目 录 (上 篇)

联系邮箱：editor@51testing.com

测试女巫之控制 Windows 上的软件篇

◆ 作者：王平平

导语： Pywinauto 此模块是使用 Python 语言，它是用于 Windows 操作系统 GUI 的测试，它主要是通过向 windows 对话框和控件发送可以实现鼠标，键盘动作来实现 windows 图形界面的自动化测试。它与另外两个模块：Pymouse 和 Pykeyboard 的区别是，对于操作对象，pywinauto 是可以通过对象的属性：例如路径，title, class name 来调用；但是 Pymouse 和 Pykeyboard 只能通过屏幕坐标的方式调用，所以可以参考第 32 期杂志中"搭建嵌入式产品自动化测试框架全过程"中使用 Monkeyrunner 的思路：对于需要操作的对象尽量不要使用坐标的方式调用，所以我们选择 Pywinauto 这个模块而不是较为简单的 Pykeyboard 和 Pymouse。

一、前言：

上一期我们总结了近几年的 Python 学习心得，其中给出了学习 Python 的历程，最重要的是给出了学习的方法，这个方法主要是抽象出来的“精华”，我们就靠着这些“精华”学习了 11 个新模块，且以 Pywinauto 为例概要说明如何学习此模块。这次我们再以 Pywinauto 为例，详细地说明如何学习这个模块，尤其是包含在此模块中的各个控件如何学习，当然女巫的一贯原则是：“授人以鱼不如授人以渔”，所以重点介绍的部分也是学习方法。这些学习方法中值得大家关注的是：如何筛选学习这个模块中的众多控件，以及这些控件该如何学习。后续大家可以根据自己的需求选择学习你所需要的模块，毕竟每个公司的情况不一样，需求不一样。我们通过 11 个模块的自学（具体是哪些模块，可以参看第 41 期杂志中的内容），发现自学这些模块，Follow 这些做法都可以比较顺利的学习，而且对于我们公司的自动化需求，Python 几乎都可以满足，所以我们的这套方法是很有价值的，所以大家一起启动学习模式，一起为改变枯燥的工作努力吧！

二、第一阶段：工作需求

公司不是学校，不能因为你感兴趣而学习，学习是需要理由的！目前女巫公司的老板已经认可了我们目前的学习成果：当然老板在乎的所谓“学习成果”就是：“可以在短期内为公司节约成本”，所以老板也赞同我们不断地学习，但是老板让女巫的团队在

汇报学习内容时，需要将学习内容与工作需求进行一一链接。也就是说每一项学习内容，必须有“工作需要”这个驱动进行驱使，否则老板不会允许你“肆无忌惮”地“利用工作时间”去学习，因为没有回报的学习对于老板来说与“肆无忌惮”地玩游戏没有任何区别！

所以对于 Pywinauto 的“工作需求”，女巫总结如下：

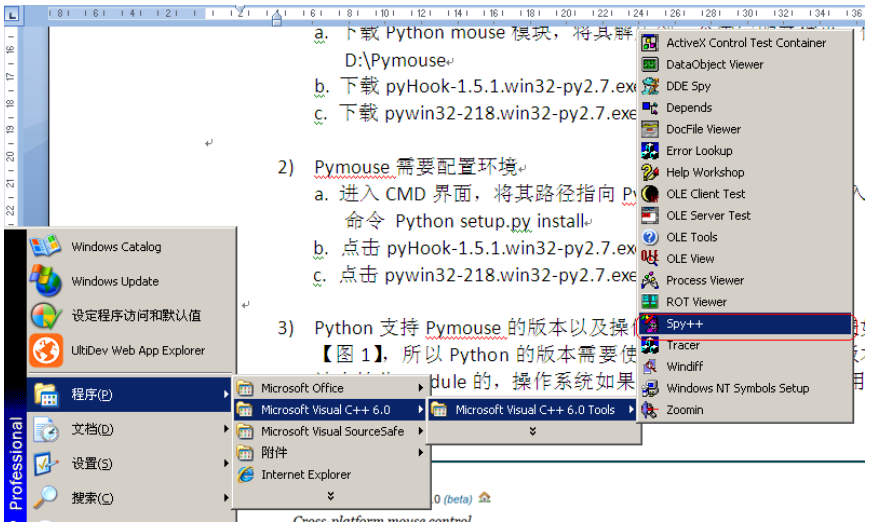
1. 在测试路由器时，通过在第 33 期学习的模块：“Selenium”无法控制的一些页面的处理，这些页面的控制需要用到 Pywinauto 这个模块。
2. 在测试路由器时，需要调用控制面板的一些应用，控制面板需要用到 Pywinauto 这个模块。
3. 需要测试厂商或者工厂提供的 Tool 的稳定性，需要做枯燥的压力测试。帮助我们脱离这个“傻傻”的压力测试工作，也要用到 Pywinauto 这个模块。
4. 测试一些 AT Command 类似的项目时，需要调用类似 Putty 这样的软件。调用这个软件，并控制 Putty 这个软件，进行输入命令，并抓取命令的 response，最后比较 response 和 Spec 是否一致，得出测试结果。这一系列操作也必须用到 Pywinauto 这个模块。

三、第二阶段：Spy++工具学习

Pywinauto 的本质是通过先通过其属性，定位到 windows 对话框以及若干个控件，然后对这些控件发送一些类似鼠标和键盘的动作来完成自动化，所以首先第一步就是：如何定位到 Windows 的对话框以及对话框中的控件。我们完成第一步，定位首先需要学习 Spy++此工具，这个工具是供我们查询对话框及其控件属性的。下一步就是说明 Spy++如何安装，如何使用。

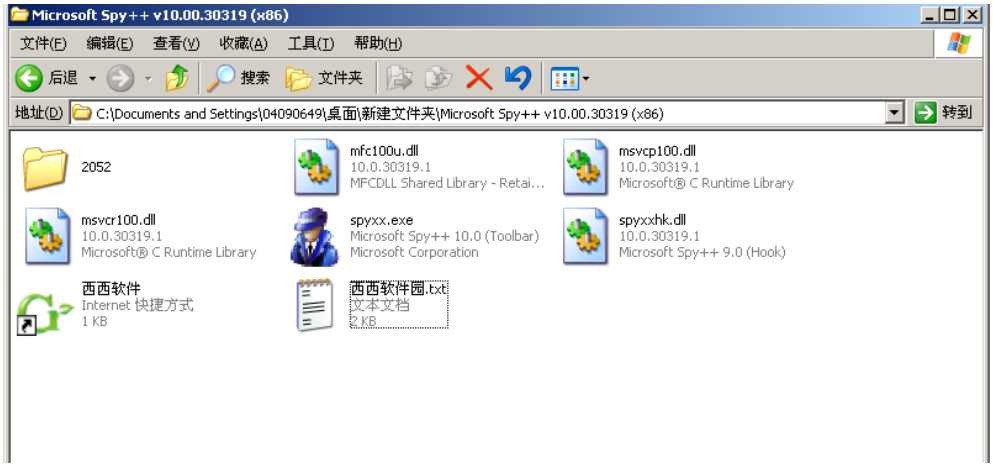
1. 配置开发环境安装说明

1) 第一种方式：安装 VC++环境，安装完毕后会在 Tools 中发现 Spy++此工具如【图 1】



【图 1】

2) 第二种方式：直接在网上下载此工具，如【图 2】



【图 2】

2. Spy++背景说明

1) Spy++是一个基于 Win32 的实用工具，它提供系统的进程、线程、窗口和窗口消息的图形视图。

2) 使用 Spy++可以做以下操作

a) 显示系统对象(包括进程，线程和窗口)之间关系的图形树

b) 搜索指定的窗口、线程、进程或者消息

c) 查看选定的窗口、线程、进程或者消息的属性

d) 直接从视图中选择窗口、进程或者消息

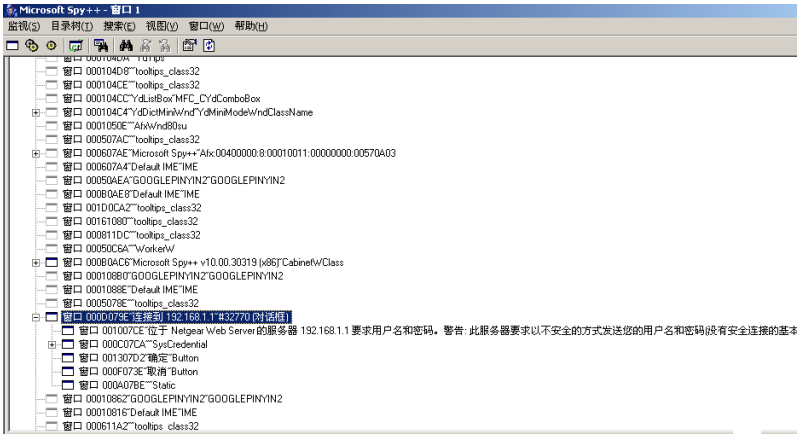
e) 通过鼠标定位，使用查找程序工具选择窗口

f) 右键视图中的某项，如果菜单中有”突出显示”此项目，则选择”突出显示”此项，打开需要定位的应用程序，就可以看出视图中的窗口对应的应用程序。

3. 工具使用说明

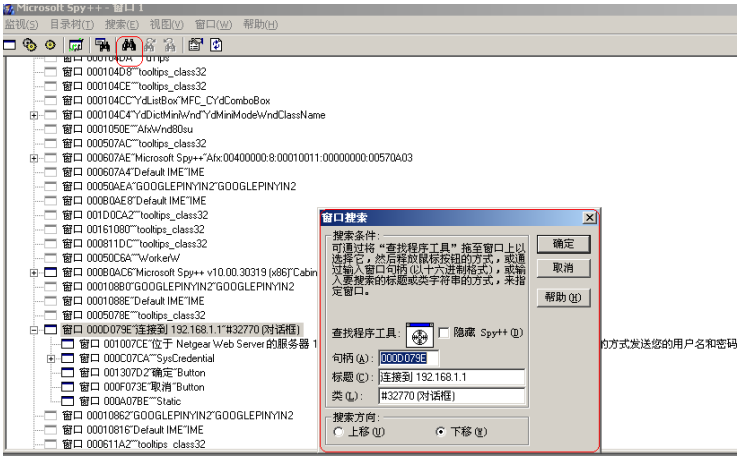
1) 对于此工具我们目前的需求主要是获得应用程序的属性：例如 Class Name，方法如下:

✓ 打开 Spy++此工具如【图 3】



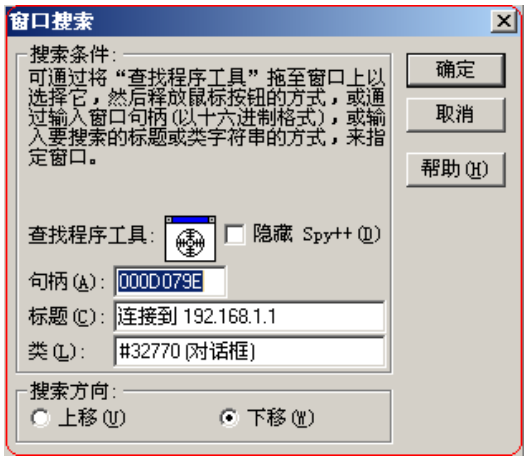
【图 3】

✓ 点击搜索按钮，会出现窗口搜索此子窗口，如【图 4】



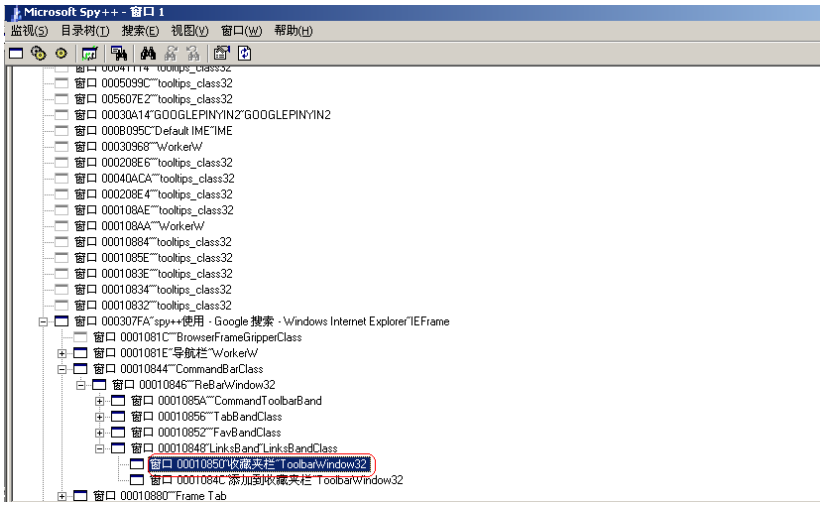
【图 4】

✓ 拖动查找工具的图标至相关的应用程序中，然后点击确认，如【图 5】



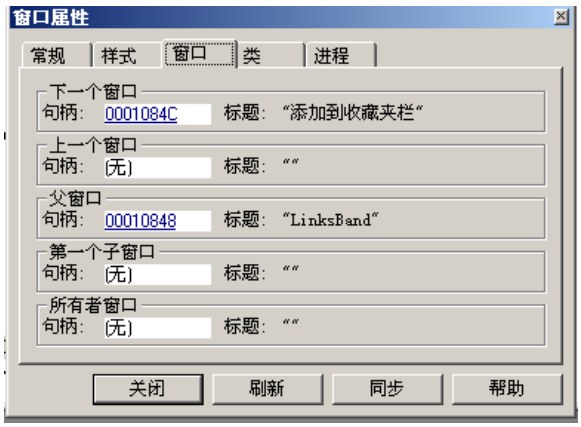
【图 5】

✓ 点击确认后会跳到此窗口的信息位置(即以高亮选中显示)，如【图 6】



【图 6】

✓ 对选中的窗口点击右键，选择属性，可以查到此窗口的属性，如句柄，类如【图 7】



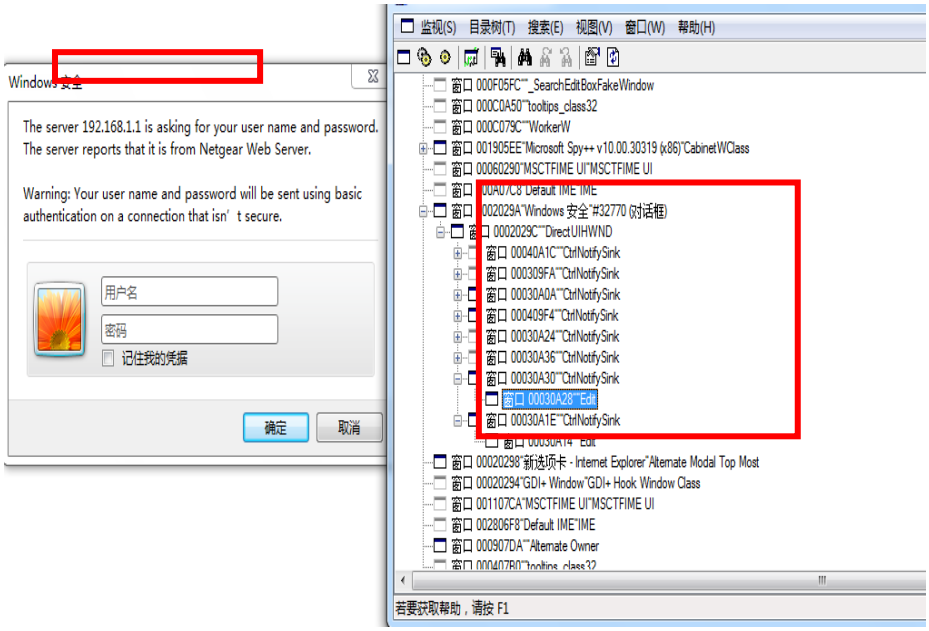
【图 7】

Notes：对于获得子窗口的相关参数，一定要注意：对于我们 Python 编程时无论是输入和输出的句柄均是十进制的，如果使用 Pyspy++此工具则获取的句柄即为十进制的；如果使用 Spy++此工具则获取的句柄为十六进制的；所以如使用 Spy++此工具，必须做一个“十六进制”转为“十进制”的额外工作

2) 获取目前窗口的层次，对于获取此窗口中指定的子窗口的句柄有非常重要的意义。此部分功能 Pyspy++是无法提供的，因为它只提供一个子窗口，所以无法获得目前窗口的层次

例如我想获得 OG2263 登录界面的窗口层次方法如下:

✓ 使用 1)中的方法将查找图标拖到登录界面的”windows 安全”位置如【图 8】，Spy++可以找到所有属于此登录界面的子窗口如【图 8】



【图 8】

可以根据 1)中的方法找到你希望定位的子窗口，例如我希望定位的子窗口为“用户名”和“密码”此两个窗口。使用

✓ 首先需要搞清楚这些子窗口之前的关系。

➢ 第一层为此窗口的窗体：即窗口名称为 0002029A “Windows 安全”#32770(对话框)

➢ 第二层为此窗体下第一层的窗口，即打开第一层窗体的“+”号后的第一层：即名称为 0002029C”” Direct UIHWND

- 第三层为第二层下的窗口，即打开第二层窗体的“+”号后的第一层，即上述窗口下有 8 个平行的窗口均为第三层窗口
- 第四层为第三层下的窗口，即打开第三层窗体的“+”号后的第一层，即第 7 个窗口打开后窗体，名称为 0003A28 “Edit”

✓ 然后了解这些窗口是如何编号的

|-0002029A “Windows 安全”#32770(对话框) ->index 无任何编号

|- 0002029C”” Direct UIHWND ->Index 为 0

|-窗口 1 ->index 为 0

|-窗口 2 ->index 为 1

|-窗口 3 ->index 为 2

|-窗口 4 ->index 为 3

|-窗口 5 ->index 为 4

|-窗口 6 ->index 为 5

|-窗口 7 ->index 为 6

|-Edit1 ->index 为 0

|-窗口 8 ->index 为 7

|-Edit2 ->index 为 0

四、第三阶段：PyWinauto 学习

1. 背景知识

Pywinauto 是适用 Window UI 自动化的模块，其实从它的名字就可以看出它的作用，“Py”代表 Python；“Win”代表可以控制 Window UI 上的软件；“Auto”代表可以自动化。

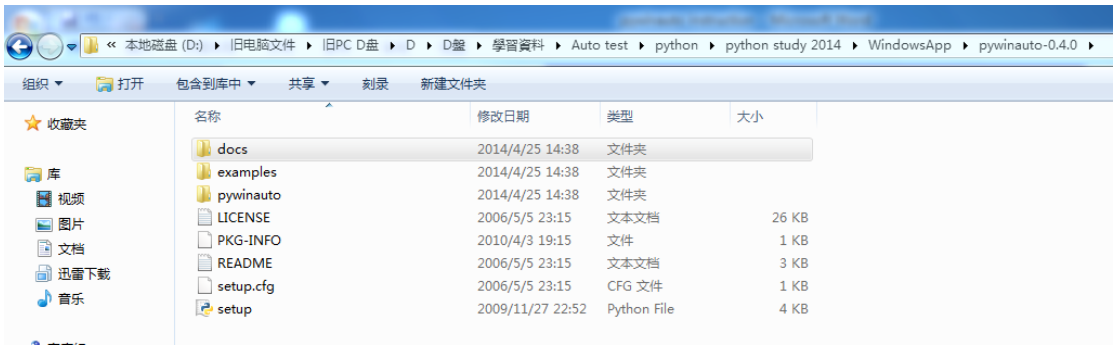
2. 安装步骤

此模块不是所有的 python 的版本都支持，Selenium 是支持 Python2.7，的所以目前我们的 Python 版本是 2.7 的版本，查看 Pywinauto 的官网如下：版本为 0.4.0 的 Pywinauto 支持的 Python 为 2.7 版本且操作系统为 windows(32-bit)，如【图 9】

【图 9】

安装模块

1) 下载 Pywinauto 后是一个文件夹如【图 10】

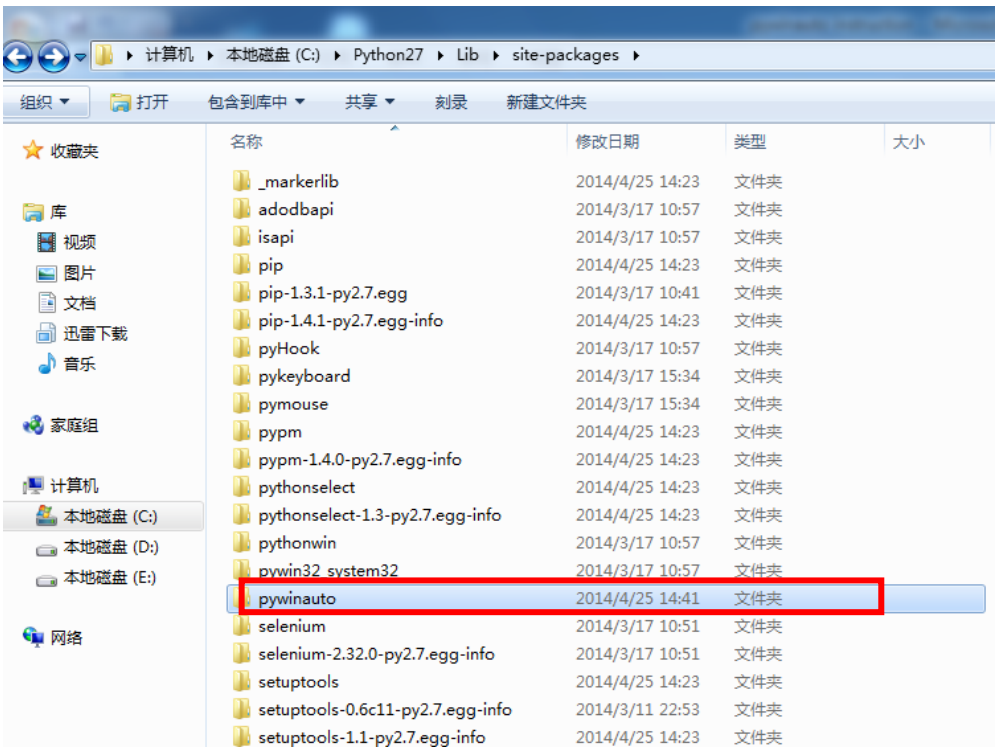


【图 10】

2) 进入 CMD->将路径指向 pywinauto 所在的文件夹，输入以下命令

Python setup.py install 即可以完成此模块的动作

3) 检查是否安装正常：进入 Python27->Lib->site-packages 路径下查看是否有 pywinauto 此文件夹如【图 11】



【图 11】

3. 连接到 Windows 上的应用软件

要控制 Windows 上的应用软件，首先要先要与其建立联系，就像你想控制一个人，首先你要与他有关系才有可能控制他吧？所以首先我们先学习如何与 Windows 上的应用软件建立联系的几个函数:

✓ **Application**

1) Application，它可以类比为 Monkeyrunner 中 MonkeyRunner，MonkeyDevice，MonkeyImage 这些子模块，需要调用其中的函数来实现操作 Windows 的各个应用程序。

2) Application 是一个实例，此实例是与所有你需要实现自动化操作的应用程序（也可以成为 Application，简称 App）连接的指针，你可以将其理解为属于 Application 模块的子模块。

3) Start: 可以理解为属于 Application 实例的函数

查看网上的函数可以看到有时此函数为 start，有时此函数为 start_

此两种形式的函数都是实现一个功能：当一个应用程序没有运行，你希望启用它，就需要使用这个函数。

官网对此函数的解释:

start_(self, cmd_line, timeout = app_start_timeout):

这两种形式的函数在使用时，会有一点不同:

a) Start 的使用

app= application.Application.start('notepad.exe')

b) Start_的使用

app= application.Application().start_('notepad.exe')

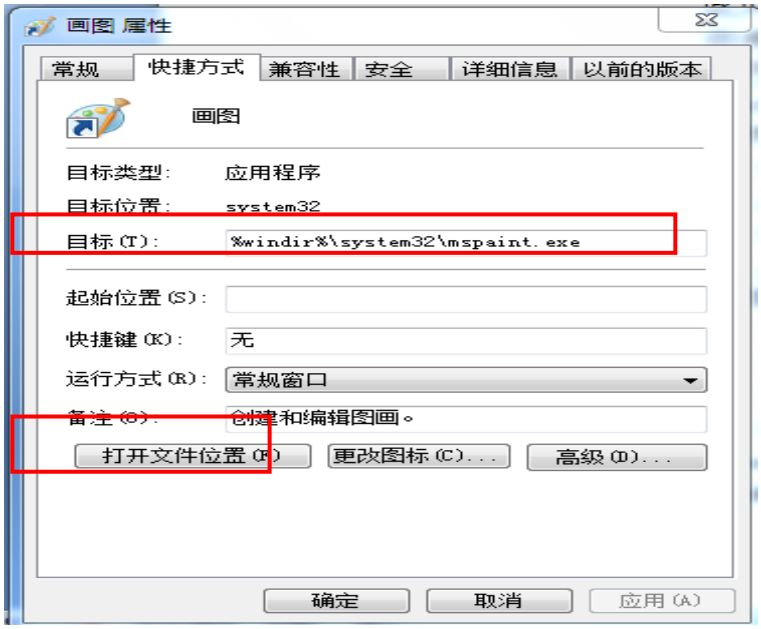
4) Connect: 同 start 一样可以理解为属于 Application 实例的函数

