1]
        print (setvalue)
    return setvalue


def caseWriteResult1(case_id,response,result):
    result = result.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apitest_apis set
apitest_apis.apiresponse=%s,apitest_apis.apistatus=%s,apitest_apis.create_time=%s where
apitest apis.id=%s;"
    param = (response,result,now,case_id)
    print ('api autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
    coon.commit()
    cursor.close()
    coon.close()

def writeBug(bug_id,interface_name,request,response,res_check):
    interface_name = interface_name.encode('utf-8')
    res_check = res_check.encode('utf-8')
    request = request.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    bugname = str(bug_id)+ '_' + interface_name.decode() + '_出错了'
    bugdetail = '[请求数据]<br />'+request.decode()+'<br/>'+'[预期结果]<br/>'+res_
check.decode()+'<br/>'+'<br/>'+'[响应数据]<br />'+'<br/>'+response.decode()
```

```
    print (bugdetail)
    sql = "INSERT INTO `bug_bug` ("\
    "`bugname`,`bugdetail`,`bugstatus`,`buglevel`, `bugcreater`,
`bugassign`,`created_time`,`Product_id`)  "\
    "VALUES ('%s','%s','激活','3','邹辉', '邹辉', '%s',
'2');"%(bugname,pymysql.escape_string(bugdetail),now)
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql)
    coon.commit()
    cursor.close()
    coon.close()
```

11.3.2 流程接口自动化测试源码

1．功能描述

注册到定时任务，运行流程接口测试用例，流程接口是指接口之间有动态关联，从而完成一个业务场景。通常下一个接口要用到上一个接口的某个参数值，脚本中要用参数化去代替这个动态取值。比如，程序清单源码中的 seturl 就是进行参数化取静态值，TaskId、preOrderSN 进行参数化取动态值。

2．程序清单

在新建的 tasks.py 文件中，加入如下内容。

```
@app.task
def apitest_testcase(self):
    sql="SELECT id,`apiname`,apiurl,apimethod,apiparamvalue,apiresult,`apistatus` from apitest_apistep
where apitest_apistep.Apitest_id=2 "
    coon = pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,
host='127.0.0.1',charset='utf8')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    print (info)
    for ii in info:
        case_list = []
        case_list.append(ii)
    #   CredentialId()
        interfaceTest(case_list)
    coon.commit()
    cursor.close()
    coon.close()

def interfaceTest(case_list):
    res_flags = []
    request_urls = []
    responses = []
    strinfo = re.compile('{seturl}')
    strinfo2 = re.compile('{TaskId}')
    tasknoinfo = re.compile('{taskno}')
 #   schemainfo = re.compile('{schema}')
    for case in case_list:
        try:
            case_id = case[0]
            interface_name = case[1]
```

```
            method = case[3]
            url = case[2]
            param = case[4]
            res_check = case[5]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        if param== '':
            new_url = 'http://'+'api.test.com.cn'+url
        elif param== 'null':
            url = strinfo.sub(str(seturl('seturl')),url)
            new_url = 'http://'+url
        else:
          #  url = schemainfo.sub(mod_config.getConfig('project', "schema"),url)
            url = strinfo.sub(str(seturl('seturl')),url)
          #  url = strinfo2.sub(TaskId,url)
            param = strinfo.sub(TaskId,param)
            param = tasknoinfo.sub(taskno,param)
            new_url = 'http://'+url
            request_urls.append(new_url)
        if method.upper() == 'GET':
          #  print str(case_id)+' ' +new_url
            headers = {'Authorization':'', 'Content-Type': 'application/json' }
            if "=" in urlParam(param):
                data = None
                print (str(case_id)+' request  is get' +new_url.encode('utf-
8')+'?'+urlParam(param).encode('utf-8'))
                results = requests.get(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is get'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (' request  is get ' +new_url+'   body is '+urlParam(param))
              #  results = requests.get(new_url,headers=headers).text #data='',=urlParam(param)

