目 录 (四十九期 下)

Python 一步步实现微信小游戏“跳一跳”外挂

◆ 作者：lamecho

去年火了一阵子的微信小游戏“跳一跳”相信大家都玩过，要说操作也着实简单，依靠手指按压屏幕让棋子在各种方格间跳跃，所谓老少皆宜的游戏也不外乎这种，没有复杂的操作，而望着大家乐此不疲的一遍遍地刷记录的时候，那是真累啊！所以我第一个想法就是如何自动的让棋子跳呢？

有了上面的想法，说干就干，让棋子自动跳那就是模拟屏幕按压就行了，有了这个思路结合前段时间写完的“pyapp”自动化测试框架，知道可以利用 adb 命令 swipe 来实现屏幕按压（大家可以参考我之前写的文章《python 自动化测试应用-第 10 篇（APP 测试）之 adb 命令》），只要掌握好按压的时间就能模拟不同的按压力度，那么接下来这个按压时间该如何确定呢？自然我们是根据两个矩形物体间的距离了，隔得远我们就需要按压时间长，反之离得近，时间就短。所以之后的重点就是如何计算出这个距离呢？我们先看一下下面这张图。

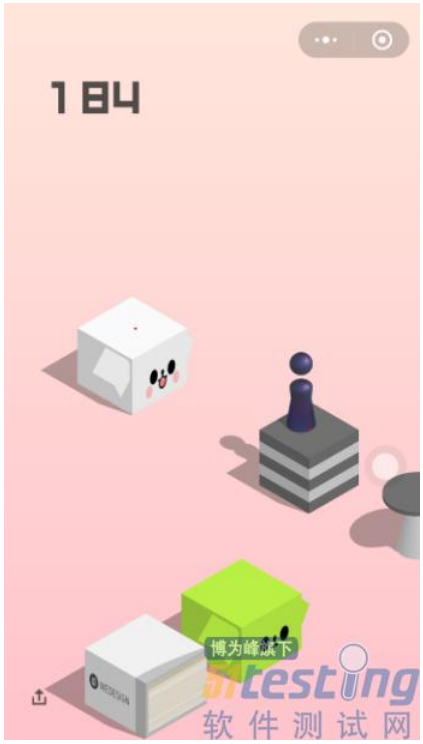


图 1

准确的说我们是要得到这两个红点之间的距离，上面这张截图就是我通过程序画出的。那么这两个红点所在位置是如何找出的呢？由于之前研究过一段时间的车牌识别，知道利用 opencv 可以分析图片，比如将彩色图片转化成灰度图，转化成灰度图后便于我们去识别图片中各种形状，这之间用到的功能函数还有 cv2.threshold（二值化），cv2.morphologyEx（图像的开，闭运算）最终我们可以将上图转化为这样。

图 2

通过以上的一些逻辑运算，得到的这张图可以清楚的将我们的棋子显示出来，接下来就可以自然的计算出棋子的坐标了。同样的方式我们看下图。

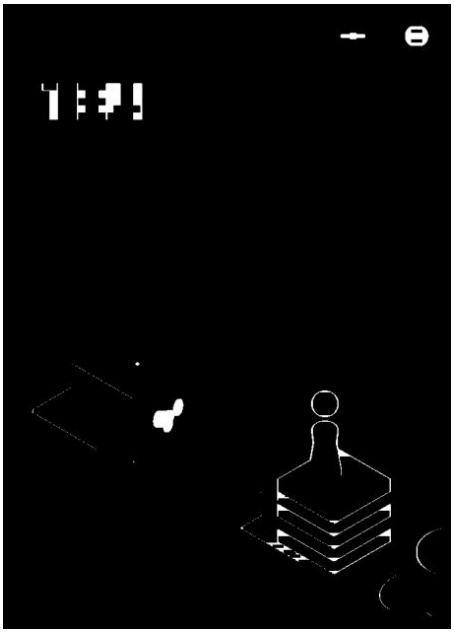


图 3

从这张图我们可以看到棋子和左边下一步要跳的矩形格子的边缘，正是利用这种关系我们就可以找到白色边缘，进一步分析出矩形格子的位置。这里着重说明一下下一步所跳矩形格子坐标位置我是如何分析的，同样我们还是按照轮询图片每一处坐标点的颜色，当然我们起始坐标不用从[0,0]开始，也就是可以屏蔽掉分数所在向上的位置，那么这样我们遇到的第一个坐标颜色是白色也就是为 0 的坐标点必然是在这个矩形范围内，那么找到这个点还没有完，因为我们要尽可能的找到矩形的中心点，因为目前我们只是找到了矩形边缘。向开篇第一张图的红点位置是如何确定到的呢？

这里我分享一下我的方法：既然我们找到了边缘点，之后我们从边缘点向上下左右四个方向继续分别获取颜色值（当然这里我们就要在原图获取颜色了），当某个方向遇到背景色时停止，这样我们就可以分析出我们这个边缘点是在矩形格子的哪个方位（是左边，还是下边，上边或右边），以这张图的情况举例，边缘点是在左边，那么自然我们就要让边缘点向右侧移动，这个偏移量自然就是我们向右找到背景色停止的那个位置的一半，因为我们是要定位中心点。这里再说明一点为什么要确定边缘点是在矩形格子的方位呢？因为跳一跳的格子它的颜色变化很多，且颜色不固定，所以我们通过 cv2 的函数运算得到图 3，矩形格子的边缘不一定是完全的边缘，所以我们要分析边缘与矩形实际的位置关系再确定中心点。

好了，知道了两个坐标点，棋子的和矩形格子的，那么他们之间的距离就可以得到

了。这个距离大家就可以理解成按压力度，因为他们是正比关系。比如我们得到的像素距离是 600，那么我们的 swipe 命令最后的参数时间就是乘以 2 即 1200 毫秒。

最后我将代码贴出来。

```
# -*-encoding:utf-8 -*-
from __future__ import division
import cv2
import time
import subprocess
import hashlib
import os,shutil
import numpy as np
from PIL import Image
import pytesseract
import math
from wg_main import *
loop_count=0
while 1:
    try:
        loop_count+=1
        shot()
        time.sleep(1)
        img=cv2.imread("d:\\cp\\screenshot.png")
        img=cv2.resize(img,(720,1280))
        hsvimg=hsv(img)
        _colorImage=img
        _colorImage1=img.copy()
        (w1,h1,_)=img.shape
        dis1=ch1(img,w1,h1)
        dis2=ch11(img,w1,h1,dis1[0])
        if dis2:
            if abs(dis1[0]-dis2[0])>100:
                (color1,_,_)=hsvimg[dis1[1],dis1[0]]
                (color2,_,_)=hsvimg[dis2[1],dis2[0]]
                if 110<color2<120 and (color1>125 or color1<110):
                    dis1=dis2
        (color1,_,_)=hsvimg[dis1[1],dis1[0]]
        if color1>125 or color1<110:
            dis3=ch111(img,w1,h1)
            dis1=dis3
        img=gry(img)#灰度转化
```

```
                img=sb(img)#Sobel 算子
                img=th2(img)#threshold 算法
                img=mo(img)#开闭运算

                pi,pj=mid_position(img,_colorImage,w1,h1,dis1)
                middle_pos=last_mid_pos(_colorImage1,pi,pj,h1)

cv2.rectangle(_colorImage,(middle_pos[0],middle_pos[1]),(middle_pos[0],middle_pos[1]),(0,0,255),4)
                cv2.rectangle(_colorImage,(dis1[0],dis1[1]),(dis1[0],dis1[1]),(0,0,255),4)
                dis=int(math.sqrt(abs(dis1[0]-middle_pos[0])**2+abs(dis1[1]-middle_pos[1])**2))
                dis=diss(dis)
                if dis<100:
                    dis=250
                jump(dis)
                time.sleep(2)
                os.remove("d:\\cp\\screenshot.png")
            except Exception,e:
                print e
```

最后，说明一点谨此篇文章请大家以自动化测试的角度去看待，游戏玩多了伤身，用外挂玩则是伤别人的身。

❖ 拓展学习

■微信小程序开发项目实战,入门速成：http://www.atstudy.com/course/265

测试女巫自动化生命进化之搞什么搞

◆作者：王平平

唉，不由得叹息一声，本来这一期我虽然心里有不舍，但是最终还是痛定思痛打算以“最近工作实在太忙”为理由放弃，但是当收到小编友好的提醒，心里那份不舍更为强烈，是的是的最近实在太忙了，女巫 2017 年 6 月份开始做前端开发了，这一个月又开始涉及后端开发，还是这个月又开始涉及 Robot framework，同时还要做自动化测试框架.....Hello 您有事吗?您还是测试女巫吗？呵呵，是的，我依然是那个测试女巫，但是从 2017 年到 2018 年女巫真的重生了，在 2017 年之前，女巫凭着不服输的劲头，带领着团队，使用 C++开发了有关硬件测试的自动化测试工具，使用 Python 有关软件测试的自动化测试工具:从开始的 anriod 手机自动化脚本，到路由器自动化.....绝大数都是自学，我挺自豪的，真的，直到.....2017 年初，我们所在组织被公司解散，重组.....是的，命运总是喜欢开玩笑，哈哈 2016 年底，女巫的老板，准确地说是“前老板”神秘地说：“你快要升职了”，额.....女巫还没来得及高兴，就在 2017 年初得到了组织重组的消息，组织都不在了，升个 Pi 啊~(此处伤害 10000 点)然后我就要面临生存问题，恰好新组织的老板对自动化开发非常感兴趣，然后经过一系列的考察，女巫写了无数的报告，然后我们团队生存下来了(当然很多原组织的人员有很多被劝退了，人生就是如此残酷.....)。

新组织的老板是个技术狂热分子，哈哈女巫跟他非常合拍，然后他就对我们来了个彻头彻尾的改造，高大上的改造，这个改造从技术到思想以及格局“海陆空全方位”的改造，到现在我反而有些庆幸原组织的解散，因为只有原组织解散，我们才能遇到这样的主管，只有这样的主管，才会让我们热血澎湃，人生苦短，我们必须要做一些事情，一些让我们热血澎湃的真的有价值的事情，不是吗？再感叹一下:人生就是这样让你猝不及防哈~

是的女巫现在做着高大上的自动化框架，一个基于前后端以及 IP 通讯流程的自动化测试框架，当然后续这个测试框架也会给大家介绍，但是测试女巫对我们这些年的“自

动化测试生命进程”很是感叹，尤其最近在微信或者 QQ 测试群组里，看到一些有着很多年工作经验的测试人，竟然提出:“为什么需要自动化测试，自动化测试还不如我手动测试快，学起来费劲，用起来麻烦~”也许所谓的自动化测试在他眼里就是沽名钓誉，没有任何价值~我很感叹，曾几何时我也遇到这样的质疑，当时我坚定了就是需要做自动化 的决心，并没有反驳他们，但是现在我突然萌生了，写一个系列吧，(希望能成为一个系列吧^_^)来与大家分享测试女巫的自动化测试生命进程，呵呵，现在在听着杰伦的“七里香”夜里 10:36 心情非常的开心。虽然女巫已经是老女巫了，但是我的心还是会热血澎湃，为了自己的进步而热血澎湃.....也许你还不明白我为什么要学前端开发后端开发，以及 robot framework 等，之前与一些“测试大牛们”交流我们目前在做什么时，得到的回应是: “哦，我觉得测试人员还是要做测试人员做的事情，你做的那些事，是开发人员的事”，额，我依然没有反驳，心里就想着“夏虫不与语冰”，但是对于我们的读者，我还是觉得要说明白，我们为什么做自动化测试? 真的不是“沽名钓誉”，测试人员也可以做得“高大上”，测试人员除了笨笨的手动测试，我们也可以设计各种高大上的自动化测试，所以请你耐心地看完我的这个系列文章吧。 (希望我也能坚持把这个系列写完)

最近看到一个文章“干掉你，与你无关”，现在方便面销量急剧下降，主要原因是外卖行业的崛起，吃方便面的人为了方便，点外卖更方便，所以方便面被干掉了，而方便面厂商也非常努力改善又味，尽量让它健康，例如推出非油炸等产品，但是有用吗?没有!这个是引子，测试人员真的需要抬眼看看外面的世界，不要等到被淘汰，而不知道为什么! 大约在 6 年前，我费劲千辛万苦从公司申请了外部培训的两次课程，这两次课程就是针对我们当时的产品: 一款 Andriod 手机实现自动化测试。大约公司一共花了 20000 元人民币左右，当时还觉得好贵，两次课程也只有两天的时间，但是现在反过来想想，这个钱花得太值了!从这个课程我们接触到了 Python 这个神奇的语言，一切的一切都是从这里开始，两次课程也就是两天的课程其它都是靠自己摸索，我也不怕丑，给大家看一下我们之前写的代码:所谓的“脚本的架构”，就是一个个的功能用文件夹包起来。

我们打开每个文件夹看一下，你就会发现，每个文件夹里面也就是一个 Python 脚本。

我们再进一步看这个脚本里面有什么，是个每个脚本里面都有一个 main 函数，每个脚本之间没有任何关系，一个脚本也就一个函数，我们的自动化测试就是需要测试人员手动一个函数一个函数的运行，说到这里，我真想捂脸哈哈，没关系，刚开始嘛。是的，对于这样的状况，我真的觉得被别人质疑：你是不是在沽名钓誉，真的是正常的，太扯了，一个脚本一个函数，想执行自动化要通过 main 函数一个个调函数来实现，真的还不如手动测试来得快，因为就是这个让人想捂脸的脚本，需要学 Python 的语法，还要学 python 的自带模块：sys 以及第三方模块 monkeyrunner，更不用说还要一个个学这些模块包含的函数用法了，最后这个函数执行完，只能产出一个非常简易的 pass 还是 fail 的测试 log.....学了这么多，出来这么个丑玩意，值得吗?没错，这个就是质疑“搞什么搞”的第一阶段阶段。

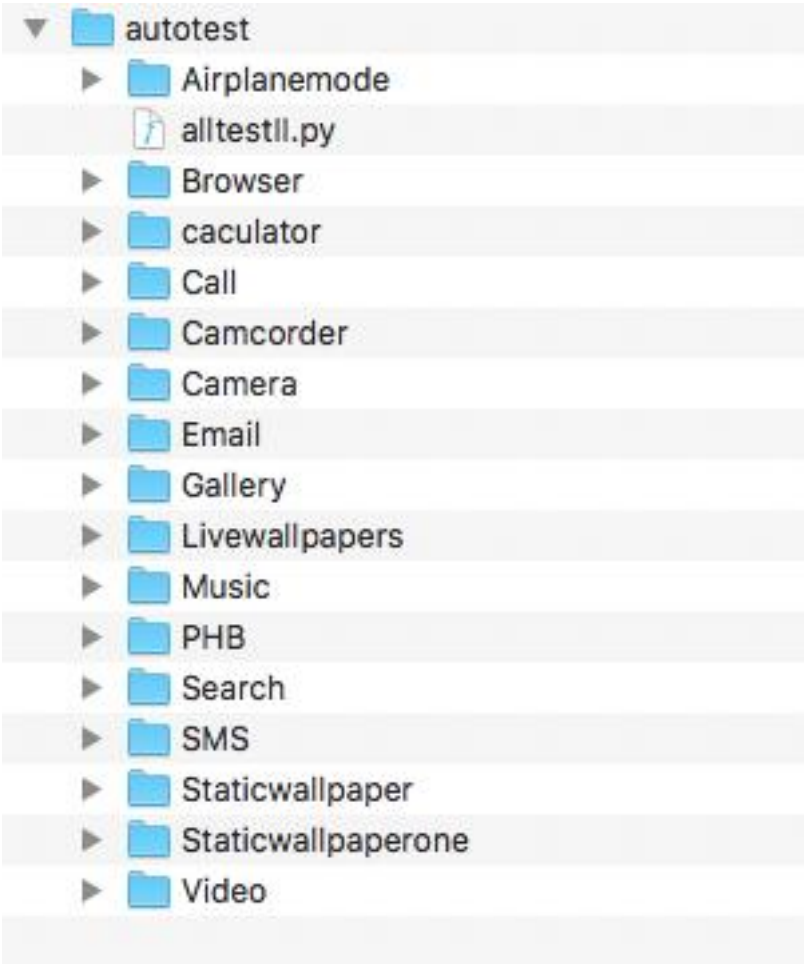
call_outgoing.py

```
#-*- coding: UTF-8 -*-
import sys
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice
def call(d):
    j=0
    k=0
    d.startActivity(action='android.intent.action.DIAL',uri='tel:10010')
#按Enter键拨打电话
    d.press('KEYCODE_ENTER','DOWN_AND_UP')
#测试需要，拨打电话持续15秒
    MonkeyRunner.sleep(15.0)
#挂断电话
    d.press('KEYCODE_ENDCALL','DOWN_AND_UP')

#拍照，将拍照结果赋值给formal
    formal.writeToFile('d:\\call_changed\\formal.png','png')
    for i in range(1,101):
        d.startActivity(component="com.android.contacts/.activities.DialtactsActivity")
#在dos上打印出“Start Activity”
        print "Start Activity"
        MonkeyRunner.sleep(2.0)
#进入拨打电话界面action，uri输入10010
        d.startActivity(action='android.intent.action.DIAL',uri='tel:10010')
        d.press('KEYCODE_ENTER','DOWN_AND_UP')
        MonkeyRunner.sleep(15.0)
        d.press('KEYCODE_ENDCALL','DOWN_AND_UP')
```

```
#拍照，将拍照结果赋值给formal
    formal.writeToFile('d:\\call_changed\\formal.png','png')
    for i in range(1,101):
        d.startActivity(component="com.android.contacts/.activities.DialtactsActivity")
#在dos上打印出“Start Activity”
        print "Start Activity"
        MonkeyRunner.sleep(2.0)
#进入拨打电话界面action，uri输入10010
        d.startActivity(action='android.intent.action.DIAL',uri='tel:10010')
        d.press('KEYCODE_ENTER','DOWN_AND_UP')
        MonkeyRunner.sleep(15.0)
        d.press('KEYCODE_ENDCALL','DOWN_AND_UP')
        test=d.takeSnapshot()
#照片名称为“test”+ str(i)，如“test1”、“test2”
        test.writeToFile('d:\\call_changed\\test'+str(i)+'.png','png')
        MonkeyRunner.sleep(1.0)
#test调用sameAS函数用来对照照片，并将结果赋值给result
        result=test.sameAs(formal,0.8)
        if result:
           j=j+1
           f= open('d:\\call_changed\\pass.log','r+')
           f.write('the actions outgoing_call had passed'+' " '+str(j)+' " '+'times')
        else:
           k=k+1
           f= open('d:\\call_changed\\fail.log','r+')
           f.write('the actions outgoing_call had failed'+' " '+str(k)+' " '+'times')
```

```
    else:
        pass
def main():
#建立host与device的连接
        device = MonkeyRunner.waitForConnection()
        if not device:
            print "Couldn't get connection"
            sys.exit()
        print "Found device"
#定义实际参数device
        call(device)

#执行主函数
if __name__ == '__main__':
    main()
```

