目 录

(五十三期·下)

安卓 Appium 自动化测试实践.....01

自动化进化论之“完结篇”.....13

如何证明你是一名优秀的测试人员.....29

通过 Java API 像 MySQL 一样查询 HBASE.....32

Appium+Python 实现 APP 启动页跳转到首页.....46

自动化测试前 ,你需要知道的 10 点.....55

如何利用 TestNG 做接口自动化测试？.....58

如何并行运行你的自动化测试.....63

影响软件测试未来的 5 件事.....68

如果您也想分享您的测试历经和学习心得，欢迎加入我们~(*^▽^*)

投稿邮箱：editor@51testing.com

安卓 Appium 自动化测试实践

◆ 作者：枫 叶

一、软件安装

1.安装 node.js

安装路径 D:\Program Files\nodejs\

可以在官网下载 https://nodejs.org/zh-cn/download/，版本号为 node-v8.12.0-x64

2.用 node 的 npm 安装 appium

npm install –g appium

官网介绍此种安装慢；亲测至少需要 15 分钟，果断中止。

可以在 https://bitbucket.org/appium/appium.app/downloads/ 下载安装包，目前最新包 AppiumForWindows_1_4_16_1，没再支持更新，如果需要有 Appium desktop 版本，目前最新版是 V1.11.0

3.安装 Java JDK

安装 jdk1.8.0_181，jdk 和 jre 的下载地址:
https://www.oracle.com/technetwork/java/javase/downloads/jre10-downloads-4417026.html;
https://www.oracle.com/technetwork/java/javase/downloads/jdk10-downloads-4416644.html

安装 jdk，如下图所示

安装 jre，注意：在安装 android-sdk 之前，否则 android-sdk 将无法安装

设置环境变量:

“我的电脑”右键菜单--->属性--->高级--->环境变量--->系统变量-->新建..

变量名：JAVA_HOME

变量值：D:\Program Files\Java\jdk1.8.0_181

变量名：CALSS_PATH

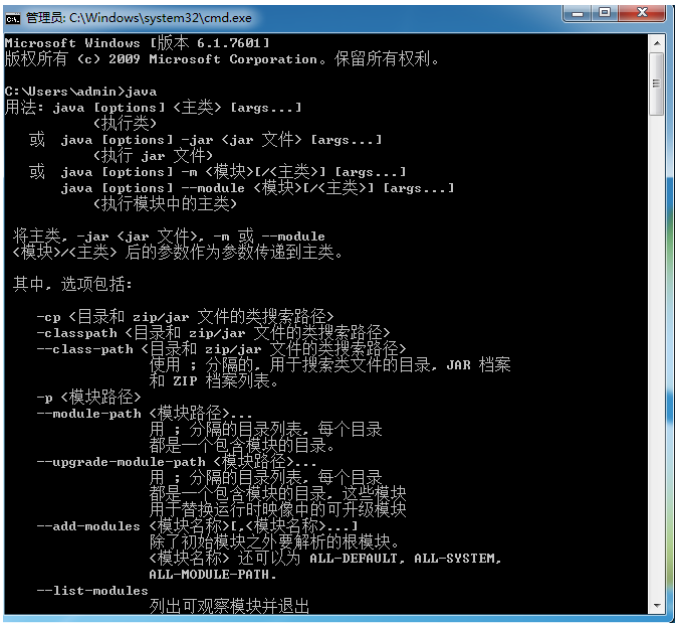
变量值：.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;

找到 path 变量名—>“编辑”，添加:

变量名：PATH

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

在 Windows 命令提示符下验证 java 是否成功：C:\Users\admin>java

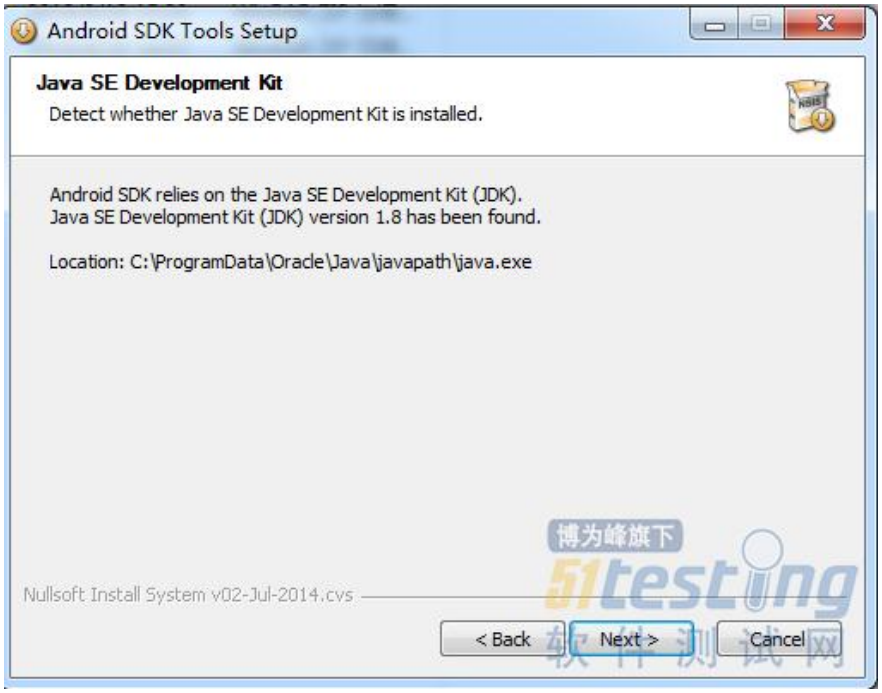


说明 java 环境安装成功。

4. 安装 Android SDK

这是 Android 开发所需的 sdk，本项目中下载的是 android5.0，下载地址

https://pan.baidu.com/s/1i33Puo1?utm_source=androiddevtools&utm_medium=website，如下图进行安装。



设置环境变量:

变量名：ANDROID_HOME

变量值：D:\Program Files (x86)\Android\android-sdk

变量名：PATH

变量值：;%ANDROID_HOME%\platform-tools;%ANDROID_HOME%\tools;

5.安装 SDK platform-Tools

http://www.androiddevtools.cn/ 网站上找到 Android SDK Platform-tools 下载链接，下载并安装。

设置环境变量：把解压出来的 platform-tools 文件夹放在 android sdk 根目录下，并把 platform-tools 文件夹中的 adb 所在的目录添加到系统 PATH 路径里。

6.安装（拷贝）apache-ant

下载 Ant 的网址 https://ant.apache.org/bindownload.cgi，然后将 Ant 的文件夹目录放到 path 变量中

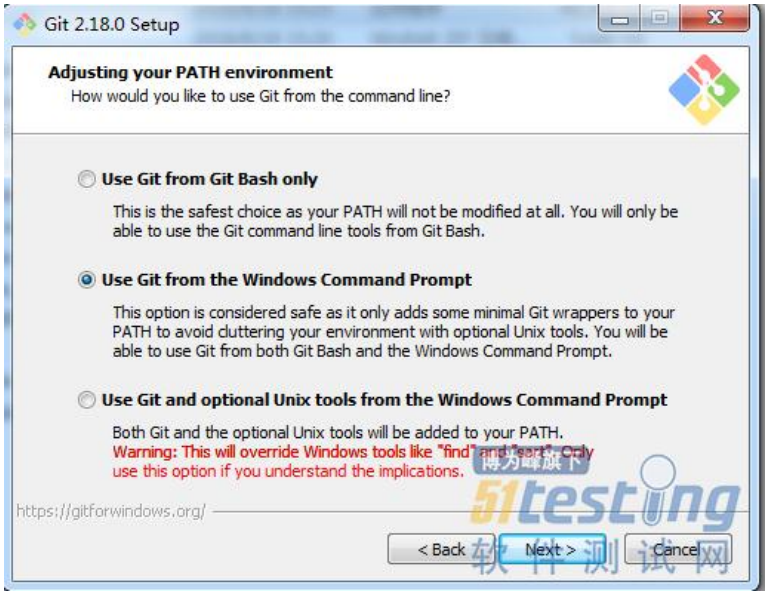
7.安装 Apache Maven

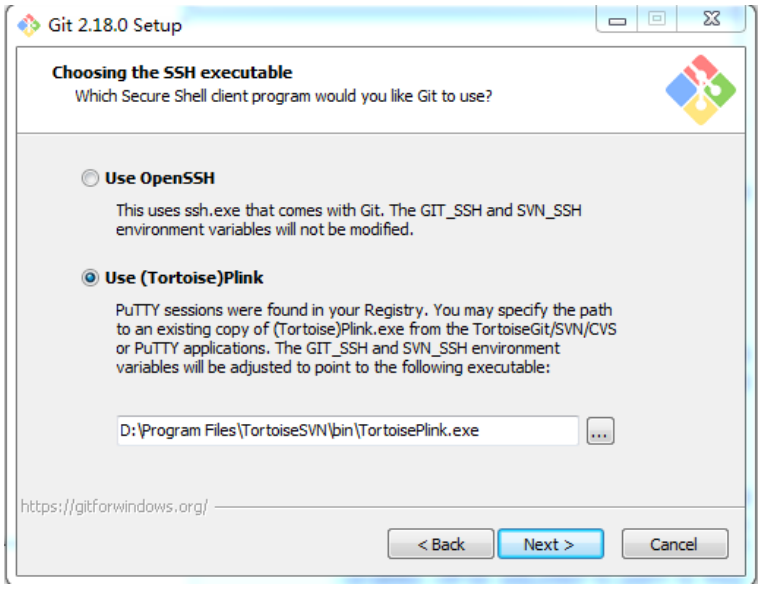
下载网址 http://maven.apache.org/download.cgi，设置 M2HOME 和 M2 环境变量，

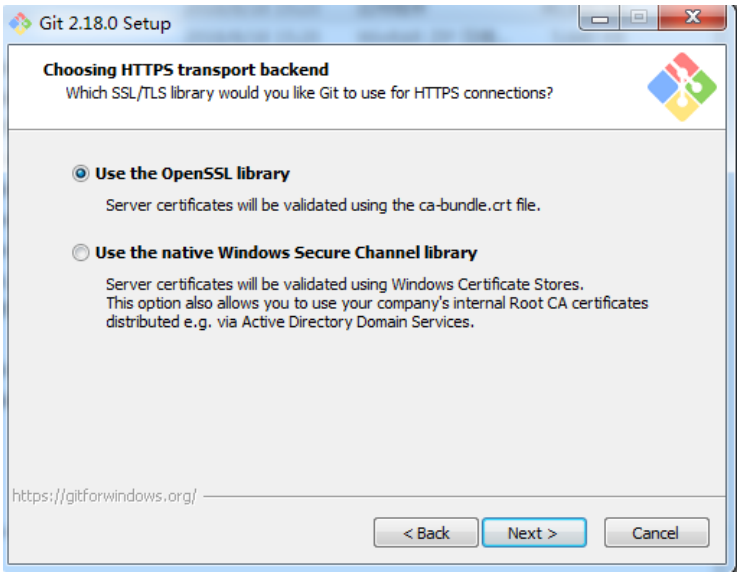
M2HOME 设为 Maven 安装目录，M2 设为%M2HOME\bin;

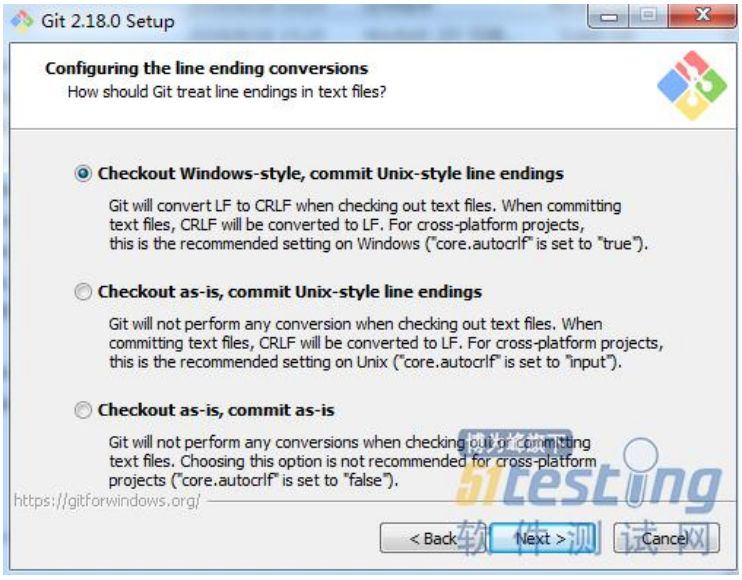
8.安装 Git

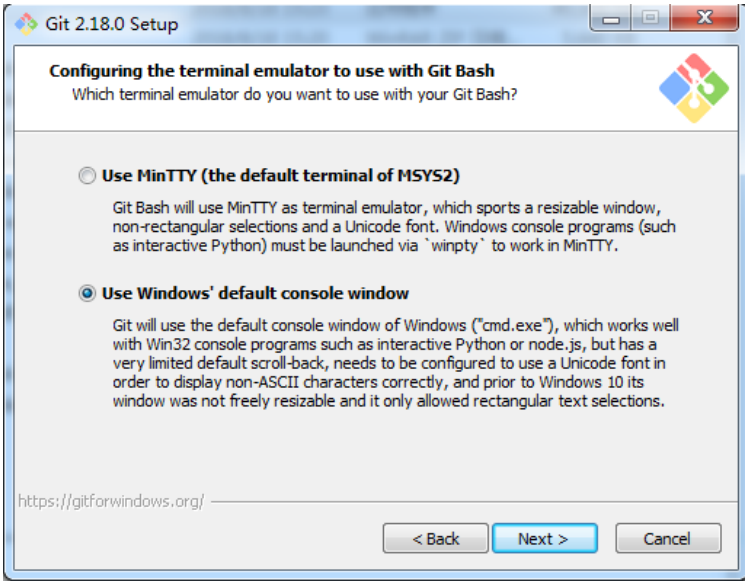
下载网址 https://git-scm.com/downloads，如下图进行安装，选择默认选项

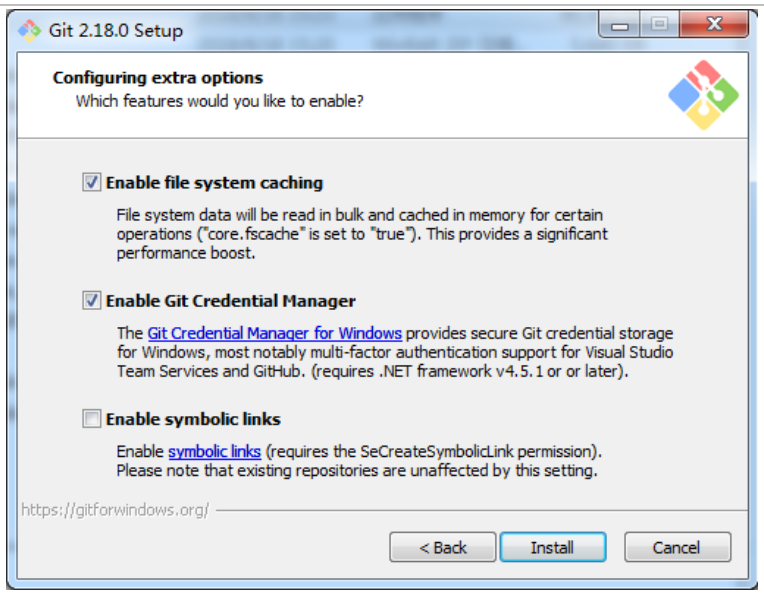





Git 2.18.0 Setup
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?
Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates
distributed e.g. via Active Directory Domain Services.
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.18.0 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing
text files, CRLF will be converted to LF. For cross-platform projects,
this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When
committing text files, CRLF will be converted to LF. For cross-platform projects,
this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing
text files. Choosing this option is not recommended for cross-platform
projects ("core.autocrlf" is set to "false").
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.18.0 Setup
Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable window,
non-rectangular selections and a Unicode font. Windows console programs (such
as interactive Python) must be launched via `winpty` to work in MinTTY.
Use Windows' default console window
Git will use the default console window of Windows ("cmd.exe"), which works well
with Win32 console programs such as interactive Python or node.js, but has a
very limited default scroll-back, needs to be configured to use a Unicode font in
order to display non-ASCII characters correctly, and prior to Windows 10 its
window was not freely resizable and it only allowed rectangular text selections.
https://gitforwindows.org/
< Back
Next >
Cancel

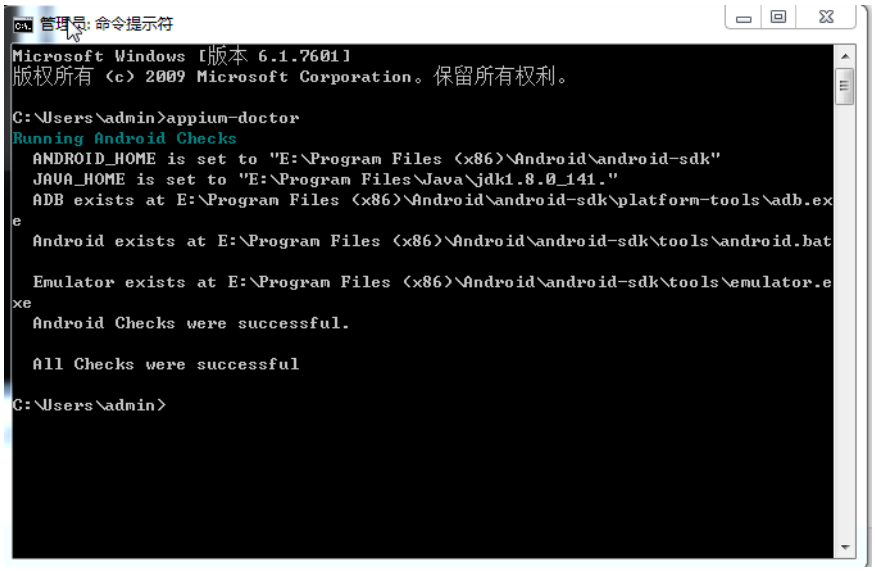


9.安装（拷贝）cURL

下载网址 https://curl.haxx.se/download.html，文件夹拷贝到电脑相应目录。

10.安装 appium

下载版本 1.4.16，安装后，打开 Windows 命令提示符，通过“appium-doctor”命令检查 appium 环境，出现下图所示，说明安装成功。



11.安装 android adt

https://www.cnblogs.com/fnng/p/4552438.html 提供下载链接，或者直接下载安装包

https://dl.google.com/android/adt/adt-bundle-windows-x86_64-20140702.zip

12. 安装 android 模拟器

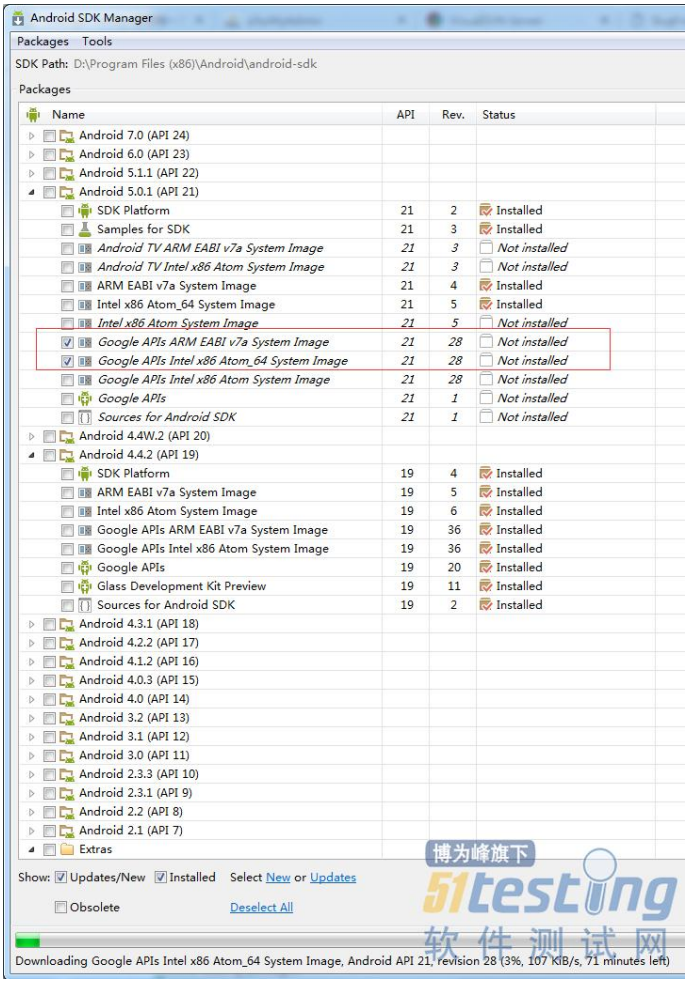
https://pan.baidu.com/s/1pJLULXh 这是 Android 开发所需的 sdk，下载并解压后，将解压出的整个文件夹复制或者移动到 .../android-sdk-windows/platforms/文件夹，然后重新打开 SDK Manager.exe

13. 安装 Samples for SDK

https://pan.baidu.com/s/1dDD19XB?utm_source=androiddevtools&utm_medium=website 下载，并放到...\Android\android-sdk\samples 文件夹下