                data = None
                req = urllib.request.Request(url=new_url,data=data,headers=headers,method="GET")
                results = urllib.request.urlopen(req).read()
                print ('   response is get ')
                print(results)
              #  responses.append(results)
                res = readRes(results,res_check)
            #print results
                if 'pass' == res:
                    res_flags.append('pass')
                    writeResult(case_id,results,'1')
                    caseWriteResult(case_id,'1')
                else:
                    res_flags.append('fail')
                    writeResult(case_id,results,'0')
                    caseWriteResult(case_id,'0')

writeBug(case_id,interface_name,new_url,results,res_check)
        if  method.upper()=='PUT':
         #   print str(case_id)+'put'+' '+'http://'+'api.test.com.cn/api'+url
          #  print param
            headers = {'Host':HOSTNAME, 'Connection':'keep-alive',  'CredentialId':id, 'Content-Type':
'application/json'}
            body_data=param
            results = requests.put(url=url,data=body_data,headers=headers)
            responses.append(results)
```

```
            res = readRes(results,res_check)
       #     print results
            if 'pass' == res:
                writeResult(case_id,'pass')
                res_flags.append('pass')
            else:
                res_flags.append('fail')
                writeResult(case id,'fail')

writeBug(case_id,interface_name,new_url,results,res_check)
            try:
                preOrderSN(results)
            except:
                print ('ok')
        if method.upper()=="PATCH":
            headers = {'Authorization':'Credential '+id, 'Content-Type': 'application/json' }
            data = None
         #   print str(case_id)+' request  is ' +new_url+'?'+urlParam(param)
            results = requests.patch(new_url+'?'+urlParam(param),data,headers=headers).text
         #   print '   response is '+results
            responses.append(results)
            res = readRes(results,res_check)
            #print results
            if 'pass' == res:
                writeResult(case_id,'pass')
                res flags.append('pass')
            else:
                res_flags.append('fail')
                writeResult(case id,'fail')

writeBug(case_id,interface_name,new_url,results,res_check)
            try:
                preOrderSN(results)
            except:
                print ('ok')
        if method.upper()=="POST":
            headers = {'Authorization':'', 'Content-Type': 'application/json' }
            if "=" in urlParam(param):
                data = None
                print (str(case_id)+' request  is get' +new_url.encode('utf-
8')+'?'+urlParam(param).encode('utf-8'))
                results = requests.get(new_url+'?'+urlParam(param),data,headers=headers).text
                print ('   response is get'+results.encode('utf-8'))
                responses.append(results)
                res = readRes(results,'')
            else:
                print (' request  is get ' +new_url+'   body is '+urlParam(param))
                data = None
                req = urllib.request.Request(url=new_url,data=data,headers=headers,method="POST")
                try:
                    results = urllib.request.urlopen(req).read()
                    print ('   response is get1 ')
                    print(results)
                except Exception as e:
                    return caseWriteResult1(case_id,results,'0')
                res = readRes(results,res_check)
                print(res)
                if 'pass' == res:
                    res_flags.append('pass')
                    caseWriteResult1(case_id,results,'1')
```

```
                else:
                    res_flags.append('fail')
                    caseWriteResult1(case_id,results,'0')

writeBug(case_id,interface_name,new_url,results,res_check)
            try:
                TaskId(results)
            except:
                print ('ok1')

def readRes(res,res check):
    res = res.decode().replace('":"',"=").replace('":',"=")
    res_check = res_check.split(';')
    for s in res check:
        if s in res:
            pass
        else:
            return  '错误，返回参数和预期结果不一致'+s
    return 'pass'

def urlParam(param):
    param1=param.replace('&quot;','"')
    return param1

def CredentialId():
    global id
    url = 'http://'+'api.test.com.cn'+'/api/Security/Authentication/Signin/web'
    body_data= json.dumps({"Identity":'test',"Password":'test'})
    headers = { 'Connection':'keep-alive','Content-Type': 'application/json'}
    response = requests.post(url=url,data=body_data,headers=headers)
    data=response.text
    regx = '.*"CredentialId":"(.*)","Scene"'
    pm = re.search(regx, data)
    id = pm.group(1)