它是当一个应用程序已经运行了，希望能够连接到此应用程序

5) 使用此函数启动不同应用程序的方法_Start

a) 对于 Windows 操作系统本身自带的程序，例如计算器和记事本，如何确认是否为操作系统本身自带的程序:

右键应用程序(例如记事本，或者计算器或者画图)，选择“属性”，可以查看目标: 此软件是放在 system32 中，既可以判断为系统自带的应用程序，如【图 12】



【图 12】

也可以点击【图 12】中的打开文件位置发现它就在 system32 文件夹中

对于打开此类的应用程序，就可以使用打开记事本的方法：

app= application.Application.start('notepad.exe')或者

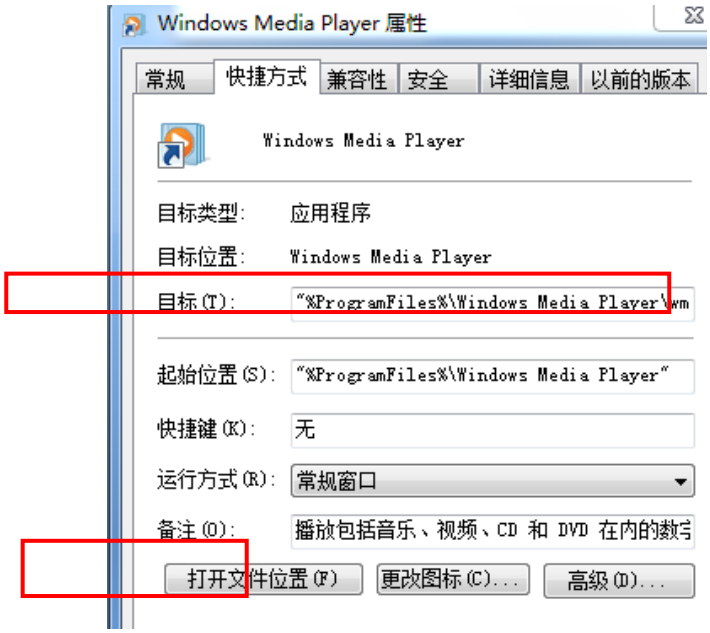
app= application.Application().start_('notepad.exe')

其中 start 括号中内容填写【图 12】中的，目标中路径的最后一段，例如对于画图此应用程序，

app= application.Application.start('mspaint.exe')或者

app= application.Application().start_('mspaint.exe')

b) 对于非 windows 操作系统自带的程序，即需要另外安装的程序，最直接的判断方法，看此应用程序的路径是否放在 system32 此文件夹中，如放在此文件夹即为操作系统。如【图 13】，Window Media player 是放在 Windows Media Player 此文件夹中就不是自带的程序。



【图 13】

✓ 对于此类的应用调用它的方法如下：

打开 Filezilla server 的步骤->

path=ur"C:\Program Files\FileZilla Server\\FileZilla Server Interface.exe"

app= application.Application.start(path)

说明:path 是此应用程序的路径，此路径在【图 13】点“打开文件位置”即可获得

此应用程序的路径

打开 Media player 的步骤->

```
path=ur"C:\Program Files\Windows Media Player\\wmplayer.exe"
app= application.Application().
```

4. 定位

当连接到应用程序后，就可以理解为将此应用实例化了，建立联系就可以开始进行进一步的较为深入的沟通了；接下来如果希望操作此应用程序，则需要做的工作是“标示应用程序的窗口”有 3 种方法应用程序的

1) 定位窗口:

a) 使用窗口标题，如

```
submenu=u‘插入’
Dialog=app[submenu]
```

还有一种方式，仅能使用在当标题全为英文:

Dialog=app.select(此时 select 即为标题名称，因为在实际运用中我们需要操作的应用程序不可能所有的标题都是英文，所以此种方式不推荐)

b) 窗口标题结合类名进行定位

dlg=app.window_(title_re="Windows Media Player",class_name="WMPlayerApp")这种方式需要通过 Spy++此工具去获得 title 和 class name 此部分知识在“Spy++ Tools Introduction”已经讲解过这里，就不再赘述。

c) 直接取最上层窗口

```
Dialog=app.top_window_()
```

此时要确保被标示的应用程序窗口为顶层窗口

推荐的方式是第 2 种，因为第三种有时会有无法预期的窗口会变为顶层窗口。

2) 定位应用程序窗口控件

窗口已经成功标示了，然后就可以定位窗口中的控件了。定位窗口中控件的方法如下:

a) 使用窗口控件标题

b) 使用 Friendly Class(这种方式当控件标题内容为空时尤为有用)

目前实践成功的是第 2 种方法，例子请看【图 14】

(1) 对于已经有标示的控件操作

此时就需要调用 pywinauto.controls 的函数，例如点击鼠标左键，输入内容等，例子请看【图 14】

【图 14】

(2) 对于使用 Class 定位窗口中的控件遇到问题及解决方案

✓ 遇到的问题

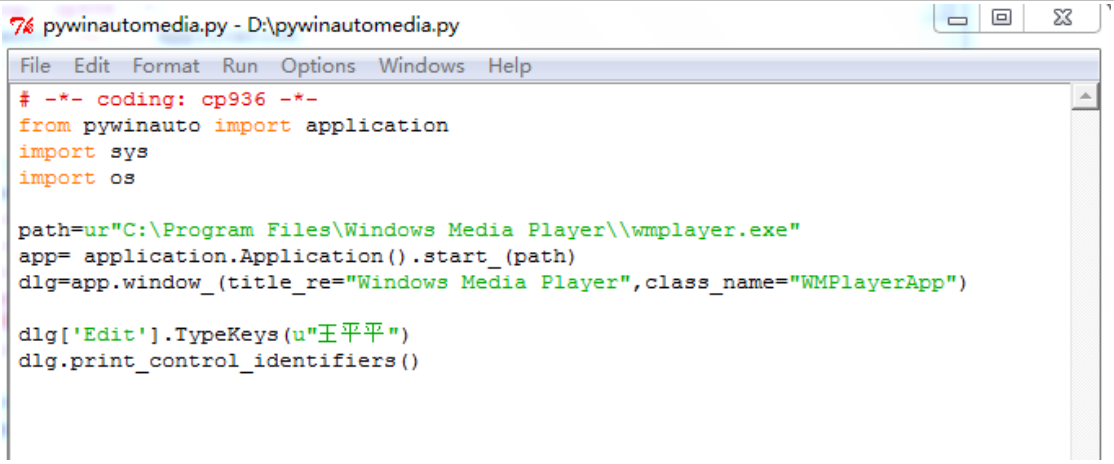
当一个窗口有相同 class 的控件若干个，那如何准确的定位到你希望定位的那个控件呢

✓ 解决方案

使用一个函数将当前定位窗口的所有控件的属性打印出来

Windowsname. print_control_identifiers()

例如【图 15】



【图 15】

✓ 打印出来内容的解读，如【图 16】

【图 16】

a) 相同 class 的不同控件的调用方法

Dialog.Button1.Click()

Dialog.ComboBox1.Select(1)

注意 Button，Button0 及 Button1 都是代表第一个 Button. 关于顺序，也要注意如果存在嵌入窗口，则其内部的子控件顺序是依据其在主窗口的顺序而定的

b) ‘’之间的内容为 Unicode 编码的中文 Title，你可以在网上搜索 Unicode 和中文之间转换的工具，将此编码转为中文

例如：'\u201c\u89c6\u56fe\u201d\u5de5\u5177\u680f'

'\u201c\u89c6\u56fe\u201d\u5de5\u5177\u680fToolbar'

转换为中文：'“视图”工具栏'，'“视图”工具栏 Toolbar'

c) 后面的（L，T，R，B）是这个控件的位置，分别对应着左上右下

d) 如果此控件的 Title 为空白，例如 Toolbar5 对应的 Title 即为空白，则它会查找此与此控件最为接近的 Title 或者 Class 名称，显示在 Title 的下一行

例如 Toolbar5 即为与 Toolbarwindow32 最为接近的控件 Class 名称

5. 如何控制软件中的控件

1) 先给出大家，女巫总结的方法，后续再以实例说明

a) 首先使用 Spy++得到此控件的名称

b) 上网搜索此控件的基本知识，要知道此控件的结构，这部分很重要，对于理解 Pywinauto 中此控件的函数非常有意义，例如 Treeview 中的 GetItem 此函数的变量，三种方式都提到了 Root(即根目录)，如果对于 Treeview 的架构不了解，就无法理解 Root 是什么。

c) 明确对于此控件，我们所需要实现的“灵魂功能”是什么，此步骤是为了在学习函数时，有很强的针对性，一个控件的函数挺多的，我们不可能每个函数都学习一遍，要有筛选。

d) 查看官网以及源代码，查看官网的目的是，了解此控件的所有函数，从中挑选与“灵魂功能”最靠近的函数。

e) 因为官网中对于每个函数的介绍非常少，如果对于官网的介绍还是一头雾水，有两种方法解决：第一是查看源代码，从源代码中可以得到比官网更多的信息。第二是对于这些函数进行实践，写个脚本去调用这些函数。有关源代码的功能介绍请看【图 17】

有兴趣的网友，还可以阅读其源代码，主要的几个程序是

```
Application.py:与app相关
Findwindows.py: 窗体的查找
Timings.py: 各种操作的时间间隔设置
win32_controls.py: 普通控件的访问
common_controls.py: TreeView/ListView...等控件的访问
Menuwrapper.py: 菜单的访问
```

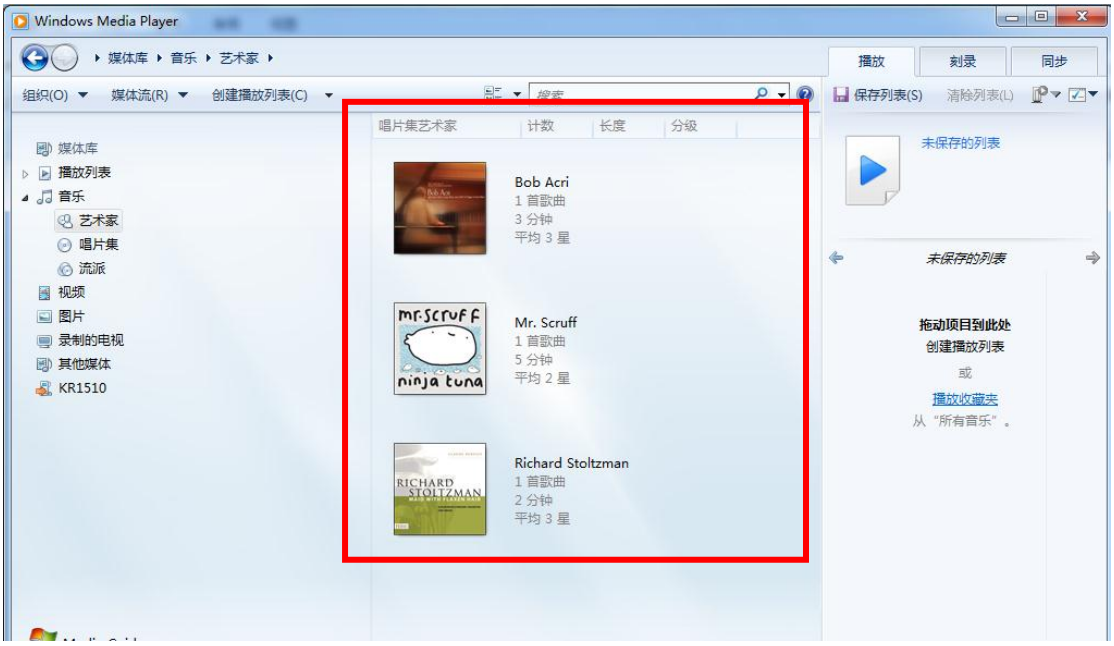
【图 17】

Win32_controls.py 和 Common_controls.py 包含了所有控件的类和函数。

2) 实例说明:

目的: 实践学习 Treeview 此控件总结学习方法

(1) 首先使用 Spy++得到 Media player 的一部分为 Listview 控件，如【图 18】



【图 18】

(2) 上网搜索 Listview 此控件的用处

Listview 控件是用来显示各项带图标的列表，如【图 18】，也可以用来显示带子项的列表如 Windows 操作系统的资源管理器如【图 19】显示

【图 19】

它的最重要的属性是 View， 此属性决定了以哪种视图模式显示控件的项，这四种模式分别为 LartIcon(大图标视图模式) ，SmallIcon(小图标视图模式)，List(列表视图模式)，Detail(详细资料视图模式)

【图 17】为大图标模式，【图 19】为列表视图模式

a) Items 属性：它是 ListView 中很重要的属性，它包含了控件所有存在的项。

b) Column 以及 ListItems 的属性介绍：

姓名 性别 年龄 成绩

小红 女 13 87

小王 男 14 69

小张 女 15 56

此时 ColumnHeaders 的集合就是上面的姓名，性别，年龄，成绩

而单个的如姓名，就为 ColumnHeader 对象。小红，小王，小张所组成的集合为 Listitems 集合。后面的“女”，13 87 则为 listsubitems 集合。

可以这样理解 Column 是不用于被用户编辑的，而 Item 是可以被用户编辑的

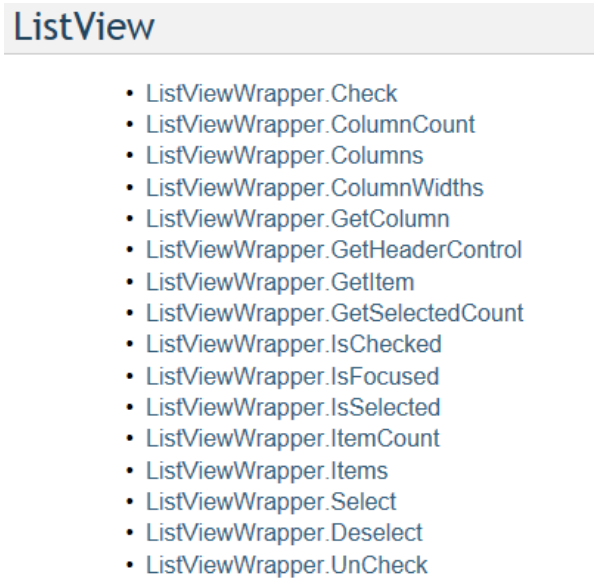
3) 明确对于此控件，我们所需要实现的“灵魂功能”是什么。

定位带图标的栏位，或者带子项的列表，然后使用鼠标点击。

4) 查看官网以及源代码

(1) 官网如【图 20】，从我们学习的 Listview 的基本知识以及“灵魂功能”

可知：有关 Column 的函数并不是我们应该关注的，因为一般我们不需要点击 Column，如【图 18】我们一般没有点击“唱片艺术家”“技术”“长度”“分级”这些 Column 的需求，我们都是需要点击各个歌曲的图片的需求，这些就是 Item，所以我们应该重点关注有关 Item 以及与选中相关的项目，例如 GetItem，IsChecked 等



【图 20】

(2) 源代码查看疑似与我们的“灵魂功能”相关的函数：GetItem()如【图 21】

```
#--------------------------------------------------------
def GetItem(self, item_index, subitem_index = 0):
    """Return the item of the list view"

    * **item_index** Can be either the index of the item or a string
      with the text of the item you want returned.
    * **subitem_index** The 0 based index of the item you want returned.
      Defaults to 0.
    """

    item_data = {}

    # ensure the item_index is an integer or
    # convert it to one
    item_index = self._as_item_index(item_index)

    # set up a memory block in the remote application
    remote_mem = _RemoteMemoryBlock(self)

    # set up the item structure to get the text
    item = self.LVITEM()
    item.mask = \
        win32defines.LVIF_TEXT | \
        win32defines.LVIF_IMAGE | \
        win32defines.LVIF_INDENT | \
        win32defines.LVIF_STATE

    item.iItem = item_index
    item.iSubItem = subitem_index
    item.stateMask = ctypes.c_uint(-1)

    item.cchTextMax = 2000
    item.pszText = remote_mem.Address() + \
        ctypes.sizeof(item) + 1
```

【图 21】

5) 对于这些函数进行实践，写个脚本去调用这些函数。因为从官网或者源代码得到的讯息有限，我们要根据这些有限的讯息进行实践确认是否是我们要的函数。

(1) 首先我们先写一个简单的脚本用于验证此函数的作用，以及返回值。

如【图 22】中红框的内容，打印出来的结果，即此函数的返回值请看【图 23】

```
# -*- coding: cp936 -*-
from pywinauto import application
import sys
import os
import time
from pykeyboard import PyKeyboard
k=PyKeyboard()
path=ur"C:\Program Files\Windows Media Player\\wmplayer.exe"
app= application.Application().start_(path)
dlg=app.window_(title_re="Windows Media Player",class_name="WMPlayerApp")
ListView=dlg.ListView
print ListView.GetItem(2,0)
time.sleep(1)
rect=ListView.GetItemRect(2)
time.sleep(1)
ListView.ClickInput(coords=(rect.left,rect.top),double=True)
```

【图 22】

```
>>> ================================ RESTART ================================
>>>
{'text': u'Richard Stoltzman', 'state': 0L, 'indent': 0, 'image': 1}
>>>
```

【图 23】

所以返回的值为一个字典类型的数据结构。

(2) 对于我们已经学习的一个控件 TreeView 的函数 GetItem 的用法如【图 24】中红框标出内容，打印出来的结果，即此函数的返回值请看【图 25】

```
# -*- coding: cp936 -*-
from pywinauto import application
import sys
import os
import time

#path=u"control ncpa.cpl"
#path=ur'control netconnections'
path=ur'C:\Program Files\FileZilla\FileZilla.exe'
app= application.Application().start_(path)
time.sleep(1)
dlg=app.window_(title_re="FileZilla version 2.2.25",class_name="FileZilla Main Window")
time.sleep(1)
#dlg.print_control_identifiers()
TreeView=dlg.TreeViewWrapper
time.sleep(2)
print TreeView.GetItem([u'桌面',u'3gp',u'H.264+AAC'])
TreeView.GetItem([u'桌面',u'3gp',u'H.264+AAC']).Click()
```

【图 24】

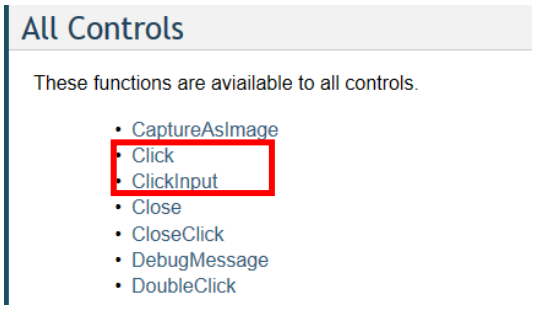
```
>>> ================================ RESTART ================================
>>>
<pywinauto.controls.common_controls._treeview_element object at 0x02262AB0>
>>>
```

【图 25】

所以返回值为一个对象。

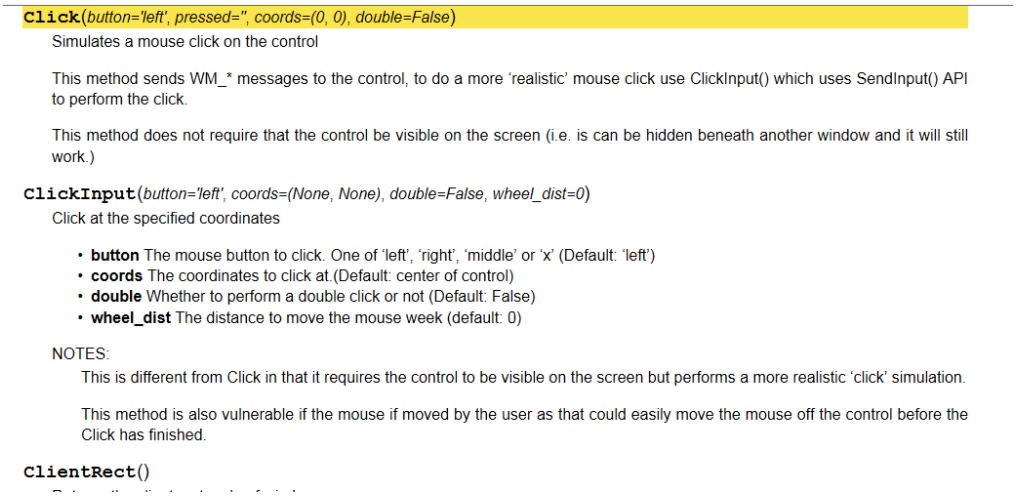
(3) 从上述两点可以看出虽然这两个控件有一个看起来一模一样的函数，但是返回值完全不一样，对于 TreeView 就可以直接使用 Click 函数，来实现我们的“灵魂功能”，因为它的返回值为一个对象。但是对于 ListView 就不能直接使用 Click 函数，因为它的返回值为一个字典数据结构。可以这样说对于 ListView， GetItem 就没什么实际的意义，因为即使得到了子 Item 也无法进一步操作，它返回的只是一个字典数据结构。

(4) 既然无法通过 GetItem 达到定位 Item 并进一步操作(点击)的要求，就需要找到另外一种定位并点击的方法。查看官网，如【图 26】



【图 26】

查看其详细说明如【图 27】



【图 27】

即从它们的解释可以看出，这两个函数本身就包含了定位(通过此两个函数的参数进行定位)，并点击(此函数本身包含了双击和单击)的作用，对于 TreeView 因为使用 GetItem 已经达到了定位的目的，所以在 Treeview 中所有参数都未设置，因为不需要定位。但是 TreeView 无法使用 GetItem 定位，所以必须使用参数进行定位的功能。

(5) 对于参数，最重要的是坐标：Coordinate。即如何获得子 item 的坐标，如果能获得子 item 的坐标就解决了这个问题。

在官网中的 ListView 函数列表中我们未能找到有关坐标的函数如【图 28】

ListView

- ListViewWrapper.Check
- ListViewWrapper.ColumnCount
- ListViewWrapper.Columns
- ListViewWrapper.ColumnWidths
- ListViewWrapper.GetColumn
- ListViewWrapper.GetHeaderControl
- ListViewWrapper.GetItem
- ListViewWrapper.GetSelectedCount
- ListViewWrapper.IsChecked
- ListViewWrapper.IsFocused
- ListViewWrapper.IsSelected
- ListViewWrapper.ItemCount
- ListViewWrapper.Items
- ListViewWrapper.Select
- ListViewWrapper.Deselect
- ListViewWrapper.UnCheck

【图 28】

按照之前的经验如果官网没有就找源代码，果然在源代码的 ListView 类中找到了一个这样的函数：如【图 29】

```
#-----------------------------------------------------------
def GetItemRect(self, item_index):
    "Return the bounding rectangle of the list view item"
    # set up a memory block in the remote application
    remote_mem = _RemoteMemoryBlock(self)
    rect = win32structures.RECT()

    rect.left = win32defines.LVIR_SELECTBOUNDS

    # Write the local RECT structure to the remote memory block
    remote_mem.Write(rect)

    # Fill in the requested item
    retval = self.SendMessage(
        win32defines.LVM_GETITEMRECT,
        item_index,
        remote_mem)

    # if it succeeded
    if not retval:
            del remote_mem
            raise RuntimeError("Did not succeed in getting rectangle")

    rect = remote_mem.Read(rect)

    del remote_mem

    return rect
```

【图 29】

(6) 根据上述的研究尝试一下实验，如【图 30】可以实现选择歌曲并双击。

```
# -*- coding: cp936 -*-
from pywinauto import application
import sys
import os
import time

path=ur"C:\Program Files\Windows Media Player\\wmplayer.exe"
app= application.Application().start_(path)
dlg=app.window_(title_re="Windows Media Player",class_name="WMPlayerApp")
ListView=dlg.ListView

print ListView.ItemCount()
time.sleep(1)
rect=ListView.GetItemRect(2)
time.sleep(1)
ListView.Click(coords=(rect.left,rect.top),double=True)
```