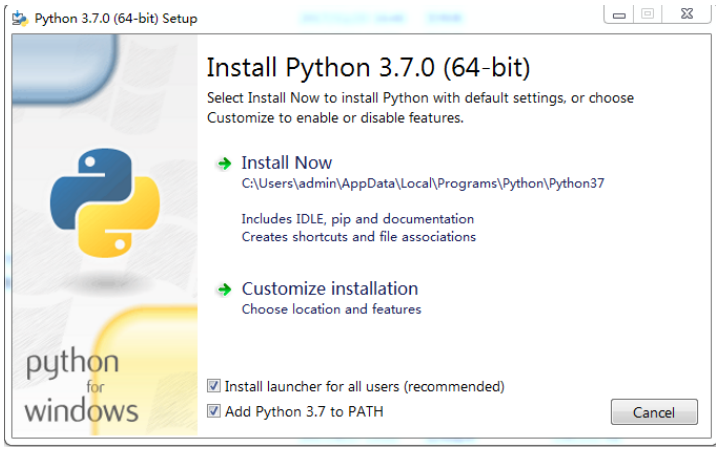
14. 安装 SDK System images

15. 安装 GoogleMap APIs SDK



为保险起见，预防日后用到而没有安装，这里把 GoogleMap APIs SDK、Android Framework Source Code 全部做了安装。

16. 安装 python3.7.0



17. 安装 Appium-Python-Client，添加 python 进系统变量

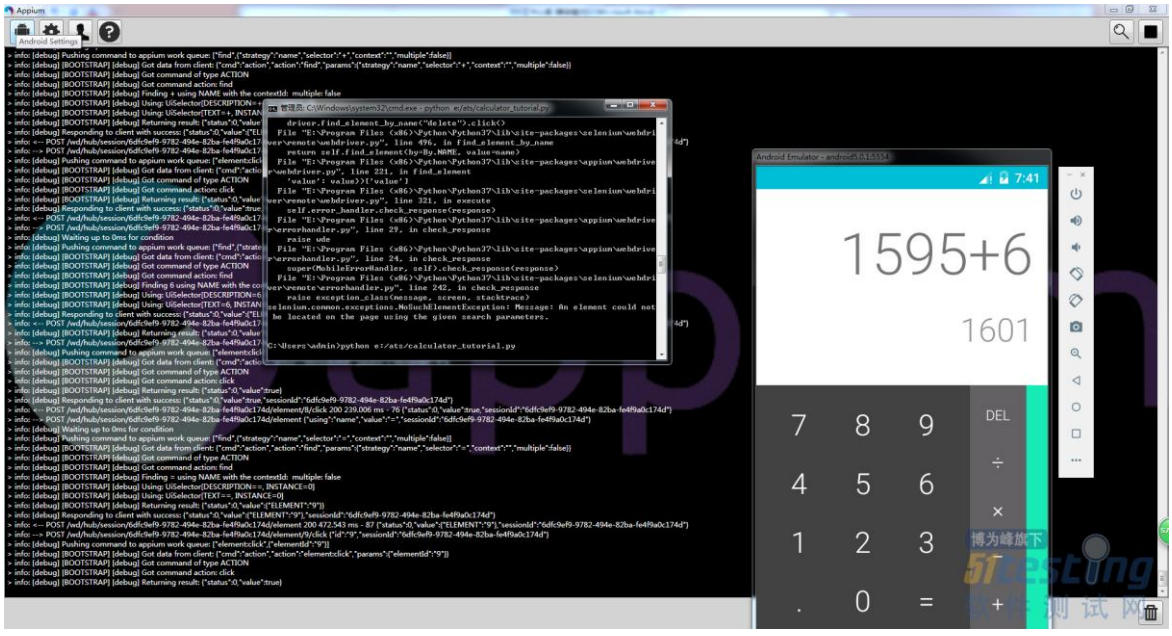
18. 安装 android studio

目前没有用到，同样地为防止日后使用，这里做了安装。一路默认 next，直到 finish。

二、启动测试

模拟器启动

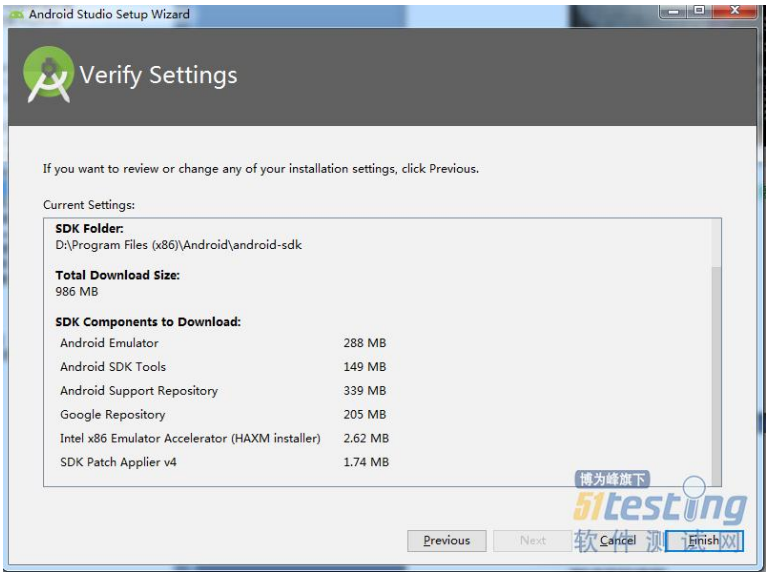
启动 AVD，注意模拟器的启动顺序：1）启动 AVD　2）启动 appium　3）执行 py 脚本



2. 真机启动

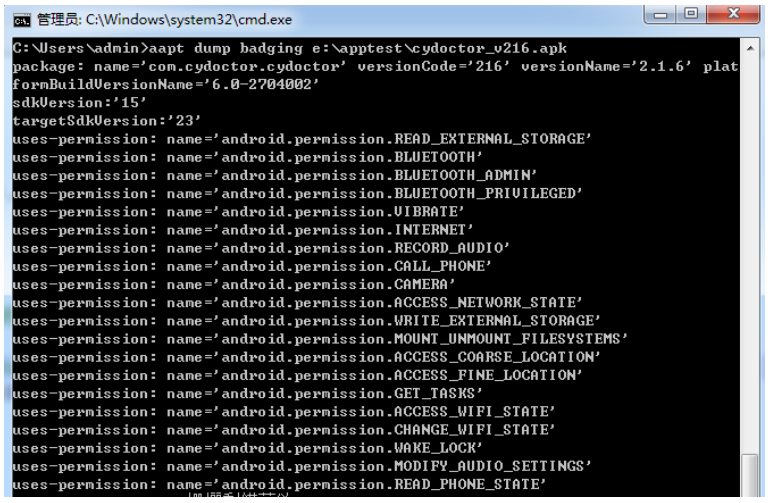
如果有真机最好，注意启动顺序：1） 连接真机 2）启动 appium 3）执行 python 脚本；

在连接真机时需要开启开发者调试模式。



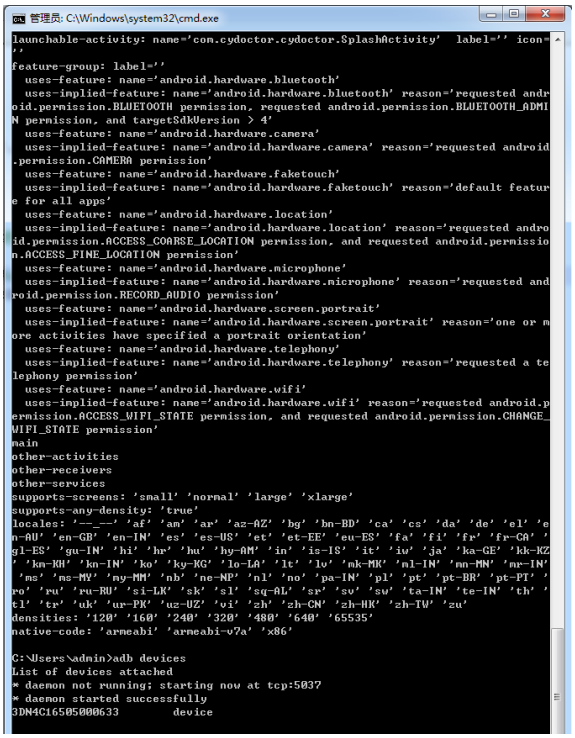
3. 启动 app 的准备

1）双击安卓 SDK-manager ，下载 build-tools 后，在 android-sdk-windows\build-tools\25.0.2 目录下找到 aapt.exe,并将该路径设置环境变量，添加到系统的环境变量 path 下；重启后在 cmd 中输入 appt，如下图即环境安装成功。



2）获取 apk 包名，获取 laucher activity

命令：aapt dump badging e:\apptest***.apk



4. 编写 python 脚本

可以在任意编辑器如 nodepad++上写 python 脚本，也可以使用 PyCharm，推荐后者，简单好用，方便调试

5. 电脑连接手机，并确认已连接，打开开发者调试模式

确认手机连接，cmd 中输入命令 adb device

6. 启动 appium

7. 可以在 cmd 执行 python 脚本，命令 python e:/ats/***.py

或者 PyCharm 中 Ctrl+F5 运行。

三、附部分代码

```
import os, time, unittest
#导入 HTMLTestRunner 报告模板模块
from HTMLTestRunner import HTMLTestRunner
from appium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import InvalidArgumentException
```

```
PATH = lambda p:os.path.abspath(os.path.join(os.path.dirname(__file__),p))
#定义一个空字典
desired_caps = {}
desired_caps = {'platformName':'Android','platformVersion':'6.0','deviceName':'HUAWEI VNS-AL00',
'app':PATH(r"E:\apptest\******.apk"),'appPackage':'com. ******.******', 'appActivity': 'com.
******.******.SplashActivity','unicodeKeyboard':True,
'resetKeyboard':True}   #unicode 编码方式发送字符串 #隐藏软键盘
driver = webdriver.Remote("http://localhost:4723/wd/hub", desired_caps) #获取 Appium client 端口值
time.sleep(5)
……
```

虽然有些步骤可能不是必须的，但是无论如何，通过上面的软件安装步骤可以把安卓 appium 自动化测试的环境搭起来，并实现安卓第一个脚本运行。

自动化进化论之“完结篇”

◆作者：测试女巫

终于到了这一系列的最后一期，每一次真的比较纠结，因为工作实在太忙了，每一次都纠结是不是要放弃，每一次又坚持了下来，为自己打个 Call，其实这个系列真的非常有意义，记录了我从只会作黑盒测试的正宗黑盒测试人员，慢慢的有了具备开发协助黑盒测试的自动化工具，这个工具从被别人鄙视“搞什么搞”的低级阶段渐渐发展成有班长，成规模，让别人觉得“有点样子”的中级阶段，再到这次真的有了界面，一个真的有模有样，让别人可以另眼相看的阶段，这个期间的痛并快乐着的感受，真的只有经历的人才会知道。但是大家千万不要以为这个完结篇就代表了女巫目前的工作 level，这个“完结篇”是 2016 年的“完结篇”，因为从 2016 年做了这个完结篇，真的主管才真的重视我们的自动化工具，然后从 2017 年开始，真的黑盒测试任务渐渐变少，到 2019 年真的几乎已经没有了黑盒测试任务，自动化工作反而成了工作的重点，所以真觉得还挺神奇的，几年前的非本职工作，被别人在质疑搞什么搞得自动化，现在变成了主业，真的很多时候你不去努力一把，你不知道自己有多厉害哈哈。

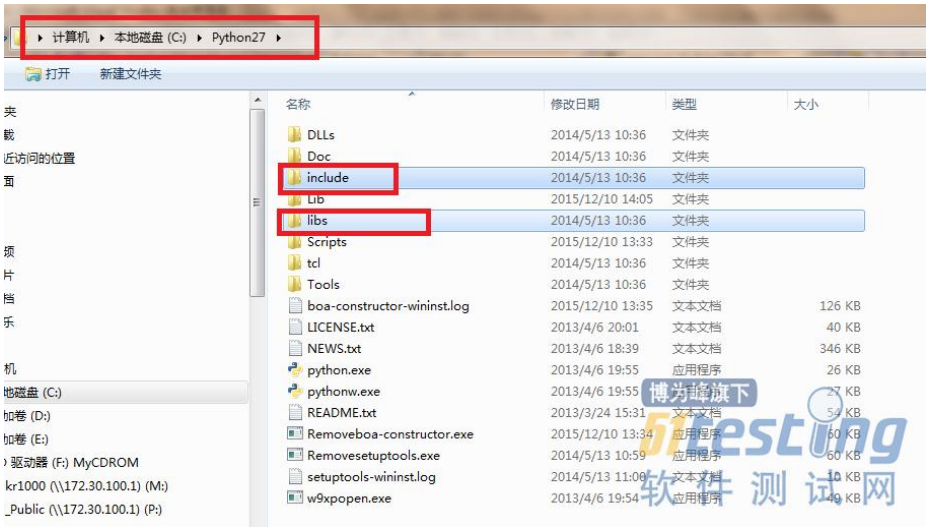
好吧，心路历程总结完毕，我们真的可以开始介绍我们的工具，我们的工具

比较厉害，有了界面，这个界面是用 C++实现，我们是在 C++中调用 Python 脚本的方式来实现我们的自动化

一、如何配置 C++与 Python 结合编程的编程环境

1.Python 的资料 Copy

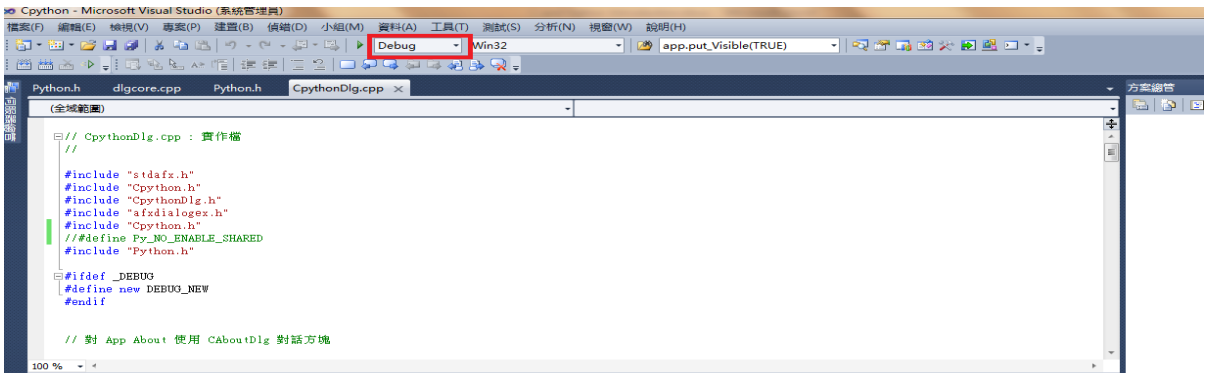
首先将 Python 安装目录下的 Libs 和 include 文件 copy 下来，如下图



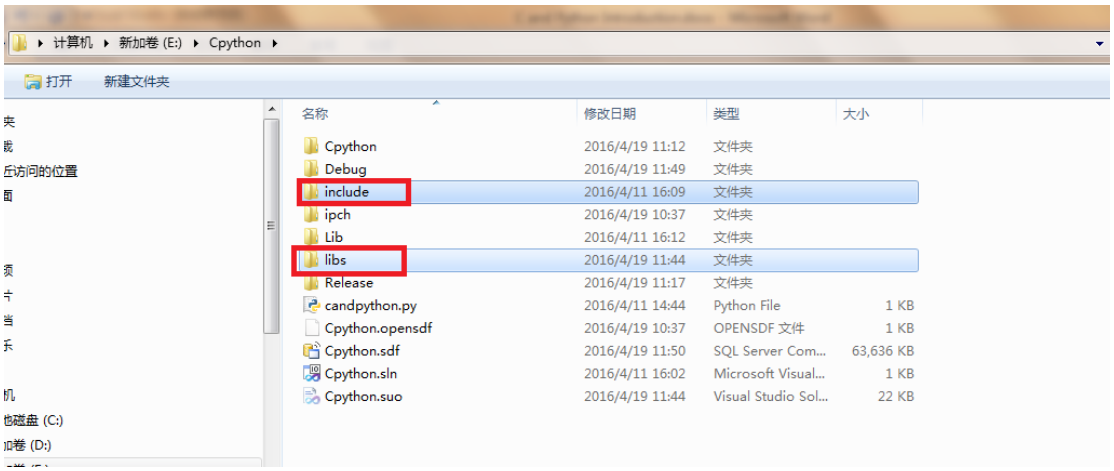
2.VS 端的设置

VS 需要创建一个 MFC 的项目，这里就不再赘述，之前的 C++已经叙述过如何创建，创建完毕请注意

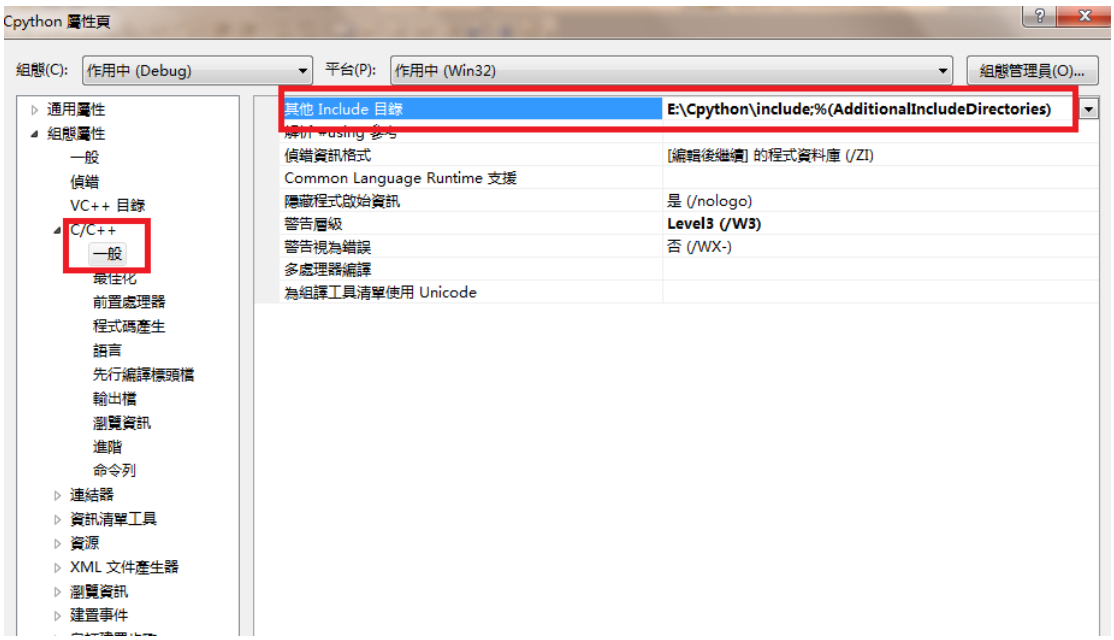
1）运行的模式如果选择 debug Mode 如下图



a.需要将 Python 的 Include 和 Libs copy 到我们建立的 C++ project 的路径下，如下图

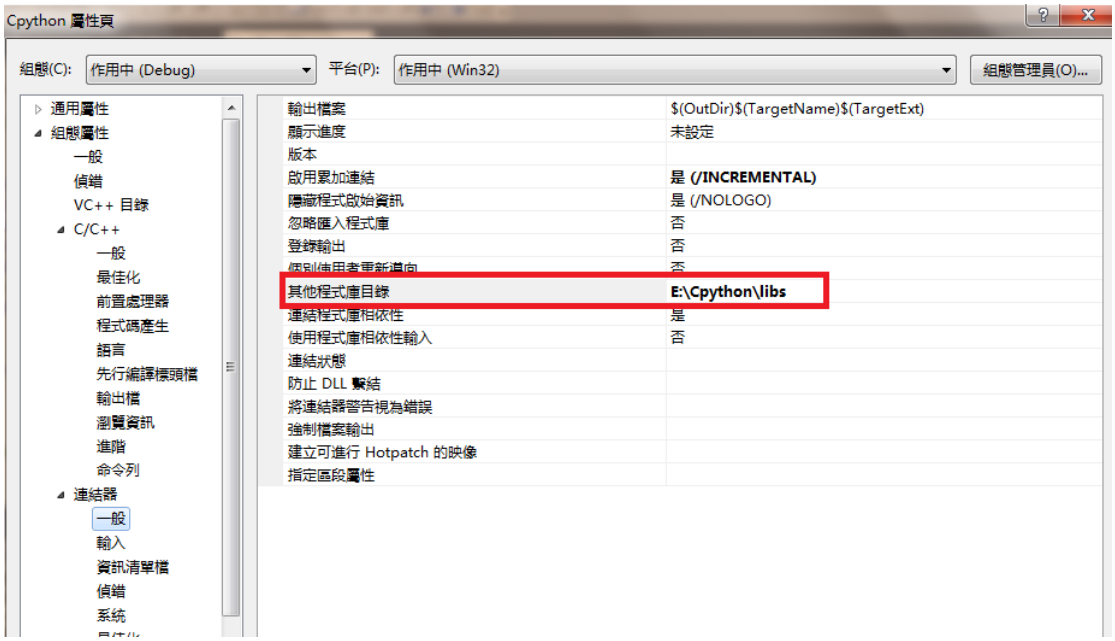


b.Include 文档的配置，选择项目->属性->C/C++的”一般”->其它 include 目录选择 a 中 copy 的 include 所在的路径



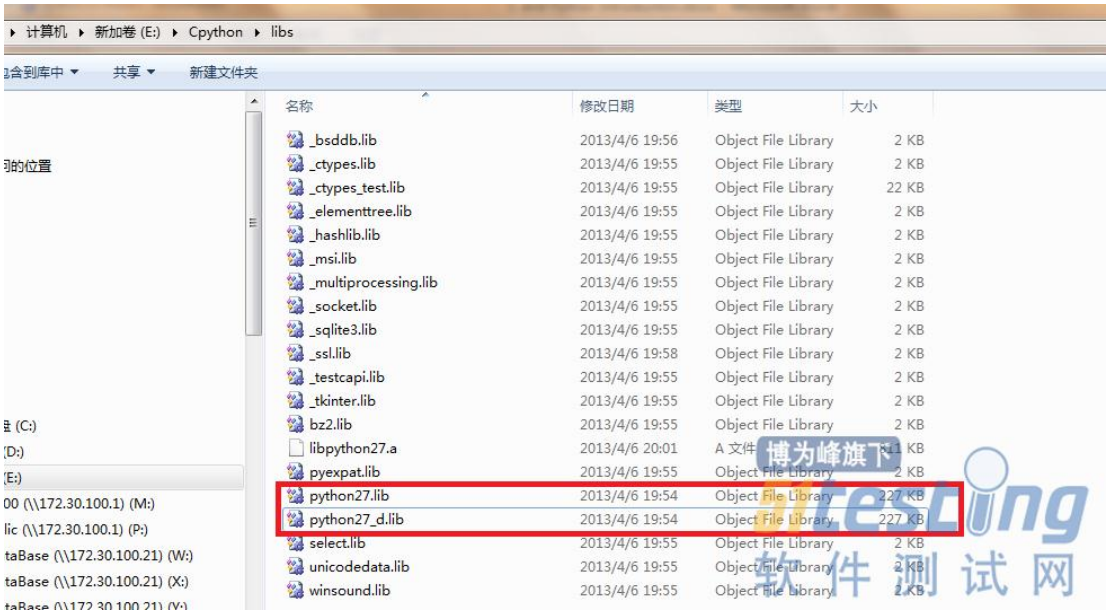
c.Libs 配置依赖项路径

选择项目->属性->连接器->一般->其它程式库目录，如下图:

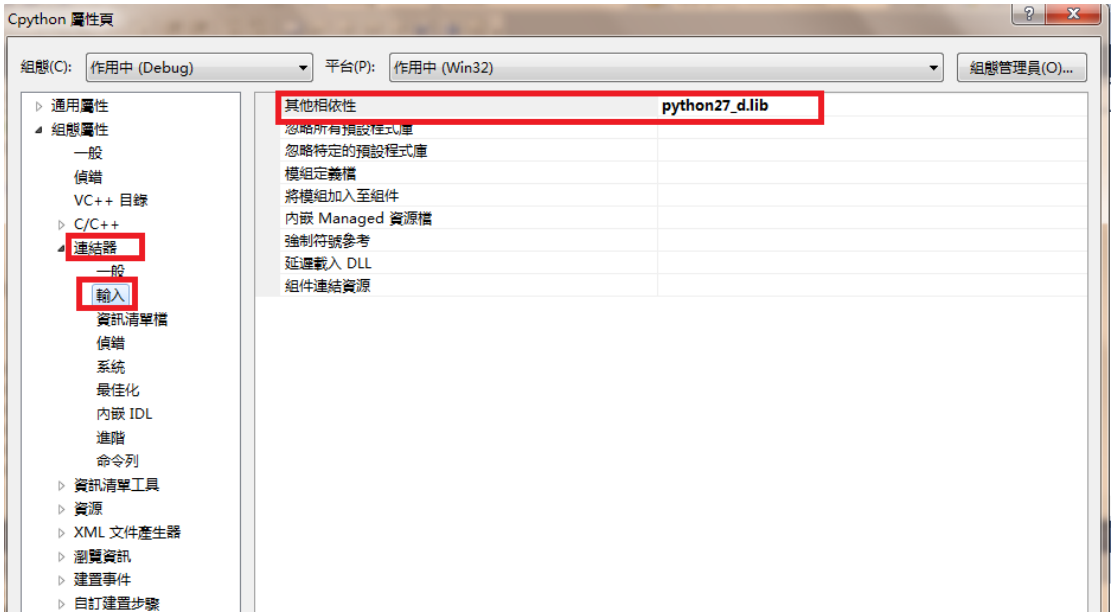


d.在 Debug 中运行，需要设置依赖项

首先需要在 libs 文件夹下的 Python27.lib 复制一个，并将其名称改为 python27_d.lib，如下图

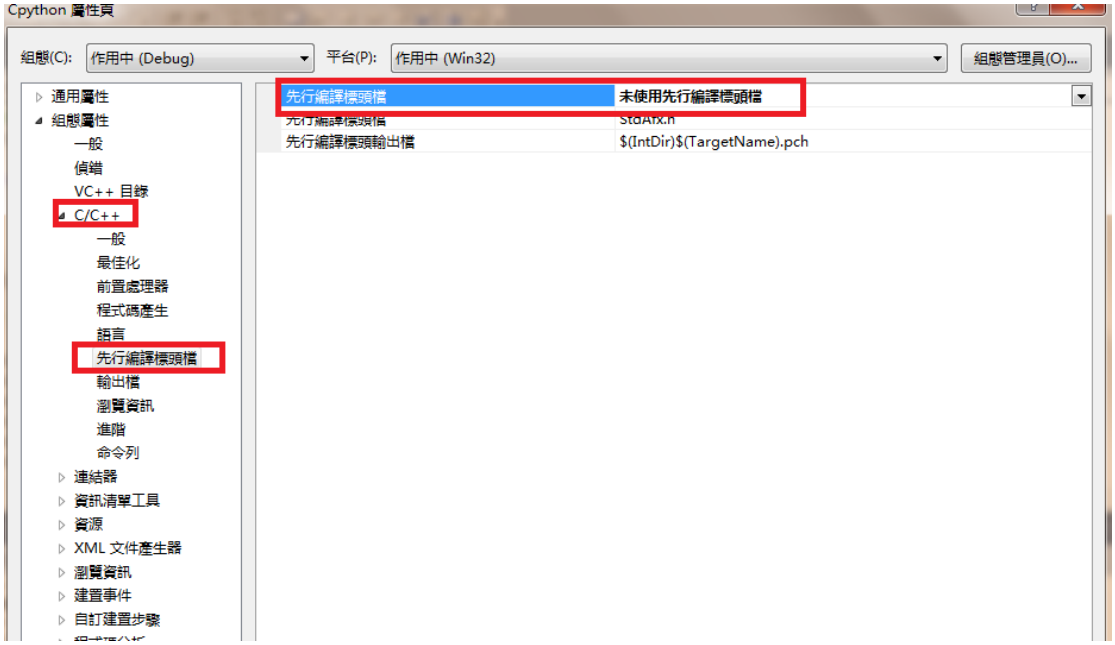


然后到选择项目->属性->连接器->输入路径下，将 python27_d.lib 设置到“其它相依性”此栏位中，如下图



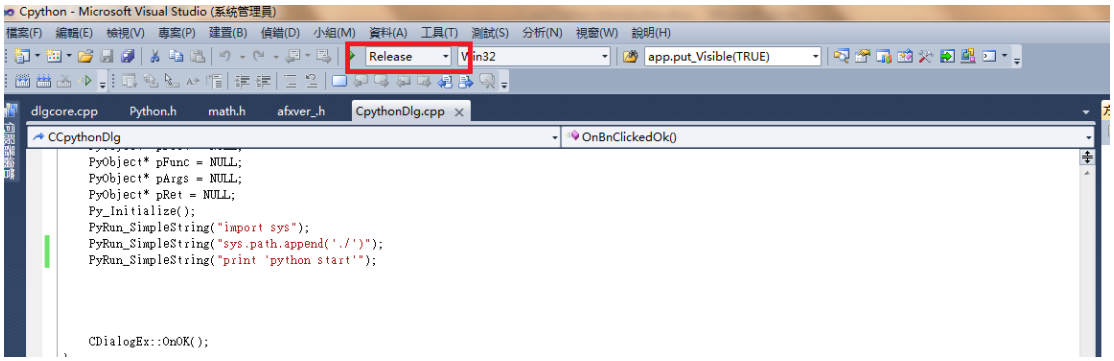
e.配置“先行编译标头档”

需要选择“未使用先行编译表头档”如下图:



2）运行模式如果选择 Release Mode 如下图

除了不需要设置依赖项即步骤 d，其它与选择 Debug 一样



3）例子说明:

下面的例子即为可以编译成功的例子

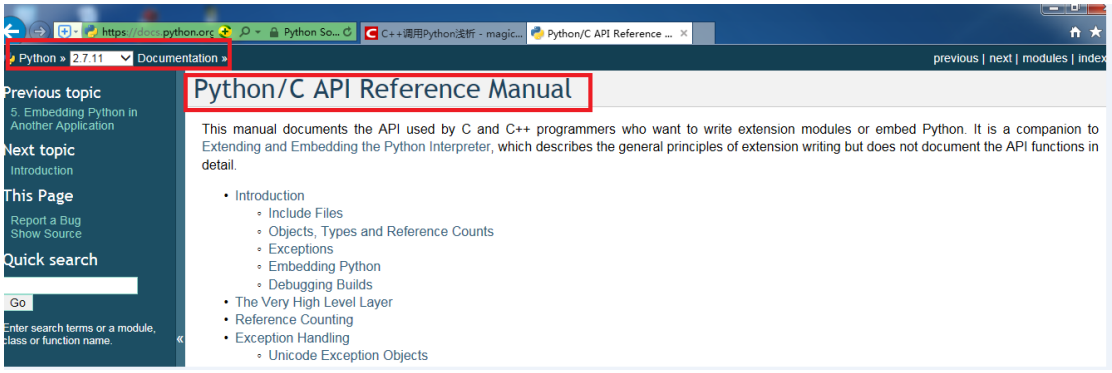
```
void CCpythonDlg::OnBnClickedOk()
{
    // TODO: 在此加入控制項告知處理常式程式碼

    PyObject* pName = NULL;
    PyObject* pModule = NULL;
    PyObject* pDict = NULL;
    PyObject* pFunc = NULL;
    PyObject* pArgs = NULL;
    PyObject* pRet = NULL;
    Py_Initialize();
    PyRun_SimpleString("import sys");
    PyRun_SimpleString("sys.path.append('./')");
    PyRun_SimpleString("print 'python start'");
    CDialogEx::OnOK();
}
```

二、Python C++ API 的学习

1.资料来源：地址

Python 的官网：Https://docs.python.org/2/c-api/index.html

注意选择的是 Python 2.7.11 这个与我们安装的 Python 的版本一样：我们安装的就是 Python27。如【图 1】



【图 1】

2.具体的 Topic

Python/C API Reference Manual

此 Topic 中一共有 10 个分类，建议不要每个分类都看，因为其中的信息量太大了，可以以实现的功能为中心，选择性的学习这些接口函数。

10 个分类分别是：Introduction；The very High Level Layer；Reference Counting；Exception Handing；Utilities；Abstract；Concrete Objects Layer；Initialization，Finalization，

3.常用接口函数

1）Initialization，Finalization and Threads

✓ **Void Py_Initialize (void)**

初始化 Python 解释器，在一个内嵌 Python 的应用软件中，在调用其它的 API 函数前，必须要调用此函数。此函数没有参数，没有返回值，且如果初始化失败就会出现致命的失败。

✓ **Int Py_IsInitalized()**

判断是否初始化成功，如果成功返回值是 True(非 0)否则为 False(0)；此函数没有参数。

✓ **Py_Finalize()**

取消所有的 Py_Initialize()此函数做的初始化，且摧毁所有已经做过的初始化。此函数没有参数，没有返回值

2）The very High Level Layer

此分类主要是执行一个文档或者缓冲区中的源代码

✓ **PyRun_SimpleString(const char *command)**

它是 PyRun_SimpleStringFlags(const char *command PyCompiler *flags)的简化版，即只需要提供一个参数即可，这句命令的意思是执行 Python 中的一段代码。

✓ **PyObject *PyEval_CallObject(PyObject*pfunc，PyObject*pargs)**

第一个参数是 Module 中需要调用的函数，是通过此函数获得 PyObject_GetAttrString（此函数在下面会讲到）pargs 是函数的参数列表，通常 Py_BuildValue(char*format，……)来构建。

✓ **Int PyRun_AnyFile(FILE *fp， const char *filename)**

该 API 的作用是运行名称为 filename 的 python 脚本

3）Concrete Objects Layer

这个分类主要是明确特定的某些 Python 的对象类型，传送一个对象必须需要搞清楚它是什么类型，同样如果你接收一个对象必须知道这个对象是什么类型的，如果不知道首先需要 check 它是什么类型的，例如如果希望确认这个对象是否为字典类型，则需要使用 PyDict_Check()这个函数。这个章节就是讲述 Python 对象类型的结构图

Python 对象类型主要分为基本的对象，数字类型对象，序列类型对象，映射类型对象，其它对象

✓ **序列类型对象中的 String/Bytes 对象类型**

这些对象包含的函数，如果获得的参数不是 string 类型的参数，则会返回一个 TypeError 的错误。

✓ **PyObject* PyString_FromString(const char *v)**

将参数中的字串常量转成一个字串对象。

✓ **Module Objects**

PyObject * PyModule_GetDict(PyObject *module)

返回一个字典类型的对象，此对象包含模块名称。此对象与模块对象的__dict__属性是相同的。这个函数不会 fail，如没有会返回

✓ **File Objects**

PyObject* PyFile_FromString(char *filename，char *mode)

返回一个打开的文件类型的 Python 对象，该对象的打开方式由 mode 指定，mode 的取值同 C 语言的函数 fopen():

mode有以下几种方式：

打开方式	说明
r	以只读方式打开文件，该文件必须存在。
r+	以读/写方式打开文件，该文件必须存在。
rb+	以读/写方式打开一个二进制文件，只允许读/写数据。
rt+	以读/写方式打开一个文本文件，允许读和写。
w	打开只写文件，若文件存在则长度清为0，即该文件内容消失，若不存在则创建该文件。
w+	打开可读/写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。
a	以附加的方式打开只写文件。若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留（EOF符保留）。
a+	以附加方式打开可读/写的文件。若文件不存在，则会建立该文件，如果文件存在，则写入的数据会被加到文件尾后，即文件原先的内容会被保留（原来的EOF符 不保留）。
wb	以只写方式打开或新建一个二进制文件，只允许写数据。
wb+	以读/写方式打开或建立一个二进制文件，允许读和写。
wt+	以读/写方式打开或建立一个文本文件，允许读写。
at+	以读/写方式打开一个文本文件，允许读或在文本末追加数据。
ab+	以读/写方式打开一个二进制文件，允许读或在文件末追加数据。

FILE* PyFile_AsFile(PyObject *p)

将与 p 相关联的文件对象作为 FILE 指针

4）Import Modules

✓ **PyObject *PyImport_ImportModule(const char *name)**

导入一个 Python 模块，参数 Name 可以是“.py”文件的文件名，类似 Python 内建函数 Import

注意：参数是 const char 类型，即调用时直接写 PyImport_ImportModule("candpython")

即可，candpython 就是需要调用的模块的名称即“.py”文件的文件名称。

✓ **PyObject*PyImport_Import(PyObject *name)**

作用与上面函数一样，但是注意其需要传进来的参数是 PyObject 类型，不是简单的 char 类型，所以需要将模块名称使用

PyString_FromString(const char *v)转换成 PyObject 类型才可以使用这个函数，这两个函数 Mary 均做过实现，可以实现相同功能

5）Object Protocol

✓ **PyObject *PyObject_GetAttrString(PyObject*o， const char *attr_name)**

两个参数，第一个参数是导入“. py ”的 module 对象，第二个参数是此 module 对象中的字串，这个函数的意义是获得此 Module 中的函数名称字串，并将此字串转成 Object 类型。

✓ **Int PyObject_HasAttr(PyObject *o，PyObject *attr_name)**

两个参数，第一个参数是导入“. py ”的 module 对象，第二个参数是此 module 对象中的字串，这个函数的意义是确认 Module 中是否有 attr_name 此字串

注意：这个函数传进去的第二个参数为 Object 所以如果使用这个函数，需要先将字串类型转成 object 类型。

如果能找到，返回值为 1 否则为 0，注意无论是否能找到，这个函数总是执行成功的，不会有错误的现象产生。

6）Int PyObject_HasAttrstring(PyObject*o， const char *attr_name)

与 2)都一样，唯一不同的是传入的第二个参数是 Const char 类型，不需要将其转成 Object。

三、代码例子

1、不需要传入参数的简单 Python 内嵌例子

```
void CCpythonDlg::OnBnClickedOk()
{
    PyObject* pName = NULL;
    PyObject* pModule = NULL;
    PyObject* pDict = NULL;
    PyObject* pFunc = NULL;
    PyObject* pArgs = NULL;
    PyObject* pRet = NULL;
    Py_Initialize();
    PyRun_SimpleString("import sys");
    PyRun_SimpleString("sys.path.append('./')");
    pName=PyString_FromString("candpython");
    pModule =PyImport_ImportModule("candpython");
    pFunc=PyObject_GetAttrString(pModule,"getlog");
    PyEval_CallObject(pFunc,NULL);
    Py_Finalize();
    CDialogEx::OnOK();
}
```

说明:

1）初始化 Py_Initialize();

2）直接运行 Python 的代码，这段代码的意思就是添加当前的路径

PyRun_SimpleString(“import sys”)

PyRun_SimpleString("sys.path.append('./')");

3）载入一个名字叫 candpython 的脚本

pName=PyString_FromString("candpython");

pModule =PyImport_ImportModule("candpython");

4）运行 candpython 的脚本中的函数

pFunc=PyObject_GetAttrString(pModule，"getlog");

PyEval_CallObject(pFunc，NULL);

5）结束初始化 Py_Finalize();

2.需要传入参数的 Python 内嵌例子之较为复杂的实现

与不需要传入参数一样的内容不再赘述。

1) 首先创建一个元组类型，元组的大小是 2

2) 将参数从 C 可以理解的内容转成 Python 可以读懂的内容

3) 将此两个参数插入到一个元组对象中

4) 调用此函数以及函数的参数即可

```
void CCpythonDlg::OnBnClickedOk()
{
    PyObject* pName = NULL;
    PyObject* pModule = NULL;
    PyObject* pDict = NULL;
    PyObject* pFunc = NULL;
    PyObject* pArgs = NULL;
    PyObject* pRet = NULL;
    Py_Initialize();
    PyRun_SimpleString("import sys");
    PyRun_SimpleString("sys.path.append('./')");
    pName=PyString_FromString("candpython");
    pModule =PyImport_ImportModule("candpython");
//  The first method _has parameters
 //  pFunc=PyObject_GetAttrString(pModule,"getlog");  //no paramters
 //  pArgs=Py_BuildValue("ii",12,14);
 //  pFunc=PyObject_GetAttrString(pModule,"add");
 //  PyEval_CallObject(pFunc,pArgs);

// Another method
    PyObject *pyParames=PyTuple_New(2);
    PyObject *pyParames1=Py_BuildValue("i",5);
    PyObject *pyParames2=Py_BuildValue("i",6);
    PyTuple_SetItem(pyParames,0,pyParames1);
    PyTuple_SetItem(pyParames,1,pyParames2);
    pFunc=PyObject_GetAttrString(pModule,"add");
    pRet=PyEval_CallObject(pFunc,pyParames);

    Py_Finalize();
    CDialogEx::OnOK();
}
```

3.需要传入参数的 Python 内嵌例子之较为简单的实现

1) 将 C 能理解数据翻译成 Python 能读懂的对象类型

注意这里是将两个参数直接翻译，并没有一个一个的翻译

2) 调用函数以及此函数的两个参数

```
void CCpythonDlg::OnBnClickedOk()
{
    PyObject* pName = NULL;
    PyObject* pModule = NULL;
    PyObject* pDict = NULL;
    PyObject* pFunc = NULL;
    PyObject* pArgs = NULL;
    PyObject* pRet = NULL;
    Py_Initialize();
    PyRun_SimpleString("import sys");
    PyRun_SimpleString("sys.path.append('./')");
    pName=PyString_FromString("candpython");
    pModule =PyImport_ImportModule("candpython");
//  The first method _has parameters
//  pFunc=PyObject_GetAttrString(pModule,"getlog");  //no paramters
    pArgs=Py_BuildValue("ii",12,14);
    pFunc=PyObject_GetAttrString(pModule,"add");
    PyEval_CallObject(pFunc,pArgs);

// Another method
//  PyObject *pyParames=PyTuple_New(2);
//  PyObject *pyParames1=Py_BuildValue("i",5);
//  PyObject *pyParames2=Py_BuildValue("i",6);
//  PyTuple_SetItem(pyParames,0,pyParames1);
//  PyTuple_SetItem(pyParames,1,pyParames2);
//  pFunc=PyObject_GetAttrString(pModule,"add");
//  pRet=PyEval_CallObject(pFunc,pyParames);

    Py_Finalize();
    CDialogEx::OnOK();
}
```

4.Python 返回值传回给 C++

该 Python 脚本执行的是一个加法运算，先从控件获得加数和被加数，将其转换为 Python 对象并传给 Python 脚本，执行脚本后得到结果，再将此结果转换为 C++数据类型回传给 C++，并在控件中显示。

```
void CAddDlg::OnBnClickedAddButton()
{
    // TODO: 在此加入控制項告知處理常式程式碼
    UpdateData(TRUE);
    PyObject* pName = NULL;
    PyObject* pModule = NULL;
    PyObject* pDict = NULL;
    PyObject* pFunc = NULL;
    PyObject* pArgs = NULL;
    PyObject* pRet = NULL;
    PyObject* result = NULL;
    Py_Initialize();

    pModule = PyImport_ImportModule("addtion");
    pFunc=PyObject_GetAttrString(pModule,"add");
    pArgs=Py_BuildValue("dd",m_editSummand,m_editAddend);
    result=PyEval_CallObject(pFunc,pArgs);
    PyArg_Parse(result, "d", &m_editSum);
    Py_Finalize();
    UpdateData(FALSE);
}
```