def seturl(set):
    global setvalue
    sql = "SELECT `setname`,`setvalue` from set_set"
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    print  (info)
    coon.commit()
    cursor.close()
    coon.close()
    if info[0][0] == set:
        setvalue = info[0][1]
        print (setvalue)
    return setvalue

def preOrderSN(results):
    global preOrderSN
    regx = '.*"preOrderSN":"(.*)","toHome"'
    pm = re.search(regx, results)
    if pm:
        preOrderSN = pm.group(1).encode('utf-8')
        return preOrderSN
```

```
    return False

def TaskId(results):
    global TaskId
    regx = '.*"TaskId":(.*),"PlanId"'
    pm = re.search(regx, results)
    if pm:
        TaskId = pm.group(1).encode('utf-8')
      #  print TaskId
        return TaskId
    return False

def taskno(param):
    global taskno
    a = int(time.time())
    taskno='task_'+str(a)
    return taskno

def writeResult(case_id,response,result):
    result = result.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apitest_apistep set
apitest apistep.apiresponse=%s,apitest apistep.apistatus=%s,apitest apistep.create time=%s where
apitest_apistep.id=%s;"
    param = (response,result,now,case_id)
    print ('api autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
    coon.commit()
    cursor.close()
    coon.close()

def caseWriteResult(case_id,result):
    result = result.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    sql = "UPDATE apitest_apitest set apitest_apitest.apitestresult=%s,apitest_apitest.create_time=%s
where apitest_apitest.id=%s;"
    param = (result,now,case id)
    print ('api autotest result is '+result.decode())
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql,param)
    coon.commit()
    cursor.close()
    coon.close()

def writeBug(bug_id,interface_name,request,response,res_check):
    interface_name = interface_name.encode('utf-8')
    res_check = res_check.encode('utf-8')
    request = request.encode('utf-8')
    now = time.strftime("%Y-%m-%d %H:%M:%S")
    bugname = str(bug_id)+ '_' + interface_name.decode() + '_出错了'
    bugdetail = '[请求数据]<br />'+request.decode()+'<br/>'+'[预期结果]<br/>'+res_
check.decode()+'<br/>'+'<br/>'+'[响应数据]<br />'+'<br/>'+response.decode()
    print (bugdetail)
```

```
    sql = "INSERT INTO `bug_bug` ("\
    "`bugname`,`bugdetail`,`bugstatus`,`buglevel`, `bugcreater`,
`bugassign`,`created_time`,`Product_id`)  "\
    "VALUES ('%s','%s','激活','3','邹辉', '邹辉', '%s',
'2');"%(bugname,pymysql.escape_string(bugdetail),now)
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    cursor.execute(sql)
    coon.commit()
    cursor.close()
    coon.close()
```

11.3.3 App 自动化测试源码

1．功能描述

注册到定时任务，实现 App 自动化进行 Android 模拟器上的计算器 1+1=2 的测试用例功能。

2．程序清单

在新建的 apptasks.py 文件中，加入如下内容。

```
from appium import webdriver

PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)

global driver

@app.task
def appauto_testcase(self):
    desired_caps = {}
    desired_caps['platformName'] = 'Android'
    desired_caps['platformVersion'] = '4.4'
    desired_caps['deviceName'] = 'emulator-5554'
    desired_caps['appPackage'] = 'com.android.calculator2'
    desired_caps['appActivity'] = '.Calculator'
#    desired_caps['app'] = PATH('csdn.apk')
    time.sleep(1)
    self.driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired_caps)
    time.sleep(1)
    sql="SELECT id,appfindmethod,appevelement,appoptmethod,appassertdata,`apptestresult` from
apptest_appcasestep where apptest_appcasestep.Appcase_id=1 ORDER BY id ASC "
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    for ii in info:
        case_list = []
        case_list.append(ii)
        apptestcase(self,case_list)
    coon.commit()
    cursor.close()
```