【图 30】

说明：先使用 ItemCount()查看界面上有多少 Item，打印出来为“3”。说明界面上有 3 个 Item，这与 UI 界面看到的一样，如【图 31】

【图 31】

Item_Index 从 0 开始计数，第三个歌曲的 index 为“2”，获得的 Rectangle 有 4 个数字：top，bottom，right，left 一般使用两个数字就可以正确定到位：left 和 top

五、总结

第 41 期女巫介绍了 Python 的学习历程，介绍了学习一个模块的学习方法，这一期我们继续介绍学习方法，我们这一期介绍的是 Pywinauto 这个模块中的控件类如何学习，当然不是按部就班的将官网和源代码的所有内容一一学习一遍，而是，如何根据你的需求进行有针对性的学习。

就像前言中所说，秉持着“授人以鱼不如授人以渔”的女巫，一直不满足将知识点讲完即可，而是一定要总结出一套方法，大家一定要仔细看“如何控制软件中的控件”这一章节，这个章节的方法真的是经过女巫以及女巫的团队验证过的，真的是屡试不爽，按照这个方法学习 Pywinauto 的所有控件类都没有问题，由于篇幅限制，这里只讲解了“Listview”这个控件如何学习。事实上对于“Treeview”，“Toolbar”，“Menu Item”这些控件都可以按照这个方法进行学习，建议大家按照这个方法学习一个控件类，如果能搞定一个控件类会大大提升你的信心，而且还提高了自学的兴趣。

我们不停找到工作中可以改善的问题，不停的解决这些问题，最重要的要不停的在总结解决这些问题的方法，久而久之我们就会觉得没有什么问题是“不可解决的”！

Jenkins 持续集成实践

◆ 作者：姜林斌

何为持续集成？

持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试)来验证，从而尽快地发现集成错误。许多团队发现这个过程可以大大减少集成的问题，让团队能够更快的开发内聚的软件。

持续集成的价值:

>减少风险

一天中进行多次的集成，并做了相应的测试，这样有利于检查缺陷，了解软件的健康状况，减少假定。

>减少重复过程

减少重复的过程可以节省时间、费用和工作量。说起来简单，做起来难。这些浪费时间的重复劳动可能在我们的项目活动的任何一个环节发生，包括代码编译、数据库集成、测试、审查、部署及反馈。通过自动化的持续集成可以将这些重复的动作都变成自动化的，无需太多人工干预，让人们的时间更多的投入到动脑筋的、更高价值的事情上。

>任何时间、任何地点生成可部署的软件

持续集成可以让您在任何时间发布可以部署的软件。从外界来看，这是持续集成最明显的好处，我们可以对改进软件品质和减少风险说起来滔滔不绝，但对于客户来说，可以部署的软件产品是最实际的资产。利用持续集成，您可以经常对源代码进行一些小改动，并将这些改动和其他的代码进行集成。如果出现问题，项目成员马上就会被通知到，问题会第一时间被修复。不采用持续集成的情况下，这些问题有可能到交付前的集

成测试的时候才发现，有可能会导致延迟发布产品，而在急于修复这些缺陷的时候又有可能引入新的缺陷，最终可能导致项目失败。

>增强项目的可见性

持续集成让我们能够注意到趋势并进行有效的决策。如果没有真实或最新的数据提供支持，项目就会遇到麻烦，每个人都会提出他最好的猜测。通常，项目成员通过手工收集这些信息，增加了负担，也很耗时。持续集成可以带来两点积极效果:

(1) 有效决策：持续集成系统为项目构建状态和品质指标提供了及时的信息，有些持续集成系统可以报告功能完成度和缺陷率。

(2) 注意到趋势：由于经常集成，我们可以看到一些趋势，如构建成功或失败、总体品质以及其它的项目信息。

>建立团队对开发产品的信心

持续集成可以建立开发团队对开发产品的信心，因为他们清楚的知道每一次构建的结果，他们知道他们对软件的改动造成了哪些影响，结果怎么样。

一：Jenkins 介绍和安装

Jenkins 是基于 Java 开发的一种持续集成工具，用于监控持续重复的工作，功能包括:

- 持续的软件版本发布/测试项目。
- 监控外部调用执行的工作。
- 跟其他持续集成相比，它的主要优点有：开源，即免费。支持多种平台（windows、linux、os x 都支持）。
- 安装、配置简单。Web 可视化管理界面，并且有丰富的 tips 帮助信息。

为什么要选择 Jenkins?

Jenins 是现在非常流行的持续集成 CI 服务器，这与它的前身 Hudson 也有着很大的关系，Jenkins 易于安装，不需要数据库的支持，直接通过 Web 界面进行配置，而且集成了 RSS/Email 的通知机制，支持分布式构建，具有丰富的插件，这些都是 Jenkins 相比其他持续集成服务器的优势所在。

对于版本控制软件的选择要看项目需要了，可能是 SVN 也可能是 Git，一般来说

Jenkins 都有提供插件支持。

1.1 CentOS 下配置 Jenkins

1.1.1 安装 jdk

新建目录 mkdir /usr/local/java

将 xvf jdk-8u77-linux-x64.gz 压缩包移动到/usr/local/java 目录下

解压 tar xvf jdk-8u77-linux-x64.gz

修改系统变量文件:

Vi /etc/profile

export JAVA_HOME=/usr/local/java/jdk1.8.0_77

export JRE_HOME=/usr/local/java/jdk1.8.0_77/jre

export CLASSPATH=.$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib:$CLASSPATH

export PATH=$JAVA_HOME/bin:$PATH

重新编译文件 使系统变量即时生效

Source /etc/profile

检查 jdk 是否安装成功

Java -version

1.1.2 安装 Jenkins

- 下载源码包

wget wget http://pkg.jenkins-ci.org/redhat/jenkins-1.656-1.1.noarch.rpm

- 安装 jenkins

rpm -ivh jenkins-1.656-1.1.noarch.rpm

- 启动 Jenkins: service jenkins start

可能会报错: Starting Jenkins bash: /usr/bin/java: 没有那个文件或目录

将/usr/local/java/jdk1.8.0_77/bin/java 添加到/etc/init.d/jenkins 文件中对应的位置

```
candidates="
/etc/alternatives/java
/usr/lib/jvm/java-1.6.0/bin/java
/usr/lib/jvm/jre-1.6.0/bin/java
/usr/lib/jvm/java-1.7.0/bin/java
/usr/lib/jvm/jre-1.7.0/bin/java
/usr/lib/jvm/java-1.8.0/bin/java
/usr/lib/jvm/jre-1.8.0/bin/java
/usr/bin/java
/usr/local/java/jdk1.8.0_77/bin/java
```

➢ 再次执行 service jenkins start 启动 jenkins 服务

1.2 Windows 下配置 Jenkins

安装过程太过简单 这里不做更多的说明，方式两种:

1: 将 jenkins.war 放到 tomcat 安装目录下的 webapp 下，启动 tomcat 即可。

2: 使用 java –jar 命令启动如下

```
//设置 jenkins 的工作目录
set JENKINS_HOME=E:\ToolsForTesting\Jenkins
cd /d %JENKINS_HOME%
//指定 jenkins 端口并启动 jenkins 服务
java -jar jenkins.war --httpPort=8080
```

二：Jenkins 插件介绍

2.1 Jenkins 集成 Svn

2.1.1 Svn 服务端部署

Step1： CentOS 上安装 Svn 服务端:

yum install svn

Step2： 创建版本库目录

mkdir -p /home/svn/svnfile

创建版本库

svnadmin create /home/svn/svnfile

生成以下目录

```
drwxrwxrwx 2 root root 4096 Mar  7 22:46 conf
drwxrwsrwx 6 root root 4096 Apr  7 18:24 db
-rw-rw-rw- 1 root root    2 Feb 28 22:42 format
drwxrwxrwx 2 root root 4096 Feb 28 22:42 hooks
drwxrwxrwx 2 root root 4096 Feb 28 23:21 locks
-rw-rw-rw- 1 root root  229 Feb 28 22:42 README.txt
```

Step3： 配置权限

进入 conf 目录下，在 passwd 文件中配置用户名/密码;

在 authz 文件中设置用户权限;

在 svnserve.conf 文件中配置 svn 服务。

```
-rw-r--r-- 1 root root 1098 Feb 28 23:31 authz
-rw-r--r-- 1 root root  322 Feb 28 23:04 passwd
-rw-r--r-- 1 root root 2272 Feb 28 22:44 svnserve.conf
```

Step4： 启动 SVN 服务

svnserve -d -r /home/svn/

2.1.2 Svn 常用命令

从服务器上签出：svn co url　(ps: co 为 checkout 缩写)

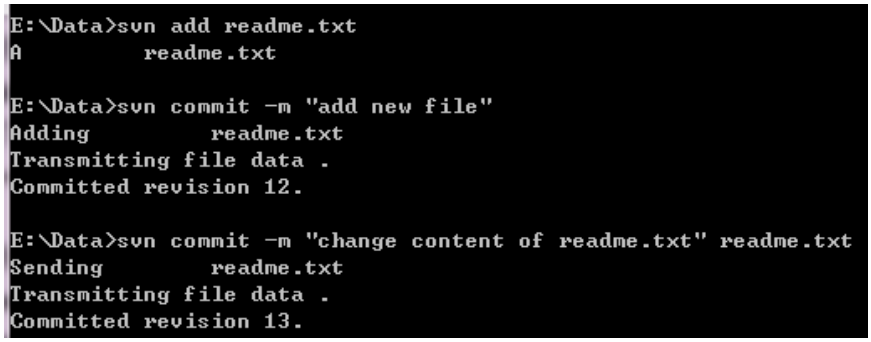
向服务器提交更新两种情况:

1) 添加新文件并提交到服务器

Step1: svn add file **Step2:** svn commit –m “comments”

2) 更改已存在的文件后提交到服务器

一步到位: svn ci –m “comments” file_name (ps:ci 为 commit 的缩写)



- 对文件/目录加锁&解锁

svn lock file_name(or path)

svn unlocl file_name(or path)

- 更新工作区

svn up(ps:up 为 update 的简写)

- 回退到指定版本

1) 先 svn up，保证更新到最新的版本。

2) svn log，查看历史版本，找出要恢复的版本。

3) 回滚到指定版本号 svn merge –r 最新版本好: 指定的回退版本号。

- 删除文件或分支

svn del/rm URL(or file) (ps: del 为 delete 缩写; rm 为 remove 缩写)

- 比较差异

Svn diff –r 版本号 1:版本号 B

- 将两个版本间的差异合并到当前文件

svn merge –r 版本号 A: 版本号 2 path

- 切换到分支

svn switch (tag/分支)URL

- 建立 tags

svn copy trunk_URL tags_URL –m "comments"

- 删除 tags

svn rm tags_URL

- 创建分支

svn cp trunk_URL branch_URL –m "comments"

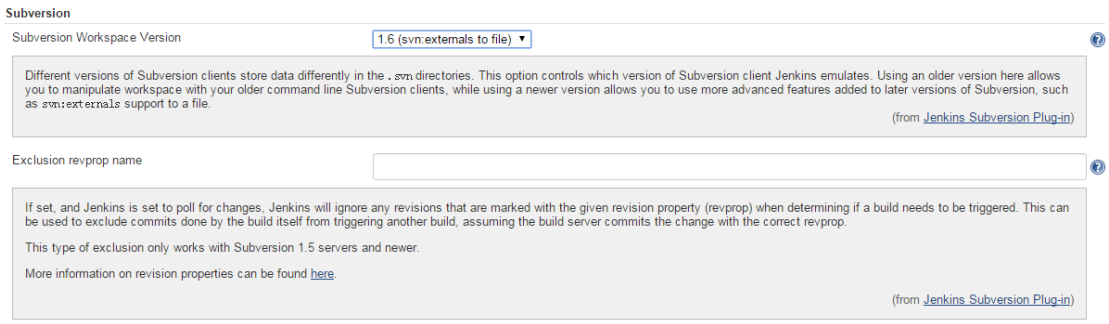
- 同步主干和分支

svn merge trunk_URL branch_URL

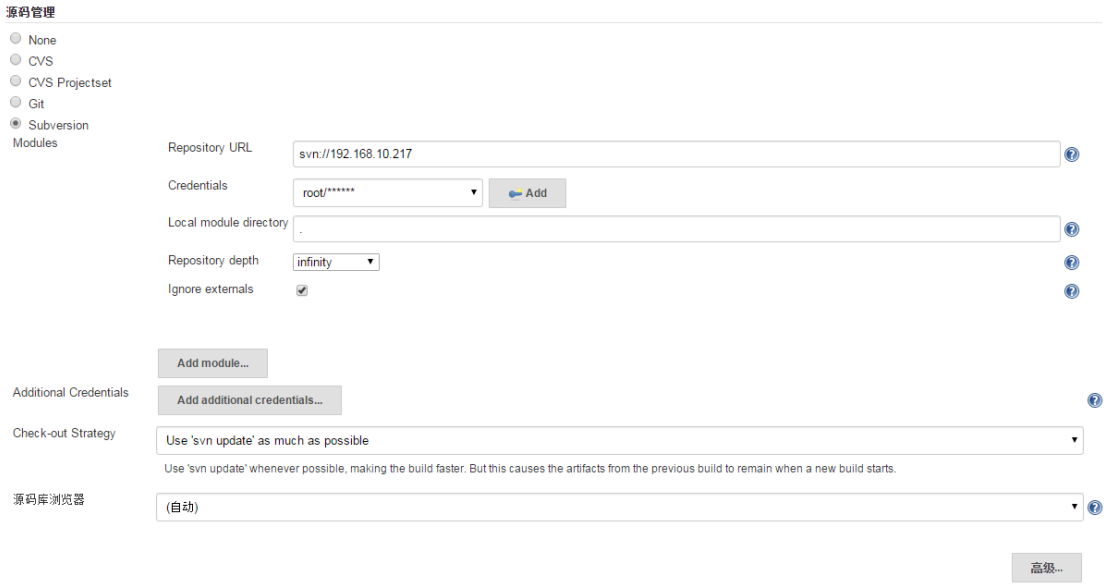
2.1.3 Jenkins 上设置 svn

Step 1: Jenkins 上设置 svn

Jenkins 上"系统管理"→"系统设置"中选择 svn 服务器端的版本



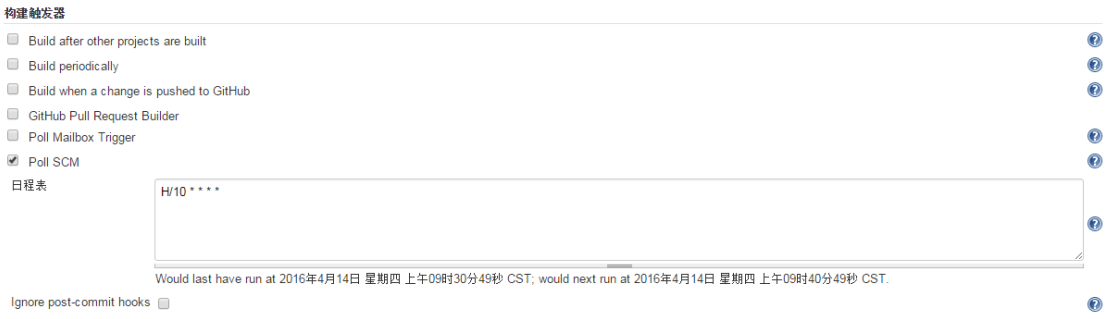
Step2：源代码管理设置



点击 Add 图标可在页面浮出层中设置 svn 的用户名密码。

Check-out Strategy 签出策略中设置为 Use ‘svn update’ as much as possible。

Step3：构建出发器



选中 Poll SCM，填写 H/10 * * * *

设置触发器为每 10 分钟比对 Svn 服务器，有新的提交则触发当前 job 的构建。

2.2 Jenkins 集成 git

2.2.1 git 的通信协议

Git 可以使用四种主要的协议来传输数据：本地传输，SSH 协议，Git 协议和 HTTP 协议。

（Ps：HTTP 协议外，其他所有协议都要求在服务器端安装并运行 Git）

本地协议

如果你使用一个共享的文件系统，就可以在一个本地文件系统中克隆仓库，推送和获取。克隆的时候只需要将远程仓库的路径作为 URL 使用，比如:

```
$ git clone /opt/git/project.git
$ git clone file:///opt/git/project.git
```

本地协议的缺点:

与基本的网络连接访问相比，难以控制从不同位置来的访问权限。如果你想从家里的笔记本电脑上推送，就要先挂载远程硬盘，这和基于网络连接的访问相比更加困难和缓慢。

另一个很重要的问题是该方法不一定就是最快的，尤其是对于共享挂载的文件系统。本地仓库只有在你对数据访问速度快的时候才快。在同一个服务器上，如果二者同时允许 Git 访问本地硬盘，通过 NFS 访问仓库通常会比 SSH 慢。

SSH 协议

SSH 也是唯一一个同时支持读写操作的网络协议；SSH 同时也是一个验证授权的网络协议。

通过 SSH 克隆一个 Git 仓库，你可以像下面这样给出 ssh:// 的 URL:

```
$ git clone ssh://user@server/project.git
```

不指明某个协议 Git 会默认使用 SSH ：

```
$ git clone user@server:project.git
```

如果不指明用户，Git 会默认使用当前登录的用户名连接服务器。

SSH 协议的缺点:

SSH 的限制在于不能通过它实现仓库的匿名访问。也必须在能通过 SSH 访问主机的前提下才能访问仓库，这使得 SSH 不利于开源的项目。

Git 协议

是一个包含在 Git 软件包中的特殊守护进程;

它会监听一个提供类似于 SSH 服务的特定端口（9418），而无需任何授权。打算支

持 Git 协议的仓库，需要先创建 git-daemon-export-ok 文件 —它是协议进程提供仓库服务的必要条件。

Git 协议的缺点：

Git 协议消极的一面是缺少授权机制。用 Git 协议作为访问项目的唯一方法通常是不可取的。一般的做法是，同时提供 SSH 接口，让几个开发者拥有推送（写）权限，其他人通过 git:// 拥有只读权限。Git 协议可能也是最难架设的协议。它要求有单独的守护进程，需要定制 — 我们将在本章的“Gitosis”一节详细介绍它的架设 — 需要设定 xinetd 或类似的程序，而这些工作就没那么轻松了。该协议还要求防火墙开放 9418 端口，而企业级防火墙一般不允许对这个非标准端口的访问。大型企业级防火墙通常会封锁这个少见的端口。

HTTP/S 协议

HTTP 或 HTTPS 协议的优美之处在于架设的简便性。基本上，只需要把 Git 的裸仓库文件放在 HTTP 的根目录下，配置一个特定的 post-update 挂钩（hook）就可以搞定。

下面的操作可以允许通过 HTTP 对仓库进行读取：

```
$ cd /var/www/htdocs/
$ git clone --bare /path/to/git_project gitproject.git
$ cd gitproject.git
$ mv hooks/post-update.sample hooks/post-update
$ chmod a+x hooks/post-update
```

这样就可以了。Git 附带的 post-update 挂钩会默认运行合适的命令（git update-server-info）来确保通过 HTTP 的获取和克隆正常工作。

HTTP/S 协议的缺点：

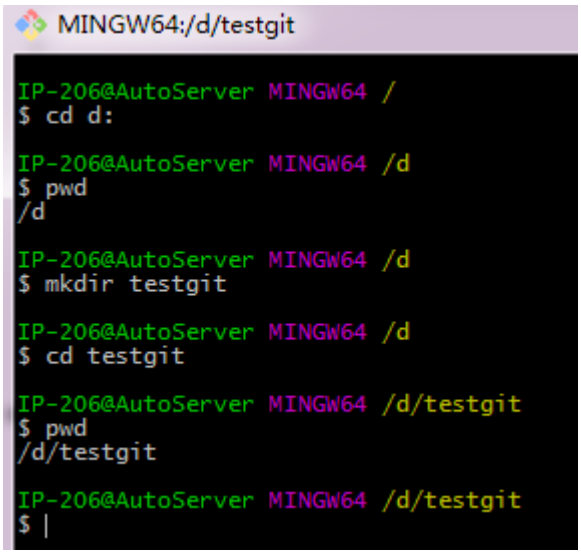
克隆或者下载仓库内容可能会花费更多时间，而且 HTTP 传输的体积和网络开销比其他任何一个协议都大。因为它没有按需供应的能力 — 传输过程中没有服务端的动态计算 — 因而 HTTP 协议经常会被称为傻瓜（dumb）协议。

2.2.2 脑补 git 命令

以下示例选择 git oschina 为 git 服务器，在 git oschina 上新建一个 private 的项目。

下载 git for windows 并安装，完成安装后打开 Git Bash。

在 D 盘新建文件夹作为仓库



➢ 初始化仓库

git init（ps: git init –bare 也可）

二者区别是:

git init 使用普通库初始化

git init –bare 使用裸库初始化

//仓库初始化完成后 d/testgit 文件夹下会有一个.git 的文件夹，它是版本库。

➢ 配置邮件和用户名

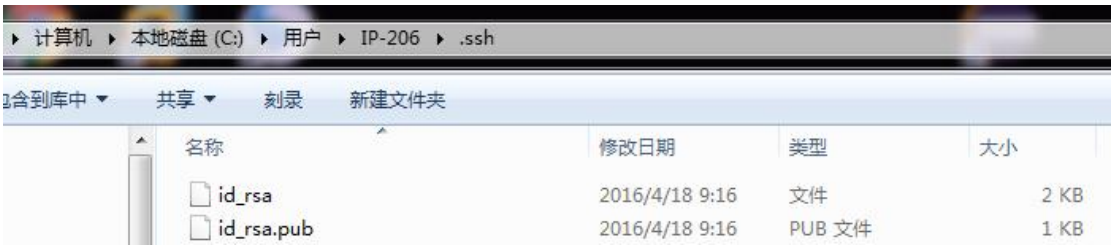
git config –-global user.email “邮箱地址”

git config –-global user.anme “用户名可随意”

➢ 创建 SSH Key

ssh-keygen -t rsa -C "你的邮箱地址"

一路回车即可，在当前用户主目录里的.ssh 下会生成两个文件 id_rsa(私钥)和 id_rsa.pub(公钥)



登录 git oschina，在个人资料----SSH 公钥栏中添加生成的公钥

➢ 克隆远程仓库

git clone 仓库 URL

如： git clone https://git.oschina.net/Share-Cherish/showgitcommand.git

git clone 和 git remote add 的区别是:

1) git clone 无需初始化仓库，它会拉取远程仓库到本地。

2) git remote add 操作必须在仓库中执行。

简要来说 git clone 等同于如下操作

```
> mkdir "exampleProject"
> cd "exampleProject"
> git init
> git remote add origin git@github.com:exampleUser/exampleProject.git
> git pull origin master
```

- 创建分支

git branch 分支名(ps: git branch 会列出所有分支)

如:git branch develop

- 切换分支

git checkout 分支名

如： git checkout master

git checkout –b 分支名(创建并切换到分支) 它等同于 git branch 分支后再执行 git checkout 分支。

- 添加新文件

新建文件 vi first_newfile.txt 在里面输入内容 add the first line

将文件添加到缓冲区 git add first_newfile.txt

将修改的内容提交到仓库 git commit –m “备注信息”

- 查看修改日志

git log

若想以精简方式显示 log 信息可以使用 git log–pretty=oneline

- 版本回退

git reset –hard HEAD^回退到上一个版本; -head HEAD^^回退到上上一个版本;

回退到 4 个版本之前 git reset –hard HEAD~4

- 撤销修改

修改了文件内容　没有执行 git add 和 git commit 操作

git checkout --需要撤销修改内容的文件名

修改了文件内容 已执行 git add 操作 想撤销修改需要两步

先执行 git reset HEAD 文件名　再执行 git checkout –文件名

- 从版本库中删除文件

先执行 git rm 文件名　再执行 git commit

- 合并分支

git merge 分支名(把指定分支合并到当前分支上)

如: git checkout master;　git merge develop(把 develop 分支合并到 master 上)

git merge 默认会使用 Fast forward 模式，该模式下，删除分支后，会丢掉分支信息。

git merge –no-ff –m “comments” 分支名:

禁用 Fast forward 模式合并分支并添加一个新的备注。

- 删除分支

git branch –d 分支名

如:git branch –d develop

- 推送本地修改到远程仓库

git push 远程仓库名　本地分支名

如: git push origin develop

➢ 拉取远程分支到本地当前仓库

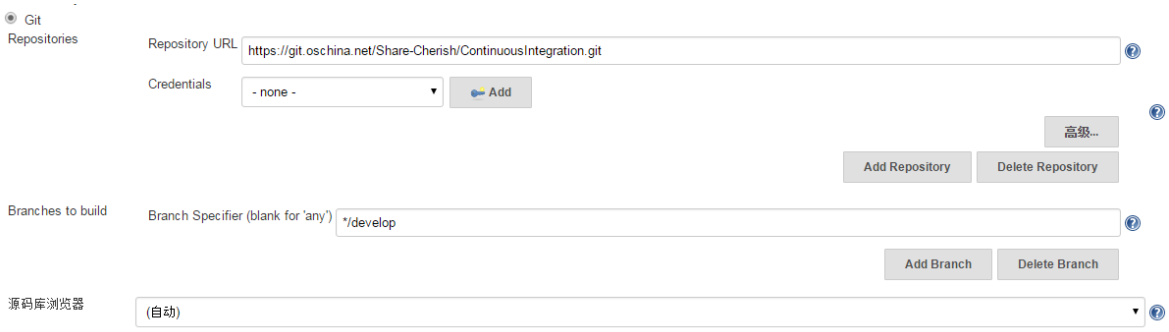
git pull

若 git pull 操作时提示“no tracking information”，则说明本地分支和远程分支的链接关系未创建，用命令 git branch --set-upstream 本地分支名 origin/远程仓库分支名来创建本地分支和远程分支的链接。

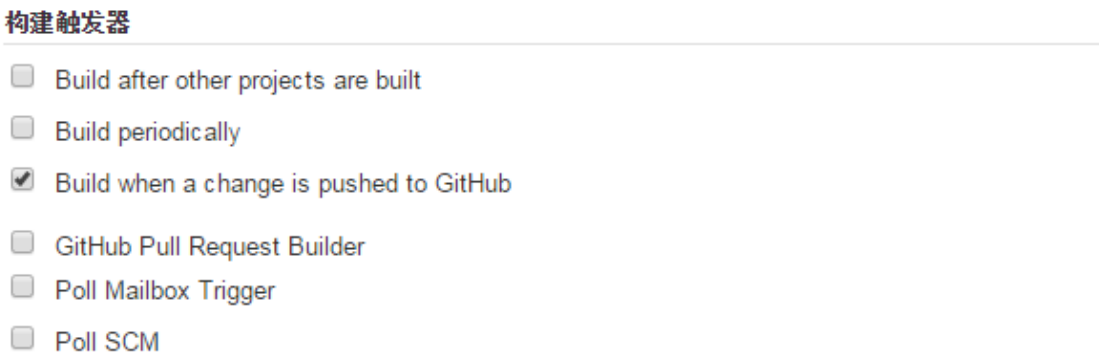
2.2.3 Jekins 上的 git 设置

在系统管理----插件管理中选择 git 插件进行安装

在 job 的源码管理栏配置远程仓库 URL



配置触发器



点击立即构建，测试 Job 是否可以构建成功

2.3 Jenkins 集成 Ant

2.3.1 Ant 安装

1) CentOS 上安装 Ant

- 下载 Ant: wget http://apache.opencas.org//ant/binaries/apache-ant-1.9.7-bin.tar.gz
- 解压 Ant 包: tar -xvzf apache-ant-1.9.7-bin.tar.gz
- 编辑/etc/profile 文件 vi /etc/profile 添加内容 export ANT_HOME=Ant 包解压的目录
- 创建链接 cd /bin 执行 ln -s -f /root/Ant/apache-ant-1.9.7/bin/ant

2) windows 上安装 Ant

下载 Ant 包，将解压后的目录作为环境变量 ANT_HOME 的值添加到系统变量即可。

2.3.2 Ant 常用配置项详解

<project>

1) name 属性：项目名称

2) deault 属性：必填项 表示默认的运行目标

3) basedir 属性：项目的基准目标

4) description 属性：项目的描述

示例如下：

```
<project name="ant_firsttest" default="dist" basedir=".">
<description>ant firsttest!</description>
```

<target>

1) name 属性：必填项 表示声明

2) depends 属性：依赖的目标

3) unless 属性：属性未设置时才知晓

4) description 属性：项目描述

示例如下：

```
<target name="compile" depends="init" description="compile the source " >
<do something/>
</target>
```

<mkdir>

```
<mkdir dir="目录名"/> 创建目录
```

<jar>

1) destfile 表示 jar 文件名

2) basedir 表示被归档的文件名

3) include 表示被归档的文件模式

4) exclueds 表示被排除的文件模式

示例如下：

```
<jar destfile="${build}/${db}/hii-${db}.jar" basedir="${build}/${db}/hii"/>
```

<javac>

1) srcdir 表示源文件的目录

2) destdir 表示 class 文件的输出目录

3) include 表示被编译的文件的模式

4) excludes 表示被排除的文件的模式

5) classpath 表示使用的类路径

6) debug 表示包含的调试信息

7) optimize 表示是否使用优化

8) verbose 表示提供详细的输出信息

9) fileonerror 表示当碰到错误就自动停止

示例如下:

```
<javac destdir="${build}/${db}/xdcpba" encoding="UTF-8" includeAntRuntime="false" debug="on">
        <classpath refid="project.classpath"/>
        <src path="${java}"/>
        <src path="${src-main}/${db}/java"/>
        <include name="com/huntto/hii/*"/>
        <include name="com/huntto/hii/model/*"/>
        <include name="com/huntto/hii/model/basic/**"/>
        <include name="com/huntto/hii/dao/*"/>
        <include name="com/huntto/hii/dao/basic/**"/>
        <include name="com/huntto/hii/dao/hibernate/basic/**"/>
        <include name="com/huntto/hii/util/**"/>
        <include name="com/huntto/hii/xdcpba/**"/>
    </javac>
```

<delete>

1) file 表示要删除的文件

2) dir 表示要删除的目录

3) includEmptyDirs 表示是否要删除空目录，默认值是删除

4) failonerror 表示指定当碰到错误是否停止，默认值是自动停止

5) verbose 是否列出所删除的文件，默认是不列出

使用示例:

```
<target name="clean-mysql-xdcpba" description="clean up xdcpba for mysql">
    <delete file="${build}/${db}/hii-xdcpba-${db}.jar"/>
    <delete dir="${build}/${db}/xdcpba"/>
    <delete dir="${dist}/${db}/xdcpba"/>
</target>
```

<copy>

1) file 表示源文件

2) tofile 表示目标文件

3) todir 表示目标目录

4) overwrite 表示指定是否覆盖目标文件，默认值是不覆盖

5) includeEmptyDirs 表示指定是否拷贝空目录，默认是拷贝

6) failonerror 表示指定如目标没有发现是否自动台停止，默认是停止

使用示例:

```
<copy todir="${build}/${db}/xg">
    <fileset dir="${java}" includes="**/*.xml" />
    <fileset dir="${src-main}/${db}/java" includes="**/*.xml" />
</copy>
```

Ant 实例详解:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="ant_web_hello" basedir="." default="deploy">
    <!-- java 源文件目录 -->
    <property name="src.dir" location="/src" />
    <!-- 构建目录 -->
    <property name="build.dir" location="build" />
    <!-- class 文件的存放目录 -->
    <property name="build.classes" location="${build.dir}/classes" />
    <!--war 包的存放目录-->
```

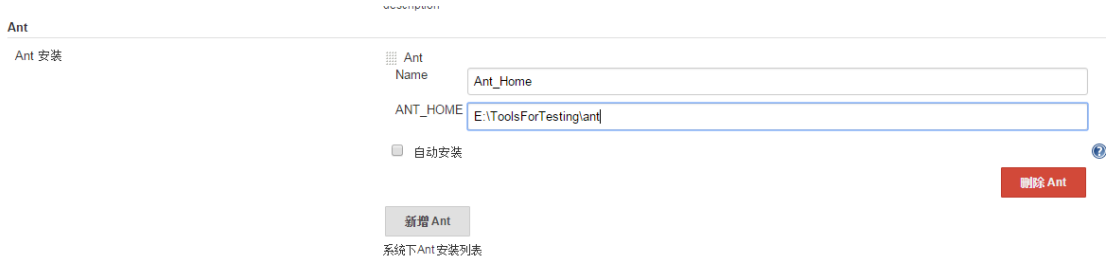
```
    <property name="build.war" location="${build.dir}/war" />

    <!-- tomcat_home 路径 -->
    <property name="tomcat.home" location="E:\ToolsForTesting\apache-tomcat" />
    <!-- <property name="tomcat.lib" location="${tomcat.home}/lib" /> -->
    <!-- web 应用名 -->
    <property name="web.name" value="anthello" />
    <!-- web 根目录-->
    <property name="web.root" value="/WebContent" />
    <property name="web.WEB-INF" location="${web.root}/WEB-INF" />
    <property name="web.lib" location="${web.WEB-INF}/lib" />
<!--Svn 远程目录-->
    <property name="svn-url" value="svn://192.168.10.217/JavaWebDemo" />

    <!-- 编译时的 classpath -->
    <path id="compile.path">
        <fileset dir="${web.lib}" includes="*.lib">
        </fileset>
        <fileset dir="${tomcat.home}/lib">
            <include name="**/*.jar" />
        </fileset>
    </path>

    <target name="svn_co">
      <echo message="hello ant!"></echo>
   <!-- 直接调用 svn exec 要确保安装了 svn 客户端-->
      <exec executable="svn" >
     <!-- 设置 svn 的命令行参数 -->
        <arg line="co ${svn-url}"/>
      </exec>
    </target>

    <target name="init" depends="svn_co" description="初始化项目构建">
```

```
        <mkdir dir="${build.dir}" />
        <mkdir dir="${build.classes}" />
        <mkdir dir="${build.war}" />
        <echo>start init task</echo>
    </target>

    <target name="compile" depends="init" description="编译任务">
        <javac destdir="${build.classes}" srcdir="${src.dir}" includeantruntime="false" fork="true">
            <compilerarg line="-encoding UTF-8 " />
            <classpath refid="compile.path" />
        </javac>
        <echo message="complate compile task"></echo>
    </target>

    <target name="war" depends="compile" description="打 war 包">
        <war destfile="${build.war}/${web.name}.war">
            <fileset dir="${web.root}" includes="**/*.*" />
            <lib dir="${web.lib}" />
            <webinf dir="${web.WEB-INF}" />
            <classes dir="${build.classes}" />
        </war>
        <echo>complate tar package of war</echo>
    </target>

    <target name="deploy" depends="war" description="发布">
        <copy todir="${tomcat.home}/webapps">
            <fileset dir="${build.war}" includes="*.war" />
        </copy>
        <echo>deploy to tomcat</echo>
    </target>

    <target name="clean" description="清理发布环境">
<deldir dir="" />
```

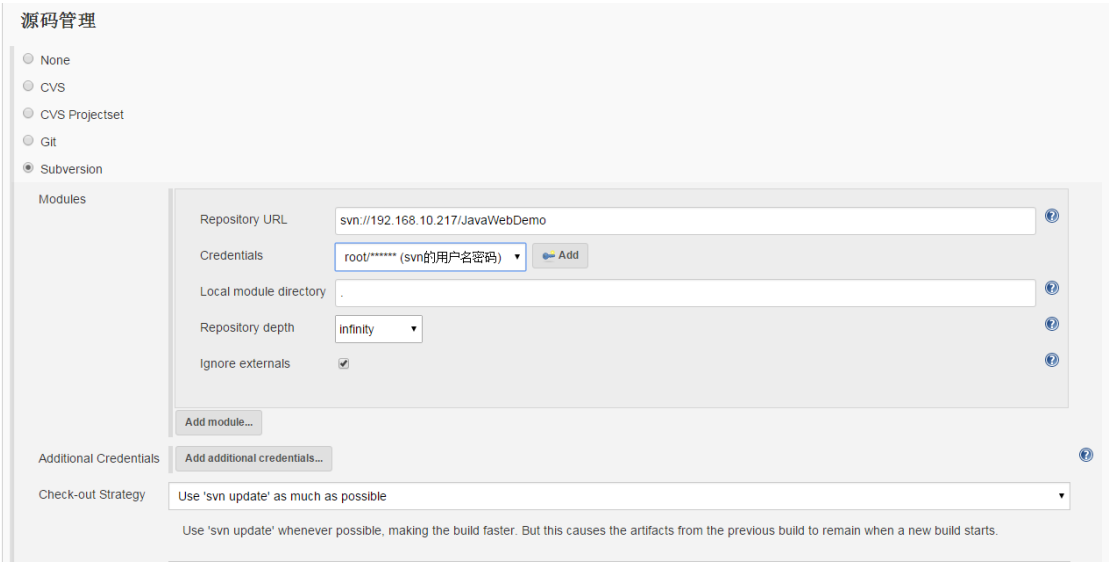
```
<deldir dir="" />
</target>


</project>
```

2.3.3 Jenkins 上的 Ant 配置使用

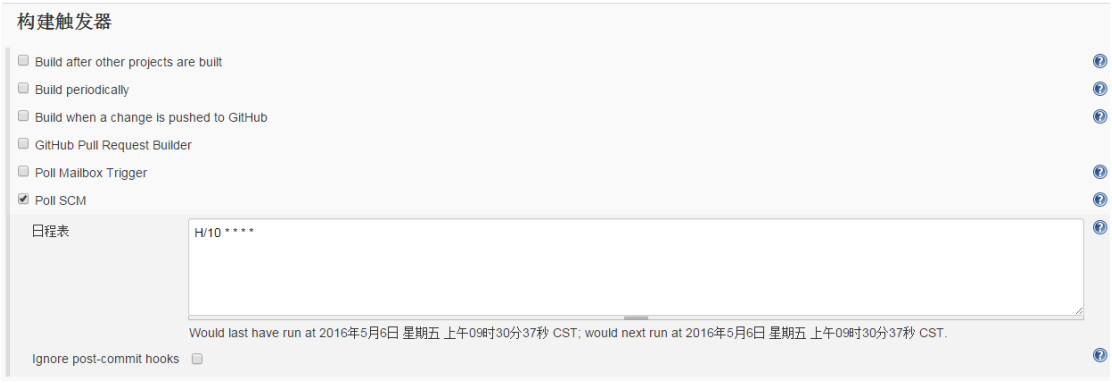
配置 ANT_HOME



新建自由风格的 Job，源码管理配置为 svn 地址:



构建触发器(每 10 分钟扫描 svn):



配置构建组件(这里我用的是环境变量 Ant_Home，target 填写 build.xml 里的 deploy):

任务构建完成:

Hello:

2.4 Jenkins 集成 Maven

2.4.1 Maven 安装和命令使用

Maven 是什么?

Maven 是一个项目管理和整合工具。Maven 为开发者提供了一套完整的构建生命周期框架。开发团队几乎不用花多少时间就能够自动完成工程的基础构建配置，因为 Maven 使用了一个标准的目录结构和一个默认的构建生命周期。

在有多个开发团队环境的情况下，Maven 能够在很短的时间内使得每项工作都按照标准进行。因为大部分的工程配置操作都非常简单并且可复用，在创建报告、检查、构建和测试自动配置时，Maven 可以让开发者的工作变得更简单。

Maven 能够帮助开发者完成以下工作:

- 构建
- 文档生成
- 报告
- 依赖
- SCMs
- 发布
- 分发
- 邮件列表

总的来说，Maven 简化了工程的构建过程，并对其标准化。它无缝衔接了编译、发布、文档生成、团队合作和其他任务。Maven 提高了重用性，负责了大部分构建相关的任务。

CentOS 上安装 Maven:

➢ 下载 maven 包:

wget http://mirrors.hust.edu.cn/apache/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz

➢ 解压:

Tar –zxvf apache-maven-3.3.9-bin.tar.gz

➢ 建立软链接:

Ln –s apache-maven-3.3.9 maven

➢ 配置环境变量:

vi /etc/profile

export M2_HOME=/usr/local/apache-maven

export PATH=$PATH:$M2_HOME/bin

- 使环境变量生效:

source/etc/profile

- 验证安装结果:

```
[root@localhost local]# mvn --version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T08:41:4
7-08:00)
Maven home: /usr/local/apache-maven-3.3.9
Java version: 1.8.0_77, vendor: Oracle Corporation
Java home: /usr/local/java/jdk1.8.0_77/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "2.6.32-431.el6.x86_64", arch: "amd64", family: "unix
"
[root@localhost local]#
```

2.4.2 Maven 常用命令介绍

mvn help:system 自动在本用户下创建 ~/.m2/repository

mvn clean compile 清理编译

mvn clean test 清理测试

mvn clean package 清理打包

mvn clean install 清理将打包好的 jar 存入 本地仓库 注意是本地仓库

mvn archetype:generate 使用 Archetype 生成项目骨架

mvn clean deploy 根据 pom 中的配置信息将项目发布到远程仓库中

Maven 项目的目录结构:

src/main/java: 正式内容包路径

src/mian/resources: 正式的配置文件路径

src/test/java: 测试包路径

src/test/resources: 测试的配置文件路径

src/main/webapp: war 资源目录

mvn dependency: list 显示所有已经解析的所有依赖

mvn dependency: tree 以目录树的形式展现依赖，最高层为一层依赖，其次二层依赖，三层依赖....

mvn dependency: analyze 第一部分显示已经使用但是未显示依赖的，第二部分显示项目未使用的但是依赖的

构件：jar 插件 war 所有依赖的 jar 构建：编译、测试、打包、发布

构建生命周期：

生命周期就是接口：表明要干什么事情

插件就是具体的实现：表明怎么干这件事情

一个典型的 Maven 构建生命周期是由以下几个阶段的序列组成的：

阶段	处理	描述
prepare-resources	资源拷贝	本阶段可以自定义需要拷贝的资源
compile	编译	本阶段完成源代码编译
package	打包	本阶段根据 pom.xml 中描述的打包配置创建 JAR / WAR 包
install	安装	本阶段在本地 / 远程仓库中安装工程包

当需要在某个特定阶段之前或之后执行目标时，可以使用 pre 和 post 来定义这个目标。

Pom.xml 实例：

```
<project
xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.temp</groupId>
<artifactId>Demo</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
```

```
    <name>Demo Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.htmlparser</groupId>
            <artifactId>htmlparser</artifactId>
            <version>1.6</version>
        </dependency>
        <!-- spring 需要的 jar 包  -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>3.2.4.RELEASE</version>
            <type>jar</type>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.2.4.RELEASE</version>
            <type>jar</type>
```

```
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>3.2.4.RELEASE</version>
            <type>jar</type>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>3.2.4.RELEASE</version>
            <type>jar</type>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>3.2.4.RELEASE</version>
            <type>jar</type>
        </dependency>

        <!-- 连接 MySQL 数据库需要的 jar 包 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.34</version>
        </dependency>

        <!-- jstl 需要的 jar 包 -->
        <dependency>
            <groupId>jstl</groupId>
```

```
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- log4j 需要的 jar 包 -->
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>

    <!-- mybatis 需要的 jar -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.3.0</version>
    </dependency>

    <dependency>
        <groupId>c3p0</groupId>
        <artifactId>c3p0</artifactId>
        <version>0.9.1.2</version>
    </dependency>

    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>1.2.2</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.5.3</version>
```

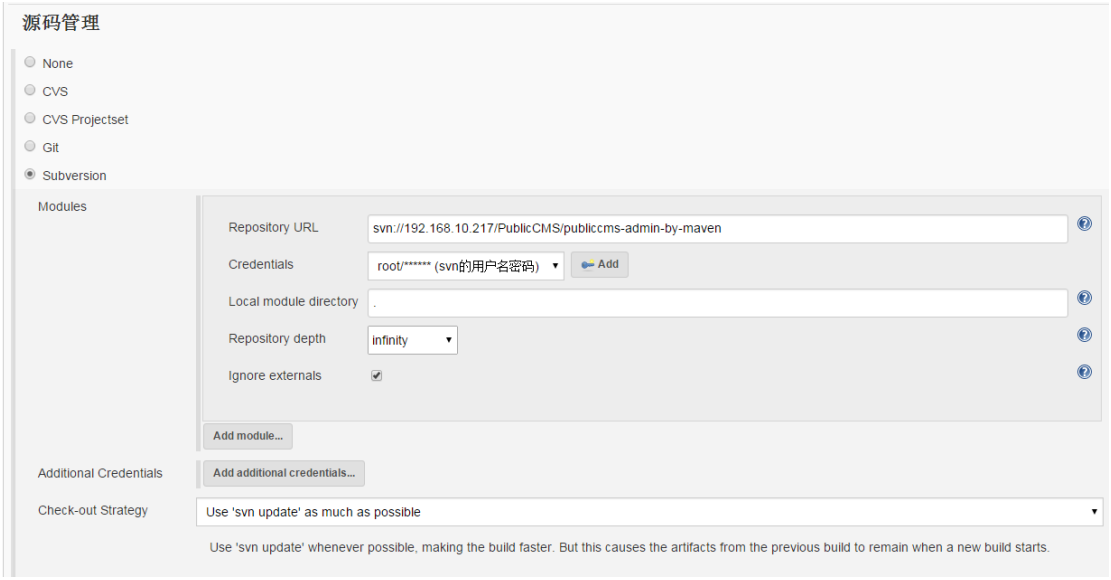
```
        </dependency>

        <!-- PDFBOX -->
        <dependency>
            <groupId>org.apache.pdfbox</groupId>
            <artifactId>pdfbox</artifactId>
            <version>2.0.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.pdfbox</groupId>
            <artifactId>pdfbox-tools</artifactId>
            <version>2.0.0</version>
        </dependency>

        <dependency>
            <groupId>com.itextpdf</groupId>
            <artifactId>itextpdf</artifactId>
            <version>5.5.1</version>
        </dependency>

    </dependencies>
    <build>
        <finalName>zfyz</finalName>
    </build>
</project>
```

2.4.3 Jenkins 上配置使用 Maven

系统管理中配置 Maven:

新建 Maven 构建项目：

设置源码管理的 svnurl



配置构建项：

立即构建验证效果：

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 13.340 s
[INFO] Finished at: 2016-05-06T14:53:32+08:00
[INFO] Final Memory: 28M/296M
[INFO] ------------------------------------------------------------------------
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving E:\ToolsForTesting\Jenkins\workspace\Jenkin+Svn+Maven\pom.xml to publiccms/admin/1.0/admin-1.0.pom
channel stopped
Finished: SUCCESS
```

2.5 Jenkin 集成 Java 静态检查工具

2.5.1 PMD 安装和使用

PMD 介绍:

PMD 是一种开源分析 Java 代码错误的工具。与其他分析工具不同的是，PMD 通过静态分析获知代码错误。也就是说，在不运行 Java 程序的情况下报告错误。PMD 附带了许多可以直接使用的规则，利用这些规则可以找出 Java 源程序的许多问题，例如:

® 潜在的 bug: 空的 try/catch/finally/switch 语句

® 未使用的代码: 未使用的局部变量、参数、私有方法等

® 可选的代码: String/StringBuffer 的滥用

® 复杂的表达式: 不必须的 if 语句、可以使用 while 循环完成的 for 循环

® 重复的代码: 拷贝/粘贴代码意味着拷贝/粘贴 bugs

® 循环体创建新对象: 尽量不要再 for 或 while 循环体内实例化一个新对象

@ 资源关闭: Connect，Result，Statement 等使用之后确保关闭掉

此外，用户还可以自己定义规则，检查 Java 代码是否符合某些特定的编码规范。例如，你可以编写一个规则，要求 PMD 找出所有创建 Thread 和 Socket 对象的操作。

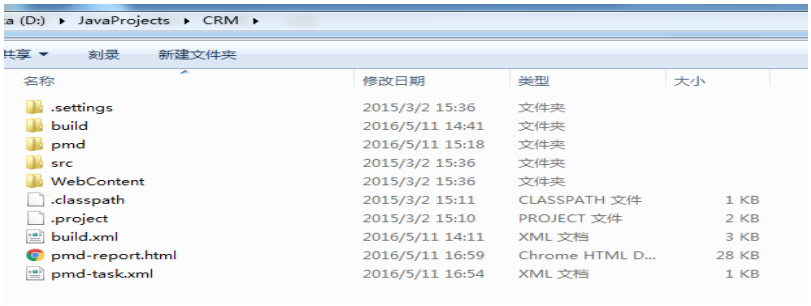
PMD 安装:

在 https://github.com/mrniko/redisson/wiki 可查看 pmd 的 wiki 信息

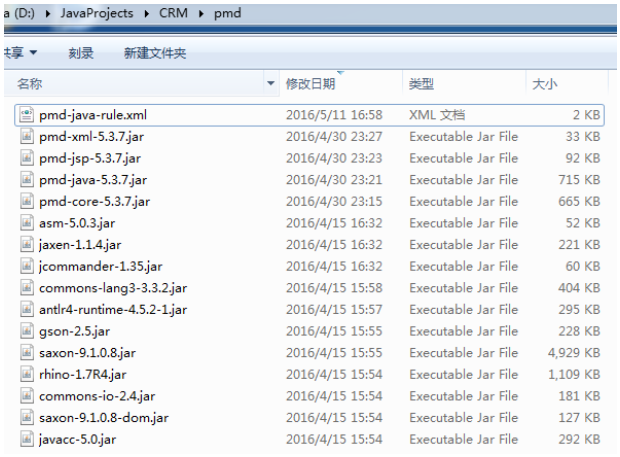
在 https://pmd.github.io/ 下载 pmd。

使用 Ant 执行 PMD 的 task:

Java 项目的目录结构:



其中的 PMD 文件夹中存放的是 pmd 依赖的 jar 包和自定义的检查规则。



Pmd 的 task 如下编写：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="pmd" default="pmd" basedir=".">
<property name="sourcedir" location="src" />
<property name="pmddir" location="pmd" />
    <!-- 定义 pmd 的 classpath，均为下载的 pmd 包解压后 lib 目录下的 jar 包-->
<path id="pmd.classpath">
    <fileset dir="${pmddir}">
      <include name="**/*.jar">
      </include>
  </fileset>
</path>

<target name="pmd">
   <taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask" classpathref="pmd.classpath"
/>
    <!-- 定义 PMD 检测规则所在的文件，规则集在 pmdrule.xml 文件中定义-->
```

```
        <pmd rulesetfiles="${pmddir}/pmd-java-rule.xml" encoding="UTF-8">
        <!-- 输出 html 格式的报告-->
        <formatter type="html" tofile="pmd-report.html" />
        <!-- 静态代码检测工程下 src 目录下的所有 java 文件-->
        <fileset dir="${sourcedir}">
            <include name="**/*.java" />
</fileset>
</pmd>
</target>
</project>
```

我们执行 ant –f pmd-task.xml:

```
     [pmd] 五月 12, 2016 9:02:05 上午 net.sourceforge.pmd.RuleSetFactory parseR
uleReferenceNode
     [pmd] 警告: Use Rule name rulesets/java/unnecessary.xml/UnnecessaryConvers
ionTemporary instead of the deprecated Rule name rulesets/java/basic.xml/Unneces
saryConversionTemporary. Future versions of PMD will remove support for this dep
recated Rule name usage.
Use Rule name rulesets/java/empty.xml/EmptyCatchBlock instead of the deprecated
Rule name rulesets/java/basic.xml/EmptyCatchBlock. Future versions of PMD will r
emove support for this deprecated Rule name usage.
     [pmd] 五月 12, 2016 9:02:05 上午 net.sourceforge.pmd.RuleSetFactory parseR
uleReferenceNode
     [pmd] 警告: Use Rule name rulesets/java/empty.xml/EmptyCatchBlock instead
of the deprecated Rule name rulesets/java/basic.xml/EmptyCatchBlock. Future vers
ions of PMD will remove support for this deprecated Rule name usage.

