# Python自动化测试学习

1. 环境安装

编译器：pycharm

语言：python3.8

第三方库安装：pip install 库名

1. 接口测试
2. 常用库名：request Pprint（精美打印）
3. 常用方法：.get(‘http:/IP地址/地址/?参数要求&另外的参数要求’,list,其他参数)，其中list表单中{}里面要包含端口要求的用户名、密码、密钥等。

.port()//相同写法，效果是添加条例。

3.返回值：返回值一般为json类型，输出需要添加.status\_code等表单下表查找相应内容。

1. 数据库读写
2. 常用库名：pymysql
3. 常用方法：
4. .connect（host=’’,user=’’,passwd=’’,db=’’,charset=’’）

//db为表单名字 charset为char显示类型一般为‘utf8’

1. 链接之后，可以使用.connect.cursor()进行游标设置。这是必须的，因为要执行sql语句（数据库控制语句）必须通过游标进行。
2. .connect.cursor.execute（‘代码内容select \*from 表名’），执行sql语句。其中的代码内容是python的字符串翻译sql语句，常用到f”string{x}”这种形式,也就是python字符串中加入显示变量的方法。

❤：在操作数据库之后要添加一条语句.connect.commit()提交的意思

1. .connect.cursor.fetchone(),这句话代表execute输入后，数据库返回的一行内容。（如果多次使用就会往后返回每一行），而外有fetchmany()可以一次读取多行。
2. cursor.rowcount,这是游标的一个属性，表示这次查询一共有多少行
3. .Connect.close关闭数据库
4. app自动化测试
5. 环境安装：
6. 安装pip install -i 清华克隆pypi Appium-Python-Client
7. 安装Appium Server：<https://github.com/appium-dexktop/releases/latest>
8. 安装Android SDK:可以去Android Stdio的官网历史中找到2.3.3版本上传的SDK进行单独安装,同时配置一下环境变量-系统变量：

Android\_Home F:\Android\_SDK\SDK(SKD安装路径)

再添加SDK中tools中的adb到path路径中（方便敲代码的时候调用不用输入全名称）

1. 安装JDK:官网下载就可以，记得将jdk路径加入环境变量
2. 获取想要测试的界面。方法可以百度，这里提供adb logcat>方法。

先创建log.txt再在相应目录下运行adb logcat>log.txt，进入app界面然后关闭cmd查看log.txt中搜索Display,找到相应界面

1. 运行adb shell时出现error: cannot connect to daemon

：先打开一个命令行窗口 adb nodaemon server 回车后

再打开另外一个命令行窗口adb shell就成功了。

1. 手机连接，我选择的是模拟器夜神模拟器（这个模拟器有个自己的nox\_adb会占用5037端口，建议删除）0：端口是62001

Adb connect localhost:62001

Adb devices

1. 建立连接
2. 打开appium，建立表单信息。AppPackage和activity属性需要借助工具：aapt,

代码;aapt dump badging 安装包路径

里面包含package和launchable-activity项目的名字就是

1. adb进行设备连接,利用adb devices进行检查，远端设备用adb connect <host>:<port> 进行连接
2. 关于AppiumInspector的功能:

1，获取元素（包含ID，非常重要，用于自动化识别）

2，

3.进行代码编写：通过获得的ID和webdriver库进行代码的编写。要注意一点，appium参数字典必须有[**'platformName'**]，[**'appPackage'**]，[**'appActivity'**]

注：以下几个Key不一定要写翻译

[**'unicodeKeyboard'**]界面是否允许输入中文

[**'resetKeyboard'**]是否有Keybord

[**'noReset'**]

[**'newCommandTimeout'**]断线时间

1. 代码编写注意事项：

1）#这个是定义实例driver

driver=webdriver.Remote(**'http://localhost:4723/wd/hub'**,desired\_caps)

2）#元素存在判断方法的定义，因为webdriver没有相关的判断，所以要自己写

def is\_element\_present(how,what):  
 try:  
 driver.find\_element(by=how,value=what)*#查找参数进行查找* except NoSuchElementException:*#如果存在则这里是* return False  
 return True