```
    coon.close()
    self.driver.quit()


def apptestcase(self,case_list):
    for case in case_list:
        try:
            case_id = case[0]
            findmethod = case[1]
            evelement = case[2]
            optmethod = case[3]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        print (evelement)
        time.sleep(10)
        if optmethod== 'click' and findmethod=='find_element_by_id':

self.driver.find_element_by_id(evelement).send_keys('test')
        elif optmethod== 'click' and findmethod=='find_element_by_name':
            self.driver.find_element_by_name(evelement).click()
        elif optmethod=='sendkey':

self.driver.find_element_by_name(evelement).send_keys()
```

11.3.4 Web 自动化测试源码

1．功能描述

注册到定时任务，实现 Web 自动化进行百度搜索：软件自动化测试开发、自动化平台测试开发的测试用例功能。

2．程序清单

步骤 1 在新建的 webtasks.py 文件中，加入如下内容。

```
  import os
from selenium import webdriver

PATH=lambda p:os.path.abspath(
os.path.join(os.path.dirname(__file__),p)
)

global driver
@app.task
def webauto_testcase(self):             #Web 测试用例
    self.driver =webdriver.Firefox()
    self.driver.get("http://www.baidu.com")
    sql="SELECT id,webfindmethod,webevelement,weboptmethod,webtestdata,webassertdata,`webtestresult`
from webtest_webcasestep where webtest_webcasestep.Webcase_id=2 ORDER BY id ASC "
    coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charset='utf8
')
    cursor = coon.cursor()
    aa=cursor.execute(sql)
    info = cursor.fetchmany(aa)
    for ii in info:
        case_list = []
        case_list.append(ii)
```

```
        webtestcase(self,case_list)
    coon.commit()
    cursor.close()
    coon.close()
    self.driver.quit()


def webtestcase(self,case list):
    for case in case_list:
        try:
            case id = case[0]
            findmethod = case[1]
            evelement = case[2]
            optmethod = case[3]
            testdata = case[4]
        except Exception as e:
            return '测试用例格式不正确！%s'%e
        print (case)
        time.sleep(5)
        if optmethod=='sendkeys' and findmethod=='find_element_by_id':
            print (evelement)
            self.driver.find_element_by_id(evelement).send_keys(testdata)
        elif optmethod=='click' and findmethod=='find_element_by_name':
            print (evelement)
            self.driver.find_element_by_name(evelement).click()
        elif optmethod=='click' and findmethod=='find_element_by_id':
            print (evelement)
            self.driver.find_element_by_id(evelement).click()
```

11.4 定时任务调用

步骤 1 在 apitest/views.py 中加入如下内容。

```
import pymysql

from .tasks import hello_world
from .tasks import apisauto_testcase
from .tasks import apitest testcase
from .webtasks import webauto_testcase
from .webtasks import webauto_testcase2
from .apptasks import appauto testcase
from .apptasks import appauto_testcase2
from djcelery.models import PeriodicTask,CrontabSchedule,IntervalSchedule
# Create your views here.

def task_apis(request):
    apisauto_testcase.delay()
    return HttpResponse("已运行")

def task_apitest(request):
    apitest_testcase(request)
    return HttpResponse("已运行")