第二阶段：对于一个项目，终于有了入又函数阶段(此时还是处于想捂脸的阶段^_*)是的，不能这样搞是吗?在被别人质疑你在搞什么搞的情况下，自己也在暗暗下决心，不能这样搞，所以第二阶段，我们终于有了入又函数:看我们的这个自动化脚本有了什么变化?没错，有了 alltest.py 脚本，哈哈这个就是解决了，我们的自动化框架需要用户一个个函数手动执行的尴尬，终于我们的自动化项目有了“班长”不再是一盘散沙，是吗?此处不应该有掌声吗?嗯，也许你觉得：你的测试报告能不能漂亮点?一个 txt 的 log 实在太丑，还有你的自动化工具连个界面都没有，还要我们来看一下:



alltest.py 中都有些什么，除了导入 monkeyrunner 这个模块外，我们也要导入自己定义的模块，这些模块就是一个个测试用例，是的我们就是通过“模块”的思路，凑出来了 alltest.py 这个“班长”。

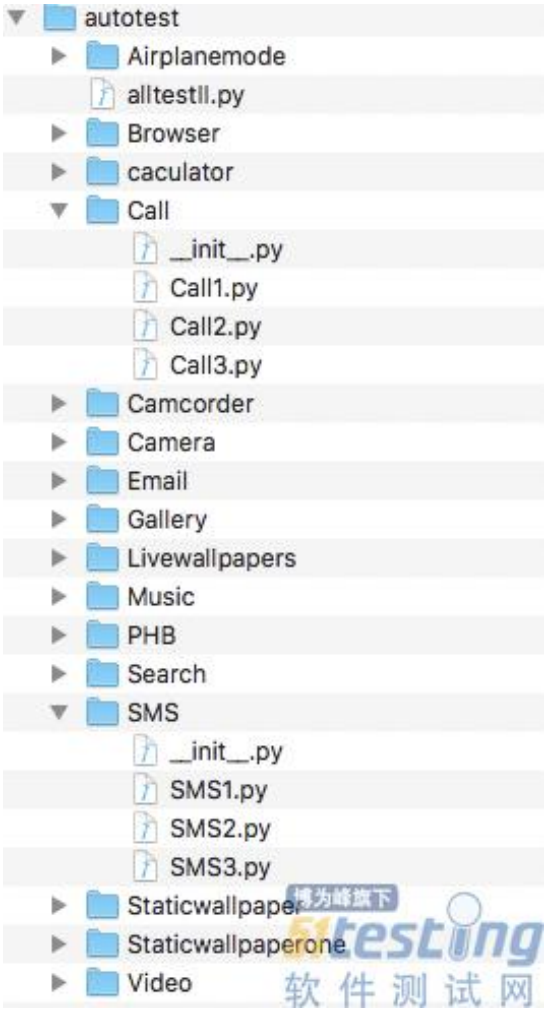
alltestll.py

```
import sys
import os
from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice,MonkeyImage

device=MonkeyRunner.waitForConnection()

try:
    from Call import Call1,Call2,Call3
    from SMS import SMS1,SMS2,SMS3
    from Email import Email1,Email2
    from PHB import PHB1,PHB2,PHB3
    from Browser import Browser1,Browser2
    from Gallery import Gallery1,Gallery2
    from Music import Music1,Music2,Music3,Music4
    from Video import Video1
    from Camera import Camera1
    from Camcorder import Camcorder1
    from Airplanemode import Airplanemode1,Airplanemode2
    from Livewallpapers import Livewallpaper1,Livewallpaper2,Livewallpaper3
    from Staticwallpaper import Staticwallpaper1,Staticwallpaper2,Staticwallpaper3,Staticwallpaper4,Staticwallpaper5,Staticwal
    from Caculator import Caculator1,Caculator2,Caculator3,Caculator4
    from Search import Search1,Search2,Search3
except ImportError:
    sys.path.append(os.path.dirname(__file__))
    try:
        from Call import Call1,Call2,Call3
        from SMS import SMS1,SMS2,SMS3
```

我们怎么实现 alltest.py 运行各个用户自定义模块呢?不是很美好的用法，就是根据程序执行顺序，顺序执行的，例如电话和短消息执行，首先使用一个字典类型的结构，给电话和短消息中需要用到的变量进行一一赋值，然后通过引入的模块调用模块中函数的方法执行，电话和短消息中的每个函数，同时将赋值的变量传到函数。

```
target_Call = {
    'package_name'      :'com.android.phone',
    'data1'              :'tel:10000',
    'data2'              :'tel:18913804378',
    'action'            :'android.intent.action.CALL_PRIVILEGED',
    'launch_activity'   :'.PrivilegedOutgoingCallBroadcaster'
}
Call1.Call1(device,target_Call)
Call2.Call2(device,target_Call)
Call3.Call3(device,target_Call)

target_SMS = {
    'package_name'      :'com.android.mms',
    'launch_activity1'   :'.ui.ConversationList',
    'launch_activity2'   :'.ui.ComposeMessageActivity'
}
SMS1.SMS1(device,target_SMS)
SMS2.SMS2(device,target_SMS)
SMS3.SMS3(device,target_SMS)
```

我们再来看看对于班里的每个成员我们如何安排的，哎呦都是 call,感觉功能多多了嘛，有了三个 python 脚本，call1，call2，call3。对于短消息，也是有三个脚本，SMS1，SMS2，SMS3。当然这个脚本的命名真的让人无语，能不能用一些让使用者一目了然的命名方式呢?对对对，怼得非常对，这个脚本的命名真的也非常非常重要~怼得非常对，这个脚本的命名真的也非常非常重要。



打开某一个脚本看一下有什么变化，哈，main 函数不见了，一个脚本也就是一个函数。

```
from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice,MonkeyImage

 #no1.main function:Call smoke test
    #call item1 test:dial out the number of 10010 and the system will connect automatic.
     ##############################################################################
passcontent='the Call item1 : The action of outgoing call and connecting automatic was passed!\n'
failcontent='the Call item1 : The action of outgoing call and connecting automatic was failed!\n'
filepath='e:\\autotest/testresult.log'

def Call1(device,target_Call):
    #generat formal picture
    #dial out the number and the system will connect automatic.
   testcase=__name__.encode('utf-8')
   runcomponent=target_Call['package_name']+'/'+target_Call['launch_activity']
   device.startActivity(component=runcomponent,data=target_Call['data1'],action=target_Call['action'])
   MonkeyRunner.sleep(10)
    #take a photo
   a=initmoduleII.take_formal_snapshot(device,testcase,'moduleIIformal')
   MonkeyRunner.sleep(2)
    #end the call
   device.press('KEYCODE_ENDCALL','DOWN_AND_UP')
   MonkeyRunner.sleep(3)
   device.press('KEYCODE_BACK','DOWN_AND_UP')
   MonkeyRunner.sleep(2)
```

```
  #generat test picture
  #dial out the number and the system will connect automatic.
device.startActivity(component=runcomponent,data=target_Call['data1'],action=target_Call['action'])
MonkeyRunner.sleep(10)
  #take a photo.
b=initmoduleII.take_test_snapshot(device,testcase,'moduleIItest')
MonkeyRunner.sleep(2)
 #invoke the function of comparing the formal picture with testing picture
initmoduleII.compare_formal_test(b,a,passcontent,failcontent,filepath)
MonkeyRunner.sleep(12)
  #end the call
device.press('KEYCODE_ENDCALL','DOWN_AND_UP')
MonkeyRunner.sleep(3)
device.press('KEYCODE_BACK','DOWN_AND_UP')
MonkeyRunner.sleep(2)
```

这次的主题是“搞什么搞”哈哈，所以这次就介绍到第二阶段，我还将持续总结，让大家清楚地看到我们是如何从“搞什么搞”的阶段迈向“高大上”的阶段，所以别人的意见真的就是进步的源泉，希望大家跟我一样，永远怀着一颗热情的心，不断挑战自己。

自动化测试投资回报率模型及其应用

◆作者：朱科伟

摘要：本文主要围绕自动化测试投资回报率 Return On Investment（ROI）进行分析讨论，设计了一套计算模型配以工具进行介绍说明，分析构成自动化测试成本的多种因素，特别对比手工测试和其对应的自动化测试，探讨在不同情况下这些因素对项目 ROI、成本、工期的影响和预测，以期帮助项目/部门/产品经理正确合理的规划自动化测试项目。

1、前言

1.1、背景

采用手工测试，还是自动化测试，在产品生命周期什么阶段开始自动化测试，各个阶段投入多少人力，如何构成最佳的人员（能力）配置，在实际的项目特别是大型软件产品项目实施过程中，由于涉及到的技术面较多，看起来比较简单的类似于 1+1 的问答题，却往往是困扰项目管理者的一道题目。拍脑袋的决策很多时候都被证明浪费了大量的人力物力，或者在产品质量出现问题时才考虑引入自动化测试加大测试频率，错过了抓虫（bug）的黄金时间点，最终影响软件产品的正常上线。自动化测试 ROI 的计算可以很好的在决策阶段起到重要的参考价值。

真实案例一:

某老牌 W 软件公司，上千人的研发团队，研发中心横跨三大洲，惯习了很多跨国企业决策慢的问题。新一代核心产品本意是为弥补其在 SaaS 上的缺失，研发过程中虽然采用敏捷开发模式，每天都有版本（build）集成发布，却一直主要依靠相对人数比例不高的手工测试团队进行软件测试/质量控制，无法最大程度尽早发现产品缺陷，工作量又让测试团队苦不堪言，最终产品缺陷 technical debt 越积越多，导致比预期的产品上线时间推迟两年以上。高层分析原因后，果断加大对自动化测试的人力物力投入，经过两年持续不断披星戴月的自动化进程推广和实施，产品质量总算步入正轨，在大刀阔斧调

整后的新日期发布。虽未错过行业红利，但给了新入的后来者时间和空间。过晚引入自动化测试给未来的竞争带来诸多不确定的因素，由垂直行业的领头羊变为展开 three kingdoms story。

➔ 选择适当的产品阶段开展自动化测试!

真实案例二:

某企业级 D 软件项目，在集成自动化冒烟测试（smoke test）中尝到了甜头，继续大力投入展开回归测试（regression test）的自动化，但由于产品业务复杂、组件众多、依赖外部服务，产品在当前集成阶段极不稳定，在数量呈几何级增加的 regression test 自动化脚本开发出来后，很多一周就遇到需求变更（feature change）或严重性能问题，自动化团队疲于修改脚本。加上测试团队与开发团队关于产品模块需求变更的信息不对称，长达半年的时间由测试脚本发现产品缺陷的目的变为由产品变更来完善测试脚本，可谓本末倒置。

➔ 可以尽早开展 smoke test automation，但在产品 feature ready 数量较少时大量开始自动化测试危险，存在不必要成本花费!

真实案例三:

某大型跨国软件 K 公司，当决定上马自动化测试项目时，高层希望尽可能的将现有所有手工全部转为自动化测试，强调自动化测试率（Automation Ratio），考虑到某些内部管理因素一刀切先以数字作为目标驱动。某些项目/产品经理从部门/个人业绩出发，对于不适合的测试类型和业务仍进行自动化，例如某些业务逻辑非常复杂、涉及场景交互很多的 scenario testing。且先不论花费不菲的人力财力，匆匆上马的决策导致这些脚本自从投入使用后通过率极低，由于涉及模块较多，任何一个模块的一点小问题都可能导致脚本不能通过。分析测试报告的错误 running failure 又带来看似无穷无尽的时间投入。这些开发人员费尽心血才孕育出来的测试脚本最后的命运自然是在数字运动后被束之高阁。

➔ 选择正确的业务和测试类型进行自动化!

1.2、计算 Automation ROI 的必要性

什么人需要/可以关注 Automation ROI?

对于预算经理，通过 (Automation) ROI 可对应不同相关类别的不同成本，得出成本预期，并划分到不同的月份进行预算预测;

对于产品/项目经理，通过 ROI 的计算和成本曲线在不同时期合理配置不同级别/要求的人员;

对于测试经理/组长，通过项目的实际情况计算 ROI，反向校验项目安排（自动化测试运行周期等）的合理性，指出问题提出意见并进行调整;

对于有思想的自动化测试开发人员，我们做的是什么测试类型的自动化工作？在什么阶段进行？自动化测试的 Scope 是什么？我还可以做什么？除了代码外，再多问几个为什么，是职业生涯进阶的必经之路。

1.3、影响自动化测试 ROI 计算的因素

1.3.1 自动化测试 ROI 基本公式介绍

自动化测试 ROI 的基本公式为

ROIAutomation(in time t) = (Savings from Automation) / (Costs of Automation)

= Δ(Savings from Automation) / Δ(Costs of Automation)

= (Costs of manual – Costs of Automation) / Costs of Automation *100%

若不考量测试的覆盖率，大部分项目实施自动化测试主要目的是为了追求自动化测试的成本优势，但在某些条件下 ROI 值可能为负数。

在基本公式中未列出的影响自动化测试 ROI 的诸多因素，如下。

1.3.2 自动化测试主要成本项

- **自动化环境搭建（一次性成本）**

包括 CI 工具（如 Jenkins, TeamCity 等）配置、测试管理工具（如 HP ALM）配置关联、代码仓库（code repository）部署等。

- **自动化测试相关软硬件（一次性成本）**

企业级的商用自动化测试工具（如 QTP），大多 license 价格不菲，同时需要考虑与其相关的技术支持费用。因此即便是很多财大气粗的大型软件公司，也越来越多采用开源的自动化工具（如 Selenium）并搭建自己的自动化测试框架平台。

- **自动化测试框架的开发/采购（一次性成本）**

除个别小型项目外，目前商用自动化测试考虑模块重用、测试管理及其效率、统计报告和系统环境集成，大多会采用或搭建一套自动化测试框架（开源或商用）。本文不详述自动化测试框架的类别，但根据其框架 feature 复杂程度都可以将其成本换算为对应的人月成本。此成本可看作为一次性成本。

以 web 自动化测试框架为例，项目需求要求支持 UI 和 API 测试，移动端（mobile）测试，考虑断言（Assertion）、Selenium-PageFactory&WebDriver&Grid、Operations、TestNG-Wrapper、测试数据模型驱动、测试数据回滚、HTML 报告及报告分析、自定义 Selenium Actions、自定义元素 Grid&日历、自定义 utils-DateParser 和 DateGenerator、增强日志功能、集成持续集成（CI）工具、录屏工具和测试管理工具（例如 ALM/TestRail/TestLink）、网页元素设计/控制、数据库支持等。

根据支持的功能数量和复杂度我们将其分别定义为基础框架、标准框架和高级框架，并赋予不同的项目人月成本/开发成本。

例如，对于 Assertion 模块，基于 TestNG 的 assert 进行加强，当发生断言 failure 时脚本仍可以继续执行。

在基础框架中封装 TestNG 的 assert，在标准框架中支持软断言（soft assertion），在高级框架中支持 soft and hard assertion。

- **自动化测试框架的维护（持续性成本）**

框架开发完成后并非一劳永逸，跟变更导致的测试脚本的维护类似，框架也会因为设计或编码导致的 defect、项目需求变更/情况变化、底层驱动变化等造成维护的工作量。此成本为持续性不间断发生成本，可分摊到每个迭代（iteration）中。

- **自动化测试脚本开发（主要成本 + 一次性成本）**

为方便统计计算，这里我们将自动化测试脚本开发和维护的成本分拆为两个类目。

自动化测试脚本开发是自动化测试中主要成本，ROI 计算中可在项目第一个迭代中即计入所有成本，或者（实际项目情况）根据预期项目周期分摊到每个迭代中。后续计算中暂采用第一种计算方法；若采用第二种计算方法，一般不影响项目整体的 ROI，手工测试成本将更早突破自动化测试成本（若平均每个迭代自动化测试成本<手工测试成本）。

➢ **自动化测试脚本维护（持续性成本）**

包括自动化测试执行、错误分析、复测（含手工和自动化复测校验）、缺陷提交、测试报告、脚本本身的 bug 或重构引起的脚本优化、框架/产品变更（含 UI 可见变更和不可见变更）引入的脚本维护等。

➢ **自动化测试执行（可忽略成本）**

对于大多数的企业级自动化测试执行而言，由于在框架和环境搭建阶段已考虑并行运行（例如采用 Selenium Grid），数据初始化也在脚本开发阶段实现，在实际的测试执行阶段，所需要的大多只是软硬件（例如虚拟机 node）投入。本文主要集中对比手工测试和自动化测试的投入产出，对两者均会发生的类似软硬件投入暂不计入。投入更多的 node，会进一步加快/节省运行时间。常规的自动化测试运行都是晚上自动运行，无需人工干预，第二天早上得到测试结果进行分析，因此可以认为基本是无需运行时间，无缝衔接，属于暂可忽略成本。

需要强调的是，此项虽在计算 ROI 时可记做隐形成本，却是自动化测试中非常重要的环节。

1.3.3 手工测试主要成本项

➢ **测试执行（主要成本）**

包括测试环境准备/搭建，测试数据准备。手工测试的执行主要依赖人力，是必需人力成本。

2、模型介绍 Automation ROI Model Introduction

2.1、ROI 计算及计算项分解

Automation test ROI = (Costs of manual – Costs of Automation) / Costs of Automation *100%

2.1.1 手工测试成本

Costs of manual = 手工测试执行时间

2.1.2 自动化测试成本

Costs of Automation = 框架开发 + 框架维护 + 环境部署 + 脚本开发 + 脚本维护 + 测试运行

2.1.3 影响成本/ROI 的关键因素

前面小节所述的成本项指的是测试的工作项（work items）的成本，每一个工作项的成本都会受到各种因素的影响。

- 测试用例的数量

根据用例用途/复杂程度分 smoke test, regression test 和 scenario test。数量越多，手工执行和自动化脚本开发的时间所需越长。

- 手工/自动化测试的运行周期

运行一个 full cycle 的频率，即测试完一轮/所有测试用例的时间频率。

大部分软件公司的 build 都是 daily build，所以通常自动化测试每天一轮。当然你十天运行一次也行，不过每天运行也不花费额外的成本，一些简单的配置而已，何乐而不为呢？

对于手工测试，通常 smoke test 每天会做一轮，regression test 有的项目/公司会每天一个 full cycle，有的则会一个 sprint（两周/三周等）一个 full cycle。

- 自动化框架功能复杂程度
- 自动化脚本开发速度
- 自动化脚本维护量

本文采用每 1000 脚本所需要维护的人力（HC）/人天（man-day）数来进行衡量。

2.1.4 影响成本/ROI 的其他因素

- 产品稳定性

产品稳定性会影响自动化脚本的运行效率。

- 产品阶段

Requirement	Design	Implementation	Verification	Maintenance

产品的不同阶段决定测试框架/脚本的修改率（需求变更/重构），即框架/脚本的维护成本。

2.1.5 前提条件 Prerequisites & Assumptions

下面汇总罗列一些关于自动化测试的常识/共识和约定，作为模型计算中的前提。在后续章节相关内容中会涉及，方便读者回溯查看。

- Environment deployment & Automation framework/script development are one-time efforts;
- Automation framework/script maintenance is a continuous effort;
- For easier calculation, count all costs of Automation framework/script development into 1st iteration;
- 为方便绘制 ROI 时间曲线，将每个迭代累计的手工测试和自动化测试成本进行比较。由于通常自动化测试频率 FA≤手工测试频率 FM，所以将手工测试的迭代作为基准迭代的时间长度进行计算。同时，纳入同一个时间长度的迭代后，可能存在每个迭代运行一个/轮手工测试 full cycle 的同时运行了多轮自动化测试;
- 同上，通常 smoke test 的运行频率 FMS ≤FMR (regression test)，那么可能存在每个迭代运行一个手工 regression test 的 full cycle 的同时也运行了多轮 smoke test;
- 自动化测试相关软硬件成本暂不纳入计算;
- 脚本开发工具软件大多采用同开发的 IDE，所以 IDE 项成本可暂不考虑;
- 框架维护的成本根据项目对框架的需求等不同而不同，可根据经验和项目不同取不同值。在本文计算模型中不做详细分解;
- The time of Automation execution only replies on hardware/client cost and isn't counted into the calculation;
- 常规的手工功能测试的测试环境准备/搭建一般相对较简单，模型中列项但暂不计算;
- 测试运行频率以工作日计算;
- 一年记 250 工作日。