3）按键使用

driver.find\_element\_by\_id(  
 **"com.lbe.security.miui:id/permission\_allow\_always\_button"**).click()#这里的fing\_element\_by\_id其实是返回了一个element对象，这里是用的是返回对象的click方法。

4）多元素查找：

Driver.find\_elements\_by\_id，注意，这里是elements，返回的是一个元素的表单，可以通过下表对里面的元素进行操作。比如：

List=driver.find\_elements\_by\_is(“ID”)

For i inrange (len(list)):

Print(list[i].test)

这段代码的意思是根据ID找到元素列表，把所有元素的text打印出来。

5）判断元素是否存在:

def is\_element\_present( how, what):

try:

self.driver.find\_element(by=how, value=what)

except NoSuchElementException:

return False

return True

#通过id is\_element\_present(By.ID,'id\_01') #通过name is\_element\_present(By.NAME,'name\_01') # 通过class\_name is\_element\_present(By.CLASS\_NAME,'class\_01') # 通过tag\_name is\_element\_present(By.TAG\_NAME,'a') # 通过

#注意，这里的how指使用的查找依据，what是查找内容

延伸知识：

APP的页面布局方式有：

1. 框架布局（FrameLayout）：所有控件都被放置在最左上的区域，下一个子控件会重复覆盖上一个控件
2. 线性布局（LinearLayout）：控件的排序方式：垂直/水平
3. 绝对布局（AbsoluteLayout）：采用从左到右，从上到下的坐标轴直接布局
4. 相对布局（RelativeLayout）：根据参照物（某控件）的位置，来确认该控件的位置
5. 表格布局（TableLayout）：通过表格方式（行列）布局控件位置

APP的页面控件分为：

1. 文本输入框（TextView）：展示文字/文本
2. 编辑框（EditText）：接收用户输入的数据
3. 图片控件（ImageView）：用来显示图片
4. 按钮（Button）：通过用户点击产生反应事件
5. 复选框（CheckBox）：复选/多选
6. 进度条（ProgressBar）：加载进度条。表示在加载中。

至此，你已经可以写一个简单的登录APP测试了，别的测试内容根据需求，编写测试用例，抽象成相应代码，所需要的库-函数等可以自行百度学习积累。

1. Web自动化测试（基于Selenuim）
2. 浏览器选择：

需要选择自动化的浏览器，因为每个浏览器的驱动不一样。用户通过驱动控制浏览器（通过代码发送Http协议）给服务器发送消息。所以要根据想要自动化的浏览器选择相应驱动。

1. 环境配置：安装相应版本的浏览器和浏览器驱动。安装selenuim库（这个库一般都装着有，没有就pip）
2. 元素查找（根据浏览器deverlop tool）重点！：
3. 代码部分：
4. from selenuim import webdriver#导入模块
5. Webdrive.Chrome(“路径”)#选择相应浏览器打开驱动，路径是驱动EXE路径

注意：这里Chrome和chrome不是一个模块别写错了！

1. Driver.get(参数)访问网址
2. Driver.fing\_element\_by\_id(这中元素返回方法的使用跟APP自动化测试中的一样，所以就不写了。）
3. CSS的使用(查找特定元素)：

在浏览器的调试模式中ctrl+f调出输入框，用#父元素 em[属性=要求],这样可以进行获取想要的拥有相应属性的特殊元素组。

用find\_elements\_by\_css\_selector(“CSS中输入的内容”)来找到放入数组中。

注：这种方法有个特别好的地方，如果你想找的特定元素在父元素下好几层，这个方法也能找到，而不会把一些其他没有相关属性的子元素放入数组中，同时也能找到多层以下的子元素。

再着，有些元素没有固定ID的时候（一般Web端上很多元素都没有固定ID，这代表着class\_name和ID这两种find方式都不好使，如果再没有tag的话，就必须使用这种方式），很明显在Web测试时，通过这种赛选方法可以精确指定单个元素。