def task_webtest(request):
    webauto_testcase(request)
    return HttpResponse("已运行")
```

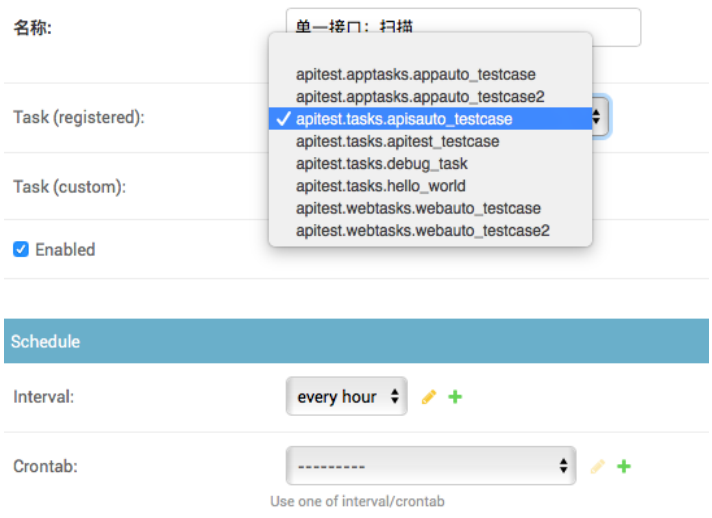
```
def task_webtest2(request):
    webauto_testcase2(request)
    return HttpResponse("已运行")

def task_apptest(request):
    appauto_testcase(request)
    return HttpResponse("已运行")

def task_apptest2(request):
    appauto_testcase2(request)
return HttpResponse("已运行")
```

步骤 2 在 autotest/urls.py 中加入如下内容。

```
from apitest import views
path('task_apis/', views.task_apis),
    path('task_apitest/', views.task_apitest),
    path('task_webtest/', views.task_webtest),
    path('task_webtest2/', views.task_webtest2),
    path('task_apptest/', views.task_apptest),
    path('task_apptest2/', views.task_apptest2),
```

步骤 3 前端添加任务。

在 Django 的 admin 后台 Djcelery-Periodic tasks 中添加注册任务，包括任务名称，如 apitest_testcase、webauto_testcase、appauto_testcase，以及执行周期，如每小时一次、每天一次等，如图 11.4 所示。



▲图 11.4

11.5 新增一个 Web 自动化用例、修改脚本、新增定时任务：

1 新增用例

单一接口 流程接口 app用例 web用例

产品	所属用例	步聚	定位方式	控件元素	操作方法	测试数据	验证数据	测试结果
蜜蜂专办	case_5_B端后台登录我的任务	第一步:用户名	find_element_by_id	loginName	sendkeys	admin	按揭帮	True
蜜蜂专办	case_5_B端后台登录我的任务	第二步:密码	find_element_by_id	loginPass	sendkeys	888888	按揭帮	True
蜜蜂专办	case_5_B端后台登录我的任务	第三步:点击登录	find_element_by_xpath	/html/body/div/div[2]/input	click	null	null	True
蜜蜂专办	case_5_B端后台登录我的任务	第四步:点击案件任务	find_element_by_xpath	/html/body/div[2]/div[1]/div[2]/ul/li[2]/a	click	null	案件&任务 aaaaaa	False
蜜蜂专办	case_5_B端后台登录我的任务	第五步:点击我的任务	find_element_by_xpath	/html/body/div[2]/div[1]/div[2]/ul/li[2]/div/a[1]	click	null	我的任务 123456	False

返回 编辑

用例 ID, 定位方式(id,name,xpath 等),操作方法(sendkeys,click 等),控件元素（ie 中 f12 抓取）
测试数据(data,null),验证数据(元素 text)

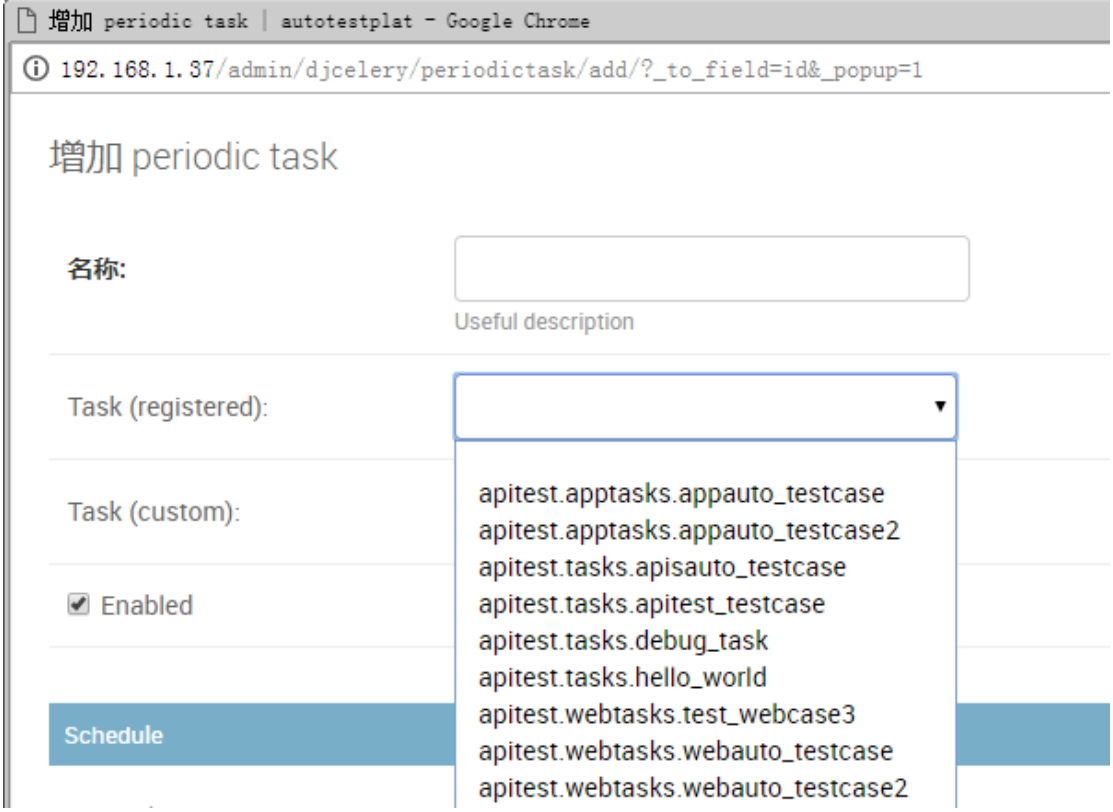
2 测试用例脚本修改

C:\Users\Administrator\AppData\Local\Programs\Python\Python36\Scripts\autotest\apitest\Webtasks.py 中, copy 并新增如下代码,并修改用例名称 webauto_testcase 为 webauto_testcaseid，以及修改 b.Webcase_id=3 为新增用例的 b.Webcase_id=id 对应。其中 id 为第 1 步中的用例 ID 数值。