2.2、模型及工具 UI

为更直观的方便读者理解，同时用 Excel 设计了一个小工具辅以介绍。

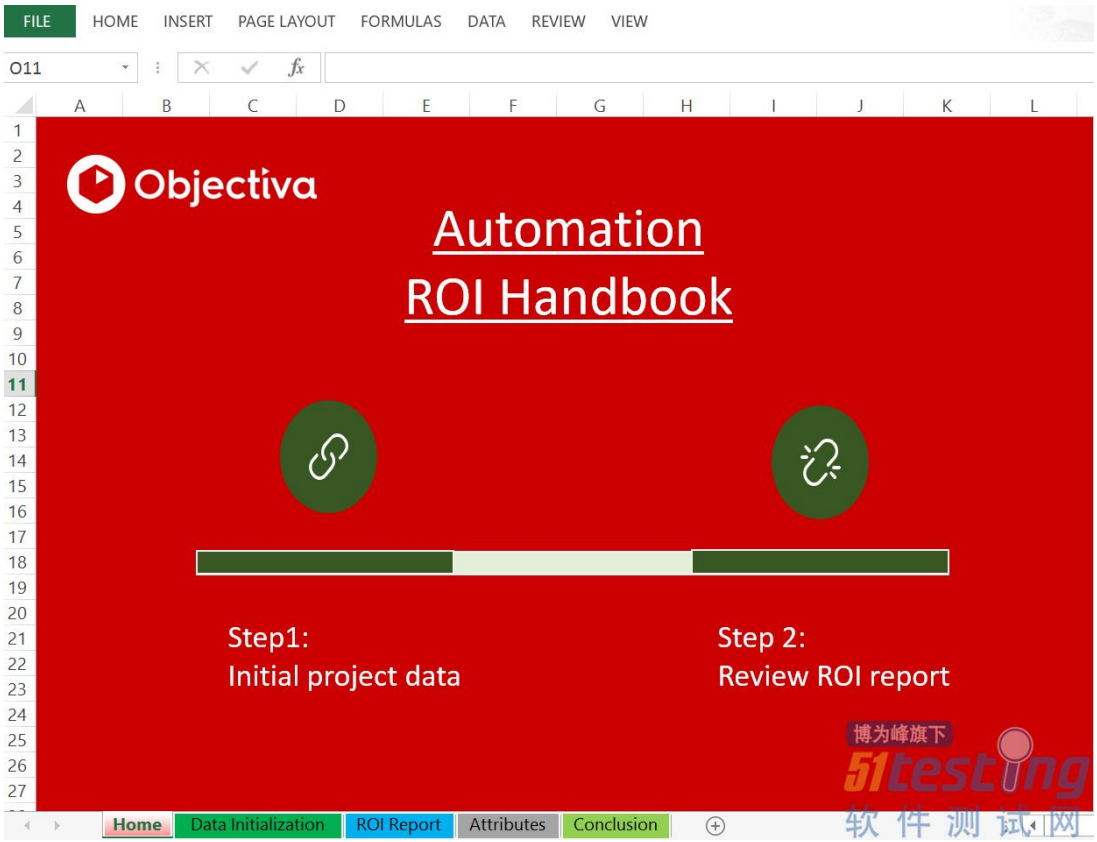


图 2.1 工具首页

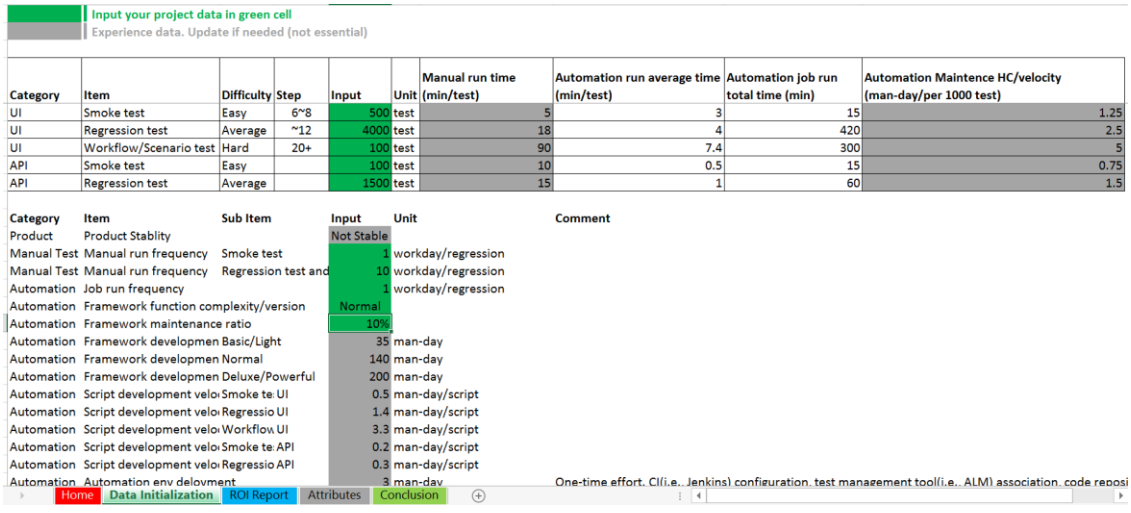
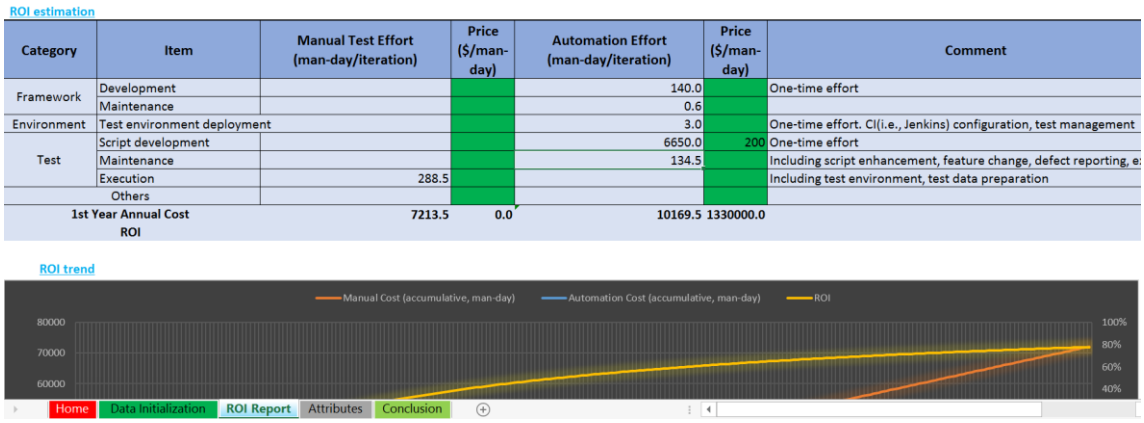


图 2.2 数据录入

‘Data Initialization’ 包含有 ROI 计算的所有数据参数，绿色单元格需要填入项目的数据，灰色单元格为经验数据（仅需要时进行修改）。



ROI estimation

Category	Item	Manual Test Effort (man-day/iteration)	Price ($/man-day)	Automation Effort (man-day/iteration)	Price ($/man-day)	Comment
Framework	Development			140.0		One-time effort
	Maintenance			0.6		
Environment	Test environment deployment			3.0		One-time effort. CI(i.e., Jenkins) configuration, test management
Test	Script development			6650.0	200	One-time effort
	Maintenance			134.5		Including script enhancement, feature change, defect reporting, e
	Execution	288.5				Including test environment, test data preparation
Others						
1st Year Annual Cost		7213.5	0.0	10169.5	1330000.0	
ROI						

图 2.3 ROI 报告

录入项目数据后，可在‘ROI Report’中查看自动生成的 ROI 数据报告包括图表。自动生成的曲线趋势为根据时间/迭代累加的成本，蓝线为手工测试成本，橙线为自动化测试成本。同时可录入实际的人月单价得出具体成本金额。

Item	Value1	Value2	Value3	Value4	Value5	Value6	Value7	Value8	Value9
Product Phase	Requirem	Design	Implemen	Verificatio	Maintenance				
Product Stablity			Not Stable	Stable	Very stable				
Service Type	UI	API							
Test Type	Smoke tes	Regressior	Workflow/Scenario test						
Framework complexity	Basic/Ligh	Normal	Deluxe/Powerful						
Company Size	Small	Middle	Big						
Priority	Low/P2	Normal/P:	High/P0						
Difficulty	Easy	Average	Hard						

图 2.4 参数设置

‘Attributes’为前面 sheet 的参数设置，可在此定义/修改为自己项目/公司的名称。

设计原则

简单、易用!

仅需两步:

1.录入项目的数据

2.得出 ROI 报告

有经验的用户可以修改经验数据/名称以更好的匹配实际的项目情况。

关注影响 ROI 的核心因素，抓住重点帮助分析。

2.3、主要输入数据及示例

Major Input Data

Item	Sample Value	Description
(UI) Smoke test (UI) Regression test	500 tests	TC amount of each test type
(Regression test) Manual run frequency	10	Per 10 days to run 1 full regression test via manual covered all TCs
Job run frequency	1	Per 1 day to run 1 full regression test via Automation
Frameworkfunction complexity/version	Basic/Standard/ Premium	The Automation framework complexity costs different development effort
Framework maintenance ratio	10%	The maturity of the framework and changes of the product cause framework maintenance effort

3、实例应用分析

下面就几个示例项目做分析演示。

Project Sample/Conclusion

3.1、项目 1

TCs below have been automated 下述列出了不同类别测试用例的数量。

- (UI) 400 smoke tests, 4000 regression tests and 100 workflow tests
- (API) 100 smoke tests, 1500 regression tests
- (Manual) run 1 full regression smoke test per day, run 1 full regression test per 2 weeks
- (Automation) run 1 full regression (smoke + regression) per day

这是一个非常典型的手工/自动化测试的运行周期，项目采用敏捷开发模式，2 周一个迭代，所以常规的回归测试（手工）每 2 周执行一轮。同时，产品的测试用例数量属于中上级别。

Product is unstable causing frequent feature change/maintenance cost increasing.

From week88 (near 2 years), the cost on Automation test is under manual test.

Automation test can monitor the product quality more frequently 10:1.

ROI estimation

Category	Item	Manual Test Effort (man-day/iteration)	Price ($/man-day)	Automation Effort (man-day/iteration)
Framework	Development			140.0
	Maintenance			0.6
Environment	Test environment deployment			3.0
Test	Script development			6650.0
	Maintenance			134.5
	Execution	288.5		
Others				
1st Year Annual Cost		7213.5	0.0	10169.5

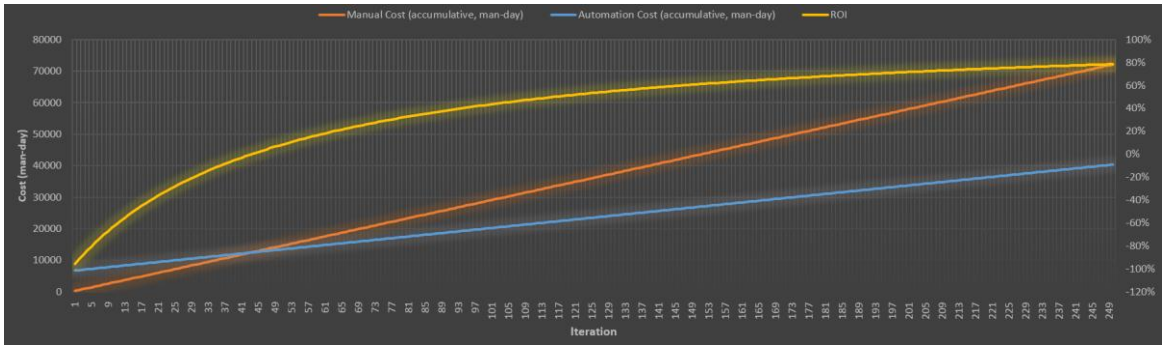


图 3.1 项目 1 成本/ROI 趋势图

从成本上，第一年末自动化测试成本仍相当于手工测试成本的 150%，接近两年成本差不多持平，再往后自动化测试成本低于手工测试成本。

从 ROI，初期由于自动化一次性成本原因为负值，40%前随时间增幅明显，达到 78%后增幅明显放缓。

从产品质量检查周期上，自动化测试对产品质量的监控/覆盖频率远高于手工测试(10:1)。

结论： 适合于大型长期的软件产品，有较高的脚本维护成本，但自动化测试框架的前期投入经过时间会摊薄许多，也可以成为其他项目的有益 legacy。

3.2、项目 2

(Manual & Automation) run 1 full regression test per day

这个项目将手工和自动化项目的测试周期拉平到同一个量级（1 full-cycle/day）进行比较。在 IT 软件业界中较重视产品质量、用户群体量较大、SaaS 平台产品为代表的微软、IBM 等公司的项目可以作为参照。

同示例项目 1 相比，采用了同样的标准测试框架和测试用例数量，主要区别在于手工测试的频率。

From 3rd month, the cost on Automation test is under manual test and will be greatly lower than Automation in long term.

ROI estimation

Category	Item	Manual Test Effort (man-day/iteration)	Price ($/man-day)	Automation Effort (man-day/iteration)
Framework	Development			140.0
	Maintenance			0.6
Environment	Test environment deployment			3.0
Test	Script development			6650.0
	Maintenance			134.5
	Execution	288.5		
Others				
1st Year Annual Cost		7213.5	0.0	10169.5

ROI trend

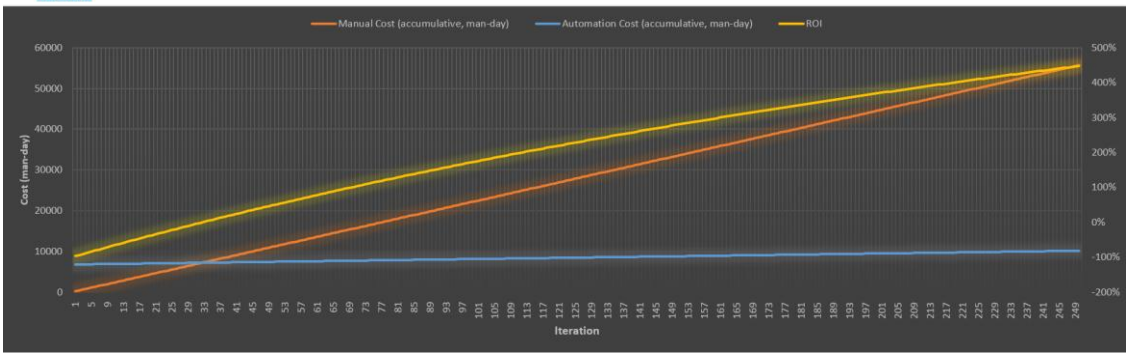


图 3.2 项目 2 ROI/成本趋势图

当手工测试和自动化测试频率拉平到同一个量级进行比较后，自动化测试的成本优势进一步凸显。在这个项目示例中仅第 3 个月（1 天/迭代）后自动化测试的成本便远远低于手工测试。第一年末自动化测试成本不到手工测试成本的 20%。

结论:

适合

- 迭代周期短（天为单位，个位数）
- 对质量更敏感
- 用户体量大或用户群体影响力强
- 更新频繁
- 轻量级软件（测试用例/脚本量级不大）

- 无需客户端部署，用户主要购买应用软件服务

3.3、项目 3

TCs below have been automated 下述列出了不同类别测试用例的数量。

(UI) 100 smoke tests, 500 regression tests and 10 workflow tests

(API) 50 smoke tests, 100 regression tests

(Manual) run 1 full regression smoke test per day, run 1 full regression test per 2 weeks

(Automation) run 1 full regression (smoke + regression) per day

同样是一个非常典型的手工/自动化测试的运行周期，项目采用敏捷开发模式，2 周一个迭代，所以常规的回归测试（手工）每 2 周执行一轮。

同示例项目 1 相比，采用了同样的手工/自动化测试频率，主要区别在于大幅降低测试用例的数量同时简化测试框架为轻量级框架。

From week35 (near 1 and half a year), the cost on Automation test is under manual test. Automation test can monitor the product quality more frequently 10:1.