至此，你已经可以从网页上，通过查找元素，判断元素而写一些简单的爬虫程序了,可以将网络元素放入EXCEL中进行保存等各种操作

1. 自动化环境部署（连接Linux系统，输入命令获取回显）
2. 环境安装：pycharm+Paramiko

pip install paramiko (如果连接不好可以-i连接克隆库)

1. 代码部分：
2. 必要连接过程：

*#创建SSHClient 实例对象*ssh=paramiko.SSHClient()  
  
*#信任远程机器，允许访问*ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())  
  
*#连接远程机器 地址 端口 用户名 密码*ssh.connect(**"端口号"**,22,**"用户名"**,**"密码"**)#其中账号密码等找管理员获取，或者自己搭建一个小服务器进行测试学习

1. ssh.exec\_command（“命令内容”），这是让给SSH输入代码内容，注意：密码输入不能用这个方法。同时这个函数的返回值是SSH数据合集，其中三个管道是{输入，输出，错误显示}，如果你给三个赋值之后可以分别通过.read().decode()函数获得其中内容

同时：我们获得回显要确定位子的时候可能会用到切片方法

A=Str.split(‘切片依据’)#这里依据一般是空格

B=[a for a in A if a!=’’]#这里的B是一个列表创建表达式,表示所有A中的飞空元素组，这里很重要的是用于定位特定回显（比如线程号等等）

注意：exec\_command()方法不能用于登录后的用户切换，因为他不进行交互，只会永久堵塞线程.要切换建议使用ssh.invoce\_shell()

3）关于paramiko库的用法可以参照文本:<https://blog.csdn.net/weixin_42252770/article/details/99697415>,

4）关于invoce\_shell()：

①介绍:invoke\_shell()方法，用于创建一个子shell线程，这样所有的操作都可以在该子shell中进行，su切换用户不受影响，但该方法没有exec\_command那种方便的ChannelFile对象，所有的标准输出和标准错误内容都通过invoke\_shell返回对象的recv方法来获取，每一次调用recv只会从上一次返回的地方开始返回，也没有直接获取命令退出状态的方法，除非自己编写相应的程序，在此不赘述

②使用：invoce.send(“shell指令+\n”)发送指令，

显示回显：

resp = s.recv(9999)#这个方法是获得一行回显中参数指定的量，（一般选择9999为了把这一行都全部回显）  
buff +=resp.decode(**'utf-8'**)这是指定回显叠加并且以utf-8解码格式输出为字符串，防止乱码  
print(buff)如果多行回显需要自己写回显命令，因为这个子shell中没有指定的方便回显函数。

所以，在不需要权限问题的情况下尽可能的不使用invoke\_shell，这种方式倾向于必须交互的时候使用。（如果是root直接登录，就可以免去这一麻烦）

5）关于文件的上传/下载：

SftpClient模块:

实例化：sftp=ssh.open\_sftp()

#需要注意，这里是通过连接的SSH用户进行开放sftp通道实例，也就是说如果用户没有上传文件权限，是不能进行上传的。同时，通过invoce\_shell()直接输入shell命令进行上传比较麻烦，所以使用自动化的时候尽量跟管理员上来开启各种权限。

上传:sftp.put(本地路径，上面路径)

下载:sftp.get

这里以防万一，留一下shell命令,方便使用shell子线程时候send使用

#### **1、上传本地文件到远程服务器**

scp /var/www/test.php [root@192.168.0.101:/var/www/](mailto:root@192.168.0.101:/var/www/)

把本机/var/www/目录下的test.php文件上传到192.168.0.101这台服务器上的/var/www/目录中

#### **2、从服务器上下载文件**

scp root@192.168.0.101:/var/www/test.txt /var/www/local\_dir（本地目录）

把192.168.0.101上的/var/www/test.txt 的文件下载到/var/www/local\_dir（本地目录）

#### **3、从服务器下载整个目录**

scp -r root@192.168.0.101:/var/www/test /var/www/

#### **4、上传目录到服务器**

scp -r test root@192.168.0.101:/var/www/

1. Python多线程操作(暂停学习)
2. 首先了解，多线程创建方法源自Threading.Thread,

所以在使用的时候我们常常定义一个自己的多线程类来继承Threading.Thread

myThread(threading.thread):