```
    @app.task
    def webauto_testcase3(self):                #流程 的 相关接口
        self.driver2 =webdriver.Firefox()
        self.driver2.get("http://bs1.beerich.cn:12102/beerich/")
        sql="SELECT
b.id,b.webfindmethod,b.webevelement,b.weboptmethod,b.webtestdata,b.webassertdata,b.`webtestresu
lt`,a.webcasename,b.webteststep from webtest_webcasestep as b,webtest_webcase as a where
b.Webcase_id=a.id AND b.Webcase_id=3 ORDER BY id ASC "
        coon =
pymysql.connect(user='root',passwd='test123456',db='autotest',port=3306,host='127.0.0.1',charse
t='utf8')
        cursor = coon.cursor()
        aa=cursor.execute(sql)
        info = cursor.fetchmany(aa)
        for ii in info:
            case_list = []
            case_list.append(ii)
            webtestcase(self,case_list)
        coon.commit()
        cursor.close()
        coon.close()
        self.driver2.quit()
```

3 新增定时任务

填入名称,周期,在 Task(registered)选择对应第 2 步脚本中的用例名称如 **webauto_testcase3**。



4 修改 view.py 文件

C:\Users\Administrator\AppData\Local\Programs\Python\Python36\Scripts\autotest\apitest\ views.py 中加入如下函数名称 task_webtest3 内容：