ROI estimation

Category	Item	Manual Test Effort (man-day/iteration)	Price ($/man-day)	Automation Effort (man-day/iteration)
Framework	Development			140.0
	Maintenance			0.6
Environment	Test environment deployment			3.0
Test	Script development			823.0
	Maintenance			16.1
	Execution	44.6		
Others				
1st Year Annual Cost		**1114.6**	**0.0**	**1383.1**

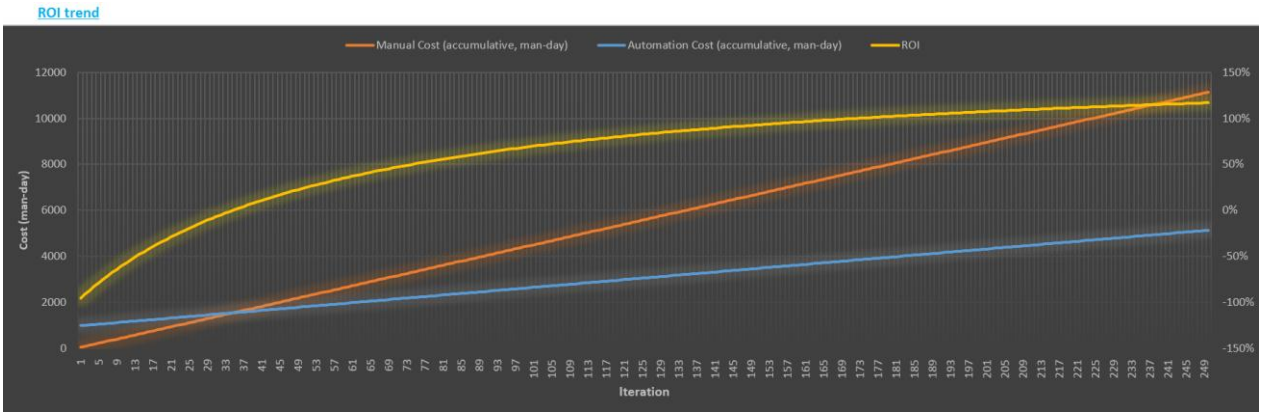


图 3.3 项目 3 成本/ROI 趋势图

当测试用例数量降低到轻量级别后，由于测试框架开发的成本占比相对依然很小，同项目 1 相比，ROI 趋于 0%/成本持平的时间提前了半年，但仍需较长的时间，主要还是由测试频率把控成本持平时间。

结论:

- 适合大型长期的软件产品
- 轻量级软件谨慎使用自动化测试
- 开展自动化测试应投放在能快速产出成效的 smoke/sanity test 上，需谨慎开展大规模的 regression test Automation
- 测试框架开发成本分摊到每个迭代后，对 ROI 影响不大，特别对于轻量级测试框架

4、其他

4.1、自动化测试 ROI 对项目的参考意义

当 ROI < 0%时，从数值上看很简单，自动化是亏本的，往往处于自动化测试的建设初期，需要根据 ROI 曲线斜率趋势去判断成本持平的时间长度。

当 ROI 曲线与手工测试成本交集前，处于稀释自动化测试初期一次性成本时期，ROI 曲线会呈现较高的斜率，此时已经呈现较明显的自动化测试成本优势。

当 ROI 曲线与手工测试成本交集后，随着时间推移相对增幅减小，甚至趋于零界点。这个时期增加自动化测试仅能扩大测试范围，对 ROI 贡献不大。

长期看对于大多数软件项目，自动化测试成本均会低于手工测试成本，所不同的是

由于各种因素引起的时间长短可能相差甚远。在考虑项目成本上，时间成本也是一个非常重要的因素，甚至会决定一个产品的成败，优秀的产品经理往往折中选择一个平衡。

4.2、依据 ROI 模型对自动化测试项目开始前的建议

为最大程度获得较高 ROI，降低自动化测试成本的 best practice:

4.2.1 自动化类型/阶段 scope

- 尽量自动化 sanity/smoke test，并将其集成到 CI 中，其对应的 pass rate 设为 build 发布的重要指标之一;
- 若回归测试的频率较频繁（如低于 1 周/轮），即每个 sprint 小于一周，可适度进行自动化;
- 若回归测试的频率中等（如高于 1 周/轮），仅考虑将其中重要 feature 进行自动化;
- 场景（scenario/workflow/E2E）测试的自动化一般用于产品较稳定阶段（UAT/α/β）。

4.2.2 功能点

- 测试频繁模块
- 手工测试容易出错的模块，例如计算公式较长或参数众多雷同的税收计算

4.2.3 人员投入时间点

不匹配/适合人员的安排，可能极大程度影响项目 ROI。

框架开发阶段:

- （初期）架构师、熟悉整体自动化框架的高级开发工程师
- （中后期）熟悉自动化脚本或某些框架功能的开发工程师

测试脚本开发阶段:

- （初期）熟悉自动化脚本设计的高级自动化开发工程师（Sr. Automation dev/QA）
- （中后期）熟悉自动化脚本开发的自动化工程师（Automation dev/QA）框架维

护阶段：架构师、熟悉整体自动化框架的高级开发工程师

4.2.4 人员能力

很多项目在进行自动化的实施上，都会考虑的是现有人员的复用。从成本/HR 角度无可厚非，但从自动化项目的成功实施并确保最终获得较好 ROI 的角度，需要慎重。除非是 BDD 非常成熟的项目（极少），或者自动化框架中对于 BDD 集成的解决方案比较完善，大多数手工测试工程师无法复用在自动化测试项目中，两者的要求也不尽相同甚至有时迥异。

除去对于从事测试行业的共性要求（良好沟通、记忆力、细心、耐心、怀疑精神等），自动化测试更多需要具有良好代码功底的工程师，手工测试更多需要具有良好发散性思维和产品测试意识的工程师。更多我们可以将实施自动化测试的人员看作是从事 QA/QE 工作的开发工程师，实施手工测试的人员看作是不写代码的 QA 工程师。

请让专业的人做专业的事!

4.2.5 测试框架

在前述章节中对于测试框架开发未大施笔墨，但实际项目实施过程中仍需考虑如下内容，否则可能造成 ROI 的极大出入。

4.2.5.1 开发语言

市面流行的开源测试框架大多支持多语言，或商用工具支持脚本多语言，一般应当采用与项目一致的语言，这样可以最大程度直接利用现有项目的 API、减少语言差异等引起的额外开销。

框架设计阶段在选择编程语言时，需考虑跟本自动化项目最关注的项目目标相关的要素，例如自动化工程师语言类别、框架核心插件/模块/第三方集成系统所采用的语言等等。同时，采用常用/流行的编程语言便于框架/脚本复用于其他项目。

4.2.5.2 框架选择

测试框架的选择是一个很大的题目，需要考虑核心模块、架构、开源/商用范围、价格、拓展性、三方集成系统、支持平台、编程语言等等。

例如若支持集成测试管理工具，那么自动化测试结果可同步到测试用例中，方便比较手工测试结果和自动化测试结果，方便比较历史测试记录。较好集成测试管理工具的

自动化框架，会降低脚本维护时间，间接提高 ROI。

附录 1 以 Selenium V1 和 V2 为实例，将录制回放类与采用高级编程语言作为测试脚本的自动化工具/技术进行了比较。

关于自动化测试框架的选择，可参考专述自动化测试框架的相关技术文章，本文不在详述。

4.2.6 持续集成（CI）与自动化测试

需要澄清一个自动化测试中常见的概念误区

Sanity / smoke test 的自动化测试 ≠ CI

如果 sanity / smoke test 的自动化未用于 CI，仅作为一个常规测试项，或未在版本发布中成为核心指标之一，那么其自动化并未发挥出最大功用。

要实现自动化测试在 CI 中的集成，除准备自动化测试脚本外，需要部署对应的 CI 环境（CI 工具配置、job 配置等）。

CI 与否并不影响自动化的 ROI，但却是体现项目自动化测试程度/层次的重要标准。

4.3、模型的不足/延伸/完善

前述章节内容提到，本模型建立初衷是为根据成本核心因素得出 ROI 快速建立自动化测试项目立项可行性分析，更多是定性分析，套用到真实项目中后还需要考虑测试框架和脚本开发的时间和成本分摊。可在本模型的基础上丰富/延伸考虑如下要素，形成更贴近真实项目的时间曲线图。

A.测试基础框架的开发需要独占项目前期阶段时间;

B.脚本开发往往也是随项目周期不断开发，并非在项目初期全部就位;

C.手工测试和自动化测试在大多数项目中同时存在，相辅相成，在进行自动化测试的同时开展手工测试。

典型的自动化测试测试项目阶段:

测试框架设计 -> 测试框架开发 – > 测试脚本开发 -> 测试执行/框架及脚本维护

5、结束语

读者可以参考前述章节的 ROI 数据模型展开不同项目类型的 ROI 计算测试，抛砖引玉探讨在更多不同要素下 ROI 与项目的关联关系。

同时，不要纯粹为了追求技术而做自动化，不要纯粹为了自动化而自动化，手工本身有自动化不具备的优势，极长一段时间内 AI 还是替代不了发散性卓越的人类思维，更多的用于手工测试很难覆盖的领域，重复性较强的地方或者频率较高的测试迭代周期。自动化测试本身是为了产品服务，综合考虑产品的质量、上线周期，这样才能把自动化测试最大的优势发挥到极致，和手工测试完美配合。

附录 Appendix

录制回放类与采用高级编程语言作为测试脚本的自动化工具/技术的比较。

以 Selenium 为例，

	Selenium v1	Selenium v2
Pre-Configurations	Complex with manual setup	Easy without manual setup
Execution	Require manual setup	Can be executed by single click
Browser support	Depend on JavaScript 由于其是由 JavaScript 实现的，所以只要是支持 JavaScript 的浏览器都可以很好的支持 它	Support browsers on mobile as well WebDriver 解决了 Selenium 存在的缺点（比如，绕过 JS 沙箱）
Reuse	Barely reuse	High reusability
Complex elements	Not support	Support coding
Limitations	Not support actions, such as drag and move elements Not support mouse or keyboard Not support popup dialog Not support basic browser operations CAN'T integrate with Test Framework	
CI	Complex when integrating with CI	

Python 下操作 MySQL 数据库

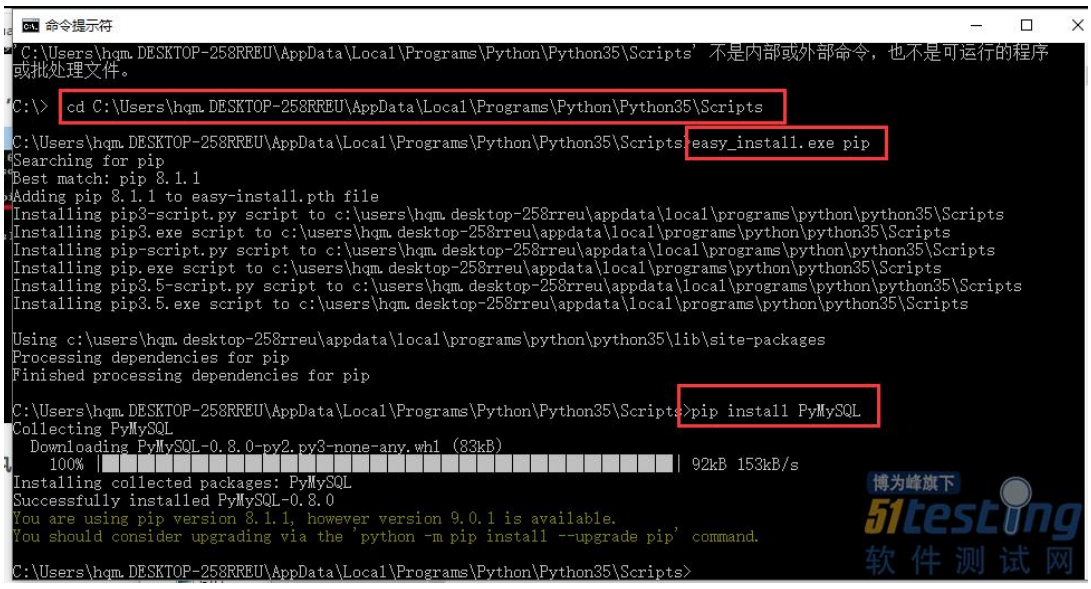
◆ 作者：呆小秋

最近刚好有用到 Python 下操作 Mysql 数据库，趁这个时候总结一下。因为我们测试过程，有可能要通过数据库保存一些我们的测试数据之类的。

要在 Python 下操作 Mysql 数据库，要先确保有安装了 PyMysql 了（电脑上安装 Mysql 的在这里就不阐述，但需要注意的是要记住一开始安装的数据库 root 的密码，另外一个就是记住数据库的端口（默认是 3306））

一、安装 PyMysql:

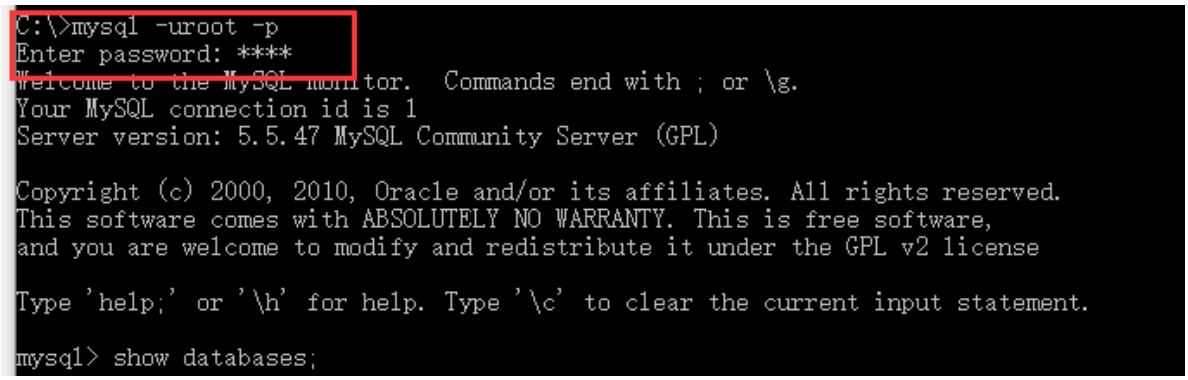
先进入到 Python 安装目录下的 Scripts 路径下，安装 pip，这个工具，然后通过 pip，安装 PyMysql，具体如下图所示。安装成功会提示 successfully installed。



二、cmd 命令模式下数据库的基本操作（语法在 Python 中也要用到）

1、启动 mysql 数据库(进入服务前要先手动启动 MySQL 服务或使用 net start

mysql 命令启动)

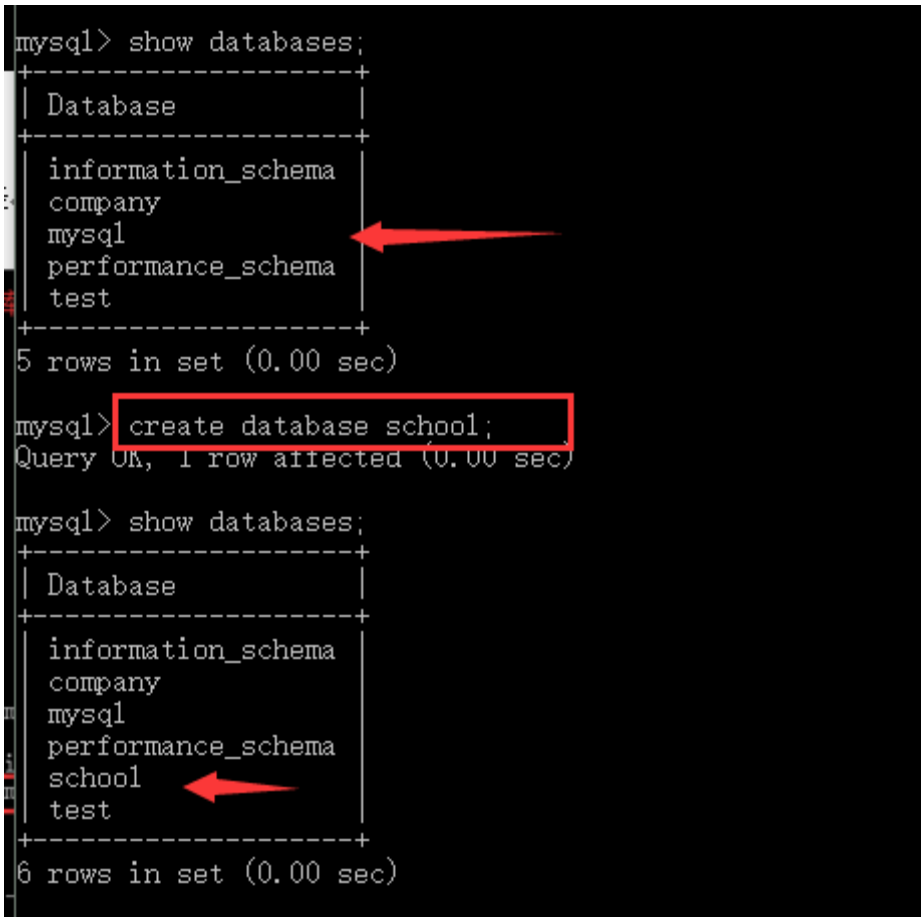


有出现如上图的一堆版本之类的，说明已经进入 MySQL 了。

注意如果你的 mysql 没有安装在 C 盘下，你需要先使用 DOS 命令进入 mysql 的安装目录下的 bin 目录中。另外，这边的 root 密码就是安装 MySQL 的时候设置的 root 密码。

2、创建数据库，查看数据库和创建表

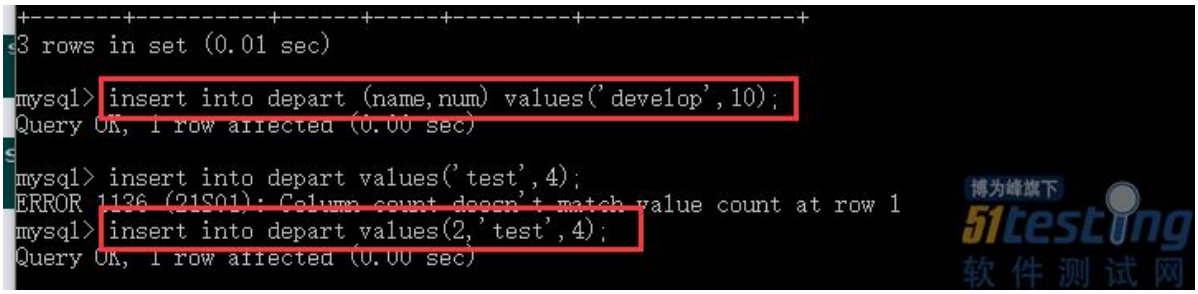
1）创建数据库:



2）查看数据库和创建表

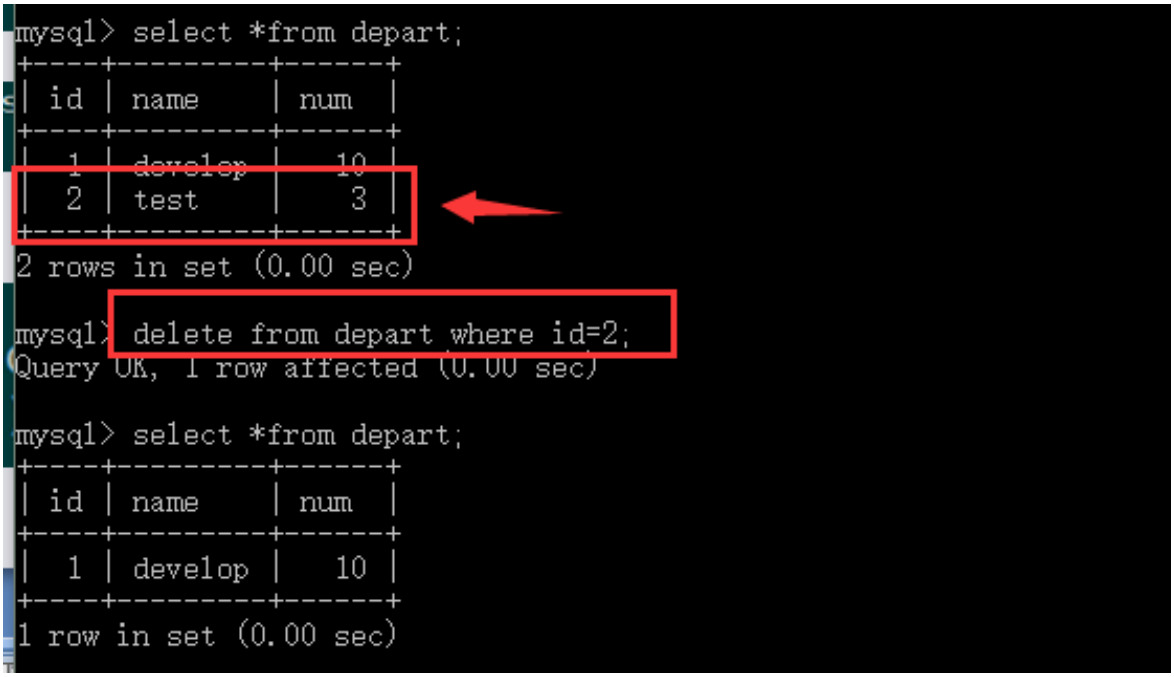

命令提示符 - mysql -uroot -p
mysql> show databases;
显示数据库
| Database |
| information_schema |
| company |
| mysql |
| performance_schema |
| test |
5 rows in set (0.00 sec)
mysql> use company
进入数据库
Database changed
mysql> use company;
Database changed
mysql> create table depart(id int primarykey auto_increment,name char (20) not null ,num int(20));
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
for the right syntax to use near 'primarykey auto_increment,name char (20) not null ,num int(20))' at line 1
mysql> create table depart(id int primary key auto_increment,name char (20) not null ,num int(20));
Query OK, 0 rows affected (0.10 sec)
创建表depart
mysql> desc depart;
显示表结构
Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
name	char(20)	NO		NULL	
num	int(20)	YES		NULL	