该脚本 Python 脚本执行的是 at command 的压力测试，先从控件获得 COM 口，将其转换为 Python 对象传给 Python 脚本，执行完脚本后，得到测试总次数，Pass 次数和 Fail 次数，再将此结果转换为 C++数据类型回传给 C++，并在控件中显示。由于需要转换的 Python 对象有多个，所以此处用的是 int PyArg_ParseTuple(PyObject *args， const char *format，…)

```
void CatcommandlDlg::OnBnClickedOk()
{
    // TODO: 在此加入控制項告知處理常式程式碼
    UpdateData(TRUE);
    PyObject* pName = NULL;
    PyObject* pModule = NULL;
    PyObject* pDict = NULL;
    PyObject* pFunc = NULL;
    PyObject* pArgs = NULL;
    PyObject* pRet = NULL;
    PyObject* result = NULL;

    Py_Initialize();
    PyRun_SimpleString("import sys");
    PyRun_SimpleString("sys.path.append('./')");//加載程序当前工作路径
    pModule=PyImport_ImportModule("atgpioreq2");
    pArgs=Py_BuildValue("(s)",m_editcomport);
    pFunc=PyObject_GetAttrString(pModule,"atcommand");
    result=PyEval_CallObject(pFunc,pArgs);
    PyArg_ParseTuple(result, "iii", &m_resulttotal,&m_resultpass,&m_resultfail); //python数据类型转换为C++数据类型
    Py_Finalize();
    UpdateData(FALSE);

    //CDialogEx::OnOK();
}
```


5.运行 Python 脚本

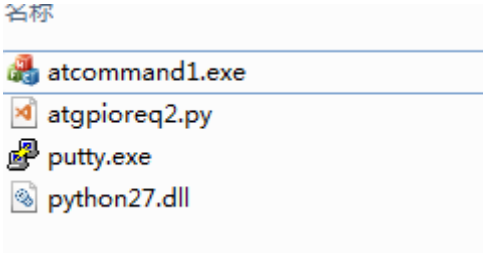
运行脚本使用 PyRun_AnyFile(FILE *fp， const char *filename)，为了得到 FILE 指针，需要先将 Python 脚本以只读方式打开并转换为 Python 对象，再将其转换为 FILE 指针，

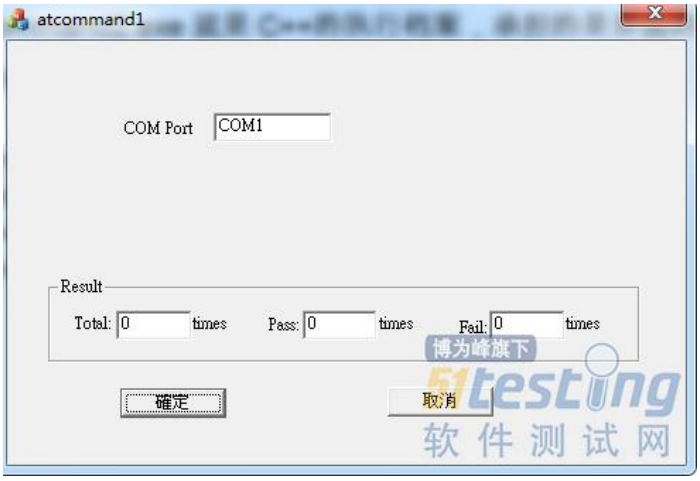
四、实际项目说明

1.简单例子

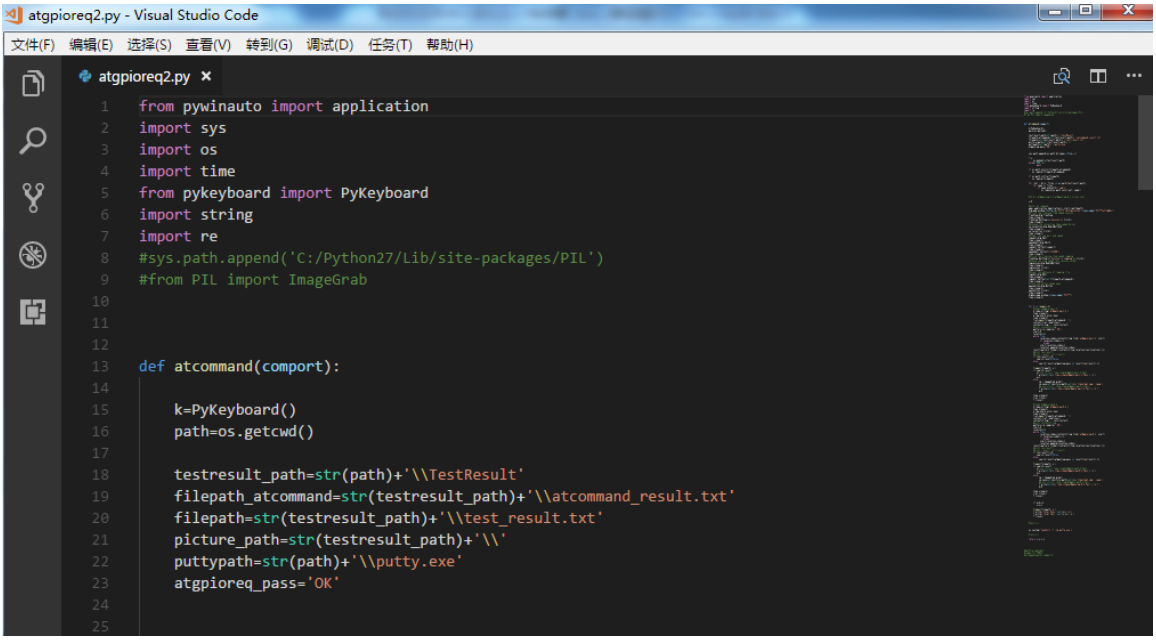
Atcommand2.exe 就是 C++的执行档案，承担的是界面的工作

Atgpioreq2.py 是界面中调用的脚本





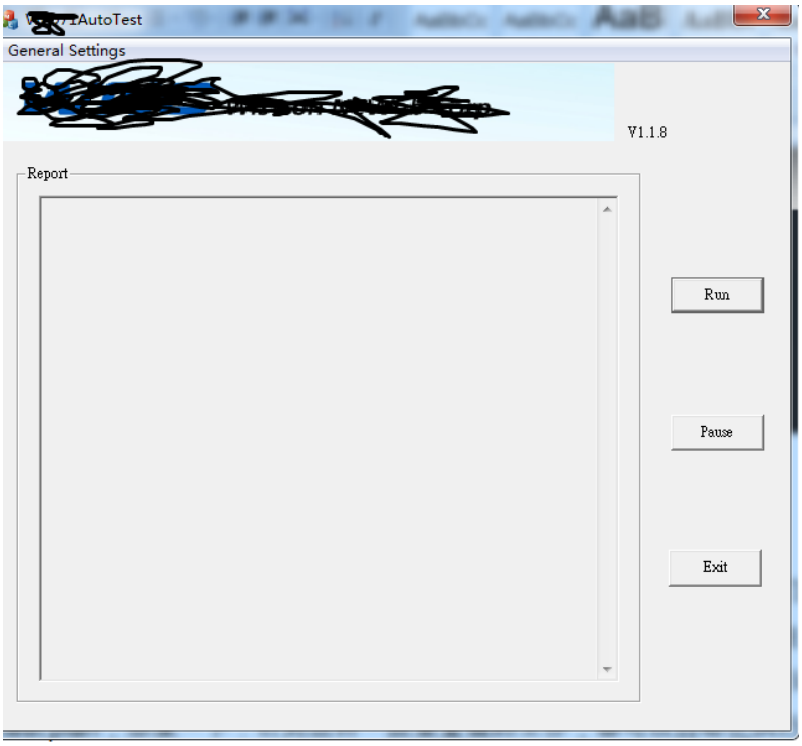
其实点击“确认”按键就是让下面的脚本执行，这样看起来是不是漂亮很多！

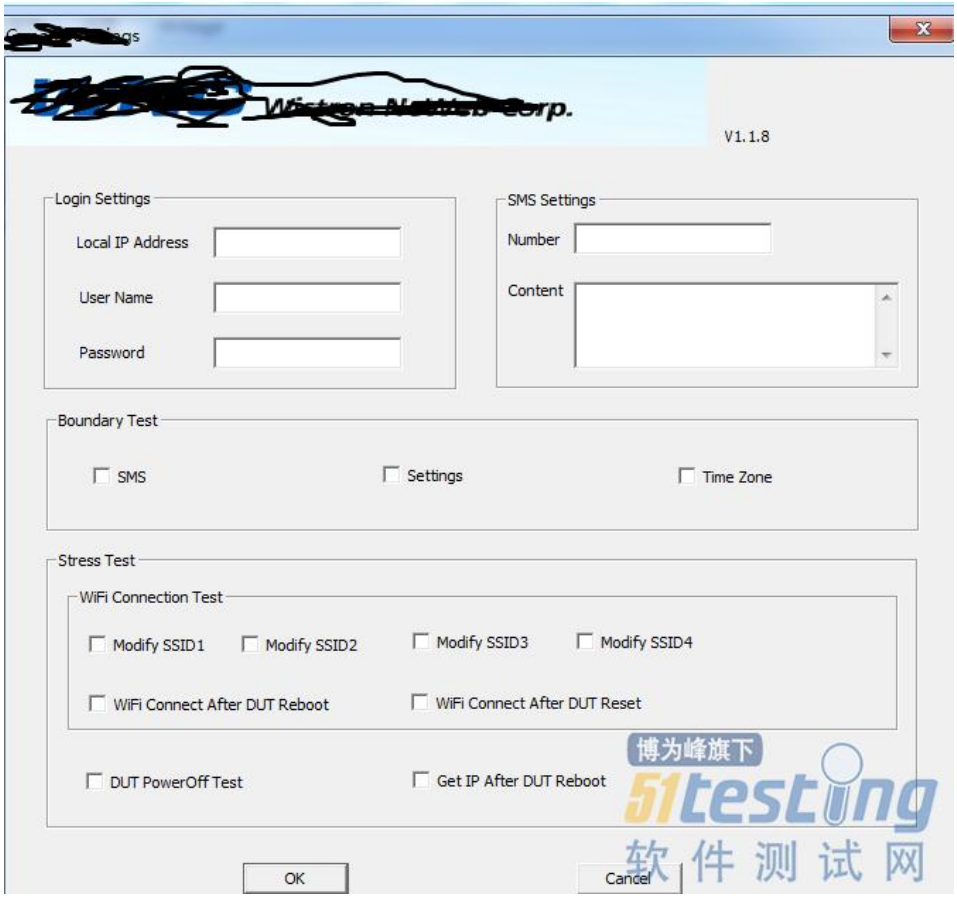


2.较漂亮的例子

如果你觉得上面的界面太简单了，看一下以下的例子:

一个非常漂亮的两个界面，其中汇集了边界测试，压力测试，而且考虑到每个项目需要将一些参数开放出来给使用者使用，对于使用者非常方便，需要测试什么项目，选择什么项目，根据项目的情况设置具体的参数，然后直接点击 run 就可以自动执行 python 的脚本，真的很酷!





这一系列告一段落，这个系列对于女巫的职业生涯真的意义非凡，对于别人的冷嘲热讽，只要内心够坚持，每天都在进步，日积月累真的就会产生奇妙的结果，当我们把这个有界面的测试工具拿出来时，老板就开始慢慢减少我们 team 的黑盒测试工作，直到现在几乎没有黑盒测试工作，还要感谢那些质疑我们搞什么搞得同事，虽然他们依然还在做着一成不变的工作……所以还是那句话：路漫漫其修远兮，吾将上下而求索……加油！

❖ 拓展学习

■自动化测试框架开发，职场精英必修课：http://testing51.mikecrm.com/hakgpyy

如何证明你是一名优秀的测试人员

◆作者：周培培

摘要:
测试人员每天都周旋于业务测试中，累的吐血，但最终月末总结的时候却发现除了 xxx 项目测试与上线外，似乎没什么有价值的东西可说了。晋升汇报时，往往感觉没什么重量级的东西可说，及时说出来感觉也没什么亮点。这些问题归纳起来就是：如何证明自己是一名优秀的测试人员。

背景

这里的测试人员不包括专门开发工具平台给测试人员使用的测试开发人员，而是专指服务于业务测试的测试人员。如果你是测试人员，是不是往往会有这样的体会：每天忙得晕头转向，不是被拉去评审需求，就是被拉去测试一个项目，但最终月末总结的时候却发现除了 xxx 项目测试与上线外，似乎没什么有价值的东西可说了。

当然了，另大多数测试人员更加“痛苦”的事情是：你每天周旋于业务中时，在最后的晋升中却发现根本 PK 不过那些每天业务测试清闲，有时间开发一些工具的人员。而在上级那里，你往往会得到这样一种说法：你说的事情属于苦劳，而别人做的事情属于功劳...但关键的问题在于，上级一方面期望你也有功劳，一方面又拼命仍给你业务，让你无暇功劳的事情。

同样作为测试人员，难道是否优秀就真的这么不容易评估吗，而你又如何证明你比别人更优秀呢？

一、诉诸于业务需求的价值

除了关心自己花了多少时间，测试了多少个项目外，请主动去了解业务需求上线后的效果，比如：PV/UV 增加了多少，购买增加了多少、业务指标增加了多少、系统技术指标提高了多少等等，这些效果本身就是你工作产出的一部分。

当然了，这方面的业务成果，和每个人负责的业务有极大的相关性：

- 有的业务非常容易评估效果，比如：广告收入、电商下单、视频网站访问量。

- 某些业务不太容易评估效果，比如：业务的技术重构、某个业务展示的改动。
- 某些业务可能指标时好时坏，比如：首页的大改版。

这个业务成果除了和业务本身有关系外，还和业务是否属于核心业务有很大的关系。比如：核心业务的一点改动、优化，很容易获得更大的关注，联动自己也会收到更多人的认可。但一个非核心业务，可能整个团队“折腾了半天”，可能产生的影响也微乎其微了。

这里的经验是：团队成员一起协商达成共识，团队的业务目标是什么，达成方式等等

二、诉诸于质量效果

这里的质量效果是指：在不断的业务产品更新迭代的周期内，产品上线质量如何。这里的质量效果最核心的一个关注点在于线上的故障数量。

当然了，线上出现故障，决定不单单是测试人员一个人的责任，但测试人员作为质量的一个重要的代言人，必然有推卸不掉的责任了。综合评估一个业务/产品的质量，可以直接去看其一个月、半年、一年、甚至更久一段时间内线上数量的变化情况。

三、诉诸于人效

人效的评估不是单纯的以多少开发测试比，每周上线多少业务，上线多少次等指标一锤定音，还应该考虑更多的因素：

相似的业务，在其他部门、业内其他公司的人员配比情况。 不同的业务，需要不同的人员配比；一个业务不同发展阶段，也需要不同的人员配比；

团队成员工作的合理性。这里举一个反面例子：一个需求涉及了小程序、APP、H5 等多个端，但由于需求经常不是一次性完成的，比如这次上小程序、H5，下次上 APP，这样的需求实现由于每次都要所有端回归，那么必然造成很多的重复性工作。

团队流程规范是否合理。流程规范对一个团队、特别是一个超过 10 个人以上的大团队来说，真是太重要了。哪怕一个小小的不合理之处都会极大降低合作效率。举一个电商售前售后的例子：售前提单，售后订单是两拨人在负责，而且两拨人会经常变动，比如今天可能在负责售前，明天可能去负责售后了。但由于大家都忙于业务测试，无暇梳理一些公共文档资料，导致的一个严重问题是——新人从介入到真正熟练上手往往需要很长时间，趟过很多坑；即便老人之间的合作也非常吃力，通常靠口口相传来沟通业务。

归纳起来就是，相较于其他人，你在人效方面有什么产出、你的人效高出了多少。

四、诉诸于影响力

只有对越来越多的人产生影响，自己的价值才会越来越大。充分扩大自己的影响力，也是提升自己个人品牌的一种方式。这里不妨问问自己以下几个问题:

- 部门、公司内有多少人认识你
- 有多少人了解你的工作
- 有多少人因为你的工作而受益
- 你的一个观点、行动能影响到多少人

五、诉诸于推动力

影响力属于一个人的自带流量，而推动力属于一个人利用自带的流量去做了些什么。这里的推动大致会有几个方面的考虑:

业务方向。比如：推动了 xxx 业务优化了什么，带来了什么效果；一起参与了 XXX 业务方向的调整，提出了什么意见;

合作方向。比如：推动落地了某某规范、流程执行，有了什么效果，合作是否更加高效；跨团队合作方面哪些问题得到了有效解决

技术方向。比如：推动了某某方案的优化，有了什么效果，哪些指标有了提升

写在最后

同样一份工作，同样的工作内容，为何不同的测试人员薪资方面会相差这么多，到底是什么起到了关键作用。除了人员本身的经验和资历外，大概就是一个人对工作的影响程度了吧。一名应届生培养一段时间可以 cover 住一个业务，而一个工作几年的资深人员来负责同一个业务的时候，对业务本身的影响、质量保证的效果、人效方面、影响力、推动力方面的工作，是一个应届生无法完成的了，而这些方面恰恰最能体现一个人的优秀程度了。

通过 Java API 像 MySQL 一样查询 HBASE

◆作者：王立东

摘要:
随着大数据的应用普及，HBASE 作为一种非常适应海量数据存储和查询的数据库也逐步流行起来。本文针对于实际测试中，存在的数据库查询需求进行说明。即使是使用 HBASE 的大数据应用，测试过程中也需要对数据进行查询来完成结果验证。HBASE 的命令行可以解决手工测试的问题，对于自动化测试中的数据验证，需要一种程序化的方法完成数据对比。关系型数据库的理念深入人心，所以如果能像查询 MySQL 一样查询 HBASE，能带来极大的便利。所以本文介绍一种使用 Java API 构造查询 HBASE 的实现思路，旨在通过程序设计统一 HBASE 和 MySQL 的处理。

本文首先简单介绍了 HBASE 和关系型数据库的差异，之后说明了 HBASE 查询命令的使用，详细给出了查询过滤器的 API，最后重点介绍了通过 Java API 实现的 HBASE 如同 MySQL 一样查询的程序实现，并对其中重要的查询给出流程说明。

1、HBASE 简介

HBase 是一个分布式的、面向列的开源数据库，它不同于一般的关系数据库，是一个适合于非结构化数据存储的数据库。另一个不同的是 HBase 基于列的而不是基于行的模式。简单理解 HBASE 就是一种数据库，和 MySQL、Oracle 等一样是用来存储数据的。但与这些关系型数据库有本质区别的是，HBASE 是一种 NOSQL 数据库，适合于非结构化数据存储的数据库。典型的 NOSQL 数据库包括 MangoDB、Redis、Memcache 等，在其中的 HBASE 是基于列数据库。

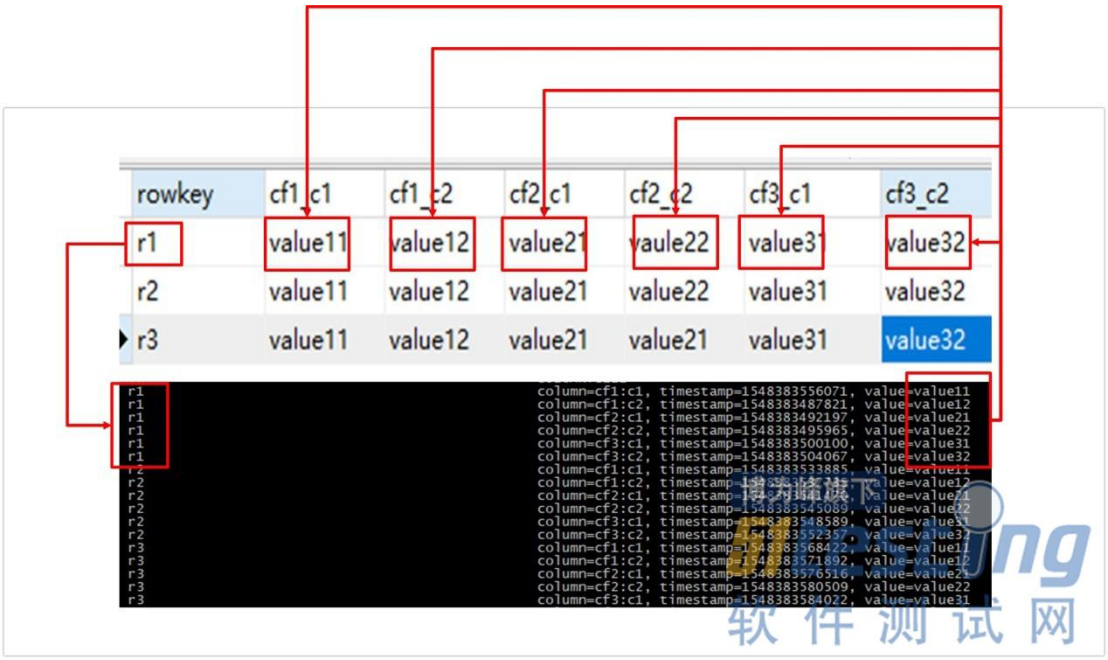
对于与关系型数据库，基于列的存储理解不是特别直接。传统的关系型数据库基于行存储，如下表 table1，具有 3 行数据 r1、r2 和 r3，表结构为 6 列，分别为 cf1_c1、cf1_c2、cf2_c1、cf2_c2、cf3_c1、cf3_c2，如下图。

rowkey	cf1_c1	cf1_c2	cf2_c1	cf2_c2	cf3_c1	cf3_c2
r1	value11	value12	value21	vaule22	value31	value32
r2	value11	value12	value21	value22	value31	value32
r3	value11	value12	value21	value21	value31	value32