```
def task_webtest3(request):
    webauto_testcase3(request)          说明：这里与第 2 步用例名称相对应
return HttpResponse("已运行")
```

修改 urls.py 文件

C:\Users\Administrator\AppData\Local\Programs\Python\Python36\Scripts\autotest\autotest\ urls.py 中加入如下内容：（函数名称相对应）

```
path('task_webtest3/', views.task_webtest3),
```

修改 periodic_task.html 文件

C:\Users\Administrator\AppData\Local\Programs\Python\Python36\Scripts\autotest\apitest\templates\periodic_task.html

页面的 2 个地方分别加入如下内容：（第 2 步中的 id，以及函数名称 task_webtest3 相对应）

```
{% elif task.id == 10 %}
<a href="../ task_webtest3" target="mainFrame">运行</a>
```

5 重启后台

python manage.py runserver 0.0.0.0:80 , 登录系统调试运行

说明：目前新增 App 自动化测试用例、脚本、任务；Api 自动化测试用例、脚本、任务都与以上步聚类似。后续将做成类似于 Jmeter，soapui 等接口测试工具可以进行参数化，尽可能减少或完全不需要代码维护。

附录A

常用软件安装包链接

1．官方网站汇总链接地址

Eclipse：http://www.eclipse.org/downloads/
JDK：http://www.oracle.com/technetwork/java/javase/downloads/index.html
SDK：http://android-sdk.en.softonic.com/
ADT：http://tools.android-studio.org/index.php
Node.js：https://nodejs.org/en/
Ant：http://ant.apache.org/
Appium：http://appium.io/
Jenkins：https://jenkins.io/index.html
Python：https://www.python.org/downloads/
Pydev：http://www.pydev.org/download.html
Postman：https://www.getpostman.com/
SVN：https://tortoisesvn.net/downloads.html
MySQL：https://www.mysql.com/

2．百度云盘汇总链接地址：

JDK 1.7 安装文件路径 https://pan.baidu.com/s/1o8B4uPs
Python 3.6.4 安装文件路径 https://pan.baidu.com/s/1diUgNzfMXgLhMQZjNNcNKA
Django 2.0 安装文件路径 https://pan.baidu.com/s/1L1dp-TJF_joLZ4ex7hVwSQ
PyMySQL 安装文件路径 https://pan.baidu.com/s/1It8L6z49nXpimmawOUEUcA
Requests-master 安装文件路径 https://pan.baidu.com/s/144Gb_aoiKX_BYwwdhrb8wA
MySQL 5.7.17 安装文件路径 https://pan.baidu.com/s/1BrZ7g7D5yslfx-rJb-AA3Q

Redis-x64-3.2.100 安装文件路径 https://pan.baidu.com/s/1QKkNm_c51xLq0COLIS1lXA
Selenium3.8.1 安装文件路径 https://pan.baidu.com/s/1fSWaqtHNQrwC51Ek9HI-GA

Appium-Python-Client-0.26 安装文件路径 https://pan.baidu.com/s/1j4emb7T22AhTeUTNM0PnSQ

Locust 安装文件路径 https://pan.baidu.com/s/1Z3gMnoK-Xo8dk_Rd5T2dJw

geckodriver.exe 安装文件路径 https://pan.baidu.com/s/15b8AQHtmLlvnGsPfAHnHZA

SVN 安装文件路径 https://pan.baidu.com/s/1o7YUHFc

Jenkins 1.64 安装文件路径 https://pan.baidu.com/s/1gfAwIen

Eclipse 4.5.2 安装文件路径 https://pan.baidu.com/s/1pK84oen

SDK r24.4.1 安装文件路径 https://pan.baidu.com/s/1cdwUce