BUILD SUCCESSFUL
Total time: 1 second

D:\JavaProjects\CRM>
```

构建 pmd 的 task 成功后 在指定的目录下会生产报告 pmd-report.html

file:///D:/JavaProjects/CRM/pmd-report.html

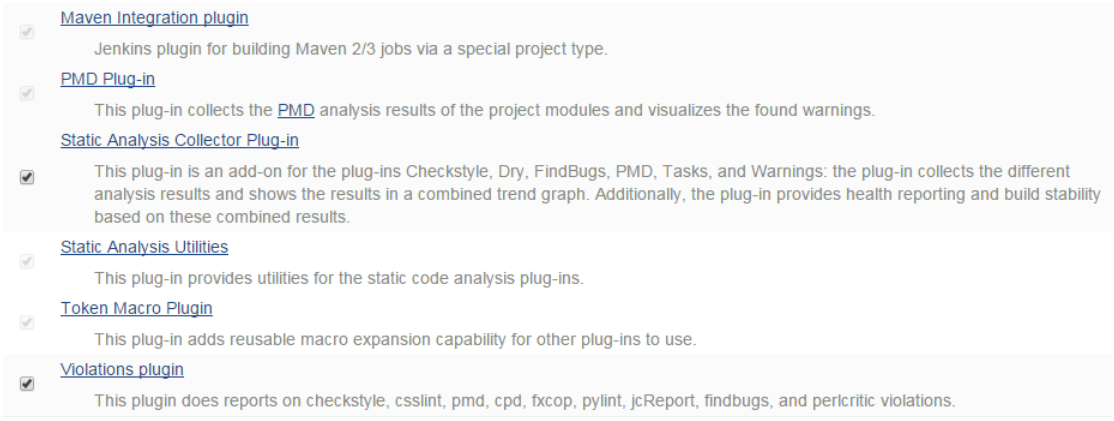
PMD report

Problems found

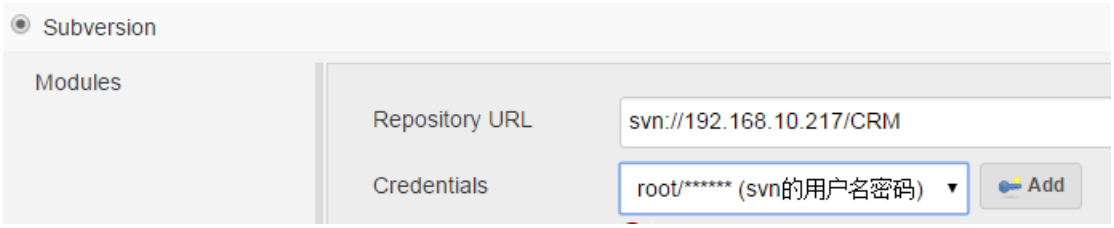
#	File	Line	Problem
1	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerCareDAOImpl.java	13	The class 'CustomerCareDAOImpl' has a Cyclomatic Complexity of 5 (Highest = 12).
2	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerCareDAOImpl.java	131	The method 'getList' has a Cyclomatic Complexity of 12.
3	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerConditionDAOImpl.java	13	The class 'CustomerConditionDAOImpl' has a Cyclomatic Complexity of 5 (Highest = 7).
4	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerConditionDAOImpl.java	85	The method 'query' has a Cyclomatic Complexity of 7.
5	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	13	The class 'CustomerDAOImpl' has a Cyclomatic Complexity of 11 (Highest = 59).
6	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	115	The String literal "select a.* ,b.condition_name, c.source_name,d.type_name ,e.user_name from customer_info a,customer_condition b, customer_source c,customer_type d,user_info e " appears 4 times in this file; the first occurrence is on line 115
7	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	131	The String literal "customer_id" appears 4 times in this file; the first occurrence is on line 131
8	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	132	The String literal "customer_name" appears 4 times in this file; the first occurrence is on line 132
9	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	133	The String literal "condition_id" appears 4 times in this file; the first occurrence is on line 133
10	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	134	The String literal "condition_name" appears 4 times in this file; the first occurrence is on line 134
11	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	135	The String literal "source_id" appears 4 times in this file; the first occurrence is on line 135
12	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	136	The String literal "source_name" appears 4 times in this file; the first occurrence is on line 136
13	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	137	The String literal "type_id" appears 4 times in this file; the first occurrence is on line 137
14	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	138	The String literal "type_name" appears 4 times in this file; the first occurrence is on line 138
15	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	139	The String literal "user_id" appears 4 times in this file; the first occurrence is on line 139
16	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	141	The String literal "customer_sex" appears 4 times in this file; the first occurrence is on line 141
17	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	142	The String literal "customer_mobile" appears 4 times in this file; the first occurrence is on line 142
18	D:\JavaProjects\CRM\src\com\sxxy\dao\impl\CustomerDAOImpl.java	143	The String literal "customer_qq" appears 4 times in this file; the first occurrence is on line 143

接下来我们将 pmd 集成到 Jenkins 中:

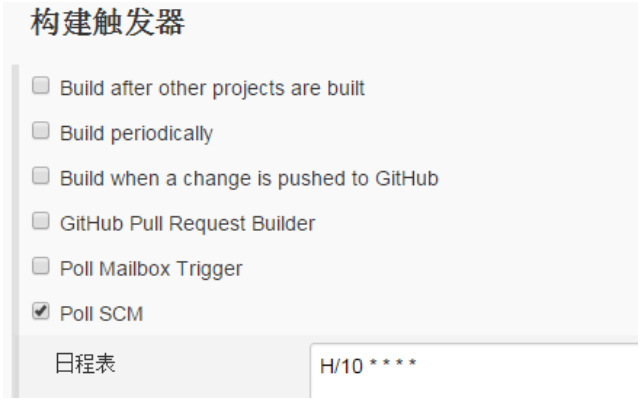
首先安装插件 PMD Plug-in，Maven Integration plugin,Static Analysis Collector Plug-in，Static Analysis Utilities。



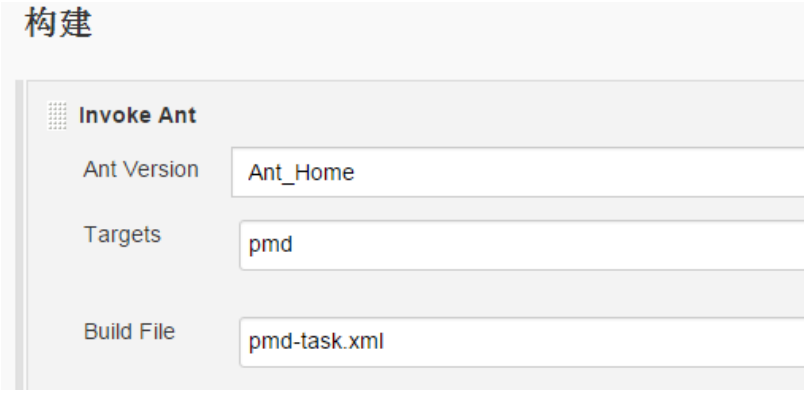
新建 Job 命名为 jenkins+pmd；配置远程源码仓库 url 地址：



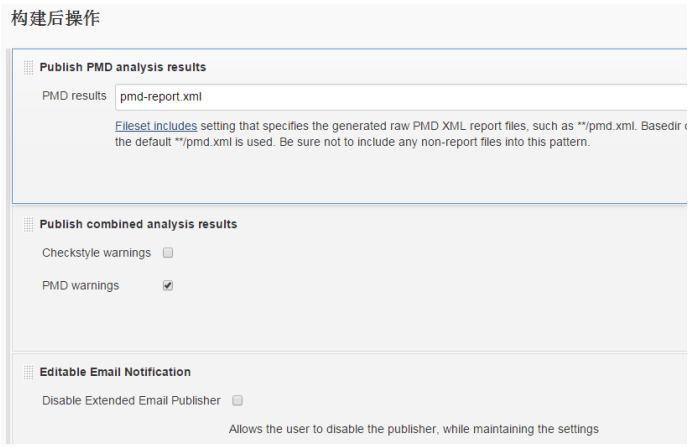
构建出发器：每 10 分钟检查源码仓库是否有更新：



配置构建任务：



配置构建后的操作:



立即构建后我们即可查收邮件并查看 pmd 的检查结果:

邮件内容:

项目名称：jenkins+pmd

构建编号：1

构建状态：Successful

触发原因：Started by user anonymous

构建日志地址：http://192.168.10.206:8080/job/jenkins+pmd/1/console

构建地址：http://192.168.10.206:8080/job/jenkins+pmd/1/

变更集:

BUILD SUCCESS

Build URL http://192.168.10.206:8080/job/jenkins+pmd/1/
Project: jenkins+pmd
Date of build: Thu, 12 May 2016 11:00:48 +0800
Build duration: 7.3 秒

查看 PMD Warnings:

PMD Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
74	74	0

Summary

Total	High Priority	Normal Priority	Low Priority
74	0	74	0

Details

Packages | Files | Categories | Types | Warnings | Details | New

Package	Total	Distribution
com.sxxy.dao.impl	69	
com.sxxy.service.impl	2	
com.sxxy.servlet.customer	3	
Total	74	

2.5.2 CheckStyle 的安装和使用

Checkstyle 介绍:

- CheckStyle 是什么?

CheckStyle 是 SourceForge 下的一个项目，提供了一个帮助 Java 开发人员遵守某些编码规范的工具。它能够自动化代码规范检查过程，从而使得开发人员从这项重要，但是枯燥的任务中解脱出来。

- CheckStyle 检验的主要内容

CheckStyle 默认提供一下主要检查内容:

- Javadoc 注释
- 命名约定

- 标题
- Import 语句
- 体积大小
- 空白
- 修饰符
- 块
- 代码问题
- 类设计
- 混合检查（包括一些有用的比如非必须的 System.out 和 printstackTrace）

从上面可以看出，CheckStyle 提供了大部分功能都是对于代码规范的检查，而没有提供象 PMD 和 Jalopy 那么多的增强代码质量和修改代码的功能。但是，对于团队开发，尤其是强调代码规范的公司来说，它的功能已经足够强大。

➢ 安装 checkstyle:

首先请务必到 github 上去读官方文档: https://github.com/checkstyle/checkstyle

将 checkstyle 的 jar 包放置在 ant 安装目录下的 lib 下即可。

➢ 创建 ant 任务(ps: 大百度搜索结果都是特么扯淡，请直接去官网看怎么写的):

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="fuck" default="checkstyle">
<property name="checkstyle.jar" location="E:\ToolsForTesting\ant\checkstyle"/>
<property name="checkrule.dir" location="E:\ToolsForTesting\ant\checkstyle"/>
<target name="checkstyle">
<taskdef name="checkstyle" classname="com.puppycrawl.tools.checkstyle.ant.CheckstyleAntTask"
classpath="${checkstyle.jar}\checkstyle-6.18-all.jar"/>
    <checkstyle config="${checkrule.dir}\sun_checks.xml" failOnViolation="false"
failureProperty="checkstyle.failure">
        <fileset dir="./src" includes="**/*.java"/>
        <formatter type="xml" toFile="./checkstyle_report.xml"/>
    </checkstyle>
</target>
</project>
```

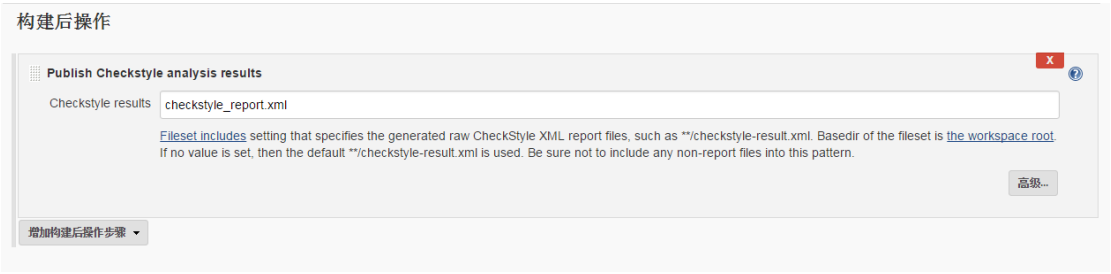
➢ Jenkins 上集成 checkstyle 插件:

我这里直接上截图哦

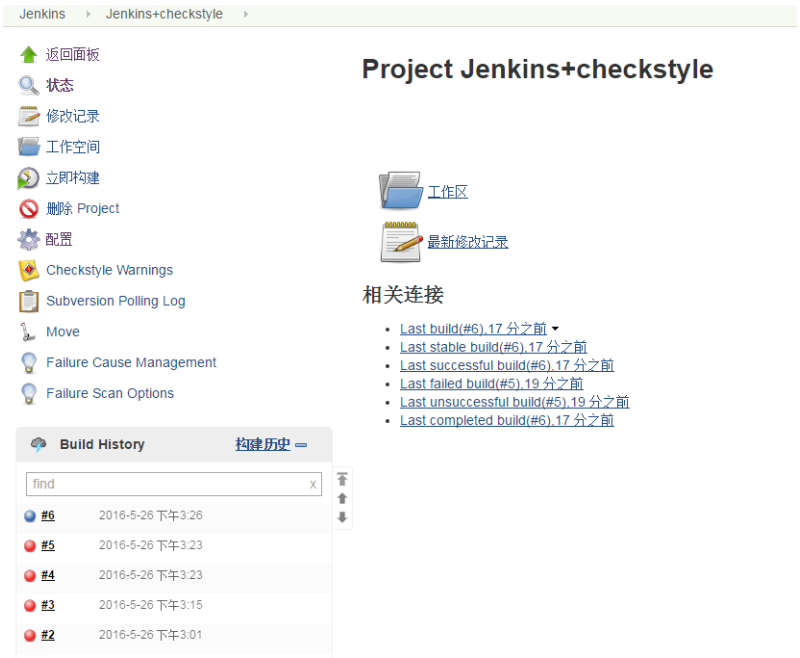
最重要的两步之一：构建(配置 ant_home，targets 和 build file)

构建后将 checkstyle 的结果发布出来:

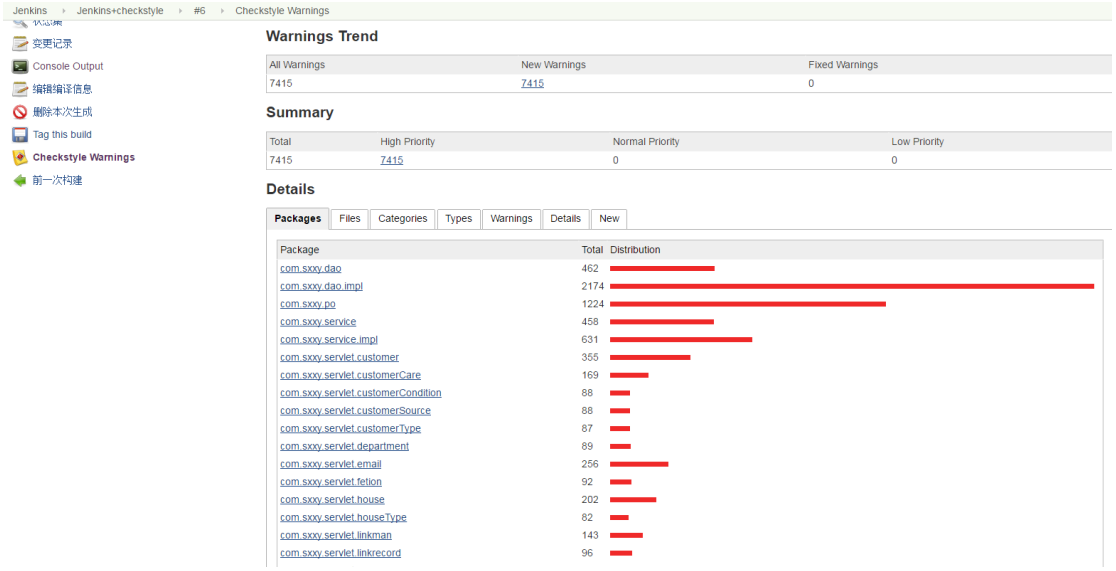
因为在 ant 的任务配置中我是直接把 checkstyle 的结果生成在项目的根目录下，所以我下面的 checkstyle results 直接写了文件名。



任务配置完成后，我们直接点立即构建可查看效果(我这里试了几次，所以前几次构建是失败的):



我们可点击 checkstyle warnings 查看 checkstyle 的检查结果:



2.5.3 Findbugs 安装和使用

FindBugs 是什么?

FindBugs 是一个静态分析工具，它检查类或者 JAR 文件，将字节码与一组缺陷模式进行对比以发现可能的问题。有了静态分析工具，就可以在不实际运行程序的情况对软件进行分析。不是通过分析类文件的形式或结构来确定程序的意图，而是通常使用 Visitor 模式（请参阅 参考资料）。图 1 显示了分析一个匿名项目的结果（为防止可怕的犯罪，这里不给出它的名字）:

在 FindBugs 的 GUI 中，需要先选择待扫描的.class 文件(FindBugs 其实就是对编译后的 class 进行扫描，藉以发现一些隐藏的 bug。)。如果你拥有这些.class 档对应的源文件，可把这些.java 文件再选上，这样便可以从稍后得出的报告中快捷的定位到出问题的代码上面。此外，还可以选上工程所使用的 library，这样似乎可以帮助 FindBugs 做一些高阶的检查，藉以发现一些更深层的 bug。

选定了以上各项后，便可以开始检测了。检测的过程可能会花好几分钟，具体视工程的规模而定。检测完毕可生成一份详细的报告，藉由这份报告，可以发现许多代码中间潜在的 bug。比较典型的，如引用了空指针(null pointer dereference)，特定的资源(db connection)未关闭，等等。如果用人工检查的方式，这些 bug 可能很难才会被发现，或许永远也无法发现，直到运行时发作…当除掉了这些典型的(classic) bug 后，可以确信的是，我们的系统稳定度将会上一个新的台阶。

以目前遇到的状况来看，FindBugs 可以有两种使用时机。

- **开发阶段**

当 Developer 完成了某一部分功能模块开发的时候(这通常是指代码撰写完成，并已 debug 通过之后)，可藉由 FindBugs 对该模块涉及的 java 文件进行一次扫描，以发现一些不易察觉的 bug 或是效能问题。交付新版的时候，开发团队可以跑一下 FindBugs，除掉一些隐藏的 Bug。FindBugs 得出的报告可以作为该版本的一个参考文档一并交付给测试团队留档待查。

在开发阶段使用 FindBugs，一方面开发人员可以对新版的品质更有信心，另一方面，测试人员藉此可以把更多的精力放在业务逻辑的确认上面，而不是花大量精力去进一些要在特殊状况下才可能出现的 BUG(典型的如 Null Pointer Dereference)。从而可以提高测试的效率。

- **维护阶段**

这里指的是系统已经上线，却发现因为代码中的某一个 bug 导致系统崩溃。在除掉这个已暴露的 bug 之后，为了快速的找出类似的但还未暴露的 bug，可以使用 FindBugs 对该版的代码进行扫描。当然，在维护阶段使用 FindBugs 往往是无奈之举，且时间紧迫。此外，如果本来在新版交付的时候就使用过 FindBugs 的话，往往意味着这种 bug 是 FindBugs 还无法检测出的。这也是 FindBugs 局限的地方。

FindBugs 出到目前的版本，功能已经相当强大，不过也有待完善的地方。从实际使用来看，有一些隐藏的 bug 并不能靠 FindBugs 直接发现。那么，可不可以撰写一个新的 Detector，来发现这种将一个未初始化的 reference 传来传去而形成的潜在的 bug 呢？理论上来讲，应该是可以的。这个 Detector 目前还未实现。哪位如果有兴趣的话，可以参考 FindBugs，Part 2：Writing custom detectors（扩展阅读）这篇文章，帮忙实现这个 Detector。实现一个新的 Detector，便可以检测出一种新型的 bug，这样不知又可以帮开发人员省去多少人工检查的时间，功德无量啊。

FindBugs 也不能发现非 java 的 Bug。对于非 java 撰写的代码，如 javascript，SQL 等等，要找出其中可能的 bug，FindBugs 是无能为力的。当然，javascript 中的 bug 似乎还不至于使系统崩溃，而 SQL 中的 bug 往往又跟业务逻辑相关，只要测试仔细一些应该是可以发现的。

FindBugs 不过是一个工具。作为开发人员，当然首先要在编程的时候努力避免引入 bug，而不要依赖于某个工具来为自己把关。不过由于代码的复杂性，一些隐藏的 bug 确实很难靠咱们的肉眼发现。这时，应用一些好的工具或许就可以帮你发现这样的 bug。这便是 FingBug 存在的价值。

为什么应该将 FindBugs 集成到编译过程中？

经常问到的第一个问题是为什么要将 FindBugs 加入到编译过程中？虽然有大量理由，最明显的回答是要保证尽可能早地在进行编译时发现问题。当团队扩大，并且不可避免地在项目中加入更多新开发人员时，FindBugs 可以作为一个安全网，检测出已经识别的缺陷模式。我想重申在一篇 FindBugs 论文中表述的一些观点。如果让一定数量的开发人员共同工作，那么在代码中就会出现缺陷。像 FindBugs 这样的工具当然不会找出所有的缺陷，但是它们会帮助找出其中的部分。现在找出部分比客户在以后找到它们要好——特别是当将 FindBugs 结合到编译过程中的成本是如此低时。

一旦确定了加入哪些过滤器和类，运行 FindBugs 就没什么成本了，而带来的好处就是它会检测出新缺陷。如果编写特定于应用程序的检测器，则这个好处可能更大。

生成有意义的结果

重要的是要认识到这种成本/效益分析只有在不生成大量误检时才有效。换句话说，如果在每次编译时，不能简单地确定是否引入了新的缺陷，那么这个工具的价值就会被

抵消。分析越自动化越好。如果修复缺陷意味着必须吃力地分析检测出的大量不相干的缺陷，那么您就不会经常使用它，或者至少不会很好地使用它。

确定不关心哪些问题并从编译中排除它们。也可以挑出 确实关注的一小部分检测器并只运行它们。另一种选择是从个别的类中排除一组检测器，但是其他的类不排除。FindBugs 提供了使用过滤器的极大灵活性，这可帮助生成对团队有意义的结果。