将上述数据转换到 HBASE 中存储，查询数据如下。

```
hbase(main):424:0> scan 'table1'
ROW                                   COLUMN+CELL
 r1                                   column=cf1:c1, timestamp=1548383556071, value=value11
 r1                                   column=cf1:c2, timestamp=1548383487821, value=value12
 r1                                   column=cf2:c1, timestamp=1548383492197, value=value21
 r1                                   column=cf2:c2, timestamp=1548383495965, value=value22
 r1                                   column=cf3:c1, timestamp=1548383500100, value=value31
 r1                                   column=cf3:c2, timestamp=1548383504067, value=value32
 r2                                   column=cf1:c1, timestamp=1548383533885, value=value11
 r2                                   column=cf1:c2, timestamp=1548383537735, value=value12
 r2                                   column=cf2:c1, timestamp=1548383541470, value=value21
 r2                                   column=cf2:c2, timestamp=1548383545089, value=value22
 r2                                   column=cf3:c1, timestamp=1548383548589, value=value31
 r2                                   column=cf3:c2, timestamp=1548383552357, value=value32
 r3                                   column=cf1:c1, timestamp=1548383568422, value=value11
 r3                                   column=cf1:c2, timestamp=1548383571892, value=value12
 r3                                   column=cf2:c1, timestamp=1548383576516, value=value21
 r3                                   column=cf2:c2, timestamp=1548383580509, value=value22
 r3                                   column=cf3:c1, timestamp=1548383584022, value=value31
 r3                                   column=cf3:c2, timestamp=1548383588080, value=value32
3 row(s) in 0.0150 seconds
```

通过下方的“3 row(s)”我们可以看出，虽然列出了 18 条数据，但事实也是属于 3 行的，所以就对应了关系型数据库中的 3 行记录。以 r1 为例，对应关系如下图所示。



这是一种方便理解的对应方式。HBASE 的优势在于不是每列都需要有值，这样就非常适合稀疏数据的存储。同时，列名可以根据需要随时增加，方便存储非结构化数据。

HBASE 目前是 Apache 的 Hadoop 项目的子项目，是一个分布式的、面向列的开源数据库。该技术来源于 Fay Chang 所撰写的 Google 论文“Bigtable: 一个结构化数据的分布式存储系统”。就像 Bigtable 利用了 Google 文件系统（File System）所提供的分布式数

据存储一样，HBase 在 Hadoop 之上提供了类似于 Bigtable 的能力。

2、HBASE 查询命令和 Java API 详解

2.1 HBASE 查询命令

HBASE 的查询命令提供两种方式:

- 按指定 rowkey 获取唯一一条记录：get 方法。
- 按指定条件获取一批记录：scan 方法。

二者都可以对 HBASE 进行过滤查询，以 get 为例，最常见的用法是如下命令：get ‘table1’,’ r1’，结果如下。

```
hbase(main):005:0* get 'table1', 'r1'
COLUMN                                CELL
 cf1:c1                               timestamp=1548383556071, value=value11
 cf1:c2                               timestamp=1548383487821, value=value12
 cf2:c1                               timestamp=1548383492197, value=value21
 cf2:c2                               timestamp=1548383495965, value=value22
 cf3:c1                               timestamp=1548383500100, value=value31
 cf3:c2                               timestamp=1548383504067, value=value32
6 row(s) in 0.3950 seconds
```

如果指定了列族和列名，可以获取指定的单元格信息，命令为：get ‘table1’,’ r1’,’ cf1:c1’，结果如下。

```
hbase(main):006:0> get 'table1', 'r1','cf1:c1'
COLUMN                                CELL
 cf1:c1                               timestamp=1548383556071, value=value11
1 row(s) in 0.0150 seconds
```

但 get 只能获取一行的数据，如果想通过 get 直接获取一个表中的全部数据是做不到的，这种情况就要用到 scan。

2.2 Scan 基本使用

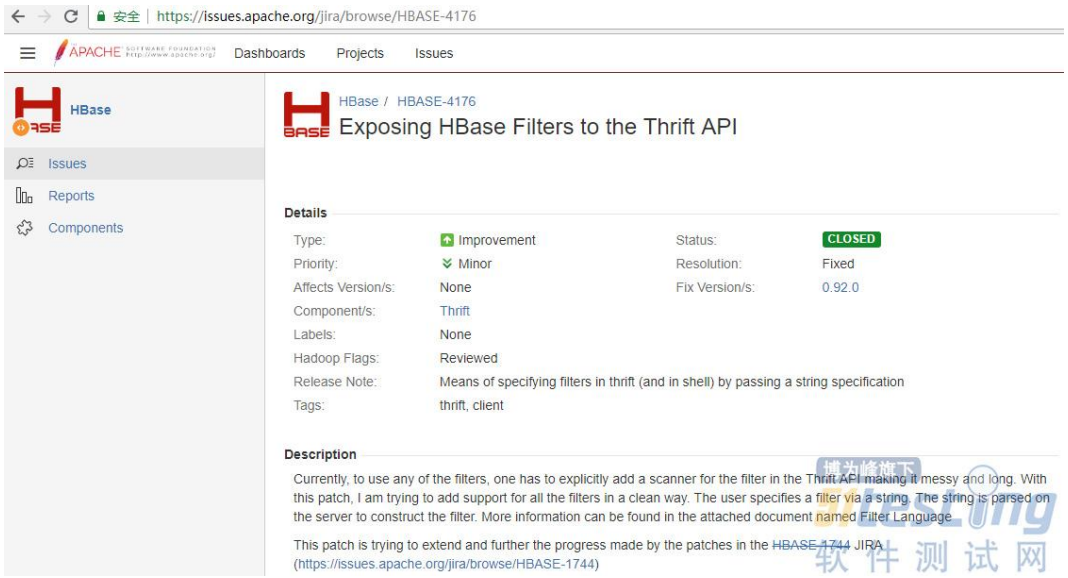
通过 HBASE shell 中的帮助，我们可以知道，scan 作用是扫描一个表。具体来说，scan 命令需要指定表名，以及可选的字典查询器。查询器中指定一个或多个如下条件：时间范围，过滤器，返回的行数，开始的行，结束的行，行前缀过滤器， 时间戳，最大长度或返回的列，缓存还是原始数据，版本，全部指标或部分指标。

如果在命令中不指定列，会返回全部的列。当需要返回所有列的时候，在列族后面不指定列限定名。

有两种方式指定过滤器:

（1）使用过滤字符串：关于这点的更多信息可以查看有关该过滤器语言的文档，在

HBASE 的 JIRA（一个需求管理系统）中查看第 4176 号需求。下图是该需求的说明。



（2）使用过滤器的全包名。

如果需要查看 scan 结果的全部指标，配置参数 ALL_METRICS 需要设置为 true。或者，我们只需要补发指标，那么可以通过指定关心的指标即可。

示例：

查看表的全部内容：

```
hbase> scan 'hbase:meta'
```

```
hbase(main):030:0> scan 'table1'
ROW                                           COLUMN+CELL
 r1                                           column=cf1:c1, timestamp=1548915064457, value=value11
 r1                                           column=cf1:c2, timestamp=1548915064504, value=value12
 r1                                           column=cf2:c1, timestamp=1548915064528, value=value21
 r1                                           column=cf2:c2, timestamp=1548915064551, value=value22
 r1                                           column=cf3:c1, timestamp=1548915064573, value=value31
 r1                                           column=cf3:c2, timestamp=1548915071822, value=value32
 r2                                           column=cf1:c1, timestamp=1548915090505, value=value11
 r2                                           column=cf1:c2, timestamp=1548915090565, value=value12
 r2                                           column=cf2:c1, timestamp=1548915090588, value=value21
 r2                                           column=cf2:c2, timestamp=1548915090613, value=value22
 r2                                           column=cf3:c1, timestamp=1548915090661, value=value31
 r2                                           column=cf3:c2, timestamp=1548915091337, value=value32
 r3                                           column=cf1:c1, timestamp=1548915096855, value=value11
 r3                                           column=cf1:c2, timestamp=1548915096916, value=value12
 r3                                           column=cf2:c1, timestamp=1548915096935, value=value21
 r3                                           column=cf2:c2, timestamp=1548915096957, value=value22
 r3                                           column=cf3:c1, timestamp=1548915096975, value=value31
 r3                                           column=cf3:c2, timestamp=1548915098034, value=value32
3 row(s) in 0.0490 seconds
```

查看表的部分列：COLUMNS 中指定需要查看的列，多列时通过逗号隔开。

```
hbase> scan 'hbase:meta',  {COLUMNS => 'info:regioninfo'}
```

```
hbase(main):021:0> scan 'table1', {COLUMNS => 'cf1:c1'}
ROW                                           COLUMN+CELL
 r1                                           column=cf1:c1, timestamp=1548915362925, value=value11
 r2                                           column=cf1:c1, timestamp=1548915368571, value=value11
 r3                                           column=cf1:c1, timestamp=1548915373712, value=value11
3 row(s) in 0.0180 seconds
```

限定行个数和开始行查看部分列：LIMIT 指定返回的行数，STARTROW 指定开始行。

hbase> scan 'ns1:t1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTROW => 'xyz'}

hbase> scan 't1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTROW => 'xyz'}

```
hbase(main):023:0> scan 'table1', {COLUMNS => ['cf1:c1', 'cf2:c2'], LIMIT => 2, STARTROW => 'r1'}
ROW                                           COLUMN+CELL
 r1                                           column=cf1:c1, timestamp=1548915362925, value=value11
 r1                                           column=cf2:c2, timestamp=1548915363001, value=value22
 r2                                           column=cf1:c1, timestamp=1548915368571, value=value11
 r2                                           column=cf2:c2, timestamp=1548915368668, value=value22
2 row(s) in 0.0230 seconds
```

设定时间戳查看部分列：TIMERANGE 指定开始和结束的时间戳

hbase> scan 't1', {COLUMNS => 'c1', TIMERANGE => [1303668804, 1303668904]}

```
hbase(main):029:0> scan 'table1', {COLUMNS => 'cf1:c1', TIMERANGE => [1548915362925, 1548915368575]}
ROW                                           COLUMN+CELL
 r1                                           column=cf1:c1, timestamp=1548915362925, value=value11
 r2                                           column=cf1:c1, timestamp=1548915368571, value=value11
2 row(s) in 0.0110 seconds
```

倒序查看:

hbase> scan 't1', {REVERSED => true}

```
hbase(main):034:0> scan 'table1', {REVERSED => true}
ROW                                           COLUMN+CELL
 r3                                           column=cf1:c1, timestamp=1548915373712, value=value11
 r3                                           column=cf1:c2, timestamp=1548915373732, value=value12
 r3                                           column=cf2:c1, timestamp=1548915373807, value=value21
 r3                                           column=cf2:c2, timestamp=1548915373837, value=value22
 r3                                           column=cf3:c1, timestamp=1548915373851, value=value31
 r3                                           column=cf3:c2, timestamp=1548915374444, value=value32
 r2                                           column=cf1:c1, timestamp=1548915368571, value=value11
 r2                                           column=cf1:c2, timestamp=1548915368637, value=value12
 r2                                           column=cf2:c1, timestamp=1548915368653, value=value21
 r2                                           column=cf2:c2, timestamp=1548915368668, value=value22
 r2                                           column=cf3:c1, timestamp=1548915368685, value=value31
 r2                                           column=cf3:c2, timestamp=1548915369171, value=value32
 r1                                           column=cf1:c1, timestamp=1548915362925, value=value11
 r1                                           column=cf1:c2, timestamp=1548915362959, value=value12
 r1                                           column=cf2:c1, timestamp=1548915362986, value=value21
 r1                                           column=cf2:c2, timestamp=1548915363001, value=value22
 r1                                           column=cf3:c1, timestamp=1548915363018, value=value31
 r1                                           column=cf3:c2, timestamp=1548915363586, value=value32
3 row(s) in 0.0210 seconds
```

查看 HBase 的全部指标:

hbase> scan 't1', {ALL_METRICS => true}

```
hbase(main):036:0> scan 'table1', {ALL_METRICS => true}
ROW                                           COLUMN+CELL
 r1                                           column=cf1:c1, timestamp=1548915362925, value=value11
 r1                                           column=cf1:c2, timestamp=1548915362959, value=value12
 r1                                           column=cf2:c1, timestamp=1548915362986, value=value21
 r1                                           column=cf2:c2, timestamp=1548915363001, value=value22
 r1                                           column=cf3:c1, timestamp=1548915363018, value=value31
 r1                                           column=cf3:c2, timestamp=1548915363586, value=value32
 r2                                           column=cf1:c1, timestamp=1548915368571, value=value11
 r2                                           column=cf1:c2, timestamp=1548915368637, value=value12
 r2                                           column=cf2:c1, timestamp=1548915368653, value=value21
 r2                                           column=cf2:c2, timestamp=1548915368668, value=value22
 r2                                           column=cf3:c1, timestamp=1548915368685, value=value31
 r2                                           column=cf3:c2, timestamp=1548915369171, value=value32
 r3                                           column=cf1:c1, timestamp=1548915373712, value=value11
 r3                                           column=cf1:c2, timestamp=1548915373732, value=value12
 r3                                           column=cf2:c1, timestamp=1548915373807, value=value21
 r3                                           column=cf2:c2, timestamp=1548915373837, value=value22
 r3                                           column=cf3:c1, timestamp=1548915373851, value=value31
 r3                                           column=cf3:c2, timestamp=1548915374444, value=value32
3 row(s) in 0.0260 seconds

METRIC                                        VALUE
 BYTES_IN_REMOTE_RESULTS                      0
 BYTES_IN_RESULTS                             684
 MILLIS_BETWEEN_NEXTS                         11
 NOT_SERVING_REGION_EXCEPTION                 0
 REGIONS_SCANNED                              1
 REMOTE_RPC_CALLS                             0
 REMOTE_RPC_RETRIES                           0
 ROWS_FILTERED                                0
 ROWS_SCANNED                                 3
 RPC_CALLS                                    3
 RPC_RETRIES                                  0
```

查看部分指标:

hbase> scan 't1', {METRICS => ['RPC_RETRIES', 'ROWS_FILTERED']}

```
hbase(main):039:0> scan 'table1', {METRICS => ['RPC_RETRIES', 'BYTES_IN_RESULTS']}
ROW                                           COLUMN+CELL
 r1                                           column=cf1:c1, timestamp=1548915362925, value=value11
 r1                                           column=cf1:c2, timestamp=1548915362959, value=value12
 r1                                           column=cf2:c1, timestamp=1548915362986, value=value21
 r1                                           column=cf2:c2, timestamp=1548915363001, value=value22
 r1                                           column=cf3:c1, timestamp=1548915363018, value=value31
 r1                                           column=cf3:c2, timestamp=1548915363586, value=value32
 r2                                           column=cf1:c1, timestamp=1548915368571, value=value11
 r2                                           column=cf1:c2, timestamp=1548915368637, value=value12
 r2                                           column=cf2:c1, timestamp=1548915368653, value=value21
 r2                                           column=cf2:c2, timestamp=1548915368668, value=value22
 r2                                           column=cf3:c1, timestamp=1548915368685, value=value31
 r2                                           column=cf3:c2, timestamp=1548915369171, value=value32
 r3                                           column=cf1:c1, timestamp=1548915373712, value=value11
 r3                                           column=cf1:c2, timestamp=1548915373732, value=value12
 r3                                           column=cf2:c1, timestamp=1548915373807, value=value21
 r3                                           column=cf2:c2, timestamp=1548915373837, value=value22
 r3                                           column=cf3:c1, timestamp=1548915373851, value=value31
 r3                                           column=cf3:c2, timestamp=1548915374444, value=value32
3 row(s) in 0.0180 seconds

METRIC                                        VALUE
 BYTES_IN_RESULTS                             684
 RPC_RETRIES                                  0
```

指定过滤器和过滤内容进行查看：scan 支持非常多、非常丰富的过滤器，通过 ROWPREFIXFILTER 指定符合条件的 rowkey，QualifierFilter 指定列限定名，TimestampsFilter 指定时间戳的起止。

hbase> scan 't1', {ROWPREFIXFILTER => 'row2', FILTER => "

(QualifierFilter (>=, 'binary:xyz')) AND (TimestampsFilter (123, 456))"}

```
hbase(main):046:0> scan 'table1', {ROWPREFIXFILTER => 'r1', FILTER => "(QualifierFilter (>=, 'binary:c1')) AND (TimestampsFilter ( 1548915362925, 1548915362926))"}
ROW                                           COLUMN+CELL
 r1                                           column=cf1:c1, timestamp=1548915362925, value=value11
1 row(s) in 0.0090 seconds
```

指定分页过滤器，查看符合条件的页：参数为返回的页数和开始的页数。

hbase> scan 't1', {FILTER =>

org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(1, 0)}

```
hbase(main):052:0> scan 'table1', {FILTER =>org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(2, 1)}
ROW                                           COLUMN+CELL
 r1                                           column=cf1:c2, timestamp=1548915362959, value=value12
 r1                                           column=cf2:c1, timestamp=1548915362986, value=value21
 r2                                           column=cf1:c2, timestamp=1548915368637, value=value12
 r2                                           column=cf2:c1, timestamp=1548915368653, value=value21
 r3                                           column=cf1:c2, timestamp=1548915373732, value=value12
 r3                                           column=cf2:c1, timestamp=1548915373807, value=value21
3 row(s) in 0.0140 seconds
```

按时间轴一致性读取：HBase 默认按强一致性读取（Consistency.STRONG），设定为 TIMELINE 可以按时间轴一致性读取

hbase> scan 't1', {CONSISTENCY => 'TIMELINE'}

```
hbase(main):059:0> scan 'table1', {CONSISTENCY => 'TIMELINE'}
ROW                                           COLUMN+CELL
 r1                                           column=cf1:c1, timestamp=1548921458090, value=value11
 r1                                           column=cf1:c2, timestamp=1548915362959, value=value12
 r1                                           column=cf2:c1, timestamp=1548915362986, value=value21
 r1                                           column=cf2:c2, timestamp=1548915363001, value=value22
 r1                                           column=cf3:c1, timestamp=1548915363018, value=value31
 r1                                           column=cf3:c2, timestamp=1548915363586, value=value32
 r2                                           column=cf1:c1, timestamp=1548915368571, value=value11
 r2                                           column=cf1:c2, timestamp=1548915368637, value=value12
 r2                                           column=cf2:c1, timestamp=1548915368653, value=value21
 r2                                           column=cf2:c2, timestamp=1548915368668, value=value22
 r2                                           column=cf3:c1, timestamp=1548915368685, value=value31
 r2                                           column=cf3:c2, timestamp=1548915369171, value=value32
 r3                                           column=cf1:c1, timestamp=1548915373712, value=value11
 r3                                           column=cf1:c2, timestamp=1548915373732, value=value12
 r3                                           column=cf2:c1, timestamp=1548915373807, value=value21
 r3                                           column=cf2:c2, timestamp=1548915373837, value=value22
 r3                                           column=cf3:c1, timestamp=1548915373851, value=value31
 r3                                           column=cf3:c2, timestamp=1548915374444, value=value32
3 row(s) in 0.0170 seconds
```

2.3 Scan 过滤器相关的 Java API

从上面的介绍可以看到，命令行下的 Scan 参数非常丰富，能够实现我们非常复杂的查询请求。我们知道，MySQL 的 JDBC API 中是可以直接执行 SQL 语句的，查询结果会以 ResultSet 对象返回。而 HBASE 不同，原生态的 HBASE 的 Java API 是以过滤器的方式提供的查询功能，无法执行原始的命令。通过环境配置，Apache Phoenix 可以将 HBASE 转换为关系型数据存储，但需要对集群服务器进行配置修改。所以还是需要使用 HBASE 的 Java API 满足查询的需求。

HBASE 的 Scan 过滤器相关的 Java API 分为三类，分别是操作符、比较器和过滤器，决定了一个查询的操作关系(是大于还是小于)，以什么方式比较(字符还是正则表达式)，以及过滤的内容（是 rowkey 还是列）。具体介绍如下。

操作符：HBASE 提供了枚举类型的变量来表示这些抽象的操作符。

操作符	含义
CompareOperator.GREATER	大于
CompareOperator.EQUAL	等于
CompareOperator.GREATER_OR_EQUAL	不小于
CompareOperator.LESS	小于
CompareOperator.LESS_OR_EQUAL	不大于
CompareOperator.NO_OP	没有操作
CompareOperator.NOT_EQUAL	不等于

比较器：比较器作为过滤器的核心组成之一，用于处理具体的比较逻辑，例如字节级的比较，字符串级的比较等。这些类都是 Comparable 接口的实现类。

比较器	含义
BigDecimalComparator	数值比较器，按数字大小比较

BinaryComparator	二进制比较器,用于按字典顺序比较 Byte 数据值。
BinaryPrefixComparator	前缀二进制比较器。与二进制比较器不同的是，只比较前缀是否相同。
BitComparator	进行二进制操作后进行比较
LongComparator	数值比较器
NullComparator	判断当前值是否为 null
RegexStringComparator	支持正则表达式的值比较
SubStringComparator	用于监测一个子串是否存在于值中，并且不区分大小写。