3、增


3 rows in set (0.01 sec)
mysql> insert into depart (name,num) values('develop',10);
Query OK, 1 row affected (0.00 sec)
mysql> insert into depart values('test',4);
ERROR 1136 (21S01): Column count doesn't match value count at row 1
mysql> insert into depart values(2,'test',4);
Query OK, 1 row affected (0.00 sec)

4、删

1）删除记录


mysql> select *from depart;
id	name	num
1	develop	10
2	test	3
2 rows in set (0.00 sec)
mysql> delete from depart where id=2;
Query OK, 1 row affected (0.00 sec)
mysql> select *from depart;
id	name	num
1	develop	10
1 row in set (0.00 sec)

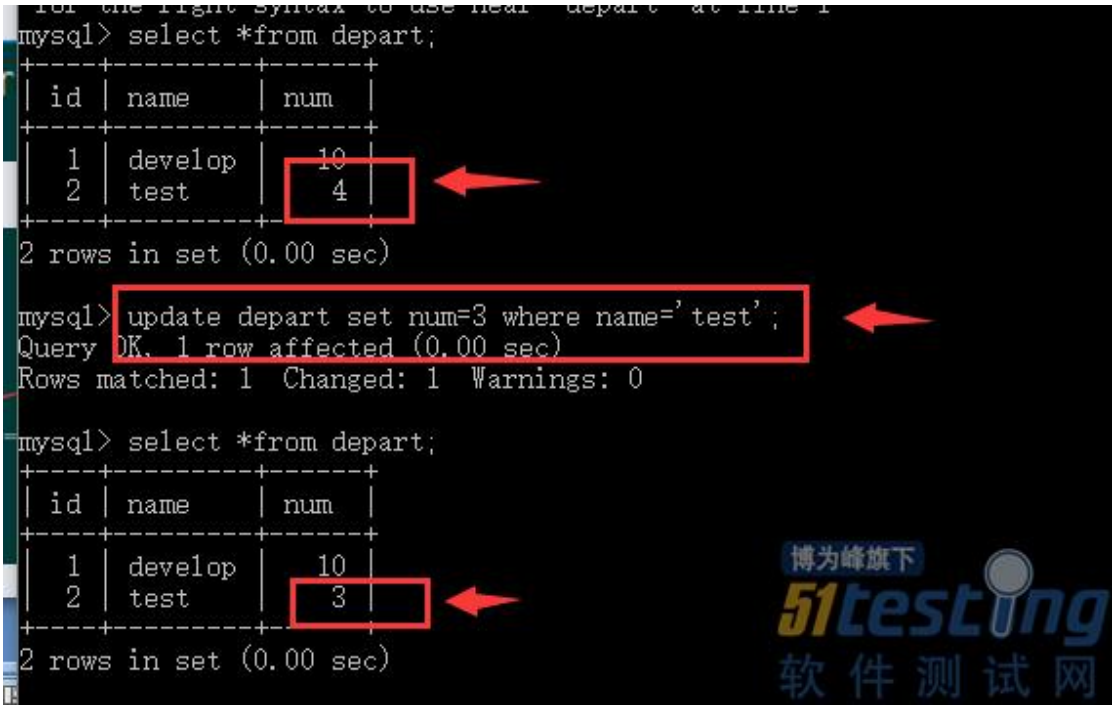
2）删表:

```
mysql> show tables;
+-------------------+
| Tables_in_company |
+-------------------+
| depart            |
| employes          |
| student           |
+-------------------+
3 rows in set (0.00 sec)

mysql> drop table depart;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+-------------------+
| Tables_in_company |
+-------------------+
| employes          |
| student           |
+-------------------+
2 rows in set (0.00 sec)

mysql>
```

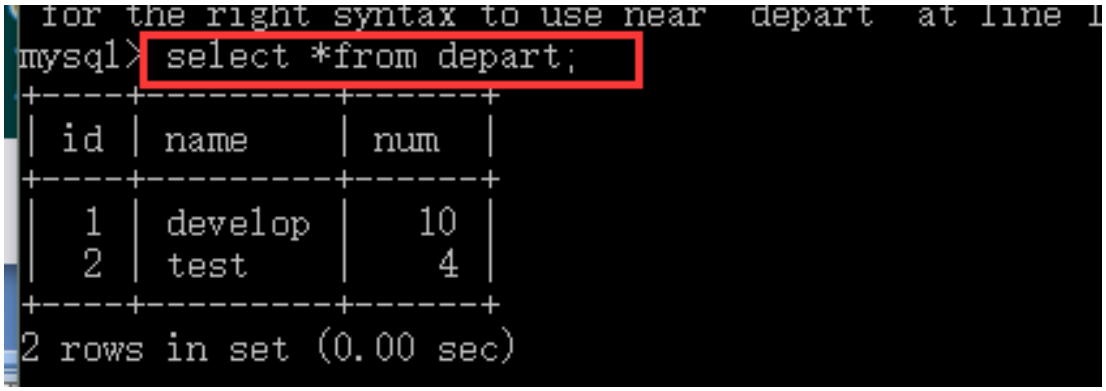
3）删数据库

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| company            |
| mysql              |
| performance_schema |
| school             |
| test               |
+--------------------+
6 rows in set (0.00 sec)

mysql> drop database school;
Query OK, 0 rows affected (0.09 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| company            |
| mysql              |
| performance_schema |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

5、改



6、查



三、Python 中操作 MySQL 数据库

以上都熟练了，在 Python 中操作就比较易懂了。

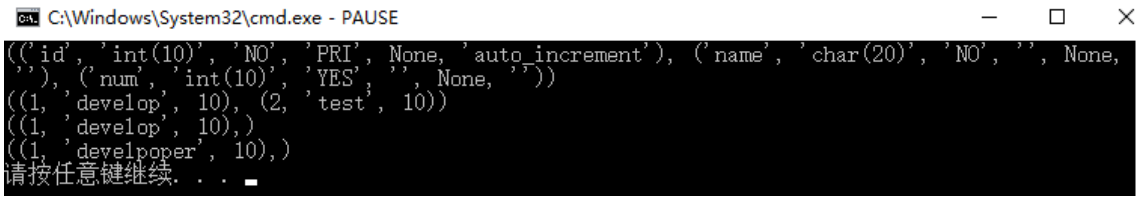
下面是我写的一个简单小程序（注意看注释部分）

代码:

```
import   pymysql    #引入 pysql 模块
#数据库连接
connect=pymysql.connect(
      db='company',          #数据库名称
      host='127.0.0.1',      #数据库的地址：本机 127.0.0.1
      user='root',           #用户名
      passwd='root',         #密码
      port=3306,             #MySQL 数据库的服务端口
```

```
        charset='utf8'          #数据库编码格式
    )
    cursor=connect.cursor()     #创建游标
    cursor.execute('drop table if exists depart;')  #如果存在该表就先删表
    def select ():              #定义查找全表的函数，后面用到
        cursor.execute('select *from depart;')      #查
        print(cursor.fetchall())      #fetchall(): 接收全部的返回结果行; fetchone(): 该方法获取下一个查询结果集，结果集是一个对象
    sql='create table depart(id int(10) primary key auto_increment,name char(20) not null,num int(10));' #创建表的 SQL 语句
    cursor.execute(sql)
    sql='desc depart;'      #显示表结构
    cursor.execute(sql)     #使用 execute() 方法执行 SQL
    print(cursor.fetchall())    #打印出查询结果
    sql="insert into depart (name,num) values('develop',10);"   #增
    cursor.execute(sql)
    cursor.execute("insert into depart(id,name,num) values(2,'test',10);")    #增 ，相比上面语句这个是直接在 execute 里面写语句。
    select()            #执行查询语句，看是否插入成功
    cursor.execute("delete from depart where name='test';")  #删
    select()            #执行删除语句，看是否删除成功
    cursor.execute("update depart set name='develpoper' where id=1;") #改
    select()            #执行改语句，看是否删除成功
    cursor.close()      #关闭游标
    connect.close()     #关闭数据库连接
```

运行代码，显示得结果如下:



附一张代码的图，可能比上面文字比较整齐清晰一点。

```
import  pymysql #引入pysql模块
#数据库连接
connect=pymysql.connect(
    db='company',     #数据库名称
    host='127.0.0.1', #数据库的地址：本机127.0.0.1
    user='root',      #用户名
    passwd='root',    #密码
    port=3306,        #MySQL数据库的服务端口
    charset='utf8'    #数据库编码格式
)
cursor=connect.cursor()  #创建游标
cursor.execute('drop table if exists depart;')   #如果存在该表就先删表
def select ():        #定义查找全表的函数，后面用到
    cursor.execute('select *from depart;')  #查
    print(cursor.fetchall())    #fetchall(): 接收全部的返回结果行; fetchone(): 该方法获取下一个查询结果集。结果集是一个对象
sql='create table depart(id int(10) primary key auto_increment,name char(20) not null,num int(10));'   #写创建表的SQL语句
cursor.execute(sql)
sql='desc depart;'        #显示表结构
cursor.execute(sql)       #使用 execute() 方法执行 SQL
print(cursor.fetchall())  #打印出查询结果
sql="insert into depart (name,num) values('develop',10);" #增
cursor.execute(sql)
cursor.execute("insert into depart(id,name,num) values(2,'test',10);")  #增 , 相比上面语句这个是直接在execute里面写语句。
select()          #执行查询语句，看是否插入成功
cursor.execute("delete from depart where name='test';")  #删
select()    #执行删除语句，看是否删除成功
cursor.execute("update depart set name='develpoper' where id=1;")  #改
select()    #执行改语句，看是否删除成功
cursor.close()  #关闭游标
connect.close()  #关闭数据库连接
```

这就是今天总结的内容啦，希望对大家有帮助。

测试过程及管理的经验总结

◆作者：呆小秋

我从一个一窍不通的测试小白，摸爬滚打到今天，在公司担任测试组长，管理公司的测试组，总觉得软件测试人员不容易。但是，还是对软件测试这个行业满腔热血（至少目前是）。

这几年来，跳的坑实在不少，有些也是自己挖。今天就来谈谈我的一些经验总结。

一、测试过程的经验总结:

1、测试过程中遇到严重的问题，除了登记在 bug 管理系统，也要反馈给领导。

为什么要反馈给领导？因为很多事情就算是让你背锅，你也背不起的，反馈给领导，领导会帮忙监督开发并给出意见。特别是紧急项目，不要在测试完的时候才给领导反馈，在测试过程就可以反馈了。

2、保管好自己的测试机子，不要轻易让软件开发人员或者其他测试人员动用了你的测试机。

有时候测试资源比较紧缺的时候，开发都是乱拿机子的，看到谁桌上有就顺走了，测试完放回来，没通知你，压根不知道版本有变换了，还在胡乱测试了那么久。所以一定要保管好测试机。不然版本会乱的。

3、发现越多问题的模块，越来更加注意

因为有可能开发在这个模块的代码逻辑有问题，所以更加花更多的时间测试，怕一些隐秘性比较高的 bug 逃逸了。

4、测试的重点依附与测试背景

电信运营对性能和并发要求高，金融行业对安全性要求高所以要搞清楚测试的重点在哪。

5、用 80%的时间花在 20%的重点模块。

比如我们公司是做车载 pos 机（也有前置系统+web 及 APP），车载 POS 最重要的就是远程升级程序和刷卡交易模块了。远程升级如果没办法保障，是要技术支持要客户现场一台一台机子升级的，可想而知那个人力物力财力得花多少，而且给客户留下不好的印象。之前我们公司出现一个案例就是版本很紧急，测试人员在最后回归测试的时候，认为开发修改的不会影响到远程升级模块，所以没有测试。只测试了修改点。结果就出现升级不了的现象。整个研发部被批斗。

6、遇到问题，不能说软件开发在调查，就没有测试的事情了，测试也要跟踪原因，做好记录。

领导问你情况，一问三不知的话，会显得你很不专业的，而且知道了原因所在，你也知道开发大概要修改那一块的代码，是否会影响到其他模块，对于回归测试是很有帮助的。

7、一个版本一个版本的测试，不要接受开发在测试过程提交新版本。

这样不仅耗时，很有可能也会造成你的测试思路被打断容易造成漏测。而且你的 bug 回归，测试文档的编写都会很麻烦。不过这个要看公司的制度了。

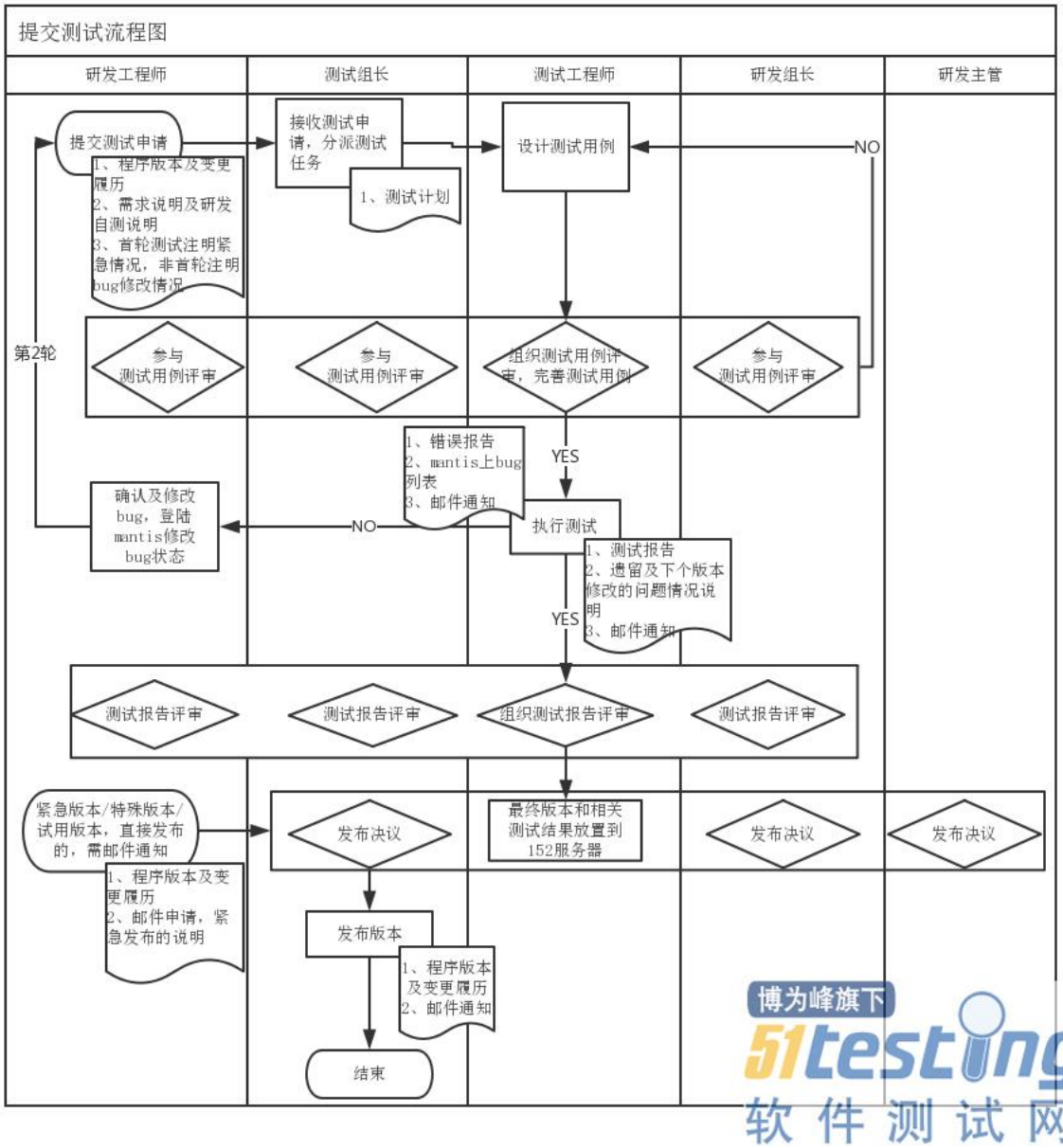
二、测试管理总结

公司的测试团队如何，要看测试流程及领导班子们的重不重视。

1、一定要完善测试流程规范包括提测试申请规范。

刚接手管理测试组的时候，整体制度是很乱的，流程也是很乱。请教了很多测试的前辈们，也上网学习了很多。然后就和我们研发总监提出要重新制定一个测试流程规范。

规范如下:



有了测试流程规范，研发总监也在公司进行宣讲，并要求强制执行了，整体流程就规范多了。

2、测试用例需要组内不定期评审和修改。

只有不断改善才能发现更到的 bug。类似杀虫剂悖论，一直使用那个牌子的杀虫剂，你家的蚊子估计都已经是有免疫了。

3、一定要和领导们强调开发自测的重要性。

开发没有自测，到测试这边一堆问题，问题一多，测试就容易乱了，很容易遗漏问题。另外就是开发没有自测过的，容易造成反复测试及版本多等现象，那测试人员就会被占用

了。那组内测测试计划就很不好安排。我是刚好被我抓到 2 次，一次开发都没有和后台调通某个功能的报文，一次是机子在刷卡交易过程直接程序就崩溃了，有开发小辫子了，和研发总监反馈的时候有理有据，现在我们总监也一直在强调开发自测的重要性。

4、发布版本的时候，一定要再三检验版本是否正确。

一道屏障是：组内强调测试人员一定要确定放置到内部服务器的程序的正确性。

二是发布到生产服务器或客户前，一定要再三确认版本是正确的，和对应测试人员核对。

发错了将直接导致整批量的产品版本都不对，发到客户那边的话问题就大了。

5、重视测试报告的评审和发布决议。

以前我们公司领导对于要发布的程序的情况一定都不了解，程序到客户那边，收到客户投诉的时候就开始追究责任了。所以有些情况在发布前一定要让领导知道，目前要发布的版本还存在哪些问题，开发要留到下个版本修改的，或者不改，或者时间问题，没有测试的，一定要和领导交代清楚。领导知道了一个大概的情况，在和客户交付的时候也好说明，不至于到客户那边上线发现问题后反馈给领导，领导一脸懵逼。

6、一定要制定测试计划或测试工作表。

制定测试计划和工作表，对一个整体组内工作的情况有个明确的概念，另外根据测试人员执行测试的情况，填入测试计划中可以直观的反馈测试人员的一些情况，是否延迟，新增测试用例数，逃逸 bug 数等。了解组内测试人员情况，可针对性培训等。

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
序号	项目	提测日期	是否紧急	测试人	测试开始日期	预计结束测试日期	实际测试结束日期	新/旧项目	发布情况	是否测试	备注	变更点	测试用例个数	新增测试用例个数	逃逸bug数	

7、做好文档的管理和分类

测试文档多，且项目一多，文档更是多得不行，测试服务器上的文档管理分类也至关重要。做好测试文档的管理和分类，能让平常的工作便捷很多，领导查看时也不会觉得一团糟。测试的工作虽然处处是坑，希望大家都能及时发现坑，不往里跳。