确定用 FindBugs 的结果做什么

可能看来很显然，但是团队中有多少加入了类似 FindBugs 这样的工具而没有真正利用它。深入探讨这个问题——用结果做什么？明确回答这个问题是困难的，因为这与团队的组织方式、如何处理代码所有权问题等有很大关系。不过，下面是一些指导:

可以考虑将 FindBugs 结果加入到源代码管理（SCM）系统中。一般的经验做法是不将编译工件（artifact）放到 SCM 系统中。不过，在这种特定情况下，打破这个规则可能是正确的，因为它使您可以监视代码质量随时间的变化。

可以选择将 XML 结果转换为可以发送到团队的网站上的 HTML 报告。转换可以用 XSL 样式表或者脚本实现。有关例子请查看 FindBugs 网站或者邮件列表（请参阅参考资料）。

像 FindBugs 这样的工具通常会成为用于敲打团队或者个人的政治武器。尽量抵制这种做法或者不让它发生——记住，它只是一个工具，它可以帮助改进团队成员代码的质量。

Findbugs 安装:

下载 findbugs https://sourceforge.net/projects/findbugs/

配置 findbugs 的 ant-task:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="crm_findbugs" default="findbugs">
  <!--set findbugs's home-->
  <property name ="findbugs.home" location="E:\ToolsForTesting\ant\findbugs"/>
  <!--set findbugs check rule file-->
  <property name="findbugs_include_filter" location="${findbugs.home}\findbugs-filter.xml"/>
  <path id="findbugs.lib">
```

```
            <fileset dir ="${findbugs.home}/lib">
                <include name ="findbugs-ant.jar"/>
            </fileset>
        </path>
        <!--set reference of task findbugs-->
        <taskdef name="findbugs" classpathref ="findbugs.lib"
classname="edu.umd.cs.findbugs.anttask.FindBugsTask"></taskdef>
            <!--define task findbugs-->
        <target name ="findbugs">
            <findbugs home ="${findbugs.home}" includeFilter="${findbugs_include_filter}" jvmargs="-
Xmx1024m" output ="xml" outputFile ="findbugs-report.xml">
                <class location =".\build\classes\com\sxxy"/>
                <auxClasspath path="${findbugs.home}/lib/findbugs-ant.jar"/>
                <sourcePath path =".\src"/>
            </findbugs>
        </target>
    </project>
```

自定义检查规则：

```
    <?xml version="1.0" encoding="UTF-8"?>
    <FindBugsFilter>
    <!--set filter rule of java class or jar files-->
    <!--all class files use HE rule to be checked-->
        <Match>
            <BugCode    name ="HE"/>
        </Match>
      <Match>
            <BugCode    name ="GC"/>
        </Match>
      <Match>
            <BugCode    name ="DM"/>
        </Match>
      <Match>
            <BugCode    name ="HRS"/>
```

```
    </Match>
  <Match>
        <BugCode    name ="SQL"/>
    </Match>
  <Match>
        <BugCode    name ="XSS"/>
    </Match>
</FindBugsFilter>
```

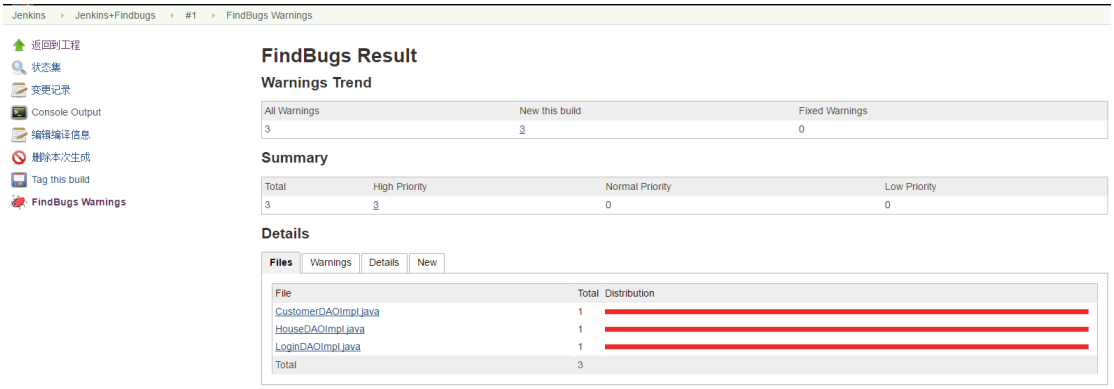
在 Jenkins 上集成 findbugs:

和 pmd 以及 checkstyle 一样创建一个自由风格的 job

源代码使用 svn 管理

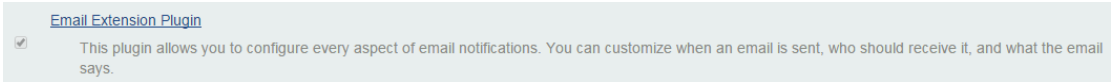
我这里直接上图:

构建成功后点击 FindBugs Warnings 查看结果:

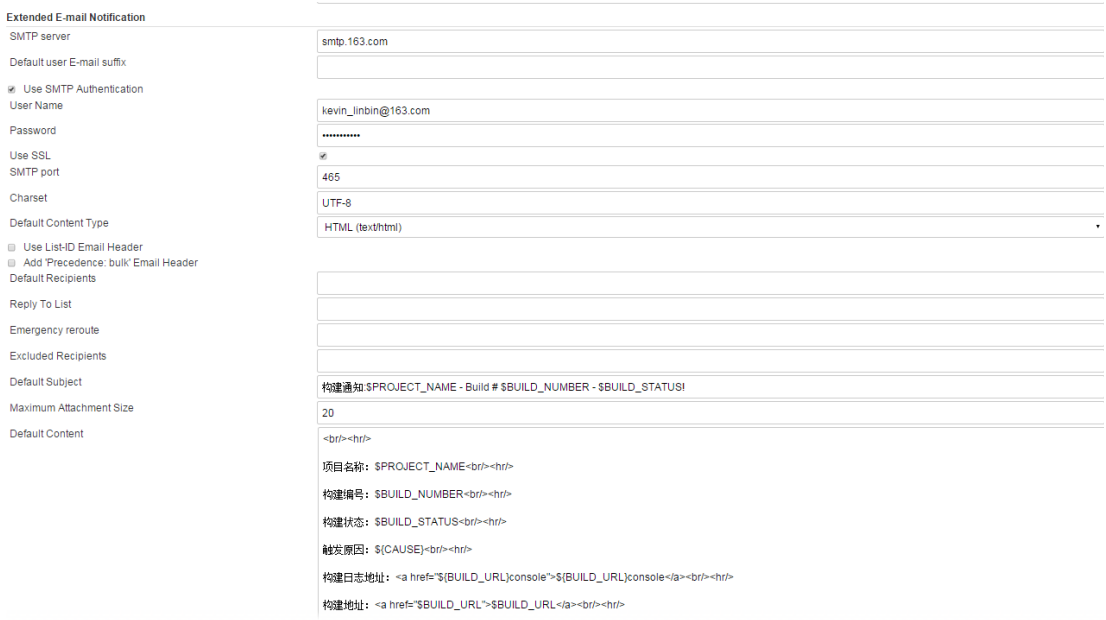


2.6 Jenkins 上的邮件扩展

安装邮件扩展插件



在”系统设置”----“Extended E-mail Notification”中配置扩展邮件项



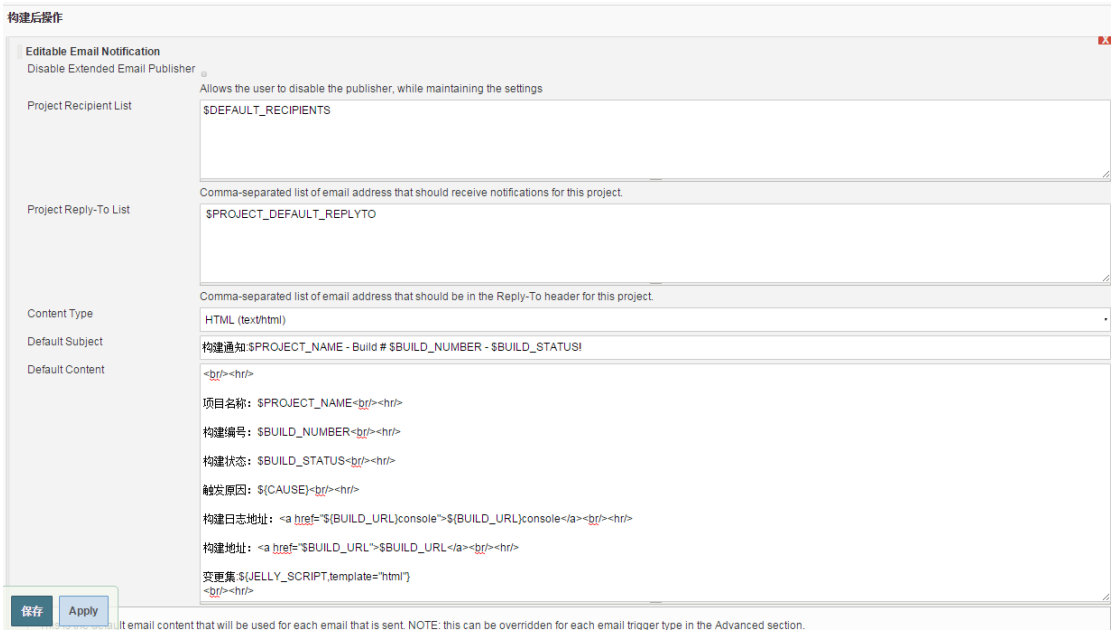
(ps: 注意这里的邮箱地址要和 Jenkins Location—系统管理员邮件地址一致)

我这里是使用 163 邮箱作示例，需要登录到 163 邮箱进行一些设置

在“设置”---“POP3/SMTP/IMAP”中开启 POP3/SMTP 服务和 IMAP/SMTP 服务

在“客户端授权密码”中设置客户端的授权密码(此处的授权密码即为 Jenkins 中的邮箱密码)

在 Job 中设置构建后的发送邮件步骤:

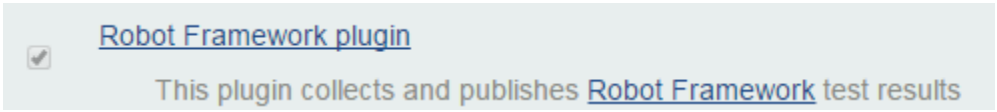


立即构建查看邮件内容:

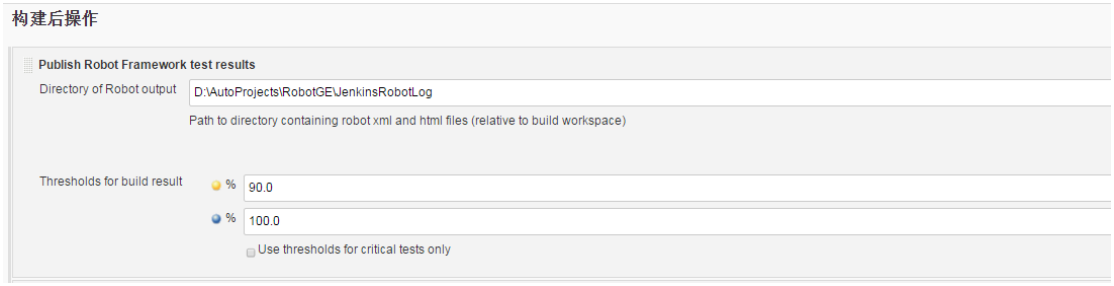
OK，大功告成！(PS：各位同学可根据自己需要来设计邮件内容模板)

2.7 Jenkins 集成 RobotFrameWork test results

首先在”系统管理”---“管理插件”中搜索 Robot 然后选择 Robot Framework plugin 插件进行安装

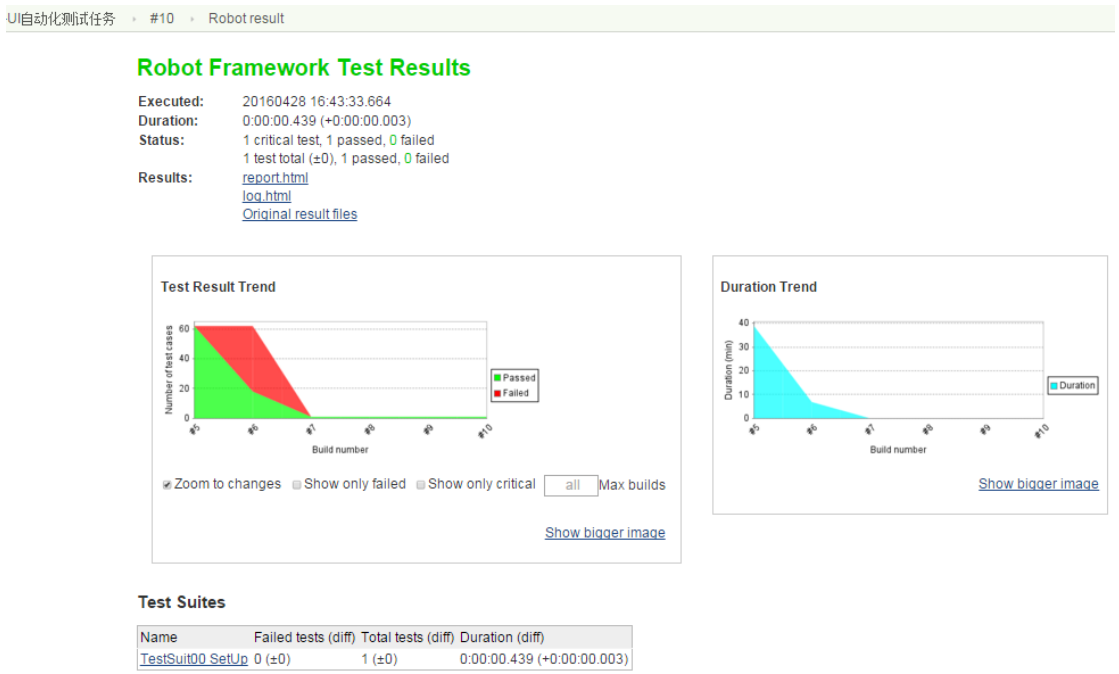


在 Job 总配置 RobotFramework 插件



立即构建该 job 来验证我们的设置

我也可以看到测试的完成情况，也可以点击 Robot Results 来查看详情



三：后记：

本想以此文作为《分层自动化测试—从入门到放弃》这门课程最后一个主题的文稿，然不忍窃为己有，遂投稿 51esting 以飨众亲； 历时数周终成此稿，不负晚上加餐后长的那二两肥膘。

《分层自动化测试—从入门到放弃》课程试听地址：http://www.atstudy.com/course/70

关于内存泄漏的总结性报告

◆作者：顾 翔

一、查看内存泄漏的方法

1. 安排有经验的编程人员对代码进行走查和分析，找出内存泄漏发生的位置

第一个步骤在代码走查的工作中，可以安排对系统业务和开发语言工具比较熟悉的开发工程师对应用的代码进行了交叉走查，尽早找出代码中存在的数据库连接使用后没有释放、对象使用后没有释放等故障代码。

2. 使用专门的内存泄漏测试工具进行测试

第二个步骤就是检测 Java 的内存泄漏。在这里我们通常使用一些工具来检查 Java 程序的内存泄漏问题。市场上已有几种专业检查 Java 内存泄漏的工具，它们的基本工作原理大同小异，都是通过监测 Java 程序运行时，所有对象的申请、释放等动作，将内存管理的所有信息进行统计、分析、可视化。开发工程师将根据这些信息判断程序是否有内存泄漏问题。

二、发生内存泄漏的原因

在 C++语言中，如果需要动态分配一块内存，程序员需要负责这块内存的整个生命周期。从申请分配、到使用、再到最后的释放。这样的过程非常灵活，但是却十分繁琐，程序员很容易由于疏忽而忘记释放内存，从而导致内存的泄露。Java 语言对内存管理做了自己的优化，这就是垃圾回收机制。Java 的几乎所有内存对象都是在堆内存上分配（基本数据类型除外），然后由 GC（garbage collection）负责自动回收不再使用的内存。

上面是 Java 内存管理机制的基本情况。但是如果仅仅理解到这里，我们在实际的项目开发中仍然会遇到内存泄漏的问题。也许有人表示怀疑，既然 Java 的垃圾回收机制能够自动的回收内存，怎么还会出现内存泄漏的情况呢？这个问题，我们需要知道 GC 在

什么时候回收内存对象，什么样的内存对象会被 GC 认为是“不再使用”的。

Java 中对内存对象的访问，使用的是引用的方式。在 Java 代码中我们维护一个内存对象的引用变量，通过这个引用变量的值，我们可以访问到对应的内存地址中的内存对象空间。在 Java 程序中，这个引用变量本身既可以存放堆内存中，又可以放在代码栈的内存中（与基本数据类型相同）。GC 线程会从代码栈中的引用变量开始跟踪，从而判定哪些内存是正在使用的。如果 GC 线程通过这种方式，无法跟踪到某一块堆内存，那么 GC 就认为这块内存将不再使用了（因为代码中已经无法访问这块内存了）。

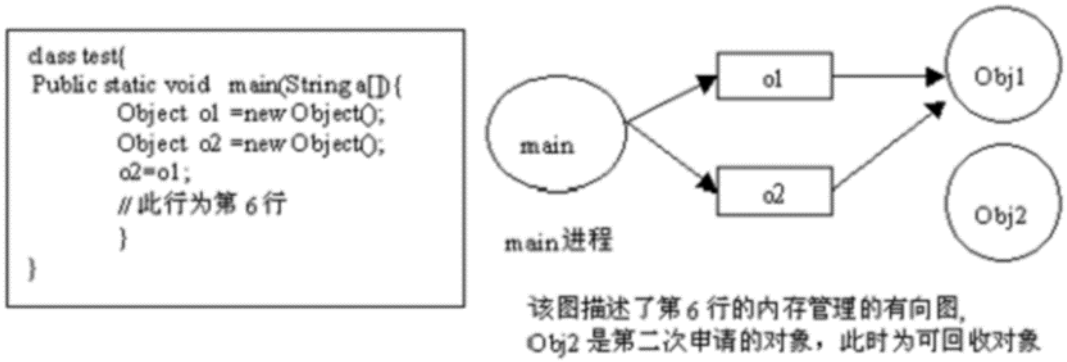


图 1 Java 发生内存泄漏的原因

通过这种有向图的内存管理方式，当一个内存对象失去了所有的引用之后，GC 就可以将其回收。反过来说，如果这个对象还存在引用，那么它将不会被 GC 回收，哪怕是 Java 虚拟机抛出 OutOfMemoryError。

Java 内存泄露类型与影响

一般来说内存泄漏有两种情况。一种情况如在 C/C++语言中的，在堆中的分配的内存，在没有将其释放掉的时候，就将所有能访问这块内存的方式都删掉（如指针重新赋值）；另一种情况则是在内存对象明明已经不需要的时候，还仍然保留着这块内存和它的访问方式（引用）。第一种情况，在 Java 中已经由于垃圾回收机制的引入，得到了很好的解决。所以，Java 中的内存泄漏，主要指的是第二种情况。

可能光说概念太抽象了，读者可以看一下这样的例子:

```
1  Vector v =new Vector(10);
2  for (int i=1; i<100; i++){
3  Object o=new Object();
4  v.add(o);
```