过滤器：过滤器是 Scan 过滤的核心，通过操作符和比较器构造。HBASE 支持的过滤器非常多，下表中加粗的为常用的过滤器。过滤器都是 FilterBase 的直接过间接子类。

比较器	含义
ColumnCountGetFilter	列个数过滤器
ColumnPaginationFilter	基于 ColumnCountGetFilter 的列个数分页过滤器。
ColumnPrefixFilter	列名前缀过滤器
ColumnRangeFilter	列名范围过滤器，比较方式是字典序。
DependentColumnFilter	参考列过滤器
FamilyFilter	列族过滤器
QualifierFilter	列限定符过滤器
RowFilter	行键过滤器
ValueFilter	值过滤器
FirstKeyOnlyFilter	首次行键过滤器
FuzzyRowFilter	模糊行匹配过滤器
InclusiveStopFilter	包含结束的过滤器。
KeyOnlyFilter	行键过滤器
FirstKeyValueMatchingQualifiersFilter	首次行键列限定名过滤器
MultipleColumnPrefixFilter	多个列前缀过滤器
MultiRowRangeFilter	多 rowkey 范围过滤器
PageFilter	分页过滤器
PrefixFilter	前缀过滤器
RandomRowFilter	随机行过滤器
SingleColumnValueFilter	单列值过滤器
SingleColumnValueExcludeFilter	单列排除过滤器继承自 SingleColumnValueFilter，过滤的方式还是按照 SingleColumnValueFilter 去过滤，但是最后的结果集去除了作为过滤条件的列
SkipFilter	跳过过滤器 根据构造器中的过滤器为基准跳过行
TimestampsFilter	使用时间戳的值来过滤值
WhileMatchFilter	当匹配到就中断扫描

2.4 HBASE 其他命令

创建表

```
create 'table1', {NAME => 'cf1', VERSIONS => 1}, {NAME => 'cf2', VERSIONS => 1}, {NAME => 'cf3', VERSIONS => 1}
```

上述命令创建了一个名为“table1”的，具有三个列族的表。

删除表

```
disable 'table1'
drop 'table1'
```

删除表要进行两步，首先 disable，再 drop。

插入数据

```
put 'table1', 'r1', 'f1', 'v1'
```

不指定列名插入数据。

```
put 'table1', 'r2','cf1:c1','value11'
put 'table1', 'r2','cf1:c2','value12'
put 'table1', 'r2','cf2:c1','value21'
put 'table1', 'r2','cf2:c2','value22'
```

指定列名插入数据。

删除数据

```
delete 'table1','r1','cf1:'
```

删除表 r1 行的 cf1 下的没有列名的数据，注意不是删除 cf1 下的全部列。

```
delete 'table1','r2','cf1:c1'
delete 'table1','r2','cf1:c2'
delete 'table1','r2','cf2:c1'
delete 'table1','r2','cf2:c2'
```

删除指定行、指定列的数据。

3、像 MySQL 一样查询 HBASE

3.1 Scan 的 Java API 使用

通过 Java API 进行 HBASE 的查询是很方便的，下面的例子是使用 RowFilter 的例子。

```
public static void main(String[] agrv) throws IOException{
    //连接HBASE
    Configuration configuration = HBaseConfiguration.create();
    configuration.set("hbase.client.retries.number", "1");
    configuration.set("hbase.zookeeper.quorum","[redacted]");
    configuration.set("hbase.zookeeper.property.clientPort","[redacted]");
    Connection conn = null;
    conn = ConnectionFactory.createConnection(configuration);
    //查询HBASE
    Table table =conn.getTable(TableName.valueOf("table1"));
    Scan scan=new Scan();
    scan.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("c1"));
    scan.addColumn(Bytes.toBytes("cf2"),Bytes.toBytes("c2"));
    Filter filter = new RowFilter(CompareOperator.EQUAL, new BinaryComparator(Bytes.toBytes("r1")));
    scan.setFilter(filter);

    ResultScanner scanner = table.getScanner(scan);
    for(Result rs:scanner){
        String rowkey = Bytes.toString(rs.getRow());
        System.out.println("row key :"+rowkey);
        Cell[] cells  = rs.rawCells();
        for(Cell cell : cells) {
            String family = new String(cell.getFamilyArray(),cell.getFamilyOffset(),cell.getFamilyLength());
            String qualifier = new String(cell.getQualifierArray(),cell.getQualifierOffset(),cell.getQualifierLength());
            String value = new String(cell.getValueArray(),cell.getValueOffset(),cell.getValueLength());
            System.out.println(family+"_"+qualifier+":"+value);
        }
    }
    scanner.close();
    table.close();
    //断开连接
    conn.close();
}
```

首先连接 HBASE，填入 IP 和端口，之后设置查询条件进行查询并打印内容，最后断开连接。在设置查询条件时，首先获取需要查询的表，之后创建过滤器 Scan。通过 addColumn 指定查询后显示的列。然后创建行 RowFilter，操作符为等于，比较器为二进制，即查询 rowkey 等于“r1”的内容。setFilter 后通过 getScanner 执行查询，最后通过 ResultScanner 的遍历打印查询结果。结果如下。

```
row key :r1
cf1_c1:value11
cf2_c2:value22
```

类比于 MySQL，上述代码相当于执行如下 SQL 语句。

SELECT cf1:c1，cf2:c2 FROM table1 WHERE rowkey=’ r1’ ；

3.2 查询 HBASE 的 Java 程序设计

通过上述示例可以看到，通过 Java API 可以完成类似 MySQL 的查询。而实际的测试中，在接口结果验证时，我们需要通过外部参数如同传递 SQL 语句一样，传递对 HBASE 的查询需求，所以需要设计类似的机制。

事实上，在测试中对数据的验证，需求相对是比较固定的，如查询某个 key 的值是否存在，满足某个条件的行数，满足某个条件的列的结果等等。所以本设计就针对实际使用设定固定的查询需求，对不同的查询需求调用适当的 Scan 过滤器。

3.2.1 总体程序设计

HbaseDbOperator

该类为提供查询服务的类，接受查询的参数，分析查询需求，调用特定的查询函数返回查询结果。

```java
public class HbaseDbOperator extends DataBaseOperator{
    private HbaseDbUtil hbaseDbConn = null;
    public HbaseDbOperator(String dbUrl,String cfgFile) throws Exception{
        HbaseDbConnetPara connPara = new HbaseDbConnetPara(dbUrl,cfgFile);
        hbaseDbConn = new HbaseDbUtil(connPara);
    }


    public String  runSql(String query) throws Exception{
        hbaseDbConn.openConn();
        String queryRst = null;
        HbaseQueryParser parser = new HbaseQueryParser(query);
        String type = parser.getType();
        switch(type){
            case HbaseQueryParser.Select_RowKey_001://select rowkey as alias from name where col1 like v1,col2 like v2,col3 like v3;
                queryRst = hbaseDbConn.queryConditionRowkey(parser);//返回内容: alias:9797dbcc-3754-42a1-9fb9-737c63b14b07
                break;
            case HbaseQueryParser.Select_Timestamp_002://select timestamp as alias from name where  col1 like v1,col2 like v2,col3 like v3;
                queryRst = hbaseDbConn.queryConditionTimeStamp(parser);//返回内容: alias:1547006755321
                break;
            case HbaseQueryParser.Select_Count_003://select count(*) as alias from name where col1 like v1,col2 like v2,col3 like v3;
                queryRst = hbaseDbConn.queryConditionCount(parser);//返回内容: alias:1
                break;
            case HbaseQueryParser.Select_Columns_004://select col1 as a1,col2 as a2 from name where col1 like v1,col3 like v3,col4 like v4;
                queryRst = hbaseDbConn.queryConditionColumns(parser);//返回内容: a1:a1-str,a2:a2-str
                break;
            case HbaseQueryParser.Select_Json_005:
                queryRst = hbaseDbConn.queryConditionJson(parser);//返回内容: value
                break;
            case HbaseQueryParser.Select_Count_Equal006://select count(*) as alias from name where col1=v1,col2=v2,col3=v3;
                queryRst = hbaseDbConn.queryConditionCountEqual(parser);//返回内容: alias:1
                break;
            default://其他需求找王立东
                System.out.println("未实现-----有需求再说");
        }

        hbaseDbConn.closeConn();
        return queryRst;
    }
}
```

HbaseDbOperator 继承 DataBaseOperator，这样可以使得多种数据库的接口一致。HbaseDbOperator 实例化时接受数据库的 url 和配置文件，其中 url 确定查询的数据 IP 和端口，配置文件是为了获取 Hadoop 的 host 名称。实际查询时调用 runSql 函数，变量为查询的语句，查询语句为 JSON 格式，类似如下。

```
{type:"select_rowkey",
  table:"table1",
  select:{rowkey:row_key},
  condition:{"cf1:c1":"r1"}}
```

HbaseQueryParser 完成 JSON 格式的解析，获取查询的类型，在后续的 HBASE 操作中也提供 JSON 格式中内容的输出。之后根据特定的查询类型调用 HbaseDbUtil 的特定函数完成查询。

3.2.2 HbaseDbUtil

该类提供对 HBASE 的连接、查询等操作，是完成功能的核心类。

```
public class HbaseDbUtil {

    private String hbaseIpAddr;
    private String hbasePort;
    private Admin hbaseAdmin;
    private Connection hbaseConn;

    public HbaseDbUtil(HbaseDbConnetPara para){

    public void openConn(){

    public void closeConn(){

    // 查询指定rowkey,返回一列的数据(列族_列名:值)----该函数目前仅作参考和调试,主流程无调用
    private void RowFilter(String tableName, String rowKeyString,String cf,String column) throws IOException {

    /**
     * 根据指定的列过滤第一个符合条件的行,返回alias:rowkeystring
     * @param parser 查询json解析器
     * @throws IOException
     */
    public String queryConditionRowkey(HbaseQueryParser parser) throws IOException{


    /**
     * 根据指定的列过滤第一个符合条件的时间戳,返回alias:timestamp_string
     * @param parser 查询json解析器
     * @throws IOException
     */
    public String queryConditionTimeStamp(HbaseQueryParser parser) throws IOException{



    /**
     * 根据查询条件返回满足条件的row个数,返回alias:cnt_res
     * @param parser 查询json解析器
     * @throws IOException
     */
    public String queryConditionCount(HbaseQueryParser parser) throws IOException{
```

HbaseDbUtil 是一个工具类，完成 HBASE 数据库的连接、断开，并根据需要进行相应查询，具体查询的时间在后面详细介绍。

HbaseDbConnetPara 和 HbaseQueryParser

HbaseDbConnetPara 和 HbaseQueryParser 是两个辅助类。HbaseQueryParser 如前面所说，完成对查询语句的 JSON 解析，将查询类型、查询提交等进行解析，供其他类调用。

HbaseDbConnetPara 是 HbaseDbOperator 构造函数中实例化的，通过数据库的 url、hosts 文件内容完成数据参数的初始化，在 HbaseDbUtil 的数据库连接中直接使用。

3.2.3 Select_RowKey_001 的实现

各类的查询实现由相似之处，以 Select_RowKey_001 的实现进行说明。这类查询面对的需求是查询特定内容的 rowkey 值，查询条件的 JSON 类似如下。

```
{type:"select_rowkey",
   table:"table1",
   select:{rowkey:row_key},
   condition:{"cf1:c1":"value111"}}
```

类比的 SQL 语句是 SELECT rowkey AS row_key FROM table1 WHERE cf1:c1 LIKE ‘value111’ ;

在 HbaseDbUtil 中实现如下。

```
/**
 * 根据指定的列过滤第一个符合条件的行,返回alias:rowkeystring
 * @param parser 查询json解析器
 * @throws IOException
 */
public String queryConditionRowkey(HbaseQueryParser parser) throws IOException{
    Table table = hbaseConn.getTable(TableName.valueOf(parser.getTableName()));
    List<Filter> filters = buildFilterList(parser.getSelectConditions());
    FilterList filterList1 = new FilterList(filters);
    Scan scan = new Scan();
    scan.setFilter(filterList1);

    ResultScanner scanner = table.getScanner(scan);
    StringBuilder rst=new StringBuilder("{");
    String alias = (new JSONObject(parser.getSelectColumns())).getString(HbaseQueryParser.RowKey_ColName);

    for (Result result:scanner) {
        byte[] row = result.getRow();
        rst.append(alias+":"+Bytes.toString(row));

    }
    scanner.close();
    table.close();
    return rst.append("}").toString();
}
```

首先根据参数获取需要查询的表，之后建立过滤器，设定过滤器之后通过 getScanner 发起查询，最后将满足条件的内容以 JSON 格式返回。

测试代码如下。

```
public static void main(String[] agrv) throws Exception{
    String url = "hbase://[redacted]";
    HbaseDbOperator hbaseOp = new HbaseDbOperator(url,"dbparam.properties");
    String query = "{type:\"select_rowkey\",table:\"table1\",select:{rowkey:row_key},condition:{\"cf1:c1\":\"value111\"}}";
    System.out.println(hbaseOp.runSql(query));
}
```

查询结果：{row_key:r1}。

3.2.4 其他查询类型

Select_Timestamp_002

查询满足条件内容的最新的时间戳，查询语句的 JSON 类似如下:

```
{type:"select_timestamp",
    table:"table1",
    select:{"timestamp":"time1"},
    condition:{"rowkey":"r1", "cf1:c1":"value111"}}
```

相当于查询如下 SQL:

SELECT timestamp AS time1 FROM table1 WHERE cf1:c1 LIKE ‘value111’ ;

Select_Count_003

查询满足条件内容的条数，查询语句的 JSON 类似如下:

```
{ type:"select_count",
   table:"table1",
   select:{"count":"cnt"},
   condition:{"rowkey":"r1", "cf1:c1":"value111"}}
```

相当于查询如下 SQL:

```
SELECT count(*) as cnt FROM table1 WHERE rowkey=’r1’ and cf1:c1 LIKE ‘value111’;
```

Select_Count_Equal_006

查询满足条件内容的条数（与 2 的区别是 like 和等于），查询语句的 JSON 类似如下:

```
{ "type":"select_count_equal",
   "table":"table1",
   "select":{"count":"cnt"},
   "condition":{"rowkey":"r1", "cf1:c1":"value111"}}
```

相当于查询如下 SQL:

```
SELECT count(*) as cnt FROM table1 WHERE rowkey=’r1’ and cf1:c1=’value111’;
```

4、总结

HBASE 在大数据系统中被普遍应用，是非常优秀的 NOSQL 数据库，但对于测试工程师而言，需要颠覆关系型数据库的理念进行测试。无形中增加了测试难度，本文主要介绍了一种使用 Java API 实现的查询 HBASE 的程序设计，可以像查询 MySQL 一样查询 HBASE。文中给出的基本的程序实现，通过这种思路可以在接口测试、功能自动化测试中使用，并且可以界面完成一个易用的测试工具。

Appium+Python 实现 APP 启动页跳转到首页

◆作者：桃 子

本文以 MSN news 为例，实现启动 APP 后跳转到首页的功能，包含使用 list 进行元素定位、try except else 进行是否首次启动 APP 判断，logging 进行日志记录等功能

一、场景:

1.启动 APP 后连续跳过 welcom、interest 、what's new 页面到首页

2.判断是否是首次启动，如果首次启动通过出现 welcom 页面，如果不是首次启动则直接进入 interest 页面

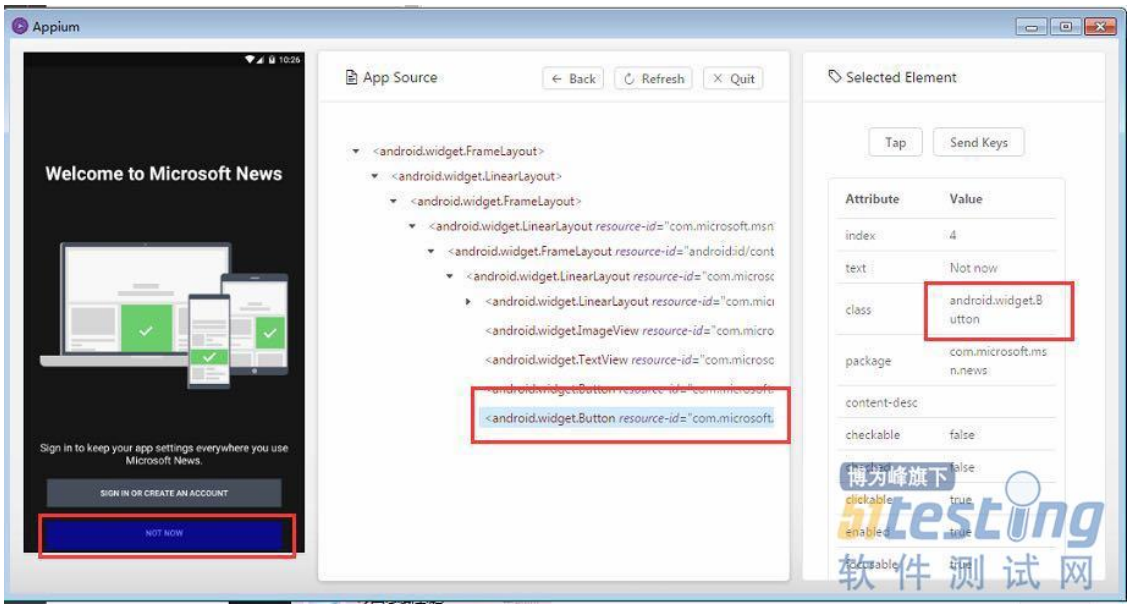
3.使用 logging 模块记录日志

Gif 图片文件:

二、实践

2.1、启动 APP 后连续跳过 welcom、interest 、what's new 页面到首页

welcom 页面



1. 首先启动 Appium 进行 session 回话

2. APP 启动到 welcom 页面，点击 not now 按钮，跳转到下一个页面

分析：这里我使用 list 元素进行定位，通过观察可以知道 class name 和 sign in name 名称相同，如果直接使用 classname 那么就会定会到当前页面首个 class name 元素，导致定位失败。

list 定位获取一组 class 名称，通过数组下标进行区分

```
a=driver.find_element_by_id()
a[1].click()
```

3. 代码：

```
skipwel=driver.find_elements_by_class_name('android.widget.Button')
skipwel[1].click()
```

2.2、判断是否是首次启动，如果首次启动通过出现 welcom 页面，如果不是首次启动则直接进入 interest 页面

首次启动 app 时，会弹出 welcome 页面，但是第二次启动时该页面就会消失，出现 interest 页面那么对于这种判断我们该如何处理呢

这里我们使用 try except 语句进行判断

try except 语法:

```
try:
<语句>        #运行别的代码
except <名字>:
<语句>        #如果在try部份引发了'name'异常
except <名字>，<数据>:
<语句>        #如果引发了'name'异常，获得附加的数据
else:
<语句>        #如果没有异常发生
```

工作原理:

✓如果 try 语句发生异常，则执行 except 匹配名称后语句

✓如果 try 语句未发生异常，执行 else 后面语句

✓如果 try 语句发生异常，except 语句后内容没有匹配成功，异常将被递交到上层的 try

处理思路:

定位 interest 页面，如果定位失败，证明出现的是 welcome，执行 welcome 语句；否则执行 interest 页面语句

代码:

```
def welcome():
  logging.info('skip welcome')
  skipwel=driver.find_elements_by_class_name('android.widget.Button')
  skipwel[1].click()
try:
  driver.find_elements_by_class_name('android.widget.Button')
except NoSuchElementException:
  welcome()
else:
  skipinterest=driver.find_elements_by_class_name('android.widget.Button')
  skipinterest[0].click()
```

interets 页面

原理同 welcom 页面

```
skipinterest=driver.find_elements_by_class_name('android.widget.Button')
skipinterest[0].click()
```

what ‘s new page

原理同 welcom 页面

```
skipwhatnew=driver.find_elements_by_class_name('android.widget.Button')
skipwhatnew[1].click()
```

2.3、使用 logging 模块记录日志

日志是用来帮我们定位问题，通过日志可以更准准确的确定问题所在，我们可以设定日志级别、日志的输出格式等

场景:

在启动 APP 时，输出 start app 日志

导航页跳转到首页过程，输出执行的页面为 interest 页面还是 welcome 页面

2.3.1、单个 py 文件日志

在单个 py 文件里日志输出形式如下:

1. 日志流程

首先，导入日志模块

然后，basicconfig 方法创建记录器，为记录做基本配置。

✓日志级别：level

✓日志输出路径：Filename

✓日志输出格式：Format

最后，在需要添加日志的地方输出日志即可，logging.info（’ start app ‘）

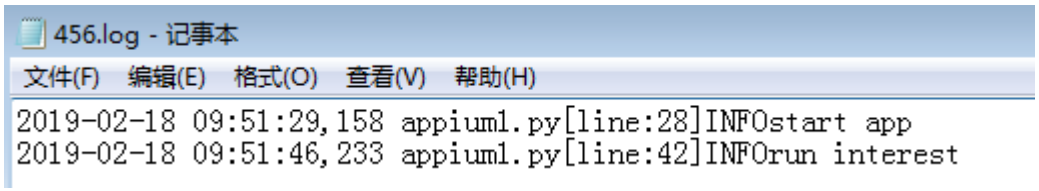
2. 代码实现

```
import logging
logging.basicConfig(level=logging.INFO，filename='456.log',
  format='%(asctime)s %(filename)s[line:%(lineno)d]%(levelname)s%(message)s')
```

```
#判断是否首次启动 App，跳过导航页到首页
def welcome():
  logging.info('skip welcome')
  skipwel=driver.find_elements_by_class_name('android.widget.Button')
  skipwel[1].click()
try:
  driver.find_elements_by_class_name('android.widget.Button')
except NoSuchElementException:
  logging.info('run welcome')
  welcome()
else:
  logging.info('run interest')
  skipinterest=driver.find_elements_by_class_name('android.widget.Button')
  skipinterest[0].click()
```