好用的 Pytest 单元测试框架

◆作者：邵君兰

单元测试框架，大家较为熟悉的有 Unittest、Nose（unittest 的扩展）。Pytest 倒是名不见经传，但是自从用了之后，就爱不释手，就像它自己描述的一样：simple powerful testing with Python。简单好用的参数化以及多种运行模式，让测试脚本简单清晰，调试运行更加方便。

一、参数化

比如我们在做自动化测试，当传入不同的参数时，期望返回的 status code 为 200。如果用 Unittest 做这块参数化比较复杂，而 pytest 会简单很多。在 pytest 中有 pytest.mark.parametrize 装饰器，轻松解决问题。我们就拿 sogo 的搜索作为例子。

用例说明:

步骤一、打开搜狗搜索（http://www.sogou.com/）

步骤二、输入 tesla 后点放大镜搜索

期望：返回正常

如果用接口来做的话，就相当于在 query 参数后面传 tesla 即可，验证状态码为 200

如果用接口来做的话，就相当于在 query 参数后面传 tesla 即可，验证状态码为 200

```
Import requests
Session = requests.session()

url = 'https://www.sogou.com/web?'
para = {'query': 'tesla'}
r = session.get(url, params=para, verify=False, allow_redirects=False)
assert r.status_code == 200
```

事实上，如果仅仅传一个搜索词就认为这个接口正常，那么很可能就出大篓子。所

以我们可能需要传 n 个搜索词做验证。也就是说，para 中的 query 后面的内容我们需要传多个不同的搜索词。比如说我们想搜索 VR,BITCOIN 等多个词的话，需要怎么做呢？这个时候就是使用 pytest 中有 pytest.mark.parametrize 装饰器，如下代码第 8 行，通过在方法外面加一个装饰器就可以了。

```
Import pytest
Import requests
Session = requests.session()

url = https://www.sogou.com/web?'
search= ['tesla','VR','BITCOIN']

@pytest.mark.parametrize('test_input',search)
def test_status_code(test_input):
  para = {'query':test_input}
  r= session.get(url,params=para, verify=False, allow_redirects=False)
  assert r.status_code == 200
```

此时有可能有同学提出疑问，对于多个参数我们用 for 循环也是可以做的，为何要用这个方法。原因是用 pytest 的参数化，传入的一个参数就是一个 case，上面的例子我们传入了三个参数，那么形成的就是三个 cases，哪个参数传入导致出错了，我们一目了然。那么要怎么做呢？我们在 cmd 里面运行脚本,假设脚本文件名为 test_sogo_search.py，位置放在 C 盘下:

```
C:\>pytest test_sogo_search.py --html=./result/Report.html
```

在 pytest 的报告中你看到的是这样的

通过这样的报告，就能很清晰的知道哪个搜索词的验证结果是正确，哪个搜索词验证的结果是错误的。比起用 for 循环来说，是要清晰很多了。另外还有运行的时长，也是能给到额外的信息。

二、运行模式

Pytest 的多种运行模式，让测试和调试变得更加得心应手，下面介绍 5 种用的比较多的模式。在介绍之前需要提醒一句，运行 pytest 时会找 test 开头的文件以及以 test 开头的方法或者 class，不然就会提示找不到可以运行的 case 了。

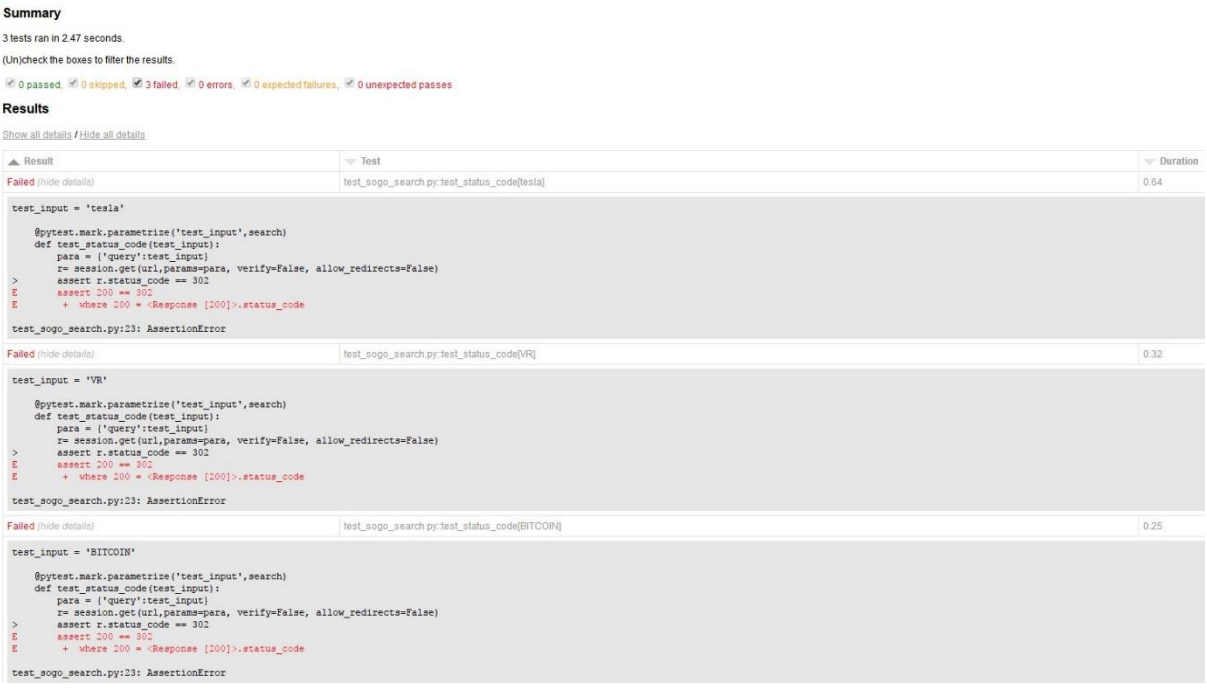
```
plugins: xdist-1.22.0, metadata-1.5.0, html-1.16.0, forked-0.2
collected 0 items

======================== no tests ran in 0.19 seconds ========================
```

1、运行后产生结果报告（htmlReport）

- 运行模式:

```
pytest test_case.py --html=./result/test_caseReport.html
```

- 效果:



Summary

3 tests ran in 2.47 seconds.

(Un)check the boxes to filter the results.

0 passed, 0 skipped, 3 failed, 0 errors, 0 expected failures, 0 unexpected passes

Results

Show all details / Hide all details

Result	Test	Duration
Failed (hide details)	test_sogo_search.py::test_status_code[tesla]	0.64

```
test_input = 'tesla'

    @pytest.mark.parametrize('test_input',search)
    def test_status_code(test_input):
        para = {'query':test_input}
        r= session.get(url,params=para, verify=False, allow_redirects=False)
>       assert r.status_code == 302
E       assert 200 == 302
E        +  where 200 = <Response [200]>.status_code

test_sogo_search.py:23: AssertionError
```

Failed (hide details)	test_sogo_search.py::test_status_code[VR]	0.32

```
test_input = 'VR'

    @pytest.mark.parametrize('test_input',search)
    def test_status_code(test_input):
        para = {'query':test_input}
        r= session.get(url,params=para, verify=False, allow_redirects=False)
>       assert r.status_code == 302
E       assert 200 == 302
E        +  where 200 = <Response [200]>.status_code

test_sogo_search.py:23: AssertionError
```

Failed (hide details)	test_sogo_search.py::test_status_code[BITCOIN]	0.25

```
test_input = 'BITCOIN'

    @pytest.mark.parametrize('test_input',search)
    def test_status_code(test_input):
        para = {'query':test_input}
        r= session.get(url,params=para, verify=False, allow_redirects=False)
>       assert r.status_code == 302
E       assert 200 == 302
E        +  where 200 = <Response [200]>.status_code

test_sogo_search.py:23: AssertionError
```

当运行出错时，在 report 里能直观的看到错误原因，上图中，预期希望的 status_code 是 302，但是实际是 200，所以就报错了。看着报告定位问题就变的非常容易。且测试结果不用整理，直接提交报告即可。

2、运行指定的 Case

- 代码块（test_case.py）

```
class TestClassOne(object):
```

```
    def test_one(self):
        x = "this"
        assert 'h' in x

    def test_two(self):
        x = "hello"
        assert hasattr(x, 'check')
class TestClassTwo(object):
    def test_one(self):
        x = "this"
        assert 'h' in x

    def test_two(self):
        x = "hello"
        assert hasattr(x, 'check')
```

- 运行模式:

模式 1: $ pytest test_case.py

模式 2: $ pytest test_case.py::TestClassTwo

模式 3: $ pytest test_case.py::TestClassTwo::test_one

说明:

模式 1: 直接执行整个 test_case 文件中在所有 case

模式 2: 运行 test_case 文件中 TestClassTwo 这个 class 下的两个 cases

模式 3: 运行 test_case 文件中 TestClassTwo 这个 class 下的 test_one

- 效果:

当我们写了较多的 cases 时，如果每次调式都要运行一遍，无疑是很浪费时间的，通过这样的简单方式我们就可以方便的指定任何一个 case 来运行了。

- 注意:

定义 class 时，需要以 T 开头，不然 pytest 是不会去运行该 class 的。

3、多进程运行 cases

```
pip install pytest-xdist
```

- 运行模式:

$ pytest tes_case.py -n NUM

- 效果:

当 case 达到一定量的时候，运行时间会变的很长，如果想缩短脚本运行的时间，那就可以用它来做。NUM 填写你想并发的进程数。速度嗖嗖嗖~不信你试试!

4、重试运行 cases

pip install pytest-rerunfailures

运行模式:

$pytest tes_case.py --reruns NUM

- 效果:

在做接口测试时，有时候会遇到 503 或者短时的网络波动，导致 case 会运行失败，而其实并非是我们想要的结果，此时可以通过重新运行 case 的方式，再次执行。重试多少次，跟你填写的 NUM 有关。

5、显示 print 内容

- 运行模式:

$pytest tes_case.py -s

- 效果:

在运行测试脚本时，为了调试或者想打印一些内容，我们会在代码中加一些 print 内容，但是在运行 pytest 时，这些内容不会被显示出来。如果带上-s 的运行模式，就可以看到 print 的内容会显示出来。

另外 Pytest 的多种运行模式是可以叠加执行的，比如说，你想同时运行 4 个进程又想打印出 print 的内容，可以用$pytest test_case.py -s -n 4

三、总结

pytest 的安装简单，学习资料丰富，框架切换成本低，有兴趣就一起来学习 pytest 吧!

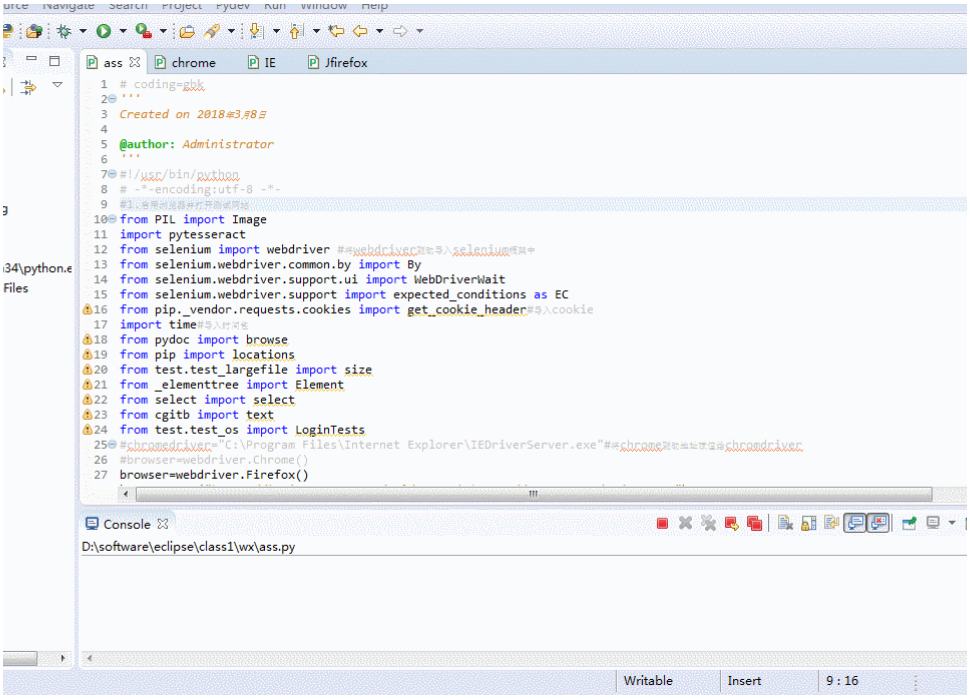
pytest 安装: pip install pytest

pytest 学习: https://docs.pytest.org/en/latest/contents.html

Selenium 实例讲解网站登录及元素定位方法总结

◆ 作者：桃 子

本文主要介绍了如何利用 Selenium 对网站进行登录，Xpath 元素定位，及窗口操作的一些常用方法，附上详细代码及解析，同时简单总结元素定位不到的原因有哪些。



一、图片验证码登录

这部分主要完成浏览器打开，通过元素定位在浏览器中输入账号、密码信息，最后通过 OCR 识别技术完成验证码的输入，保证登录成功。

1.1、启用浏览器并打开测试网站

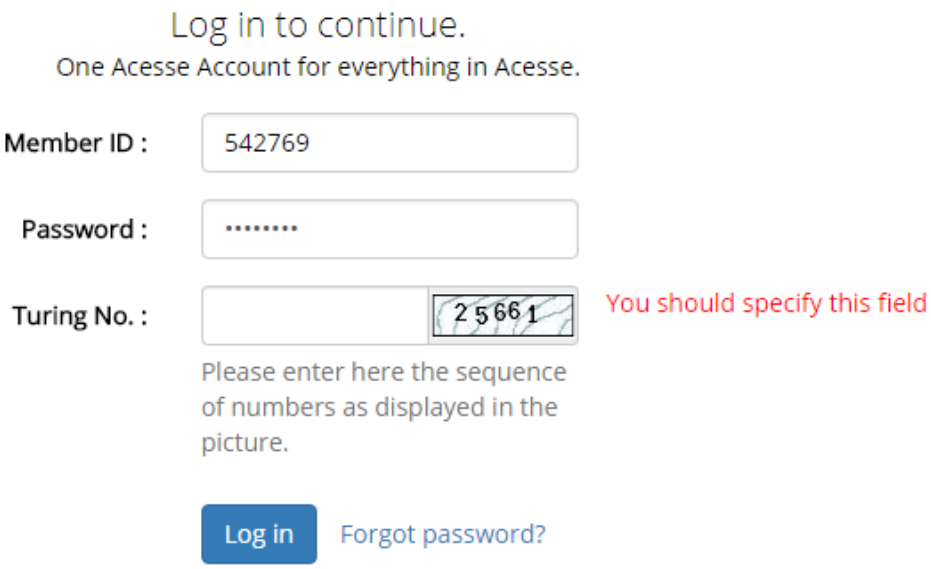
代码:

```
from selenium import webdriver #将 webdriver 驱动导入 selenium 框架中
from pip._vendor.requests.cookies import get_cookie_header
```

```
#导入 cookie（获取 cookie 时使用）
import time#导入时间包（后续 time 函数使用）
#将 chrome 驱动地址赋值给 chromdriver
browser=webdriver.Firefox()#打开 Firefox 浏览器
browser.get("https://login.xxxx.com/en?dest_url=https://xxxx.com/en/contact")
#打开测试网站（此网站地址需要填写自己测试的网站地址）
```

1.2、设置等待时间

在上面登录过程中网页加载慢，出现了还没等图片完全加载出来就进行图片识别的现象，导致识别失败，如下图



那么我们应该如何操作去避免提前加载的现象呢？

如果给它加一个条件满足（图片验证码显示完全）时：再进行图片识别操作是否可行呢，我们来看看。

Selenium 有 3 种等待时间:

名称	方法	特点
强制等待	Thread.sleep()	执行到此时不管什么就固定的等待三秒之后再接着执行后面的操作
隐式等待方法	implicitlyWait()	隐式等待采用全部设置，此方法针对执行脚本的所有对象，等待 10 秒
显示等待方法	WebDriverWait()	明确的要等到某个元素的出现或者是某个元素的可点击等条件,等不到,就一直等,除非在规定的时间之内都没找到,那么就跳出 Exception

这里使用 WebDriverWait()方法:

```
WebDriverWait(driver,timeout,poll_frequency=0.5,ignored_exceptions=None)
```

driver ： 浏览器驱动名称

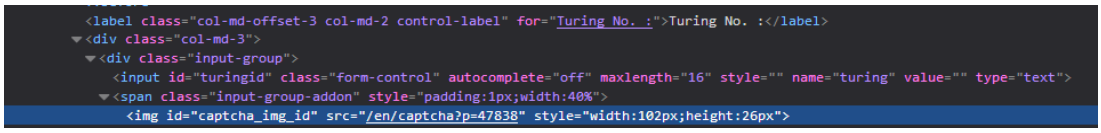
timeout ： 最长超时时间，默认以秒为单位。

poll_frequency ： 检测的间隔（步长）时间，默认为 0.5S。

ignored_exceptions ： 超时后的异常信息，默认情况下抛 NoSuchElementException 异常

WebDriverWait()一般由 until()或 until_not()方法配合使用

HTML 结构:



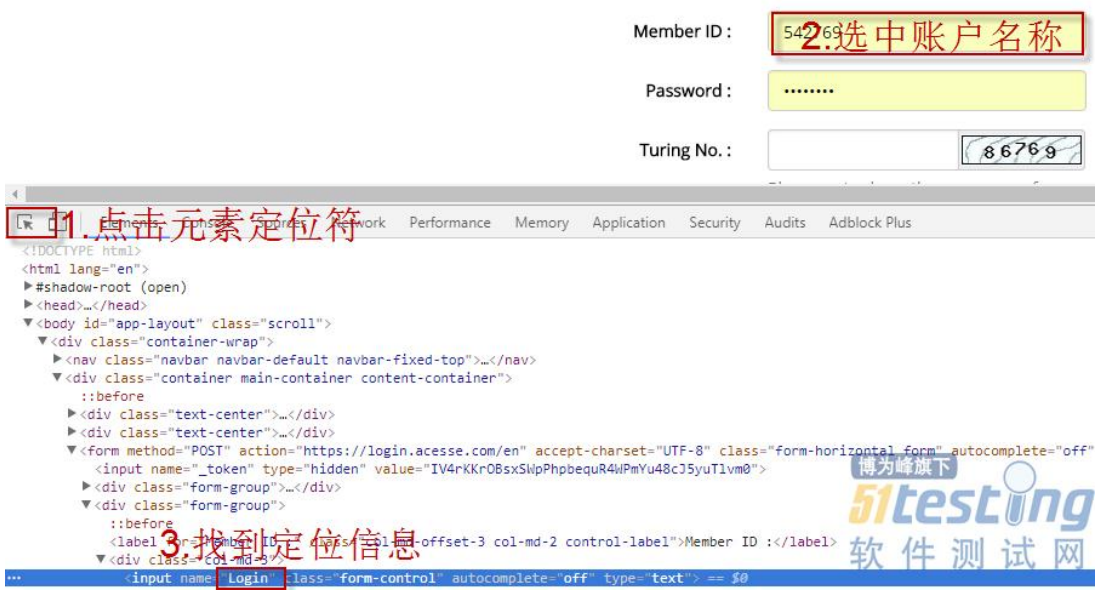
代码:

```
Element=WebDriverWait(browser,100).until(
    EC.presence_of_element_located((By.ID,"captcha_img_id"))
)
```

代码意思是：ID 名称为"captcha_img_id"的元素显示等待 100s，如果没有出现，抛出异常，通过上面方法就可以处理等待页面元素加载完全后进行相关功能操作。

1.3、账号、密码元素定位并输入内容

首先通过 F12 打开代码界面，点击元素定位图标，选中要定位的“账号”文本框内容，在代码区域找到定位的元素代码信息，如下图中的 name 名称“login”，密码元素同理



代码:

```
browser.find_element_by_name("Login").send_keys("xxxx")
#通过 name 定位登录文本框，通过 send_keys 输入账户信息
browser.find_element_by_name("Password").send_keys("xxxx")
#通过 name 定位密码文本框，通过 send_keys 输入密码信息
```

1.4、验证码处理-使用 OCR 自动识别

看网上介绍验证码处理大致有 4 种方法，1.让开发把验证码代码注释掉 2.让开发设置万能验证码 3.通过添加 cookie 方式绕过图片验证码 4.OCR 自动识别，其中 1-2 因为这个网站与开发接触不到，不予考虑方法 3 一般适用于记住登录状态的网站才适合，这里我使用第 4 种方法，这种方法适合于处理比较简单的验证码

OCR 自动识别的原理是什么呢?