ADT 安装文件路径 23.0.7 https://pan.baidu.com/s/1nvtp39B

Nodejs x64 安装文件路径 https://pan.baidu.com/s/1dEIYaDr

.Net 4.5 安装文件路径 https://pan.baidu.com/s/1milPZ5Y

Autotestplat 自动化平台源代码包路径 https://pan.baidu.com/s/1d_A9a11fEeJmd0O8dM36uQ

Autotestplat 自动化平台初始化数据 autotest.sql 路径 https://pan.baidu.com/s/1RxFwFBFjeFtjTPeXy82BKA

3．Github 下载链接地址

Autotestplat 自动化平台源代码包路径：https://github.com/testdevhome/autotestplat

Autotestplat 自动化平台各依赖安装包路径：https://github.com/testdevhome/package

Autotestplat 自动化平台初始化数据文件路径：https://github.com/testdevhome/package

附录B

Autotestplat 使用指南

1．下载或安装软件包

安装 Python 3.6.4
安装 Django 2.0
安装 PyMySQL
安装 Requests-master
安装 MySQL 5.7.17
安装 Django-bootstrap4-master
安装 Django-celery-3.2.2
安装 Celery-with-redis-3.0
下载 Redis-x64-3.2.100
下载 Autotestplat v1.0
安装 Selenium3.8.1
安装 Firefox 57.0.4
安装 Appium-Python-Client-0.26
安装 Locust

以上安装文件下载地址：https://pan.baidu.com/s/1j58bo_UGiTZ9kIOz_XCFHQ

安装方法：分别进入压缩后的相应目录，运行 CMD，输入 Python setup.py install

2．配置启动服务

步骤 1 运行 CMD，切换到相应目录，创建 Autotest 项目，python manage.py startproject autotest。

步骤 2 从百度网盘下载文件：https://pan.baidu.com/s/1d_A9a11fEeJmd0O8dM36uQ，然后解压缩，把 autotestplat v1.0 覆盖到 Script 目录下。

步骤 3 通过 MySQL 客户端工具 Navicat 连接上新建数据库 autotest。

Python manage.py createsuperuser
Python manage.py runserver

步骤 5 运行 CMD，进入 Redis-x64-3.2.100 解压缩目录，输入 redis-server redis-windows- conf 启动 Redis 服务。

步骤 6 运行 CMD，进入到 Autotest 项目目录，启动 Celery 定时任务。

Python manage celery worker –l info
Python manage.py celery beat
Locust -f performance.py --host=http://127.0.0.1

3．登录 Autotestplat 平台

在浏览器中输入 http://127.0.0.1:8000/login，用户名为 admin，密码为 test123456。

4．导入 autotest.sql 数据到平台或录入产品、用例，增加定时任务，调试，运行。

5．Autotestplat 官网及体验地址：http://www.autotestplat.com。

后 记

在 IT 互联网行业，只有更快、更好的创新，才能使得企业保持持续的竞争力。国外的苹果、微软、IBM、谷歌等公司，每年都在更新软件和硬件，而国内的百度、阿里、腾讯、华为等公司，也在不断地快速扩张，保持产品的迭代更新。

自动化技术也在不断迭代。

第一代自动化测试技术，即最早的录制—回放技术，它们对系统和环境的依赖性太强。

第二代自动化测试技术，即脚本驱动化，用模块化和库实现。

第三代自动化测试技术，即数据驱动、关键字驱动，出现了自动化框架。

第四代自动化测试技术，即自动化平台，能够在独立的系统上完整地运行整个自动化过程，测试人员只需维护一些用例元素和脚本等。

目前接口 API、WebUI、AppUI、性能都能集成到 Autotestplat 平台上，如果只使用或展示其中一个，可以在相关前端页面导航栏的文件中或开发过程中屏蔽相应功能。虽然目前它不是最优的，但是很多自动化测试平台的思路是相通的，且目前此方案是可行、可实践、可落地、易上手的。我把本平台当前版本定义为 Autotestplat_V1.0，后续将会不断地快速更新、迭代优化、升级。自动化平台测试开发技术知识体系是我近期工作中接触到的内容，并在业余时间中进行整理和总结而来，通过和读者朋友分享，能起到抛砖引玉、给大家一点启迪思考作用或能直接应用于工作实战中，这就够了。

一起加油吧！