3.运行

在代码中 456.log 中查看日志输出信息



2.3.2 日志模块

上面讲述的是单个功能日志的流程，整个 APP 测试下来有很多功能都需要用到日志，如果每个功能都重复编写，会造成代码冗余

所以，下面我们把日志的配置参数抽离出来，需要时直接调用即可

1.首先建立 log.conf 文件，存储配置参数等信息

log.conf 文件代码:

```
[loggers]
keys=root，infoLogger
[logger_root]
level=DEBUG
```

handlers=consoleHander，fileHandler

[logger_infoLogger]

handlers=consoleHander，fileHandler

qualname=infoLogger

propagat=0

[handlers]

keys=consoleHander，fileHandler

[handler_consoleHander]

class=StreamHandler

level=INFO

formatter=form02

args=(sys.stdout，)

[handler_fileHandler]

class=FileHandler

level=INFO

formatter=form01

args=('456.log'，'a')

[formatters]

keys=form01，form02

2. 在需要引用 l o g 的 py 文件中引用 log.conf 文件

```
import logging
import logging.config

CON_LOG='log.conf'
logging.config.fileConfig(CON_LOG)
logging=logging.getLogger()
```

将上文中如下代码删除

logging.basicConfig(level=logging.INFO，filename='456.log'，

format='%(asctime)s %(filename)s[line:%(lineno)d]%(levelname)s%(message)s')

更换为:

CON_LOG='log.conf'

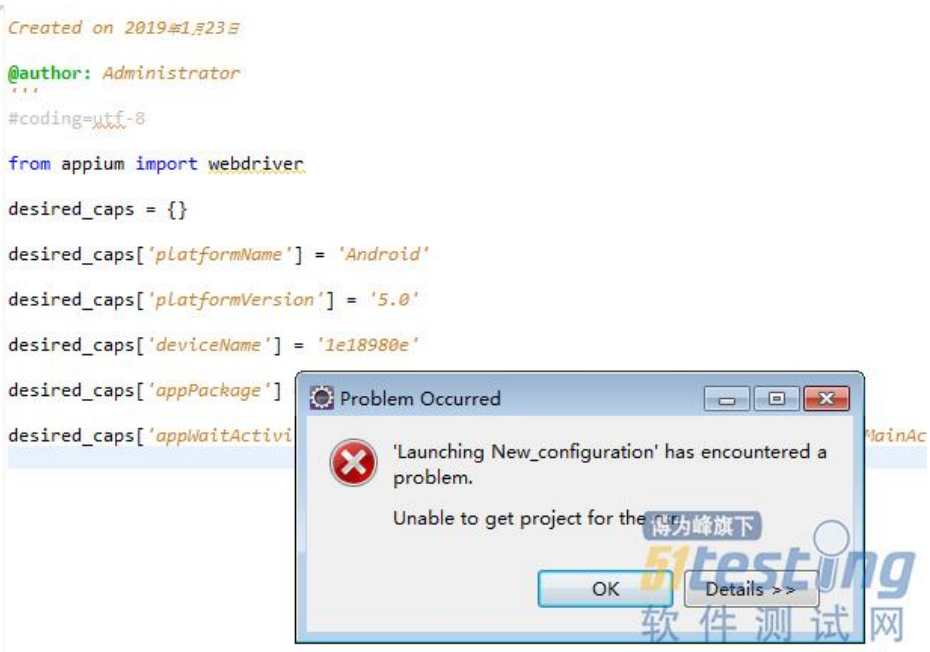
logging.config.fileConfig(CON_LOG)

logging=logging.getLogger()

三、报错提示

总结运行过程中报错提示及解决方法

1. Eclipse 运行 启动 app 代码，提示 launching new configuration



Eclipse

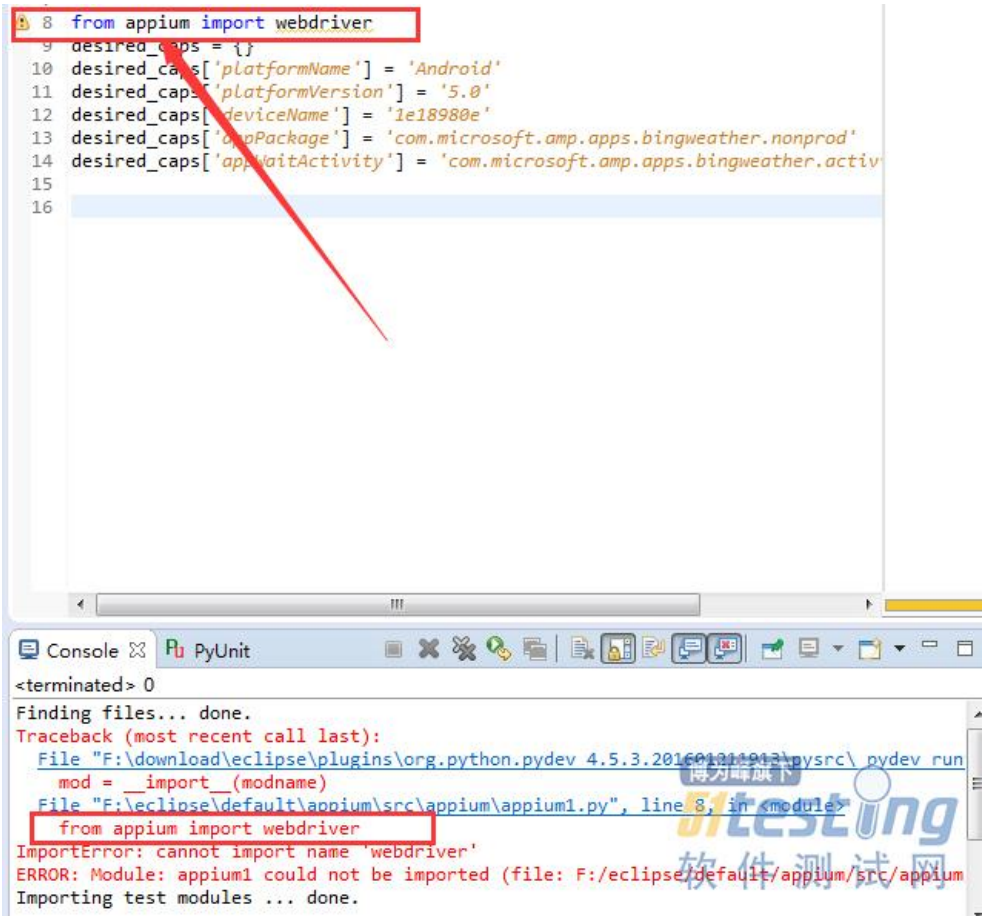
解决方法:

Project -> Properties -> Run/Debug Settings:

1. select "Launching New_configuration5"

2. Delete

2. ImportError: cannot import name 'webdriver'



解决方法:

from selenium import webdriver

3. TypeError: 'WebElement' object does not support indexing

```
  exec(compile(contents+"\n", file, 'exec'), glob, loc)
File "F:\eclipse\default\appium\src\appium\appium1.py", line 29,
  skip[1].click()
TypeError: 'WebElement' object does not support indexing
```

解决: 把 find_element 改为 find_elements

4. IndexError: list index out of range

```
  pydev_imports.execfile(file, globals, locals)  #
File "F:\download\eclipse\plugins\org.python.pydev
  exec(compile(contents+"\n", file, 'exec'), glob,
File "F:\eclipse\default\appium\src\appium\appium1
  skipnews[1].click()
IndexError: list index out of range
```

解决:

有可能是数组越界，里面数值从 0 开始标记

再有可能 list 是一个空的，没有一个元素

进行 list[0]就会出现该错误

5. module 'logging' has no attribute 'basicConfig'

```
    import logging
  File "F:\eclipse\default\appium\logging\logging.py", line 2, in <mod
    logging.basicConfig(level=logging.INFO)
AttributeError: module 'logging' has no attribute 'basicConfig'
```

解决:

自己就命名了一个 logging.py 的文件，在代码中 import 的是自己新建的这个，找不到 basicConfig，文件名改成 logging_test.py 后，正常运行。

6. int' object is not callable

```
<terminated> F:\eclipse\default\appium\logging\logging_test.py
Traceback (most recent call last):
  File "F:\eclipse\default\appium\logging\logging_test.py", line 4, in <moc
    logging.DEBUG('debug info')
TypeError: 'int' object is not callable
```

解决：发现代码中写的是 logging.DEBUG，应该为 logging.debug()

❖ 拓展学习

■Python 全栈测试开发，每天 1 小时，搞定名企工作！http://testing51.mikecrm.com/hakgpyy

自动化测试前，你需要知道的 10 点

◆作者：周培培

摘要:
从步入测试领域开始，无论哪个公司，哪个团队都在谈论自动化测试、动手实现自动化测试，从而让测试显得更加“高大上”。那么是不是所有的业务都适合自动化？是不是自动化做的越多，效果越好呢？下面就自己一些经验和感悟，聊聊自己的一些体会。

一、引言

我一直以来获得了太多关于何时实现测试自动化、以及如何实现测试自动化方面的问题。不再一一针对每个人的问题来回答，这里针对这些问题一些讨论吧。

下面我将就何时自动化，如何自动化，是否应该自动化谈一谈自己的看法。

我知道一些读者比我要聪明。因此，对这样一个大的主题进行的讨论是值得的，以便从不同领域的专家那里获得深入的想法和思考，以及他们在自动化测试方面的经验。

二、为什么自动化测试?

1) 在测试时，你进行了新的部署、bug 修复，这是你如何保证新 bug 没有被引入老功能？你需要测试之前的功能。

因而，每当有 bug 修复，或新功能添加时，你都要手工测试所有功能？考虑到花费、资源、时间等等因素，你这么测试不是高效的。

因而自动化有了需求:

-当你有太多回归测试工作要做时，请自动化你的测试工作

2) 当你正在测试一款 web 应用时，与此同时，这个应用可能有数千用户正在使用。

你将如何测试这样的 web 应用？你将如何使用手工方式，同时模拟这些多的用户呢？这是一件十分困难的手工操作。

-模拟众多虚拟用户，来测试应用的负载容量时，请将你的负载测试自动化

3) 你正在测试一款代码被频繁修改的应用，虽然 GUI 几乎一样，但功能变动越多，需要的测试“维修”就越多。

-当你的 GUI 几乎不变，功能频繁发生变化时，请将你的测试工作自动化。

三、关于自动化测试，有哪些风险?

在一些不同的情况下，你可以考虑自动化测试工作。这里我介绍自动化测试的一些风险。如果你已经下定决心要做自动化或者想要更早地采取措施，那么请先考虑以下问题。

1) 你能找到有经验的人力吗?

想要自动化，你需要有一些编程经验的人员。

考虑一下你的人力资源。他们有足够的自动化测试经验吗？如果没有，他们有技术能力或编程背景来轻松应对新技术吗？你打算投资建立一个好的自动化团队吗？如果你的答案是肯定的，那么考虑自动化你的工作吧。

2) 自动化的初始成本非常高

我赞同这个观点：由于要雇用熟练的手动测试人员，因而手动测试的相关成本很高。但如果你正在考虑将自动化作为方案，请三思而后行。

自动化的初始新建成本太高，例如：自动化工具的购买，测试脚本的培训和维护。

很多自动化工具用户都会后悔做自动化。如果你花费了很高的成本，却只得到了一些好看的测试工具和一些基本的自动化脚本，那么自动化的用途是什么？

3) 如果 UI 不是一成不变的，不要试图自动化

自动化测试用户界面前务，请必要小心。如果用户界面正在大范围发送变化，那么自动化脚本的维护成本将会非常高。在这种情况下，基本的 UI 自动化就足够了。

4) 你的应用是否足够稳定，可以支持你的自动化测试工作?

在早期的开发周期中自动化测试工作将是一个坏主意(除非它处在一个敏捷的环境)。在这种情况下，脚本的维护成本将非常高。

5) 你正在考虑 100%自动化?

别异想天开了，你不可能 100%将测试工作自动化。当然，有一些领域，如性能测试，回归测试，负载/压力测试，你可以有机会达到接近 100％的自动化。但用户界面，文档，

安装，兼容性和恢复等领域，必须手动完成测试。

6) 不要自动化只执行一次的测试任务

某些识别应用领域和测试用例，可能只需要运行一次，并且不需要包含在回归测试中。避免自动化此类模块或测试用例。

7) 你的自动化套件会长期使用吗?

每个自动化脚本套件都应该有足够长的使用寿命，其新建成本应该绝对低于手动执行成本。然而分析每个自动化脚本套件的有效成本有点困难。

对于单独的构建（一般假设，取决于具体的应用程序的复杂性），大约应该使用或运行至少 15 到 20 次自动化套件，才能获得良好的 ROI。

四、总结

自动化测试是实现大多数测试目标和有效利用资源和时间的最佳方式。但在选择自动化工具之前，你应该谨慎。在决定自动化测试工作之前，请确保有熟练的人力。否则，您的工具只是一个空架子，无法获得 ROI。

将昂贵的自动化工具交给非技术人员会带来失望。在购买自动化工具之前，请确保该工具最适合你的要求。你不太可能拥有与你的要求 100％匹配的工具。

你需要找出最符合你要求的工具的局限性，然后使用手动测试来克服这些测试工具的限制性。开源工具也是开始自动化的好选择。

不是 100％依赖于手动或自动化，而是要使用手动测试和自动化测试的最佳组合。这是每个项目的最佳解决方案（我认为）。自动化套件不会找到所有的错误，也不能替代真正的测试人员。在许多情况下，随机测试也是必要的。

如何利用 TestNG 做接口自动化测试？

◆作者：小鱼儿

摘要:
本文主要依据之前测试经验，涉及接口自动化测试，文章旨在帮助小公司想开展自己独立的接口自动化测试环境，主要从接口自动化测试是什么，接口自动化测试适应哪些公司哪些项目，接口自动化环境怎么搭建以及怎么进行接口自动化测试展开，最终达到帮助测试人员写成一条接口自动化测试用例的目的。

测试按照不同的测试标准有不同的测试划分，按照测试阶段，可以划分为：单元测试，集成测试，系统测试，验收测试；按照测试升入层次，可以划分为：黑盒测试，白盒测试，灰盒测试；按照测试对象，可以划分为：性能测试，安全测试，兼容性测试，文档测试，易用性测试（用户体验测试），业务测试，界面测试，安装测试；按照测试方法，可以划分为：自动化测试，功能测试；不知道大家目前在做什么类型的测试工作呢？可能大部分测试人员日常测试工作都是涉及上面的多个方面，今天我主要讨论下接口测试，涉及的是接口自动化测试，接口测试涉及程序的输入输出，接口自动化测试是利用 TestNG 测试框架，对程序的输入输出进行校验的测试工作。

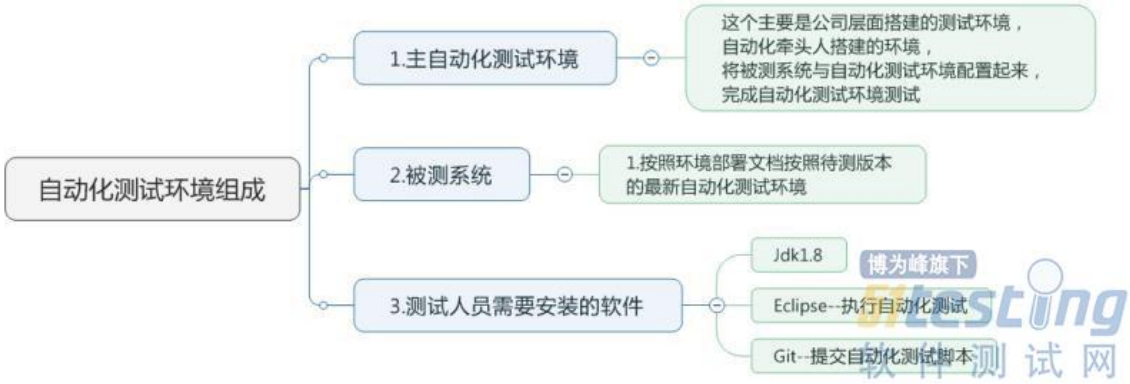
那么接口自动化测试可以为哪些公司提供测试服务呢？接口自动化测试其实和手工测试本质上测试内容是一致的，一般自动化测试都是针对比较稳定的程序，对于不经常进行变动的程序开展测试工作，接口自动化测试工作也是如此，主要是对已经完成大部分程序开发和测试的模块，对其中的部分改造的存量功能开展的测试工作，主要适用于接口类的测试工作，测试用例主要从手工测试用例中挑选，从中选择可以进行接口自动化测试的用例开展接口自动化测试工作。如果你们公司项目相对比较稳定，大部分项目主要是二次开发那可能就比较适用，最好是一个项目可以拆分为一个一个的相互之间相互独立又相互影响的应用，因为接口自动化测试大部分是涉及程序之间的接口应用；但是对于一个版本的改动导致程序改的面目全非，或者自动化主要是为了界面类服务的就不适用这套框架。

搭建这套自动化测试环境复杂吗？搭建这套自动化测试环境主要分为以下三个方面，主要为：

一.主环境，公司层面搭建的测试环境，主自动化测试环境；

二.被测系统，主要是之前进行手工测试需要的系统环境，最好能有专门进行自动化测试的环境，如果资源有限也可以使用目前在用的被测环境；

三.本地测试人员测试需要安装的软件；JDK，Eclipse 等，具体可以参照以下图解：



如果你们公司的系统适应这套自动化测试环境，并且测试人员已经按照要求把自动化测试环境搭建好了，现在怎么写接口自动化测试用例呢？接下来会讲到一条自动化测试用例怎么编写，以及怎么将自动化测试用例提交到环境上让自动化牵头人统计到，首先写自动化的时候可以首先完成下面材料的准备工作：接口文档(类名，方法名)，jar 包文件；接口文档可以问开发人员要，也可以自己去触发程序检查日志获得；Jar 包文件可以直接让开发人员打包发给你，这个自己不太好获得，写完自动化测试用例之后自己上传到测试环境中。这里首先需要说明几点注意事项，第一：虽然我们是自动化测试，也会用到开发人员使用到的 Eclipse，Jdk 等工具，但是这里面并不会涉及到很复杂的代码要求，并不需要测试人员和开发人员那样了解代码，代码要求其实不高，甚至也可以说没有代码要求，所以测试人员不需要担心自己没有大码底子，写不了这套框架的自动化测试用例，担心到时候写自动化的时候无从开展；第二：对数据库要求比较高，如果数据库基本薄弱这块相对会有点问题，所以写自动化之前前提是你是位比较了解数据库的测试人员；

写自动化测试用例涉及哪些步骤呢？编写这套自动化测试用例测试人员需要准备什么呢？步骤有哪些呢？主要有以下操作步骤：

第一：自动化测试环境:

如果在上面你已经把自动化测试环境搭建好了，其实节省了我们很多时间，然后需要确定好准备写哪个功能，哪个模块，最好能确定写哪个类下面的哪个方法，计划写什么用例，校验哪些点，对应前台的界面是哪些等，问开发人员要到对应功能的 Jar 包和接口文档等;

第二：下载自动化测试工程(project):

这个工程是我们的环境基础，我们在对应自己应用下面新增本次测试用例;

第三：自动化编写涉及哪些文件:

3.1.JAVA 文件（主程序，调用入参和数据准备文件）;

3.2.XLS 文件（数据池文件）;

3.3.Dubbo 文件（配置文件，配置自动化调度）;

3.4.Jar 包（上面提到的问开发要的自动化 Jar 包，打包的程序，主要是一些类和方法）;

3.5.调度文件（自动化统计配置文档，包括每日新增自动化数量，新增模块，谁新增的以及失败自动化整改情况）; 3.6.接口定义文件（定义请求通讯区和应答通讯区）;

第四：自动化编写

步骤一：将工程下载下来之后将 Jar 包上传，然后进行加载;

自动化编写步骤二: 可以复制和自己模块差不多功能点的 JAVA 文件，修改里面的 ID，类名，方法名;其实除了类名，方法名，其他的代码调用公共方法，引用相同的包这些都是一样了，为了减少工作量，其他的复制就可以;

第五：自动化工作流程

XLS 文件（数据池文件）里面主要是进行自动化编写的数据准备，包括请求通讯区和应答通讯区，因为我们的自动化工作流程主要为以下图示工作流程图，所以会涉及到请求通讯区和应答通讯区的数据和文件，数据池文件主要包括一些自动化案例执行前的 Before class 数据准备和自动化用例执行后的 After class 数据清理，是一系列的 SQL 代码，这里要求对数据库的 SQL 语句比较熟悉:

接口自动化测试工作流程:

通过 JAVA 文件调用 JAR 包文件-->通过接口定义文件和数据池文件-->拼成一个报文（也可以叫 map 文件）-->往服务器（被测系统）发送-->等待服务器响应-->