在这里我们需要使用 pytesseract，它是一款用于光学字符识别（OCR）的 Python 工具，即从图片中识别出其中嵌入的文字。**整个过程分为截取登录页面->获取验证码的位置坐标->打开截图->从截图中截取验证码的区域->使用 pytesseract 工具识别验证码。**

代码:

```
browser.save_screenshot('f://a.png')#截取当前网页，该网页有我们需要的验证码
yzm=browser.find_element_by_id("captcha_img_id") #定位验证码
location=yzm.location#获取验证码 x,y 轴坐标
size=yzm.size#获取验证码的长宽
```

rangle=(int(location['x']),int(location['y']),int(location['x']+size['width']),int(location['y']+size['height'])) #截取的位置坐标

i=Image.open("f://a.png") #打开截图

frame4=i.crop(rangle) #使用 Image 的 crop 函数，从截图中再次截取我们需要的区域

frame4.save('f://frame4.jpg')#将截取到的验证码保存为 jpg 图片

qq=Image.open('f://frame4.jpg')#打开 jpg 验证码图片

text=pytesseract.image_to_string(qq).strip() #使用 image_to_string 识别验证码

browser.find_element_by_name("turing").send_keys(text)#将识别的图片验证码信息输入到验证码输入文本框中

browser.find_element_by_class_name("btn").click()#点击登录按钮

运行代码后可能会遇到提示“系统找不到指定文件”

```
ass.py [C:\Users\wx\AppData\Local\Programs\Python\Python36-32\python.exe]
    return run_and_get_output(image, 'txt', lang, config, nice)
  File "C:\Users\wx\AppData\Local\Programs\Python\Python36-32\lib\site-packages\pytesseract\pytesseract.py", li
    run_tesseract(**kwargs)
  File "C:\Users\wx\AppData\Local\Programs\Python\Python36-32\lib\site-packages\pytesseract\pytesseract.py", li
    proc = subprocess.Popen(command, stderr=subprocess.PIPE)
  File "C:\Users\wx\AppData\Local\Programs\Python\Python36-32\lib\subprocess.py", line 707, in __init__
    restore_signals, start_new_session)
  File "C:\Users\wx\AppData\Local\Programs\Python\Python36-32\lib\subprocess.py", line 990, in _execute_child
    startupinfo)
FileNotFoundError: [WinError 2] 系统找不到指定的文件。
```

这个问题困扰了我好久，最后处理方案如下:

首先保证 pytesseract 环境安装正确-参见 pytesseract 环境安装文章:

http://www.51testing.com/html/47/n-3725847.html

其次，打开文件 pytesseract.py，找到如下代码，将 tesseract_cmd 的值修改为全路径如:

tesseract_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesseract'**#这里一定要用\\不能用\，在程序里\\表示转译，如果只使用\是没用的。**

运行代码，图 1 中的整个自动登录功能就实现了，怎么样有没有一种要飞的感觉……

二、Firepath 工具方法定位元素

2.1、实现的主要功能

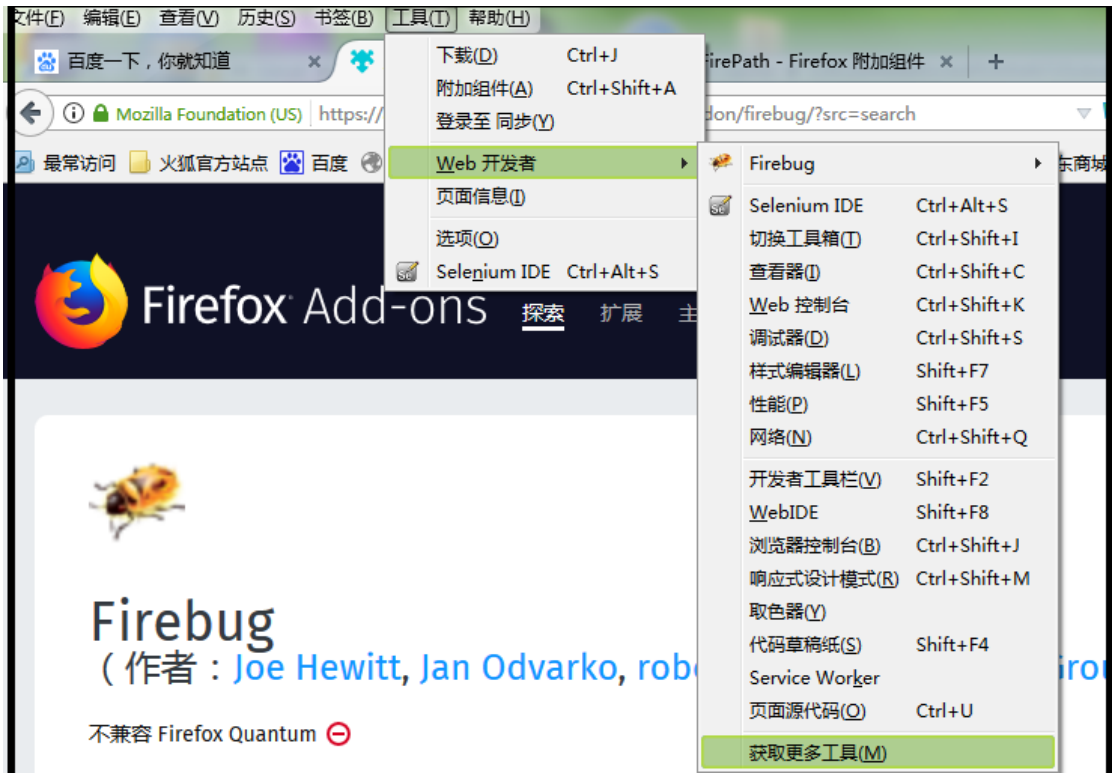
点击用户名称，选择选中下拉菜单选项进入详细页面

借助 Firebug 和 Firepath 工具，方便我们使用 Xpath 对元素进行定位，这里我们使用

Xpath 定位，一般都通过 Xpath 结合属性值进行定位元素，95%以上的定位都能通过此方法解决。

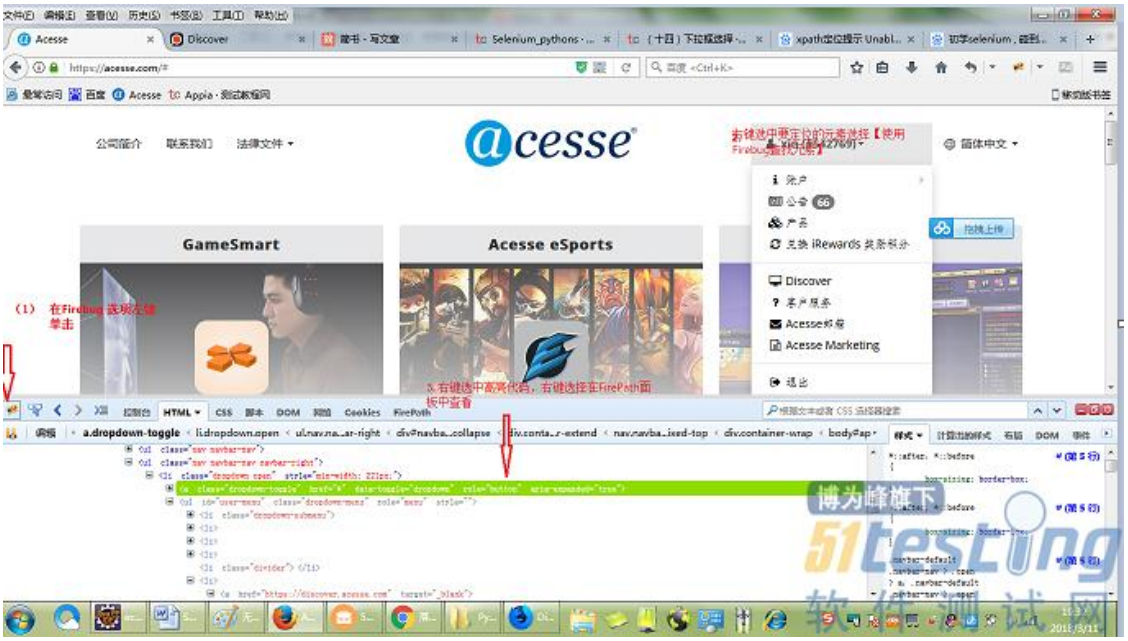
2.2、具体操作

1、首先下载 Firebug 和 Firepath 工具,下载步骤：工具-web 开发者-获取更多工具-搜索框搜索 Firebug-添加到 Firefox 即可【Firepath 同理】



2、工具使用，firefox 中按[F12]

- 在 Firebug 选项左键单击
- 右键选中要定位的元素选择【使用 Firebug 查找元素】
- 右键选中高亮代码，右键选择在 FirePath 面板中查看
- FirePath 下文本框内容就是 Xpath 定位命令,拷贝命定到代码编辑器中（elipse）



HTML 代码:

```
<html lang="en">
 <head>…</head>
 <body id="app-layout" class="scroll">
  <div class="container-wrap">
   <nav class="navbar navbar-default navbar-fixed-top">
    ::before
    <div class="container container-extend">
     ::before
     <div class="navbar-header">…</div>
     <style>…</style>
     <div class="logo-box">…</div>
     <div id="navbar" class="navbar-collapse collapse">
      ::before
      <ul class="nav navbar-nav">…</ul>
      <ul class="nav navbar-nav navbar-right">
       ::before
       <li id="notification">…</li>
       <li class="dropdown open" style="min-width: 191px;">
        <a class="dropdown-toggle" href="#" data-toggle="dropdown" role="button" aria-expanded="true">…</a>
        <ul id="user-menu" class="dropdown-menu" role="menu" style="">
         <li class="dropdown-submenu">…</li>
         <li>…</li>
         <li>…</li>
         <li>…</li>
         <li class="divider"></li>
         <li>
          <a href="https://discover.acesse.com" target="_blank">…</a>
```

3.定位 Xpath 元素，实现点击按钮

browser.find_element_by_xpath("//*[@id='navbar']/ul[2]/li[1]/a").click()

其中“//*[@id='navbar']/ul[2]/li[1]/a”这句话表示什么意思呢，跟着我一步一步看，

（1）//表示在文档的全部层级进行查找。

（2）[@id='navbar']表示定位 id='navbar'元素下

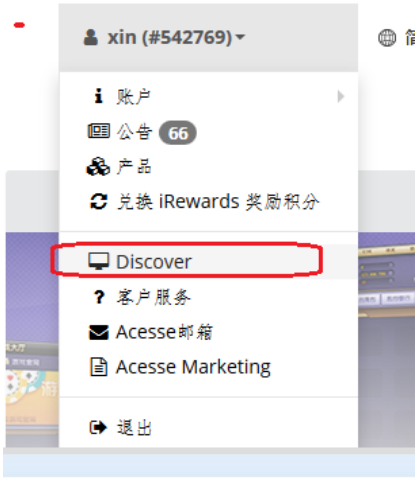
（3）ul[2]表示定位到第二个无序列表的样式 ul 下

（4）li[1]表示定位到第一个列表内行的样式 li 下

（5）a 表示定位到超链接下

从（1）到（5）是逐级展开的

4、通过 link text 定位元素实现跳转



实现代码：browser.find_element_by_link_text("Discover").click()

- by_link_text：文本链接方式
- click()：鼠标点击功能

Firepath 工具可以帮助我们轻松定位元素，对于初学者是一个非常不错的工具，熟练之后再慢慢练习自己写。

三、元素定位不到原因分析--窗口句柄

在程序运行过程中，一直提示元素定位不到“unable to locate element”，这个问题我解决了大概有一个星期，下面是我的解决思路。

页面元素结构：

```
<!DOCTYPE html>
<html lang="en">
▶ <head>…</head>
▼ <body id="app-layout">
  ▼ <div class="container-wrap">
    ▶ <nav class="navbar navbar-default">…</nav>
    ▼ <section class="top-panel">
      ▼ <form action="/adview" method="get" target="_blank">
        ▼ <div class="container inline-reset" style="padding-left: 0; padding-right: 0;">
            ::before
          ▶ <div class="col-lg-6 col-md-6 col-xs-12">…</div>
          ▼ <div class="col-lg-3 col-md-3 col-sm-6 col-xs-12">
```

定位代码:

browser.find_element_by_xpath("//form/div/div/descendant::a[@href='/adview']").click()

1. 无论怎么定位都提示 “unable to locate element”，如下图

```
    self.error_handler.check_response(response)
  File "C:\Python34\lib\site-packages\selenium-3.8.0-py3.4.egg\selenium\webdriver\remote\errorhandler.py", line 237, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.NoSuchElementException: Message: Unable to locate element: //form/div/div/descendant::a[@href='/adview']
```

于是就开启了找问题出在哪里

3.1、网上查定位元素不到的原因有哪些

首先，我在网上查元素定位不到的原因，因为根据提示，我们知道是元素没有定位成功

网上的思路大概都是:

1.定位的 ID 是动态，这里不涉及 ID，所以排除

2.iframe 原因定位不到元素

3.不在同一个 frame 里边查找元素，这里不涉及 iframe 元素，所以 2/3 排除

4.Xpath 描述错误

刚开始我以为是我元素定位的错误，所以花了很多时间在换各种方式进行定位：包括使用 Selenium IDE 工具、使用 Link（）方法、利用 Firepath 中的 Xpath Css 定位，仍提示定位不成功。

5.点击速度过快 页面没有加载出来就需要点击页面上的元素

我在语句前面加上加载时间 time.sleep(5)等待，还是不行

6.firefox 安全性强，不允许跨域调用出现报错

错误描述： uncaught exception: [Exception... "Component returned failure code: 0x80004005 (NS_ERROR_FAILURE) [nsIDOMNSHTMLDocument.execCommand]" nsresult: "0x80004005 (NS_ERROR_FAILURE)" location:

根据错误描述，这点也不符合，排除。

3.2、如何排除不是语句错误

根据之前操作，感觉不应该定位语句错误的原因，所以如何排除判断是不是语句错误呢。

1.因为这个页面属于登录成功后调用的第二个页面，所以我从上一页面开始，把之前的代码前加上注释#后运行

2.结果调用成功了，直接跳转到第三个页面了，可以说明语句没有编写错误

3.这里还有一个方法，就是你换其他浏览器试一试，但是我觉得还得重新配置太麻烦了

3.3、罪魁祸首-窗口句柄

既然语句编写的没有问题，那么我又开始怀疑人生了，到低哪里出错了

1. 继续网上找资料，发现是从 a 页面到 b 页面，窗口句柄还停留在上一个页面，所以无法定位元素

窗口句柄是什么

想要在某个窗口做一些事情，你就得让操作系统知道你是在哪一个窗口做这些事情，而窗口的句柄就能起到识别哪一个窗口的作用。

所以刚刚进入到一个页面后，将窗口进行重新定位应该就可以了

```
time.sleep(5
browser.switch_to_window(browser.window_handles[1])
browser.find_element_by_xpath("//form/div/div/descendant::a[@href='/adview']").click()
```

总结：以后再遇到跳转新页面的时候不要忘记加窗口句柄!!!