```
5    o =null;
6    }
```

在这个例子中，代码栈中存在 Vector 对象的引用 v 和 Object 对象的引用 o 。在 For 循环中，我们不断的生成新的对象，然后将其添加到 Vector 对象中，之后将 o 引用置空。问题是当 o 引用被置空后，如果发生 GC，我们创建的 Object 对象是否能够被 GC 回收呢？答案是否定的。因为，GC 在跟踪代码栈中的引用时，会发现 v 引用，而继续往下跟踪，就会发现 v 引用指向的内存空间中又存在指向 Object 对象的引用。也就是说尽管 O 引用已经被置空，但是 Object 对象仍然存在其他的引用，是可以被访问到的，所以 GC 无法将其释放掉。如果在此循环之后，Object 对象对程序已经没有任何作用，那么我们就认为此 Java 程序发生了内存泄漏。

尽管对于 C/C++中的内存泄露情况来说，Java 内存泄露导致的破坏性小，除了少数情况会出现程序崩溃的情况外，大多数情况下程序仍然能正常运行。但是，在移动设备对于内存和 CPU 都有较严格的限制的情况下，Java 的内存溢出会导致程序效率低下、占用大量不需要的内存等问题。这将导致整个机器性能变差，严重的也会引起抛出 OutOfMemoryError，导致程序崩溃。

三、内存泄漏的发生方式

常发性内存泄漏。发生内存泄漏的代码会被多次执行到，每次被执行的时候都会导致一块内存泄漏。

偶发性内存泄漏。发生内存泄漏的代码只有在某些特定环境或操作过程下才会发生。常发性和偶发性是相对的。对于特定的环境，偶发性的也许就变成了常发性的。所以测试环境和测试方法对检测内存泄漏至关重要。

一次性内存泄漏。发生内存泄漏的代码只会被执行一次，或者由于算法上的缺陷，导致总会有一块且仅有一块内存发生泄漏。

隐式内存泄漏。程序在运行过程中不停的分配内存，但是直到结束的时候才释放内存。严格的说这里并没有发生内存泄漏，因为最终程序释放了所有申 77 请的内存。但是对于一个服务器程序，需要运行几天，几周甚至几个月，不及时释放内存也可能导致最终耗尽系统的所有内存。所以，我们称这类内存泄漏为隐式内存泄漏。

四、内存泄漏工具

1. ccmalloc－Linux 和 Solaris 下对 C 和 C++程序的简单的使用内存泄漏和 malloc 调试库。
2. Dmalloc－Debug Malloc Library.
3. Electric Fence－Linux 分发版中由 Bruce Perens 编写的 malloc()调试库。
4. Leaky－Linux 下检测内存泄漏的程序。
5. LeakTracer－Linux、Solaris 和 HP-UX 下跟踪和分析 C++程序中的内存泄漏。
6. MEMWATCH－由 Johan Lindh 编写，是一个开放源代码 C 语言内存错误检测工具，主要是通过 gcc 的 precessor 来进行。
7. Valgrind－Debugging and profiling Linux programs, aiming at programs written in C and C++.
8. KCachegrind－A visualization tool for the profiling data generated by Cachegrind and Calltree.
9. Leak Monitor－一个 Firefox 扩展，能找出跟 Firefox 相关的泄漏类型。
10. IE Leak Detector (Drip/IE Sieve)－Drip 和 IE Sieve leak detectors 帮助网页开发员提升动态网页性能通过报告可避免的因为 IE 局限的内存泄漏。
11. Windows Leaks Detector－探测任何 Win32 应用程序中的任何资源泄漏(内存，句柄等)，基于 Win API 调用钩子。
12. SAP Memory Analyzer－是一款开源的 JAVA 内存分析软件，可用于辅助查找 JAVA 程序的内存泄漏，能容易找到大块内存并验证谁在一直占用它，它是基于 Eclipse RCP(Rich Client Platform)，可以下载 RCP 的独立版本或者 Eclipse 的插件。
13. DTrace－即动态跟踪 Dynamic Tracing，是一款开源软件，能在 Unix 类似平台运行，用户能够动态检测操作系统内核和用户进程，以更精确地掌握系统的资源使用状况，提高系统性能，减少支持成本，并进行有效的调节。
14. IBM Rational PurifyPlus－帮助开发人员查明 C/C++、托管.NET、Java 和 VB6 代码中的性能和可靠性错误。PurifyPlus 将内存错误和泄漏检测、应用程序性能描述、代码覆盖分析等功能组合在一个单一、完整的工具包中。

15. Parasoft Insure++－针对 C/C++应用的运行时错误自动检测工具，它能够自动监测 C/C++程序，发现其中存在着的内存破坏、内存泄漏、指针错误和 I/O 等错误。并通过使用一系列独特的技术（SCI 技术和变异测试等），彻底的检查和测试我们的代码，精确定位错误的准确位置并给出详细的诊断信息。能作为 Microsoft Visual C++的一个插件运行。
16. Compuware DevPartner for Visual C++ BoundsChecker Suite－为 C++开发者设计的运行错误检测和调试工具软件。作为 Microsoft Visual Studio 和 C++ 6.0 的一个插件运行。
17. Electric Software GlowCode－包括内存泄漏检查，code profiler，函数调用跟踪等功能。给 C++和.Net 开发者提供完整的错误诊断，和运行时性能分析工具包。
18. Compuware DevPartner Java Edition－包含 Java 内存检测,代码覆盖率测试,代码性能测试,线程死锁,分布式应用等几大功能模块。
19. Quest JProbe－分析 Java 的内存泄漏。
20. ej-technologies JProfiler－一个全功能的 Java 剖析工具，专用于分析 J2SE 和 J2EE 应用程序。它把 CPU、执行绪和内存的剖析组合在一个强大的应用中。JProfiler 可提供许多 IDE 整合和应用服务器整合用途。JProfiler 直觉式的 GUI 可以找到效能瓶颈、抓出内存泄漏、并解决执行绪的问题。4.3.2 注册码：A-G666#76114F-1olm9mv1i5uuly#0126
21. BEA JRockit－用来诊断 Java 内存泄漏并指出根本原因，专门针对 Intel 平台并得到优化，能在 Intel 硬件上获得最高的性能。
22. SciTech Software AB .NET Memory Profiler－找到内存泄漏并优化内存使用针对 C#，VB.Net，或其它.Net 程序。
23. YourKit .NET & Java Profiler－业界领先的 Java 和.NET 程序性能分析工具。
24. AutomatedQA AQTime－AutomatedQA 的获奖产品 performance profiling 和 memory debugging 工具集的下一代替换产品，支持 Microsoft, Borland, Intel, Compaq 和 GNU 编译器。可以为.NET 和 Windows 程序生成全面细致的报告，从而帮助您轻松隔离并排除代码中含有的性能问题和内存/资源泄露问题。支

持.Net 1.0,1.1,2.0,3.0 和 Windows 32/64 位应用程序。

25. JavaScript Memory Leak Detector－微软全球产品开发欧洲团队(Global Product Development- Europe team, GPDE) 发布的一款调试工具，用来探测 JavaScript 代码中的内存泄漏，运行为 IE 系列的一个插件。

五、解决方法

我们应该:

- 首先从代码审查开始着手，审查代码中可能出现内存泄漏的地方;
- 再次利用一些免费或开源的 Java 内存泄漏检查工具进行检查;
- 当以上方法都行不通的时候可以采取购买付费的内存泄漏检查工具。

关于内存泄漏检查上有一句名言:

内存泄漏分析是一项比较复杂的工程，人对内存泄漏的深刻理解才是最重要的，因为工具有时候无法检测出内存泄漏，所以关键工作要做好第一步

六、建议

在公司内部建立起定期的代码检查制度，结合公司的自身的情况，可以每两周或者每月搞一次，每次执行约为四到五天时间（第一天发布检查的代码及本次检查的重点（比如内存泄漏），第二到三天进行代码检查，第四天进行代码检查结果讨论及结果存储，以后一到二周时间进行代码修改及复审），代码的检查贵在平时的积累，而不是靠一时的突击。

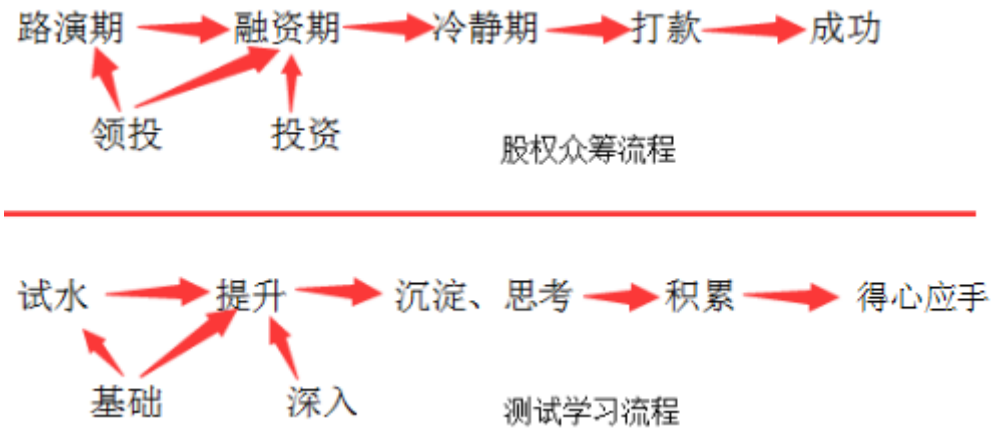
我的测试"众筹"

◆作者：流氓贵族

摘要：众筹一词对于大多数人来说已经不陌生了，现在也不仅仅只有一些小众人群才可以接触众筹，随着人们的思想越来越开放，对于理财方面也不在是存钱的保守方式，人们希望通过多种方式来积累和创造财富。在此背景下，许许多多的众筹平台如雨后春笋般的冒了出来，这就加大了金融监管部门的工作量。除了人为方面的监管，对于众筹平台技术方面的质量保证更加是不容忽视的。

我常常思考测试的测试的技术点在哪里？测试的难点在哪里？我认为测试的技术难点一定不仅限于学会几种测试方法，或者测试工具。都说实践出真知，在工作中，我逐渐明白如果把整个测试比作旅行，测试方法是导航，测试工具是交通工具，业务知识是旅行者的思维。测试人员需熟知测试的业务流程，运用各种测试方法，配合测试工具的使用完成整个测试。

一个合格的测试人员需要掌握的不仅仅是以上的几个技能，下面我就把测试人员所需掌握的技能以股权众筹业务流程的形式介绍一下。在介绍技能的同时，让大家了解一下股权众筹的业务流程。



图中红线之上为股权众筹的流程，红线之下为测试学习的进级阶段。

路演期/试水

发起一个股权众筹的项目，平台审核通过后首先进入路演阶段。在项目的路演阶

段，具有领投人身份的用户可以申请领投项目，经项目发起人和和平台审核后，即可申请领投人成功；路演阶段的项目不可以投资，只能够申请领投、约谈项目发起人。项目路演的主角是项目方，每个项目都有相关人员就项目的发展向投资者做介绍，以便吸引投资者投资。路演期的效果好坏对于项目能否成功有很大的作用。

测试人员首次接触测试领域，这一阶段对于测试人员来说就相当于测试生涯中的“路演期”。如果首次试水的经历很愉快，就会真正的踏入测试这个行业，如果入行不是很顺利，就会有放弃从事测试这个职业。

领投/基础

领投人在股权众筹中的地位很重要。大部分投资人在选择投资项目时，都会看看领投人是谁，牛逼的领头人将对跟投人的决策产生决定性影响。如果领投人非常知名，其他投资人跟投的愿望就更加强烈，项目的融资也就更顺利。深入来看这种追星效应，大家会发现领投人的作用：一是背书，二是资源。用自己的经验和人气给企业站台，给其他投资人带来信心；除了背书外，领投人还承担了对所投资项目的投后管理。领投人不仅向企业投资资金，还给企业带来了一定的资源、人脉和管理指导等经验。领投期按照投资金额收取领投人一定比例保证金。

测试学习过程中的学习测试需要掌握的知识就可以看作领投。需要扎实的基础和有效的技巧才能保证测试的顺利进行；另外，熟练的业务知识也是不可或缺的一项。总结起来，“测试”这个项目的“领投人”为测试基础知识、测试技巧、业务知识三位。有了这三位领投人的保驾护航，“测试”项目才能够顺利进行下去。

融资期/提升

路演期结束后，项目进入待融资审核阶段，平台审核通过后，进入融资期。在项目的融资阶段，具有领投人身份的用户可以申请领投项目，经项目发起人和和平台审核后，即可申请领投人成功；具有投资资格的用户可以项目进行投资；用户约谈项目发起人以便进一步了解项目的进展。如果融资期结束，融资金额没有达到募集金额的下线，该项目融资失败。由此可见，路演期的项目宣传以及领投人的选择至关重要。

测试入门之后，测试人员往往不满足于仅仅执行功能测试，希望能够有进一步的提高，这就涉及到了测试人员的转型。想要继续走技术道路的，可以选择自动化测试、性能测试、安全测试等方面；除此，还可以走管理道路，测试主管、项目经理都是不错的

选择。无论往哪条路上转型都需要增加知识储备，强化已有的技能，在这个阶段，需要我们注重自身的专业知识和素质的提升。

投资/深入

股权投资指通过投资取得被投资单位的股份。企业（或者个人）购买的其他企业（准备上市、未上市公司）的股票或以货币资金、无形资产和其他实物资产直接投资于其他单位，最终目的是为了获得较大的经济利益，这种经济利益可以通过分得利润或股利获取，也可以通过其他方式取得。在融资期，具有投资资格的用户可以根据项目领投人和项目的发展来判断是否对该项进行投资。投资期按照投资金额收取投资人一定比例保证金。

在测试的提升阶段，测试人员可以深入的理解测试职位的核心价值和所需的能力，补充编程、数据库等方面的知识，加强沟通、文字方面的能力。通过自身的专业知识和素质的提升来严把测试产品的质量关，提高自身的竞争力。

冷静期/思考、沉淀

股权众筹确实可以为我们带来高额收益，但却不要忽视了它的高风险性。所以股权众筹需要冷静期，创业者也要冷静思考自己是否适合股权众筹。

项目进入冷静期，分两个阶段，冷静期间，冷静期后。

融资中项目到期后，系统自动判断是否进入冷静期，条件为募集金额需大于等于最低融

资金额。在冷静期间（一般为 3 天）内，已经投资的投资人可以退出，退出后扣除相应违约金，剩余保证金解冻。已经投资的投资人也可以进行补投，补投时，冻结相应的补投保证金。3 天后，系统自动判断是否能进入冷静后期，条件为大于等于最低融资金额，达到条件的进入冷静期后，达不到的进入冷静失败项目列表。

冷静期间过后，进入冷静期后，发布人可以勾选投资人，勾选的投资人成功投资，未勾选的投资人退出项目，投资金额返回投资人账户。勾选完成发布人提交打款审核申请。提交条件为融资金额必须在最低融资金额和最高融资金额之间。

在测试的入门和提升阶段，我们学习了很多的测试知识，在工作中也逐渐积累了一些测试的经验。对于这些知识和经验，我们要学会取其精华，去其糟粕，对于知识的掌

握应该求深不求广。通常到了这个阶段，我们也许会遇到瓶颈，进步没有入门和提升阶段那么的快，那么利用这段时间，思考什么对于自己才是有益处的，学会沉淀浮躁。

打款期/积累

发布人提交后，项目进入待打款阶段。投资人在 10 天内打款，打款金融等于投资金额减去保证金差额部分。超过 10 天未打款投资人扣除全额保证金。10 天后项目进入打款期后复审。复审通过后，项目进入待划转列表。复审不通过，项目金融打款期后审核失败项目列表，冻结的款项全部解冻，未打款的投资人保证金扣除。

经过了沉淀和思考以后，这个阶段我们的心态稳定、没有了初绽头角时的莽撞，此时的我们也大都了解自己真正想要的、确定了发展的方向。这个阶段就需要我们整理和总结掌握的知识和经验。

成功/得心应手

投资人在打款期，补齐剩余带有款项后，由平台进行资金划转操作。此时的项目才完全众筹成功。项目众筹成功后，由平台或项目发起人对项目进行信息纰漏。信息披露主要是指公众公司以招股说明书、上市公告书以及定期报告和临时报告等形式，把公司及与公司相关的信息，向投资者和社会公众公开披露的行为。

经过的知识的不断补充和精炼，对于本职工作逐渐得心应手。大神都是从小白阶段一步一步走过来的，在这个阶段除了小白自身的努力，各个行业前辈创造的环境也对小白的成长有很大的帮助。穷则独善其身，达则兼济天下。要懂得感恩和分享，不吝惜传授自己的知识和经验。

测试经济学

◆译者：于 芳

概述：我们所做的每件事都有经济上的影响因为我们所做的事情有成本和收益在里面。测试时有关快速获得真实的反馈，消减浪费的测试活动，和在我们的程序面前放一面镜子。去理解这些活动的成本并将精力投资导向最有利的地方因此变得具有风险。

可能你会认为当某人决定写一本书的时候，所有的东西已经在他大脑中了。错误。拿我有一些话题来开头的单元测试的书来说，它在四个月左右的时间后才有了一个普通的思路浮现。这个思路就是经济学，甚至让我自己都吃惊。但是我已经开始看到经济学怎样充斥进我们在软件开发过程中所做的一切，具体到测试上。

如果你曾做过某种专业的开发工作，你很可能开发了某些特定的技能。当我们应用这些技能时，我们称之为工作。我们不会在工作的时候对其思考过多--有时候在工作后甚至更少--但是每件我们做的事情都有经济学影响因为我们做的事情有成本和收益因素。测试活动，不管谁在实际做这项工作，也是同样的道理。

历史

软件是一个相当年轻的行业，而测试作为一个职业更是如此。我们需要从一开始在测试人员存在之前就正确测试。在二次世界大战后，软件行业增长得越来越庞大，而伴随而来的是注入质量和维护成本以及名誉损失的经济风险。回到那个时候，开发人员测试软件；那是工作的一部分。那个时候，在 20 世纪 80 年代，事情发生了变化：计算机变得越来越便宜，而操作系统最终稳定而很适应。而现在软件编写便宜多了。

至少，如果你曾拥有过开发人员。开发人员在以前是一个相当稀少的资源。现在市场需要更多的开发人员。这导致行业的最有逻辑性同时又愚蠢的决定之一。很多机构决定他们会通过雇佣更廉价的人取代那些做测试工作的人，保持现有的开发人员在功能开发上使力。这在经济学上说得通，但是一旦开发人员从对代码负责的桎梏中脱离开来，软件质量多数时间被牺牲了。

第一批测试人员是缺陷捕捉者。今天的测试人员做的更多：他们从各个方面汇报产品的状态。为此，他们需要了解风险在哪，市场情况和用户情况。他们通过探索产品上的不确定区域来降低这些风险，证明预想，并且建议修改。他们定义测试策略，知道他们在这个世界上并不总是有时间因而他们需要对自己的活动做优先级排序。为了最大化测试效率，他们也需要对测试技能和他们团队里的活动有所了解。好的测试人员理解经济学概念。

一个测试人员身上要聚集这所有的资质？难怪我们找不到很多好的测试人员，而那些我们能够找到的要价堪比开发人员（嘘！）。

机构

让我们看下测试活动怎样影响底线，并从测试人员开始。我们都知道在敏捷团队中，测试人员是嵌入到团队里的。这样做有经济上的益处：当我们与团队搭配一切工作时，反馈流向很快且合作加速，变得更加有效率。然而，成熟的敏捷团队现在面临另一个挑战。

想象我们有 5 个工作在一个网站应用程序的团队，其中一个测试人员是性能测试专家。所有的人都需要他的帮助，但是我们怎样管理这件事呢？我们可以创建不同的工作模型来满足不断增长的组织里的性能测试需求。我们可以培养每个团队里的每个测试人员性能测试的技能（尽管那样做可能会过度--应当每个人都需要知道所有事吗?）那名测试专家可以在团队间轮转，但是他并不属于任何一个团队。或者他可以提供其咨询服务给其他团队来限制在当前团队的工作时间。

那不是一个流程问题。退回去，你会看到更大的问题：招聘，放置和对测试人员的保留应当怎样在一个敏捷的时代运作。我们需要多少，而他们的技能集又应当是怎样的？他们在团队里的地位是怎样的？对开发人员我们应不应当问同样的问题呢？这是一个比较大的可以造就或毁坏一个机构的人力资源问题。

我们都知道缺陷是垃圾。我们需要重现他们，用文档记录他们，然后修复他们。

但是那不是全部：在我们将他们记录下来之后，我们需要在一个优先分配会议上讨论他们。如果他们能够经过一轮两轮的挑战，我们会再讨论他们。产品拥有者然后围绕备份日志一遍又一遍讨论商议他们。

紧急缺陷更糟糕。我们确实需要快速修复缺陷，从而不会有更多其他的讨论。但是

当我们将所有事情都放来修复缺陷，我们要浏览内容切换，通宵工作，而且无需通知地进入一些新的缺陷因为我们很匆忙。我们本意是好的，但是我们损害了质量，这会在以后让我们付出更多代价。

调试是修复缺陷最有欺骗性的部分因为我们已经习得认为它非常有用我们离不开他。然而，精通并不总是能够加快事情的进展。我们轻易投入到调试的时间可以用测试来减少。这可以解放一些时间来编写更多新的功能。

测试这里并不是对的解药。合适的预防，需要开发人员具备测试技能，代码评审能力以及跟测试人员合作的能力。

测试架构

大多数情况下，架构只创建一次。也许我们在创建他之前会思考良久。或者也许架构会需要打包很多次。不管如何，头脑中没有可测性，就会变得难以测试。这要么会导致不去测试难以测试的领域，要么是为了测试他要花费巨大精力。在经济学术语里，我们要么引入大的风险要么放缓开发工作。两个我们都不想要。当我们有一个可测的架构，我们可以根据我们拥有的需求对其修改。它可能仍然会让我们花费很多，但是已不足够去完全遗弃那个主意。让自动化测试来围绕架构，我们可以做必要的变动。

围绕该程序做测试也能够帮忙延迟“大的重写”。我们今天编写的代码应当能够持续使用 20 年。没有测试，它会一直使用直到开发人员决定从头从底层开始写程序比继续维护现有的代码更快。大的重写耗费巨大并且风险很大，而这样我们会想要尽可能延误他们。测试帮助我们做到那些。

策略

一个好的测试人员具有一个项目经理的技能。当修订一个策略时，他需要将开发人员怎样测试他们的代码考虑进去这样他便可以集中精力在覆盖较少测试的部分。他持续不断地检查反馈循环来确认哪里可以减少。他会看自动化测试执行的长度并且想办法来减少它，哪里可以在继承测试上投资，这样可以给予他更多自信但是也会耗费更多时间和资源来创建和调试程序，而需要多少手工测试时给予自动化测试的数量的。

如果他决定使用验收性驱动测试开发绑定测试驱动开发作为其策略的一部分，他对经济影响推动有明确的想法。使用测试第一的方式将每个人跟业务需求排列在一起。那意味着是更多应用程序想要的代码，更少你不会需要的代码。测试驱动开发也会帮助到

这个，因为它让开发人员在他们实际编写代码之前去思考。在编写代码前思考是我所知道的最好的预防缺陷的方法之一。

一个策略并不会结束于测试活动，却会以理解市场而结束。今天这点对很多机构都很重要，尤其是刚创立的公司，要将市场放在首位。那需要不同的策略。与对自动化进行大投资相对的是，我们的测试人员只做足够的手工测试。开发人员根本不会去自动化可能不会成为最终产品的代码，因此一个足够好的被手工测试过的产品足够用来做试探。

我们职业人员通常瞧不起这样低质量的产品。然而，有时它可能是一个对的决定，而我们总可以后来添加质量，尽管可能会付出更大的代价。

经济无处不在

我们不管做什么事都是有价值和成本在的。测试是快速获取真实反馈，减少浪费的测试活动，并将镜子放在我们的程序之前的事情。去理解这些活动的成本并将精力投资在最值得投资的地方变得很冒险。

成为一个职业人士需要理解经济学。你是职业人士吗？开始思考那些数字吧。

如何在合适的时机引入接口测试 V0.3

◆ 作者：刘铭辉

一、背景介绍

首先要简单说下推荐系统，这是一个采用 thrift 协议为其他服务提供相关推荐数据的后台服务系统，该服务主要包含 q2u（为用户推荐感兴趣问题），u2u（为用户推荐感兴趣的用户）等多个子服务，要求针对每个子服务做负载测试，寻求系统最大所能承受压力。作为小白的我思考之后很快制定了测试方案：

有过重大专项测试经验的朋友一定会有这样的体会：‘功能测试初期发现的缺陷（Bug）比预期的多，多到了让人怀疑程序是否通过了前置的单元测试’。虽然这样的情况屡见不鲜，但事实并非如此，以至于开发人员来说，大量的缺陷‘提交-修复－再提交’的恶性循环，使得整个项目进度受到了影响。

经过分析，我们发现这样的场景往往发生在多模块松耦合架构下系统间交互接口的通信或相互调用上。尤其是涉及到多系统间接口差异大、报文流转复杂、业务场景多样化的时候。虽然开发人员通过了程序模块的单元测试，甚至通过了一些基本的集成测试，但是业务场景覆盖性方面难以保证。如果直接进行‘关注度在交易级的’功能测试，而忽略‘接口级’的覆盖，测试初期的缺陷（Bug）逃逸率也必然相对较高，如果不及时处理，越靠近实施后期，修复这些缺陷的成本和风险就越大。

不过也不用担心，这些问题是有规律的：

1. 被触发的条件简单

虽然利用简单的测试方法和正常的业务流程就可以使一部分缺陷暴露出来，但并不能使问题暴露达到预期的最大化。

2. 缺陷集中趋势表现成模块化发展

程序的缺陷往往在某一个或某几个模块中表现非常突出，正所谓“缺陷的区域性”

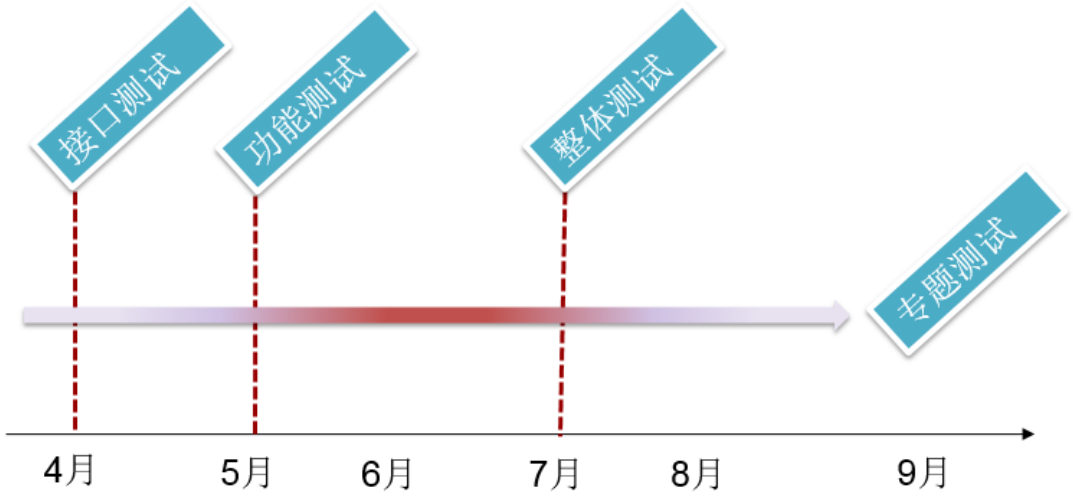
特点。

3. 缺陷主/子流程节点叠加

前一缺陷修复后，往往会暴露更深层次的问题。一些主流程节点的缺陷被修复后，子流程节点和报文流转的下游往往出现更多或更深层次的问题。

4. 接口不规范、需求理解差异化导致存在缺陷

对于需求不明确、接口定义模糊的情况会造成测试侧重点偏离的情况，使一些问题不能及时发现。



在哪个时间引入“接口测试”最“划算”呢？单元测试之后和功能测试之前。因为：

1）可以扫清低级接口问题，使得缺陷在早期暴露；