-->如果上送的内容与远程服务器一致，断言通过，测试用例执行通过;

-->如果上送的内容与远程服务器不一致，断言不通过，测试用例执行不通过;

第六：断言:

这里面有个概念：断言，可能写过自动化测试用例的同行比较熟悉这个概念，没有接触过的可能就比较生疏了，这里统一解释下，断言其实对于我们这套接口自动化测试来说就是一个个独立的程序分支，可以借助于以下场景去帮助理解：业务场景：一个新增界面，要求新增用户，约束规则：已经注销的用户不能新增，新增过的用户不能重复新增；那么这个用例你会怎么设计呢？思路：写的类：1.针对重复用户，Before class:首先在用户信息表里面先数据库插入一条用户数据，需要先清理，怕插入时报主键冲突等错误，保证环境是干净的，清理是请求通讯区的数据准备操作 Before class，然后执行插入操作，最后在应答通讯区在对应字段里面写入和插入数据完全一致的记录，这样在调用程序的时候就会报错：“该用户已经存在”，error code 和 error message 会依据之前已经定义的接口文档抛出对应的错误，这条用例就是我们的一个断言，也就是一条接口自动化测试用例，执行完这条自动化之后我们需要对插入数据库的代码执行清理操作，这也是一个比较好的编写自动化测试用例的习惯，主要也是怕给数据库造成太多的脏数据，导致测试环境混乱，不利于后面测试。同理我们对注销的操作也是一样的，只是数据准备存在差异，涉及的表不同，这里面的话程序的判断接口是注销，则我们需要在注销涉及的表里面插入一条记录，然后前台在新增的时候会有对应的 error message：“已经注销的用户不能新增”。

第七：自动化测试用例统计:

已经写好的自动化测试用例我们需要进行提交，提交之前必须确保本地执行通过，没有失败的测试用例提交到环境上面，如果有失败的测试用例需要按照 console 的输出提示进行调试，调试完成没有问题提交 GIT，这里面还是得说下这个 git 是个很好的东西，它只会出现那些你新增的内容在提交界面，然后你选择你刚刚编辑过的新增和有改动的代码提交到环境上面，这样后续就可以让自动化统计工具自动统计到，因为我们之前是每月有自动化编写要求的，所以必须在规定的时间将自动化测试用例提交到环境上去，至此我们的自动化测试用例算新增完成，后期的工作主要是维护，后期如果测试环境部署新的代码也是需要定期更新我们的自动化测试脚本。

上面已经简单介绍了接口自动化测试用例是什么，接口自动化测试用例适合哪些公司的什么类型系统，接口自动化测试环境怎么搭建，怎么开展接口自动化测试，以及怎么编写一条接口自动化测试用例，希望本次介绍的接口自动化接口测试-TEST-NG 框架可以帮助你们顺利完成自动化测试框架选择和搭建，可以对大家后期的自动化测试工作有所帮助，也希望大家可以分享自己公司目前在使用的自动化测试框架，最终我希望每家公司每个产品都可以找到一个适合自己本公司本产品的自动化测试框架，进而最终解放人力成本，让人力资源去做更多更有意义和更有价值的工作。

如何并行运行你的自动化测试

◆作者：咖啡猫

在自动化测试过程中的某些时刻，你将遇到一个共同的问题，那就是：需要花费很长时间去执行大量的 UI 自动化测试用例。最为有效地加速用例执行的方式就是并行地运行自动化测试。不幸地是，使各个自动化测试用例并行地执行并非一件容易的事情。典型地，线程安全就没有融入到自动化测试中去。线程安全可以确保代码编写完成后，程序可以运行并操作各种数据结构而不产生意想不到的结果。

为了测试可以并行地运行，需要满足一些强制性的要求。这些要求同我们的一些客户每天用来帮助运行五万条自动化测试用例所需的条件相同。四个强制要求如下：

1. 测试必须是原子的

你的自动化测试用例应该形成一个单一的不能简化的单元。这就意味着你的测试必须极其集中并且每个测试用例只做一件事。一个单独的自动化测试用例应该不能测试端到端的功能。

在我的团队中使用的一个比较好的经验办法就是：自动化验收测试在你本地资源上运行的时间不能超过一分钟，实际上当然是运行越快越好，但是也不要太着急、慢慢来，可以逐步改进运行时间。如果你的测试用例运行时间超过一分钟，那将是很危险的事。

有一个例子，在这个存储库中可用的非原子测试（NAT）。本篇文章将自始至终地引用此测试库中的内容。

```
19          [Test]
20          public void EndToEndTest()
21          {
22              SauceReporter.SetBuildName("AntiPatternTests3");
23              var loginPage = new SauceDemoLoginPage(Driver);
24              //test loading of login page
25              loginPage.Open().IsLoaded.Should().BeTrue("the login page should load successfully.");
26              loginPage.UsernameField.Displayed.Should().BeTrue("the page is loaded, so the username field should exist");
27              loginPage.PasswordField.Displayed.Should().BeTrue("the page is loaded, so the password field should exist");
28
29              //test login with valid user
30              var productsPage = loginPage.Login("standard_user", "secret_sauce");
31              productsPage.IsLoaded.Should().BeTrue("we successfully logged in and the home page should load.");
32              productsPage.Logout();
33              loginPage.IsLoaded.Should().BeTrue("we successfully logged out, so the login page should be visible");
34
35              //test login with Locked Out user
36              productsPage = loginPage.Login("locked_out_user", "secret_sauce");
37              productsPage.IsLoaded.Should().BeFalse("we used a locked out user who should not be able to login.");
38
39              //test login with Problem user
40              productsPage = loginPage.Login("problem_user", "secret_sauce");
41              productsPage.IsLoaded.Should().BeTrue("we successfully logged in and the home page should load.");
42              productsPage.Logout();
43
44              //test login with invalid username
45              productsPage = loginPage.Login("fake_user_name", "secret_sauce");
46              productsPage.IsLoaded.Should().BeFalse("we used a invalid username who should not be able to login.");
47
48              //test login with invalid password
49              productsPage = loginPage.Login("standard_user", "fake_pass");
50              productsPage.IsLoaded.Should().BeFalse("we used an invalid password, so the user should not be able to login");
51
52              //validate that all products are present
53              productsPage = loginPage.Login("standard_user", "secret_sauce");
54              productsPage.IsLoaded.Should().BeTrue("we successfully logged in and the home page should load.");
55              productsPage.ProductCount.Should().Be(6,
56                  "we logged in successfully and we should have 6 items on the page");
57
58              //validate that a product can be added to a cart
59              productsPage.AddToCart(Item.Backpack);
60              productsPage.Cart.ItemCount.Should().Be(1, "we added a backpack to the cart");
61
62              //validate that user can checkout
63              var cartPage = productsPage.Cart.Click();
64              var checkoutOverviewPage = cartPage.Checkout().
65                  FillOutPersonalInformation();
66              checkoutOverviewPage.FinishCheckout().IsCheckoutComplete.
67                  Should().
68                  BeTrue("we finished the checkout process");
69          }
```

- **原子测试可以快速失败**

为什么要使用原子测试？首先，它让你尽早地快速地失败。这就意味着你将得到及其快速的有针对性的反馈。如果你想检查一个特征功能的状态，原子测试只会花费你不到一分钟的时间就能让你拿到结果。

- **原子测试可以降低薄片行为**

第二，写原子测试用例减少了薄片，因为它降低了测试过程中可能出现的断点数量。薄片是自动化测试领域的一个概念，它是指测试展示出一种无效的测试结果，例如原本失败的用例结果显示通过了或者原本通过的用例结果显示失败了。一个没有薄片的自动化测试是一个可以正确传达系统状态的测试---无论被测系统是否正常运行。

- **原子测试考虑到了更好的测试**

原子测试第三点好处就是一旦一个原子用例失败了，它不会阻止你进行其他功能的测试。比如，一个普通测试在第 25 行失败了，假如你不手动执行测试用例去验证剩余的功能话，那么在 25 行之后的功能点都不会被测试到。所造成的结果就是，如果你正进行大范围的测试，由于有一些特性没有被测试到，实际上就降低了测试覆盖率。

下图是一个从上面的测试集中摘取的原子测试用例，如果这个用例失败了，你仍然可以接着测试其他功能点，这样整个测试过程不会中断，自然也不会影响到测试覆盖率。

```
26          [Test]
27          public void ShouldBeAbleToLoginWithValidUser()
28          {
29              _loginPage.Open();
30              var productsPage = _loginPage.Login("standard_user", "secret_sauce");
31              productsPage.IsLoaded.Should().BeTrue("we successfully logged in and the home page should load.");
32          }
```

2. 测试必须是自主的

一个自主的自动化测试是指测试本身并不依赖其他测试的结果。一种常见的反面模式就是将所有的用例连接到一起执行以避免重复执行一些测试步骤，这种反面模式是从加速执行测试集演变而来的。

例如，你要先执行登陆测试、再执行搜索测试、最后执行结算测试。为了进行结算场景的测试，所有的测试项目都应该按照正确的顺序进行。

这对于平行化来说就是一个很大的挑战了，因为它意味着你的测试永远不可能失序的进行。如果不按顺序执行第一步到第三步，那测试结果将失---但是这种失败并不是因为应用系统里有 BUG，而是设计逻辑本该如此，是因为结算的场景里需要一个已登录的用户，而具体测试时由于打乱了三步的顺序致使登陆操作是失败的。

3. 正确管理你的测试数据

测试数据的管理对完成并行测试来说是非常关键的。管理测试数据最为有效的方式就是使用实时数据。这就意味着你在测试过程中有很多种方式来创造和销毁数据。一个 RESTFUL 风格的 API 接口可以实现这一功能，并且这是最优的方法。

遗憾的是：在我的 WEB 应用里没有现成的 RESTFUL 风格的 API 可供使用。因此，我和开发人员一起合作，容许 JavaScript 注入来控制应用程序的状态和数据。这样我们测试团队就可以解决如何在不需要事先登陆、搜索商品、添加商品到购物车的情况下进行购物车结算了。

通过适当的测试数据管理，我可以直接打开购物车页面。结果就是我可以旁路掉/忽略掉登陆页面和搜索商品页面。如下图所示我注入了一个用户和一个商品到购物车里：

```
public CartElement SetCartState()
{
    ((IJavaScriptExecutor)_driver).ExecuteScript("window.sessionStorage.setItem('session-username', 'standard-user')");
    ((IJavaScriptExecutor)_driver).ExecuteScript("window.sessionStorage.setItem('cart-contents', '[4,1]')");
    _driver.Navigate().Refresh();
    return this;
}
```

现在我就可以使用应有程序的 UI 界面来完成结算过程并且测试这一功能是否 OK，如下图：

```
[Test]
public void ShouldBeAbleToCheckOutWithItems()
{
    //Arrange
    var shoppingCartPage = new YourShoppingCartPage(Driver).Open();
    //We don't need to actually use th UI to add items to the cart.
    //I'm injecting Javascript to control the state of the cart
    shoppingCartPage.Cart.SetCartState();
    //Act
    var checkoutOverviewPage = shoppingCartPage.Checkout().
        FillOutPersonalInformation();
    //Assert
    checkoutOverviewPage.FinishCheckout().
        IsCheckoutComplete.Should().BeTrue("we finished the checkout process");
}
```

最后，这种方式能够让你有能力旁路掉/忽略掉应用系统中任何与测试无关的 UI 界面。在你使用 API 或者 JavaScript 操作应用程序的状态后，你就可以使用任意的 UI 自动化工具执行相关的测试步骤。

4. 不能使用静态关键字

这里有一个一般性的经验法则：在你的自动化代码里不要使用静态关键字。这既是最简单的需要遵循的规则，同时也是最危险的。

在面向对象的程序设计语言中一个静态关键字会要求程序在代码运行期间内给一个变量分配一个单独的内存空间，这就意味代码运行期间这个空间的内容不能改变。任何使用静态变量的程序都将信息存储在单独的内存空间里。

一个单一的错位的静态关键字实例会毁掉你所有并行测试的希望。也许这个经验法则（指“不要使用静态关键字”）会有很多例外情况，但是确保安全总比意外发生后懊悔不已好得多。

例如，在你的浏览器中创建一个静态的浏览器驱动关键字，当你尝试并行地运行所有

测试用例时，这些用例都将尝试使用唯一的浏览器驱动。就会造成一种局面：一个用例正在准备键入账号密码时，另一个用例尝试打开一个不同的页面等等，结果这些用例会将浏览器驱动实例带到不同的方向上去，最终造成各种稀奇古怪的错误。

使测试又快又准确

自动化需要快速地进行才能发挥它的作用，同时并行测试是快速执行完测试用例的一种最好的方式。但是并行运行自动化测试的时候，你需要原子的、非依赖性的用例，并且这些用例里不能包含任何静态关键字，并且你需要正确地管理测试数据集。

影响软件测试未来的 5 件事

◆作者：枫叶

摘要:
从我们看待软件、评估风险、考虑复杂性、设计我们的测试方法和策略，以及帮助向用户发布一个稳定的产品的方式来看，技术已经对我们测试软件的方式产生了影响。而且这种影响只会随着技术进步而持续。在高层次上，我们已经看到了决定软件测试的未来的 5 件重要事情。

在过去的几十年里，软件测试已经根据用于执行不同活动的工具和使用这些工具的人的心态发生了变化。那时用于软件测试的工具很少，但是现在我们有很多的工具可以选择，从专有的到开源的。同样地，人们开始把软件测试者当作信息代理者而不是看门人，并且在敏捷的世界中已经出现很多积极的开发团队，这些开发对团队在软件开发生命周期中遵循的流程进行了重要的更改。技术的进步要感谢这些进化。

从我们看待软件、评估风险、考虑复杂性、设计我们的测试方法和策略，以及帮助向用户发布一个稳定的产品的方式来看，技术确实对我们测试软件的方式产生了影响，并且这种影响将只会随着技术的进步而继续。在高层次上，我们已经看到将决定软件测试未来的 5 件重要事情。

1. 人工智能

大约 5 年前，每个人都在谈论“移动优先”，并为用户提供使用手机网页、本机和混合应用程序的移动体验。现在，新的流行词是人工智能。它在自动驾驶汽车、家庭助理（人们当然喜欢 Alexa）、计算机视觉、健康保健、金融，以及现在的软件测试领域都有使用。

现在，在市场上很少有可靠的工具使用机器学习来帮助编写程序和执行功能测试、端到端测试和回归测试。它们主要集中在基于用户界面的测试自动化——用户创造的测试越多，算法变得越智能，这使得测试更稳定。

幸亏有人工智能，有一些我们可以期望开始看到的在测试中的好处:

- 更容易编写测试代码
- 降低测试脚本的维护工作
- 减少片状测试
- 使非技术人们开始进行自动化
- 更容易集成 CI/CD
- 更多可复用测试

举个例子，我用 Cucumber、 Java 和 Appium 构建了一个自动化框架。虽然我有一个健壮的框架，并且在编写自定义代码来执行各种操作时具有很大的灵活性，但我经常遇到维护方面的常见问题。当开发人员更改我的自动化测试已经覆盖的元素的属性时，测试开始失败。结果，我花了很多时间来维护这些测试，而不是编写新的自动化代码来覆盖实现的新功能。

这个问题现在可以通过使用人工智能从文档对象模型提取的动态定位器来解决。在实时的情况下，人工智能分析会分析 DOM 中的所有对象树和属性，并为特定元素创建不同属性的列表。因此，当一个元素的属性变化时，人工智能会尝试进入列表中的下一个属性来定位该元素，并一直遍历列表，直到找到该元素为止。这种测试更加稳定，测试程序的编写和执行速度会快得多，而且测试者在维护上花的时间会更少。

2. 开发运营

开发运行帮助软件开发团队和运营团队更好地协作，从而确保在整个软件开发生命周期(SDLC)中有持续的自动化和监视，包括基础设施管理。

您可能会问，这将如何影响软件测试？答案是：作为测试的一部分，我们所做的一切都会改变。我预计的变化包括：

- 需要在软件开发生命周期的开始时就启动自动化，并且确保几乎所有的测试用例都是自动化的
- 所有的质量保障工作都需要对齐，以确保 CI/CD 周期的顺利进行
- 需要高水平的协作，以确保在生产环境有持续的监控
- 所有的 QA 环境都需要被标准化

- 测试思维从“在此模块上完成测试”转变为“在发布候选版本中已经减轻了哪些业务风险？”

以上所有变化的关键是自动化。开发运营和自动化手携手并进——缺少其一，另一个将无法工作。这就是聪明的人类和工具能帮助缩短和更可靠的发布周期的地方。

我曾在一家公司工作，那里的开发、测试、运营团队之间的协作很少。我们在软件开发生命周期里发现了很多缺陷，比如更多的 bug 进入生产环境，不稳定的 CI/CD 基础设施，以及对生产监控和统计的不可见性。注意到这些差距，团队决定实施开发运营实践，每个人都开始在软件开发生命周期的每个阶段进行协作和贡献。这从需求收集开始，一直扩展到产品发布和监控上。

这种增强的协作文化开始对团队士气产生积极影响，更多自动化开始产生，整个团队开始作为一个单元一起工作。

3. 质量保证即服务

就像我们有软件即服务、基础设施即服务、平台即服务一样，我们现在也有质量保障即服务。在过去的几年里，这已经成为公司满足软件测试需求的一种流行方式。

拥有质量保障即服务解决方案的公司可以通过以下方式使软件测试过程的不同方面变得更简单:

- 测试用例管理和维护解决方案
- 测试自动化工具，减少编码需求
- 强大的测试报告功能，包括日志、视频录制和屏幕截图
- 易于与 CI 系统集成

在过去 7 年做自动化的过程中，像手机、虚拟机、安全网络和测试人员等资源里，我经常遇到的一个大问题是，必须维护自己的服务器来运行自动化测试。服务器的机器有不同的问题，如存储空间，一个片状的互联网连接，处理速度慢的测试正在运行持续整个星期，和需要频繁更新的最新操作系统，构建工具，安全补丁、集成开发环境等等。这些问题可以通过质量保障即服务的提供商解决，因为他们可以为您完成所有这些活动，因此团队成员可以将精力集中在更关键的任务上。

将来，质量保障即服务的供应商将考虑更多的方法来改进他们的产品，以保持领先于

他们的竞争对手，这也将使软件测试人员受益。

4. 物联网

随着可穿戴设备、智能家居、联网汽车和其他基于云技术的出现，物联网已经开始成为一个大的讨论的主题。这些设备的惊人之处在于，每秒钟都有如此多的通信和集成发生。

让我们来分析一下，在高水平上，可穿戴健身追踪器发生不同通信。首先，手机 app 和健身追踪器需要相互沟通。你的移动应用程序捕获的数据与该应用程序的桌面、移动 web 和平板电脑版本无缝集成，所有这些跨设备的通信都应该实时发生。所有的数据都在云、设备和应用程序之间来回传输。人们还可以通过应用程序组成小组，互相竞争，所以这些计算和通信也需要实时进行。根据触发的不同事件，需要在正确的时间向正确的用户发送正确的通知。所有这些通信都发生在互联网上。

假设您是测试这个健身跟踪器的测试人员。从哪里开始呢？您将如何设计您的测试策略和方法？

物联网将其自身的复杂性引入软件测试。它将影响我们对测试的看法，特别是因为集成测试需要比单独测试每个组件的旧方法给予更多的关注。

举个例子，当我在一家旅游预订公司工作时，我们为 Apple Watch 开发了一款新的应用程序，它使用的是 WatchOS (Apple Watch 最初由 Apple 推出)。该应用程序具有有限但有用的功能，比如查看通知和奖励信息、预定以及定位酒店、航班和租车位置的能力。在测试这个应用程序时，我注意到当 Apple Watch 应用程序连接到我手机上的同一个应用程序时，出现了一些奇怪的问题:当我将手机上的应用程序最小化时，Apple Watch 一片空白，只有一个黑屏;但当我再次在手机上打开应用程序时，黑屏消失了，Apple Watch 应用程序运行正常。

这是一个很好的例子，说明了集成测试的重要性。随着越来越多的设备进入市场，这对于组织和用户来说将是至关重要的。

5. 机器人

现在有做测试的机器人。有些人可能认为这是可怕的工作保障，但我仍然相信，人类的思想是无法取代的。仍然需要人类来监控机器人，以确保它们在做人们期望它们做的事情，并为它们编写程序。这种可扩展性有多强？只有时间才能证明。

总之，技术的进步已经开始影响我们进行软件测试的方式。这也导致公司重新思考他们的组织结构：QA 团队正在向嵌入开发团队的方向发展，并且整个团队将拥有质量。研究和开发团队与开发团队的频繁互动也将变得非常重要，以使产品更智能，对客户更有用。

还需要有处理大量数据的程序，以及适当的计算能力来梳理这些数据以获得有用的信息和反馈。最后，为了使这一切成为现实，公司需要采用精益流程，并且更加透明，以防止成为创新的一个障碍。精益转型对有效增长至关重要。

重要的是改变我们看待系统的思维方式，并相应地进行测试。我们可以选择忽略它，也可以选择接受它。你将会怎么做?

《51 测试天地》(五十三) 上篇 精彩预览

- 开源性能测试工具大比武
- PYSNMP 模拟器实战
- 记一次难忘的腾讯面试经历
- 机器学习与数据挖掘十大经典算法之 PageRank 算法
- 测试人员不得不小心那些职场套路
- 软件测试证书知多少
- 由测试的历史追溯测试发展
- APP 崩溃类问题总结
- 打破测试惯例

马上阅读