Def \_\_init\_\_(self,threadID,name,q):

Threading.Thread.\_\_init\_\_(self)#构造器构造时先调用父类的初始化方法，即是拥有父类的各种方法，比如运行.start()方法

Self.threadID=threadID#别的属性定义

1. 用一个类中的run():方法写好分配执行的代码之后，在调用的时候通过thread.start()进行线程开启。
2. 通过thread.append(thread0)把线程加入线程列表
3. 通过thread.join()等待进程完成，#如果不等待代码会直接结束从而杀死进程
4. 进程锁：首先你需要考虑你的多个进程里面，哪个是需要做什么工作的，需要哪些顺序，如果是进程1完成之后进程2才能执行这种情况可以用进程锁来实现。

具体方法就是在run（）方法内通过，threadLock.acquire()获得进程锁，在这之前得先定义threadLock=threading.Lock()实例

然后在run（）的最后添加threadLock.release()进行锁的释放，允许开启下一个线程。

这种方法的原理大概能理解为，如果进程运行到进程锁检查的时候如果已经有进程获取了锁，则会堵塞当前进程直到可以获得锁为止。所以在代码端上先启动的进程就会先获得锁，从而实现想要的顺序要求。

1. 优先级队列（Queue）

Python 的 Queue 模块中提供了同步的、线程安全的队列类，包括FIFO（先入先出)队列Queue，LIFO（后入先出）队列LifoQueue，和优先级队列 PriorityQueue。

这些队列都实现了锁原语，能够在多线程中直接使用，可以使用队列来实现线程间的同步。

Queue 模块中的常用方法:

* Queue.qsize() 返回队列的大小
* Queue.empty() 如果队列为空，返回True,反之False
* Queue.full() 如果队列满了，返回True,反之False
* Queue.full 与 maxsize 大小对应
* Queue.get([block[, timeout]])获取队列，timeout等待时间
* Queue.get\_nowait() 相当Queue.get(False)
* Queue.put(item) 写入队列，timeout等待时间
* Queue.put\_nowait(item) 相当Queue.put(item, False)
* Queue.task\_done() 在完成一项工作之后，Queue.task\_done()函数向任务已经完成的队列发送一个信号
* Queue.join() 实际上意味着等到队列为空，再执行别的操作

这里给一个代码例子帮助理解：

#!/usr/bin/python3  
  
**import** queue  
**import** threading  
**import** time  
  
exitFlag = 0  
  
**class** myThread (threading.Thread):  
    **def** \_\_init\_\_(self, threadID, name, q):  
        threading.Thread.\_\_init\_\_(self)  
        self.threadID = threadID  
        self.name = name  
        self.q = q  
    **def** run(self):  
        **print** ("开启线程：" + self.name)  
        process\_data(self.name, self.q)  
        **print** ("退出线程：" + self.name)  
  
**def** process\_data(threadName, q):  
    **while** **not** exitFlag:  
        queueLock.acquire()  
        **if** **not** workQueue.empty():  
            data = q.get()  
            queueLock.release()  
            **print** ("%s processing %s" % (threadName, data))  
        **else**:  
            queueLock.release()  
        time.sleep(1)  
  
threadList = ["Thread-1", "Thread-2", "Thread-3"]  
nameList = ["One", "Two", "Three", "Four", "Five"]  
queueLock = threading.Lock()  
workQueue = queue.Queue(10)  
threads = []  
threadID = 1  
  
# 创建新线程  
**for** tName **in** threadList:  
    thread = myThread(threadID, tName, workQueue)  
    thread.start()  
    threads.append(thread)  
    threadID += 1  
  
# 填充队列  
queueLock.acquire()  
**for** word **in** nameList:  
    workQueue.put(word)  
queueLock.release()  
  
# 等待队列清空  
**while** **not** workQueue.empty():  
    **pass**  
  
# 通知线程是时候退出  
exitFlag = 1  
  
# 等待所有线程完成  
**for** t **in** threads:  
    t.join()  
**print** ("退出主线程")