本篇文章就到这里，附上所有运行代码：

```
# coding=gbk
'''
Created on 2018 年 3 月 8 日
@author: Administrator
'''
#!/usr/bin/Python
# -*-encoding:utf-8 -*-
#1.启用浏览器并打开测试网站
```

```
from PIL import Image
import pytesseract
from selenium import webdriver #将 webdriver 驱动导入 selenium 框架中
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from pip._vendor.requests.cookies import get_cookie_header#导入 cookie
import time#导入时间包
from pydoc import browse
from pip import locations
from test.test_largefile import size
from _elementtree import Element
from select import select
from cgitb import text
from test.test_os import LoginTests
#chromedriver="C:\Program Files\Internet Explorer\IEDriverServer.exe"#将 chrome 驱动地址赋值给
chromdriver
#browser=webdriver.Chrome()
browser=webdriver.Firefox()
browser.get("https://login.acesse.com/en?dest_url=https://acesse.com/en/contact")#打开测试网站
Element=WebDriverWait(browser,100).until(
    EC.presence_of_element_located((By.ID,"captcha_img_id"))
    )
#2.对账号、密码元素进行定位
browser.find_element_by_name("Login").send_keys("xxxxx")
browser.find_element_by_name("Password").send_keys("xxxxxxx")
#3.验证码处理-使用 OCR 自动识别
#browser.maximize_window()
def login():
    browser.save_screenshot('f://a.png')#截取当前网页，该网页有我们需要的验证码
    yzm=browser.find_element_by_id("captcha_img_id") #定位验证码
    location=yzm.location#获取验证码 x,y 轴坐标
    size=yzm.size#获取验证码的长宽
```

```
        rangle=(int(location['x']),int(location['y']),int(location['x']+size['width']),int(location['y']+size['height'])) #截取的位置坐标
            i=Image.open("f://a.png") #打开截图
            frame4=i.crop(rangle) #使用 Image 的 crop 函数，从截图中再次截取我们需要的区域
            frame4= frame4.convert('RGB')
            frame4.save('f://frame4.jpeg')#讲截取到的验证码保存为 jpg 图片
            qq=Image.open('f://frame4.jpeg')#打开 jpg 验证码图片
            text=pytesseract.image_to_string(qq).strip() #使用 image_to_string 识别验证码
            browser.find_element_by_name('turing').send_keys(text)#将识别的图片验证码信息输入到验证码输入文本框中
            browser.find_element_by_class_name("btn").click()#点击登录按钮
        login()
        browser.find_element_by_xpath("/html/body/div[1]/nav/div/div[3]/ul[2]/li[2]/a").click()
        browser.find_element_by_xpath('/html/body/div[1]/nav/div/div[3]/ul[2]/li[2]/ul/li[6]/a/span').click()
        #browser.implicitly_wait(30)
        time.sleep(5)
        browser.switch_to_window(browser.window_handles[1])
        browser.find_element_by_xpath("//form/div/div/descendant::a[@href='/adview']").click()
```

❖ 拓展学习

■从入门到高薪玩转 Selenium 自动化：http://www.atstudy.com/course/270

版本控制系统--Git

◆作者：非比君

版本控制系统（Version Control System，VCS）是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。在一个项目团队中，开发人员需要对代码做版本控制，而测试人员编写自动化测试脚本后，同样需要对这些内容的版本做维护和管理。目前比较热门的版本控制系统是：Git。

本文主要介绍以 Windows 操作系统为例，在 GitLab 上使用 Git 来管理测试版本（包括自动化测试脚本、测试用例、测试文档）的一系列操作，包括：Git 安装、GitLab 配置 SSH 连接、GitLab 上创建仓库、上传本地项目、克隆代码到本地以及推送（Push）更新后的脚本到 GitLab 等。

1、Git、GitHub 和 GitLab

在介绍具体操作之前，我们先来了解：Git、GitHub 和 GitLab 究竟是什么，这三者之间有什么关系。

Git: 正如上面提到的，是一个版本控制系统，它是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。Git 的发明者正是 Linus Torvalds--Linux 操作系统的开创者。

GitHub: Github 和 Git 是两回事。Github 是在线的基于 Git 的代码托管服务。GitHub 是 2008 年由 Ruby on Rails 编写而成。GitHub 同时提供付费账户和免费账户。这两种账户都可以创建公开的代码仓库，但是只有付费账户可以创建私有的代码仓库。

GitLab: 在很大程度上 GitLab 是仿照 GitHub 来做的，它们都是在线的基于 Git 的代码托管服务，提供了分享开源项目的平台。但是 GitLab 解决了 GitHub 付费的问题，可以创建私人的免费仓库（PS：GitLab 可免费创建私有代码仓库，但是更高级的服务仍要收费）。相较于 GitHub，GitLab 允许免费设置仓库权限；允许用户选择分享一个 project 的部分代码；允许用户设置 project 的获取权限，进一步的提升安全性；可以设置

获取到团队整体的改进进度；通过 innersourcing 让不在权限范围内的人访问不到该资源。

2、安装 Git

Git 有许多种安装方式，主要分为两种，一种是通过编译源代码来安装；另一种是使用为特定平台预编译好的安装包。本文推荐第二种安装方式，到 GitHub 的页面上下载支持 Windows 系统的 exe 安装文件并运行，下载链接：http://msysgit.github.com/。

安装 Git.exe 之后，开始菜单中会有 Git 文件夹，目录下有：Git Bash、Git CMD 和 Git GUI。

3、GitLab 配置 SSH 连接

把本地的项目传到 GitLab 上，需要先配置 SSH Key。当然也可以不需要 SSH Key 连接，可以通过 HTTPS 的方式上传本地项目，只是 HTTPS 连接时，每次要输入用户名、密码。

（1）首先，在开始菜单里找到“Git”->“Git Bash”并打开。输入如下命令以生成一个新的 SSH:

```
$ssh-keygen -t rsa -C "email" #gitlab 注册的邮箱
```

然后直接回车，不用填写东西。之后会让你输入密码（可以不输入密码，直接为空，这样更新代码不用每次输入 id_rsa 密码了）。然后就会在“C:\Users\电脑用户名”路径下生成一个目录.ssh ，里面有两个文件：id_rsa , id_rsa.pub（id_rsa 中保存的是私钥，id_rsa.pub 中保存的是公钥）。

（2）在如下图的 GitLab 上打开 SSH keys->add key 页面，把 id_rsa.pub 公钥的内容复制进去就可以了。

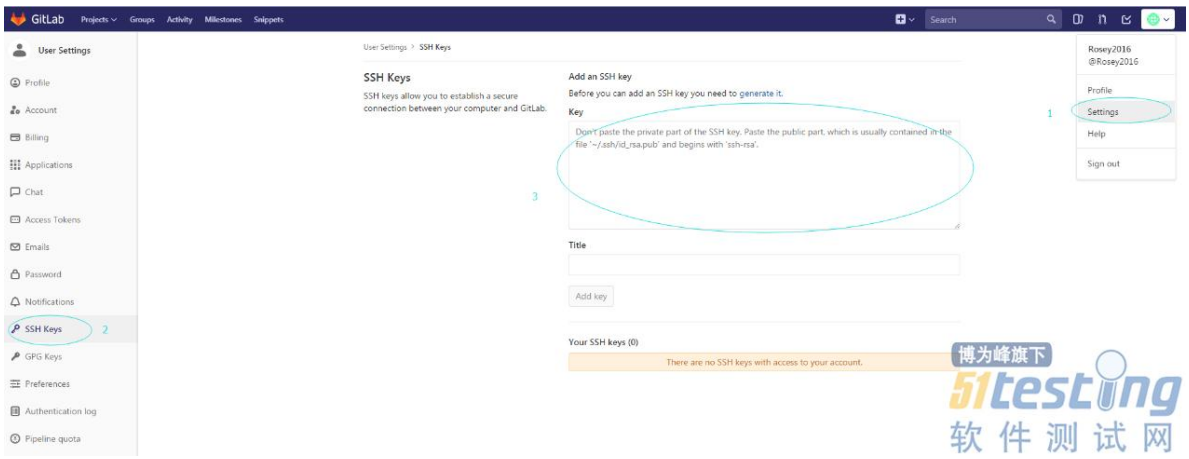


图 GitLab1

（3）在 Git Bash 上配置账户，命令如下:

```
$ git config --global user.name "your_username"   #设置用户名
$ git config --global user.email "your_registered_github_Email"   #设置邮箱地址(GitLab 注册的邮箱)
```

（4）最后，打开 IDEA 或 Pycharm 编辑器，就可以通过 SSH 连接来 Clone 已有的仓库代码了。具体操作，下面会详细介绍。

4、GitLab 上创建仓库并上传本地项目

（1） 在 GitLab 上创建仓库，如下图。首先进入 GitLab 选择 Create a Project，填写 Project name、Project description（optional）、Visibility Level。然后点【Create project】按钮创建。

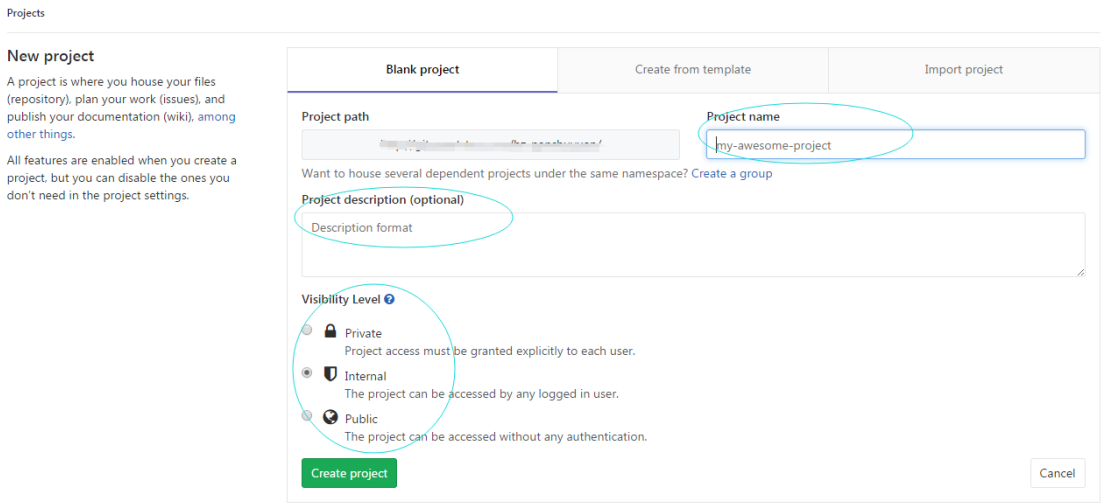


图 GitLab2

（2） 然后在 GitLab 上创建 README 文件。打开本地电脑中的测试脚本所在文件夹，将.idea 文件的 IDE 配置信息删除，这样是为了保证 IDE 配置信息不会上传到 GitLab 中。

（3）选中测试脚本文件夹，鼠标右键打开 Git Bash Here，依次输入如下命令:

- 输入 git config --global user.name "你的用户名"
- 输入 git config --global user.email "你的邮箱"
- 输入 git init
- 输入 git remote add origin 配置好的 SSH 地址
- 输入 git add .
- 输入 git commit -m "Initial commit"
- 输入 git push -u origin master 将代码推送到 GitLab 端

当然也可以根据 GitLab--创建仓库后会在 GitLab 页面上展示出命令语句，根据这些语句来一步一步地操作。上传完成之后，本地项目就出现在 GitLab 上。

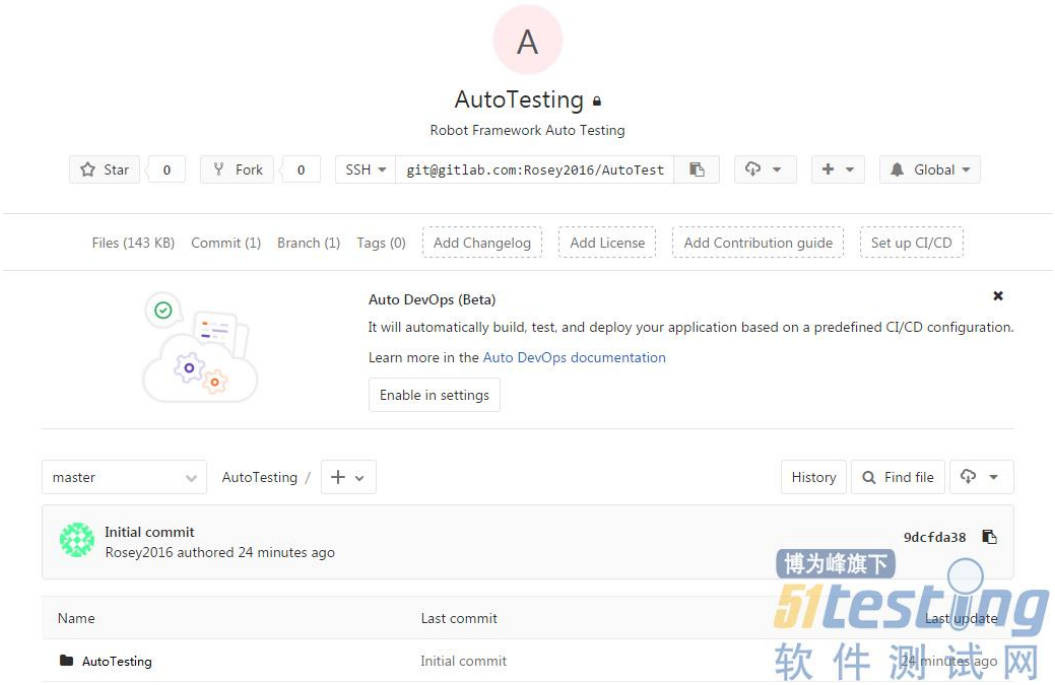


图 GitLab3

CLONE 脚本到本地

配置 SSH Key 之后，就可以把 GitLab 上的脚本直接 Clone 到本地。打开 Pycharm 或者 IDEA 工具，在“Check out from Version Control”中选择“Git”，然后在“Git Repository URL”中输入 GitLab 上的 SSH Key 链接；在“Parent Directory”输入 Clone 项目到本地的存放路径；然后在“Directory Name”中输入项目的名称，点击【Clone】按钮，就会将 GitLab 仓库导入到本地。

每一个 Git 克隆都是一个完整的文件库，含有全部历史记录和修订追踪能力，不依赖于网络连接或中心服务器。其最大特色就是“分支”及“合并”操作非常快速、简便。

图 GitLab4

6、推送（Push）更新后的脚本到 GitLab

本地推送更新后的脚本到 GitLab，在配置了 SSH Key 连接后，使用 Pycharm 或 IDEA 工具进行操作是非常地方便。详情搜索网上相关工具的操作手册，此处不再详细介绍。

总之，版本控制系统—Git，可以协助软件测试人员管理测试脚本等内容。Git 具有版本库本地化，支持离线提交，相对独立不影响协同开发等优点。每个编写自动化脚本的测试人员都拥有自己的版本控制库，在自己的版本库上可以任意的执行提交代码、创建分支等行为，不用担心影响到其他测试人员提交的测试脚本。大家快去试试吧。

揭露测试中隐藏的边界值

◆ 译者：Alice

概述：边界值分析是测试设计一个稳定的部分，但是对黑盒测试人员来讲有时候边界并不是那么明显。这些不明显的边界被称作隐藏的边界。本文提供几个隐藏的边界的例子，还有一些以让隐藏边界显露来设计测试计划的要点方法。

使用边界值分析和等价类划分是测试设计的基础做法之一。该理论的含义是对于一个特定的输入，测试中更常使用的值是在这些输入范围的边界附近的值。在边界之间的一类值当用到测试时常被认为是“等价的”。

举个例子，如果你的 APP 有一个功能是让输入一个价格的打折值，有效的范围很可能就是 0 百分比到 100 百分比。将要测试的值就会是那些在边界附近的值，或-1 百分比，0 百分比，100 百分比和 101 百分比。你很可能也会选择测试一个“正常的”20%的折扣。如果那些工作正确，你不大可能通过测试其他可能的折扣值来发现缺陷。你在寻找在边界和超过边界的正确行为。

然而，有时候有些边界并不那么显而易见。我曾经因为隐藏的边界栽过跟头，导致缺陷跑到了顾客面前。这种情形不常有，但是当其出现时，我就会听到那个有魔力的话：“你没测过这个吗？”

在本篇文章中，我会详细叙述隐藏边界的几个引起因素：潜在的执行中的数据类型边界（比如，16 比特数据类型到 32 比特数据类型）；信任边界，尤其是在一个重新设计或重构中；有特别含义的数据值和软件中有趣的边角。

Data-Type Boundaries 数据类型边界

我测试过的一个电信管理平台有一个功能是在远程设备上配置不断变化的参数。其中一个参数有个范围是 1 到 1024000。前端 UI 将这个参数储存在一个 32 位的整型数种

并正确地处理了输入的验证和系统的行为，或者说期望的边界值（0，1，1024000 和 1024001）和几个会被顾客使用的正常值。我们并没有穷举式地测试每个值，因为在边界值之间的值被视作彼此都是等价的。

但是，在该管理平台和远程设备之间的通信网络有个协议将数据类型限制在 16 比特位。通信模块将 32 位的值撕成 2 个 16 比特的值并分开将他们传输。这是缺陷所在的地方。

当顾客将那个参数设置成 65536，它就被通信模块设置成 0。代码本应该将发送两个值作为该参数，65535 和 1。65535 是 16 比特所能代表的最大数值。撕开该值的代码有一位的偏差；它尝试将 65536 放进那个 16 比特的值中。

65535 和 65536 之间的边界在 UI 端部那么明显。它在软件说明书或测试计划中并没出现。只有参与做该设置理解整个系统的人这个边界才显而易见。

现在，当我看到一个要输入较大数值的数字型输入，我就会测试可能会出现的数据代表（256，65535 和 429467295）。

Trust Boundaries 信任边界

软件架构的一个要义是定义信任区。如果一个客户请求来自信任区，接收的 API 接口可以认为这些请求没有恶意意图并可以略过一些输入验证。这对提高系统性能和开发者效率非常有用。

有一个项目，我们正在按照商议好的约定快速向通过 Web 服务的外部公司展露我们潜在的商业逻辑。这个项目称作服务启动并在大量将 Web 服务包装在我们已有的 API 请求附近。

这些 API 请求原来只对我们程序的内部模块提供，而且他们被设计做内部模块。结果，那些输入被认为是来自信任源。输入验证以节省编码和处理时间。

然而，对这个服务启动项目，有些 API 请求被暴露给了外部的请求者。“快乐路径”测试用例使用跟我们的程序一样的输入，一切都很好--直到我们开始从 Web 服务层面测试并对输入产生怀疑。

一个特定的字段在该 API 中有读和写的能力，但是逻辑上，它只应该有读的能力。该条商业逻辑基于来自数据库的数据来计算那个值。前端 UI 只读到了该值，但是当开

发人员暴露那个 API，我们能够写入那个值，将不一致的数据写入数据库。

任何时候信任边界移动或被假定了，就很有可能是有严重的缺陷。明确地与开发人员检查信任边界然后相应地设计测试计划是一个好的工作做法。

Special Data Values 特殊数据值

很多 APP 对特殊数据值有特殊的行为。这些行为可能在软件说明书或者界面上不可见，但是他们可能导致缺陷。

一个系统对测试有特殊规定，使用一个唯一的由那个用户个人资料中的信用卡号和其他数据组成的组合来付款。本来是不要对测试人员和测试账号收款的。然而，你猜到了--一个有关特殊信用卡检查的缺陷导致很多顾客可以免费使用我们的服务。

这种现象并不普遍或者会有灾难性的后果，当然顾客们并不在意，但是它却将使用特殊处理过的数据来猜测该账户是测试账户的危险暴露了出来。一个更好的设计应当是明确将那些账户指定为测试账户。

Easter Eggs 复活节彩蛋【软件中有趣的边角】

一个复活节彩蛋是建到一个 APP 里的有些特殊的功能--也许是一些有趣的事，像迷你游戏或有团队名称的信用卷。软件中的首个复活节彩蛋是在阿他瑞 2600 游戏毛线。如果游戏玩家拿起一个特定的不可见的物体，他们将进入一个秘密房间看到来自游戏创造者留的信息。这个房间不会在任何文档或测试中出现，因为没有人知道它。

复活节彩蛋很有趣，但是他们可能暴露系统的脆弱性，尤其是没有测试彻底的情况下。我见过的一个缺陷彩蛋是一个在特别的天里将我们在 app 上的正常品牌替换掉的特殊的标志。举个例子，在一个如独立日这样的节日里，该 logo 可能变化颜色有炮竹图像。尤其是除了那段时间它不工作，而用户看到的是一个 404 的错误。图像崩溃了，我们在发布之前没有那个特定的节日日期。

对隐藏边界可以做什么

基于隐藏边界的特性，他们难以发现，尤其是在黑盒测试里。随机地搜索隐藏边界是困难的而且有很少的成功可能。

但是，不是一切都迷失了。这里有一些可以处理隐藏边界的方法。

- 参加设计评审，寻找数据类型的不匹配情形和任何数据转换的情形。询问有关转换

的事情然后合适滴设计测试。

- 对数字数据输入，测试由潜在的数据类型定义的潜在边界。
- 学习代码包，寻找任何可能是复活节彩蛋的文件或资源。
- 问开发人员可能存在的边界和任何复活节彩蛋。

隐藏边界的潜在存在给黑盒测试同行们充足的理由去测试程序中有些潜在的架构和执行。希望本篇文章启发你去探索你的程序并暴露出一些潜在的问题。

《51 测试天地》（四十九）上篇 精彩预览

- 全职妈妈重返测试岗的经验分享
- 测试中具备什么更加重要：性格 VS 能力
- LoadRunner 实战（上）-了解 LR 使用过程
- LoadRunner 实战（下）-Analysis 性能数据分析
- 浏览器端性能提升之 DNS Prefetch
- 一文告诉你 ATDD,TDD,BDD 的区别
- 测试人员如何帮助软件成功上线
- 如何快速提高自己的测试能力
- VMware 10 Workstation 软件在测试中的应用
- 测试岗校招之旅--面试官的几个小问题

马上阅读