2）可以提升代码质量，变向缩短功能测试、专题测试、整体测试周期；

3）可以通过接口分析，能分析出不易想到的业务场景；

4）接口级问题修复/测试，高速迭代时间成本更低，程序修复速度更快，投入产出较高；

5）更贴近灰盒测试，有助于测试人员将测试从抽象到具象，让业务、技术能力同步提高。

而且，接口测试和功能测试的关注点有所不同。

接口测试相比功能测试颗粒度更细，且覆盖的偏重点在于交易的报文层面，它并不是黑盒测试那样在功能界面上通过交易按钮触发程序流程，覆盖测试案例。接口测试一般安排在功能测试的前期，相比功能测试，有着案例覆盖简单、接口级的粒度覆盖更细、底层问题暴露几率较大的优点。多系统交互的项目测试活动中，在功能测试前期引入接口测试，能较大的降低缺陷逃逸率，减少后期底层代码的修改而造成的项目风险。同时，也能避免逃逸的缺陷影响到其他交易模块，被其他模块接口适配，造成潜在的风险。

所以，在涉及多系统交互的重大专项功能测试前期，我们要引入接口测试。

二、那么如何进行接口测试?

1. 接口测试前期的准备工作

- 业务需求说明书
- 接口规范标准文档
- 数据字典
- 常用日志查询工具（SecureCRT/Xshell/flashfxp 工具……）

2. 设计接口测试案例:

接口规范文档定义程序接口规则，报文按照接口规定的格式要求来流转。测试人员依据接口规范来文档、数据字典、业务需求说明书等文档来设计测试案例。案例设计的原则“重报文流转，轻业务场景”，案例执行通过报文日志查询结合前端交易结果来验证测试结果。

案例编写需要在报文层面与业务场景相结合往来账，充分考虑正常异常场景:

- 边界值验证：报文所允许的边界值，使用在最小值、略高于最小值、正常值、略低于最大值、最大值和超大值处取输入变量值，记为：min、min+、nom、max-、max、max+
- 有效值、无效值验证：有效值略。无效值验证输入无效的数值类型（空格、符号）
- 最大长度输入：根据报文标准，输入报文标准中要求域的最大长度。如地址、附言、金额域等等

- 全角半角验证
- 户名、地址、附言等字段的特殊字符验证
- 涉及到金额的关注金额是否被放大、被缩小、小数点错位等现象
- 报文域是否强制项判断。报文标准中关于报文的结构说明“属性”一比如[1..1]表示该域最少个数为 1，最多个数为 1，也就是该域是强制项，报文中必须要有的；如[0..1]表示该域最少个数为 0，最多个数为 1，也就是说该域费强制项
- 业务种类与业务类型、报文类型对应关系检查
- 报文流转明确，明确自己测试的报文流转，确定各种流转的场景：如正常、（被对手行/被人行）异常、拒绝等等
- 确认会计分录是否正常
- 账户正常、异常状态的判断

报文体域确认						往帐测试结果	来帐测试结果	测试负责人	问题记录
报文要素	域名	属性	类型	场景	预期目标				
Message root	<FIToFICstmrCdtTrf>	[1..1]		报文正常被人行受理	报文流转正确无误，被正常轧差			刘铭辉	
				报文被人行拒绝	报文流转正确无误，接收通用处理确认报文			刘铭辉	
				对手行回应拒绝	报文流转正确无误，接收通用处理确认报文			刘铭辉	
				报文被人行丢弃	报文流转正确无误			刘铭辉	
.GroupHeader	<GrpHdr>	[1..1]						刘铭辉	
.--MessageIdentificat	<MsgId>	[1..1]	Max35Text	正常场景				刘铭辉	
				报文标识号重复被人行911丢弃				刘铭辉	
.--CreationDateTime.报文发送时间	<CreDtTm>	[1..1]	ISODateTime					刘铭辉	
.--BatchBooking.批量报文标识	<BtchBookg>	[1..1]	Indicator		固定填写false			刘铭辉	
.--NumberOfTransactions.明细业务总笔数	<NbOfTxs>	[1..1]	Max15NumericText	贷记业务明细笔数正常值1，系统处理正常	1.报文连通性处理正确 2.交易界面展现正常			刘铭辉	
.--ControlSum.明细业务总金额	<CtrlSum>	[1..1]	DecimalNumber	报文所允许的边界值，使用最小值，记为：min	1.报文连通性处理正确 2.交易界面展现正常			刘铭辉	
				报文所允许的边界值，略高于最小值，记为：min+	1.报文连通性处理正确 2.交易界面展现正常			刘铭辉	
				报文所允许的边界值，正常值，记为：mon	1.报文连通性处理正确 2.交易界面展现正常			刘铭辉	
				报文所允许的边界值，略低于最大值，记为：max-	1.报文连通性处理正确 2.交易界面展现正常			刘铭辉	
				报文所允许的边界值，最大值，记为：max	1.报文连通性处理正确 2.交易界面展现正常			刘铭辉	
				报文所允许的边界值，超大值，记为：max+	报文不能被处理系统判断合理			刘铭辉	
				最大长度验证：根据报文标准，输入报文标准中要求域的最大长度	1.报文连通性处理正确 2.交易界面展现正常			刘铭辉	
				金额带小数点				刘铭辉	
				金额不带小数点				刘铭辉	
				金额全角验证	非法输入系统判断正确			刘铭辉	
				金额半角验证	合法输入系统判断正确			刘铭辉	

（截图去掉一半）

3. 接口测试案例执行流程：

（以网上银行系统报文测试流程为例）

往帐报文:（交易界面发起→业务系统中后台→前置机→对端系统）

1.报文从交易界面发起发出后，检查网银交易明细、核心客户账务处理是否如报文发起一致

2.对端系统正常接收报文无报错

3.在不同的测试案例场景下，对端系统查询报文各个域的值与发起的报文要素一致（如果某些要素不体现在页面上，则通过查询系统日志和报文要素的方式验证）

来帐报文:（对端系统→前置机→业务系统中台→核心系统/交易界面）

1.查询日志报文接收到的报文与对端系统发起一致，对比各个要素的值正确无误（如果某些要素不体现在页面上，则通过查询日志和报文要素的方式验证）

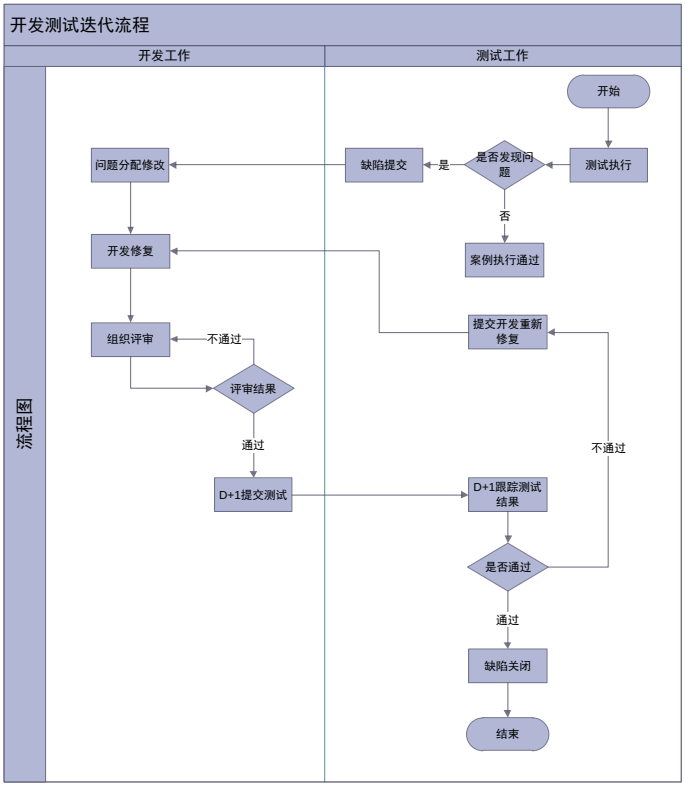
2.查询核心账务处理正确，会计分录正确

3.前台界面展现正确无误

4.核心数据库查询表中的数据与对端系统发起的数据一致

4. 接口测试过程中的问题跟踪复测-测试/开发缺陷迭代流程

为了保证项目进度，此测试阶段采用开发/测试迭代的方式进行。推荐的流程方法:即每天测试人员发现的问题在当天下午约定的时间点提交缺陷，开发人员第二天约定的时间点前解决问题并提交测试，迭代周期为 D+1。详见下图:



这么做有两方面好处:

1、有效的迭代能够使问题高效的解决，修复速度提升的同时，结对测试正交矩阵交叉案例组合也能够使一些低级问题得到最大程度的暴露，这些低级缺陷早期暴露能够

使后期测试更有针对性，为后续组织专题、整体测试打下基础；

2、新搭建的开发测试团队在连通性测试阶段彼此高密度的沟通磨合有助于形成团队默契。

三、总结

在功能测试初期引入接口测试，能够使大部分低级的问题暴露，从测试活动的源头避免后期更多的问题。同时在测试过程中，也顺便将接口规范需求进行了二次梳理，明确后期的开发测试需求。接口级别的测试使测试人员对业务和系统有了更深层次的理解。

以前项目从敏捷中学到的 7 个教训

◆译者：王宏瑜

概要

通过 11 年的软件开发实践敏捷已经成为有效的方法。这些方法不是对以前的项目运行方法做了全面的变革，而是融入了以前的软件开发生命周期的内容。

主要课题

●敏捷软件开发是与以前项目完全不同的流程，是以一些极为有效的开发的做法展开的。

●以前的项目，区别于至今为止的开发流程的观点，有必要找到在现在的流程中也可以被采用的有效的开发方法。

推荐事项

- 从敏捷中获得的开发方法的进步不容忽视。
- 方法的评价和试点操作，对本公司来说是找到了有效的做法。
- 将开发和测试整合到一个时间段。

分析

敏捷开发破坏性的存在，常常是软件开发中产生摩擦的重要因素。虽然在围绕最好的软件开发流程的议论中迷失了，但迄今为止多数的创新的有效的开发的做法，还是在敏捷项目中被用来进行开发了的。本研究报告是对敏捷的开发方法中可以适用于以前开发过程中的 7 个方法的验证。这些方法大多数，如果要追述到遥远的过去的话，是可以追溯起源到 2001 年的「敏捷・软件开发宣言」的，所以把他们的产生归功于敏捷的话可能不算公平。但是，敏捷项目中被广泛使用的这种方法，是被提炼的结果。这 10 年间这些做法是通过敏捷的措施结合起来才变得这么强大。

软件方法论，大致区分以下两个要素（参照图 1)。

- 项目运营中使用的流程
- 产品开发中用于设计/开发/测试的一连串的做法

敏捷项目，虽是采用与以前项目从根本上就不同的流程，但做法很多是也能够适用以前项目的。本研究报告是对可以适用于以前项目的适用效果，敏捷开发的 7 个做法进行解说。

图 1 流程 + 实践方法 = 方法论

简单的设计与编码

敏捷开发采用的最主要的开发方法是尽量用简单的代码。敏捷・软件开发宣言将这种简单表现为「最大限度的做到不浪费」，为了适用这个概念，追求开发功能用最简单的设计和编码。

在以前的项目中，尽可能设计通用性的代码，重视注释。根据这种方式虽然扩展性得到了优异的代码，但是通常会变成更复杂抽象化的代码。扩展性大多不能得到活用，产生了相当大规模的投机开发。投机开发就像在买住房之前就先开发了旁边。近年不景气的结果就是在一大片荒地上只有住宅。（日本土地私有，买的土地归自己所有。）投机性的软件开发经常不被使用，结果就是增加了不使用的编码的开发，测试，维护的成本。

对敏捷开发的批判一般比较错综复杂，是会产生难维护的代码。虽然实践不充分的敏捷项目可能会有那样的危险，但是最好的敏捷团队是不会产生那样的代码的。只要简单的软件设计和编码被正确的适用的话，得到快速而复杂的解决方案是没有问题的。简单的编码里是泥团似的结构设计或者编码的话就完全没有意义了，简单的编码应该具备漂亮，马上就能明白的功能性。再以宅地开发最为例子的话，简单的设计，虽然开始时建造的住宅和道路少，但是随着需求增加明显存在有增加的空间的。同样，很好的简单的软件，是正好对应现在的需求，扩展也容易，具备明确的扩展路线的。

敏捷的教训:

- 实现能满足现在需求的最简单的设计和编码。
- 代码要设计能够对应扩展，但是扩展的开发有必要拖延到需要的时候。

新信息的反映

敏捷项目鼓励在项目中途就接受变化。敏捷项目的本质是以从问题显现开始，解决问题最好用最简单的解决方案为目标的。在开发过程中，新信息相应的编码重构多次是很正常的。以前的项目，重复作业都是浪费的。在敏捷项目中，重构是与代码改善是关联的，所以被视为生产性。熟练的敏捷开发者，将复杂的代码使用更简单的解决方案进行重构是感到自豪的。废弃加入大量的复杂代码，用简单的合理代码进行替代，当然被视为积极的开发。如果没有支援的话，恐怕会容易变成无聊的工作，所以许多集成开发环境加入了重构工具。

以前的项目，具体说明从开发解决方案开始。这样的在前期进行整体设计的方式，设计的大部分情况还在不是充分理解时就定下来了。Domain 驱动设计的提案者 Eric Evans 先生，描述前期设计是在「混沌中被封闭的事情」。项目过程中判明新信息有以下两种类型。

- 外部设计的洞擦力-为了最好的解决方案，应该如何与用户互动的最新理解：如何发挥外部设计的洞擦力是敏捷流程的核心能力。在以前的项目过程中，发挥外部设计洞察力在这种程度上是很难的。如果与用户的互动不从根本上跟着变更的话，变更管理流程可能变得很容易，但是同时也限制了解决方案。

- 内部设计的洞擦力- 关于解决方案最好的构建方法的最新理解：内部设计的洞察力是在开发期间对设计进行重构的活用。经验丰富的设计者，要有前期设计必要性和根据项目过程中详细设计可能变无效来做调控的掌控能力。

敏捷的教训:

- 在开发期间获得的新知识，每次发现更合适的简单的解决方法都重构代码和设计。

- 不是为了回避重复作业，鼓励为了解决方案改善的重构。熟练的设计者对于事前设计的必要性，开发过程中（特别是比较详细的层次）设计变更的的现实情况有平衡掌

控的能力。

根据模型设计的示范

敏捷项目，对于软件的终端用户或者其代理人来说，极为重视定期进行演示。正是这个演示，是在敏捷中能不断提供终端用户解决方案的主要原因。根据演示，用户对设计变更，在更廉价的时期能具体的把握解决方案。敏捷项目中对需求如何处理，可以参照【敏捷和 Rest 架构：需求定义与管理】

在以往项目中，项目结束应该开发实现的功能，通常在需求式样书上做了详细的记载。但是，大部分的情况，终端用户不能想象出解决方案，在理解这些式样书时就承认了需求。

敏捷的教训

- 式样书尽可能做成视觉性的东西。用户界面方案的模型，设计样式对终端用户来说时切身的东西很有用。

- 开发阶段反复进行演示。把做成不合适解决方案的作为锻炼时机，不是开发途中的好就是好，而是比解决方案展开后会更好。

组对合作

敏捷，软件开发是一个协调性的过程。组对・编程设计是说软件工程师配对一同合作开发的过程。这样的话，能够持续进行代码修改，也鼓励了有着不同技能的工程师之间合作和交叉训练。可能看起来两个工程师进行一个工作是浪费的，但调查显示，组对合作编程的生产性和个人编程是同程度的（参考依据 2）。配对合作编程对交叉训练和提高代码品质有好处。

敏捷的教训:

- 鼓励配对合作编程。大多软件工程师本质是内向的，如果强迫搭档有可能会损坏生产性。但根据只针对配成对的工程师分配特别的工作环境，通过口头促成组队还是可能的。

- 对工程师的配对工作进行划分，设定对两者（至少对代码审查等）的期望值，促进合作编程设计。

开发和测试的融合

敏捷项目的开发过程的编码阶段和测试阶段是不分离的，测试在流程中是要尽可能早开始的。开发和测试融合成在同一个阶段了，组织构成也要变化，开发小组要有开发和测试人员构成。测试人员，从开始就要参与项目，这样可以加深对开发需求的理解。一般代码开发者以单体测试的形式记录白盒测试。根据功能/验收测试的自动化形式进行的黑盒测试，则由专职的品质保证（QA）测试人员在编码人员支援下进行。为了保证测试适度的独立性，大部分的企业采用矩阵结构，面向 QA/测试人员准备另一个管理体制。

大多敏捷团队，进一步推进这个流程，在编码前进行测试设计。这样做的话，团队中的 QA 人员，就离开发更近，测试时就可以控制住大方向偏了的风险。编码前测试设计，测试要确认开发是否与当初的需求要件是否一致。另外，对于因为误解需求引起的缺陷，也更容易在早时期发现修改。

如果采用自动化测试的话，测试设计的成本高，但是执行成本要远远的低。从初始测试记录开始，测试就获得了最大的价值。因为测试能够发现编码开发中途的缺陷。在单体测试级别，测试初始记录被称为测试驱动开发。白盒测试方式进行的功能/验收测试的情况，被叫做动作驱动型开发或者验收测试驱动型开发（ATDD）。敏捷测试详细，请参照【敏捷和 Rest 架构:软件质量与测试】

大多以前的的开发项目发展，变成可以开发阶段和测试阶段并行执行。但是多数情况，具体的测试用例的编码还是为了让测试 QA 能够继续完成流程的。结果，开发和测试至今仍是分别封闭独立的。因为测试延迟导致发现缺陷也跟着延迟，缺陷发现时修改成本膨胀。因为缺陷数，修改那些缺陷的劳动力都是未知的，所以大多数以前项目，测试阶段日程安排什么的都是完全失败的。

敏捷的教训

- 组成执行开发和测试的岗位横断的开发团队。
- 流程早期，如果可以的话，在编码前进行自动化测试设计。

持续集成

敏捷项目，为了尽可能早期发现缺陷，采用持续集成和测试。这样的话，废除项目尾盘安定化阶段。采用测试框架记录自动化测试。持续性集成服务器连接代码版本控制

系统，每次代码变化提交就开始打包。敏捷项目持续集成是如何进行的，详细请参照 2011 年 9 月 9 日 APP-11-89 调查研究【敏捷要求的 3 个 C（连续）】

夜间打包的想法，在以前项目中也不新奇。如果没有伴随自动化进行打包的话，是无法捕捉代码基盘安定性的全体像的。而且进行一天一次的打包，复数个变更可以同时测试，让打包的缺陷原因的查明变困难。以前项目的测试，因为是在打包工程后执行，所以是在 bug 产生了以后缺陷才被发现。

敏捷的教训：

- 采用自动化测试框架，每次打包执行自动化测试。
- 使用持续集成服务器，迅速打包和发现测试缺陷。持续集成和测试，在通过维持项目全体代码基盘安定性上是有用的，能够大大缩短安定化阶段，大幅减少不确定性。

定期执行回顾

敏捷项目开发要定期回顾，是关于以前遇到的课题和成功的话题。根据这个，让软件开发采用流程和方法实践不断完善变成可能。

回顾以往，都是在项目最后进行事后分析的。但是，这种场合，导致最终项目延迟的原因是什么变成了焦点。因此，与过程和方法的改善相比，会议成了在追究责任所在。

敏捷的教训：

- 贯穿整个项目，进行定期回顾。
- 着眼于流程方法相关的问题，不以追究个人责任为目的。

构成敏捷开发项目的方法，没有全面变革软件开发流程，也可能适用于以前的开发项目。（参照图 2）


简单地做
·设计和编码
反映
·新情报
模型示范
·根据模型设计
合作
·组对工作
融合
·开发和测试
集成
·持续的
回顾
·贯穿项目定期的

图 2 敏捷的 7 个教训

Loadrunner 学习笔记（二）

◆ 作　者：脚踏实地

一、脚本回放

1、初次回放

脚本录制完成之后，点击菜单栏上蓝色三角形按钮，进行脚本回放，观察日志是否报错。

2、设置指标回放

对于一个脚本，如果仅仅执行一次，可能发现不了问题。此时需要我们迭代回放，增加迭代次数，对显示日志进行设置，观察比对结果是否出现问题。

1）设置迭代次数

（1）点击菜单栏上的 Edit Runtime Settings 或者按 F4 即可打开设置界面;

（2）选择 Run Logic,设置 Number of Iterations 即为迭代次数，并且这个迭代次数是作用于 Action，一般一个脚本的 Init 和 End 都只执行一次，而 Action 可以执行多次。

（3）根据需要可以对 Action 进行增加删除，理清整个逻辑。

（4）若设置成随机执行，点击 Properties，选择 Random，但是 Init、Run、End 总和必须为 100%;

（5）当然，在左侧选择 Pacing 可以根据需要设置迭代间隔时间。

2）设置回放日志格式

（1）点击菜单栏上的 Edit Runtime Settings 或者按 F4 即可打开设置界面;

（2）点击 Log，通常采用的是 Always sendmessage，且默认采用的是 Standard Log 即标准的日志，可以选择 Extended Log 中的 Parameter substitution 即参数替换，这样可

以详细地看到参数的替换过程，而不仅仅是最终的结果，方便我们排查错误；

3）设置完毕，点击 ok。如果设置完成没有点击 ok，仅是关闭窗口，则设置未保存；

4）再次回放脚本，观察日志上 Action 执行的次数，以及各个参数替换过程。

3、注意事项

1）对于有唯一性要求的参数，在进行初次回放之时，需要修改参数的值，以免因唯一性导致脚本回放出错；

2）设置指标回放，要根据设置找到相应的反馈，否则也视为回放失败；

3）脚本回放之前要有预期，要与预期的结果进行比对，才能辨别脚本是否回放成功；

二、参数化

如果不做处理，录制的脚本仅是对某一特定的操作有效，不具有代表性。若想模拟真实用户的操作和创建现实的结果，我们需要进行参数化。

1、需要参数化的情况

数据参数化能帮助解决一些问题，也是一些操作的必然要求。主要有两种情况需要参数化:

1）业务系统的要求（如注册、登录等）;

2）操作对性能结果有要求（如查询）。

2、参数化步骤

数据参数化更贴切地模拟真实用户，创建现实的结果。具体参数化的步骤如下:

1） 确定需要参数化的数据;

2） 选择数据，鼠标右键选择 Replace with a parameter;

3） Param List 中设置参数值和参数更新方式。

3、参数设置

1）可直接点击菜单栏上带有<P>的标志，打开 Param List，查看参数化的数据;

2）可以选择参数化的类型形成一定的参数文件，并保存在指定或者特定的位置;

3）在 Param List 中可以点击 Add Column、Add Row 对参数的取值进行编辑，也可以点击 Edit with Notepad 进行文本编辑，既方便又快捷;

4）选择下一行的方式（Select next row）包括顺序取值（Sequential）、随机取值（Random）、唯一取值（Unique），根据不同的需要进行相应的选择;

5）参数变更条件(Update value on)包括每次迭代变更（Each iteration）、每次出现变更（Each occurrence）、仅一次变更（Once），具体选择什么条件要根据实际情况进行选择;

6）当选择下一行的方式（Select next row）为唯一取值（Unique）时，会出现用户参数不够用的时候（when out of values），三种取值方式包括停止虚拟用户（Abort Vuser）、循环取值（Continue in a cyclic manner）、取最后的值(Continue with last value)

7）只有当选择下一行的方式（Select next row）为唯一取值（Unique），参数变更条件为每次迭代变更（Each iteration）的时候，当参数不够用时才能选择自动分配块大小（Automatically allocate block size）。

4、Select next row 与 Update value on 的设置

仅以一个虚拟用户为例

Select next row	Update value on	实际结果
Sequential	Each iteration	依次序进行选择，第一次迭代取第一行值，第二次迭代取第二行值，以此类推
	Each occurrence	在脚本中使用参数的地方，依次进行选择。脚本中出现要使用参数的话，参数值就更新一次，迭代一次值再更新一次。
	Once	用户在所有的迭代中，只用一个值（即参数中的第一行值）
Random	Each iteration	每次迭代随机取值，可能相同可能不相同
	Each occurrence	脚本中出现要使用参数的话，随机取值一次，迭代一次再随机取值一次
	Once	随机取值一次，有几次迭代就取多少次该值
	Each iteration	所有迭代的取值唯一且相同

Unique	Each occurrence	脚本中要使用参数以及迭代几次，就取唯一值几次
	Once	用户在所有的迭代中，只用一个唯一值

5、参数文件遵循的原则:

1）每个参数与参数文件一一对应，即每一个参数只能读取一个参数文件，每一个参数文件可以被多个参数读取;

2）每个参数文件中可存取多列值;

3）每个参数只能读取参数文件中的一列值。

6、注意事项

1）明确参数与变量的区别：参数可以随用随定义，变量必须定义才能使用，且在代码开头；参数需要借助“{}”进行使用，变量则不能使用“{}”，否则将被视为字符串处理；参数为全局的，而变量默认是局部的，也可以定义全局变量。

2）不是所有的问题都需要使用参数化，需要根据实际情况进行正确处理;

3）对于已经进行参数化的脚本，建议选择参数替换日志 Parameter substitution，便于观察参数的替换过程。

三、建立事务

所谓事务(TRANSACTION)，就是在脚本定义中定义的某段操作(ACTION)，更确切的说，就是一段脚本语句。定义事务时，首先在脚本中找到事务的开始和结束位置，然后分别插入一个事务起始标记，这样，当脚本运行的时候，LOADRUNER 会自动在事务的起始点计时，脚本在运行到事务结束点时计时结束，系统会自动记录这段操作的运行时间等性能数据；在脚本运行完毕后，系统会在结果信息中单独反映每个事务运行结果。

1、建立事务的方式:

1）在录制事务前点击第一个时钟图标开始事务（Start Transaction），在结束事务后点击第二个时钟图标，结束事务（End Transaction）;

2）脚本录制完成时，找到事务开始的位置，直接在菜单栏上点击开始事务图标，同样在事务结束位置直接点击结束事务图标，软件会自动生成函数

lr_start_transaction("Transaction Name"); lr_end_transaction("Transaction Name ", LR_AUTO);

3）直接在脚本事务开始和结束的位置插入开始事务和结束事务的函数。分别在事务开始和结束的位置鼠标单击右键，选择 Insert 下的 Start Transaction 和 End Transaction。或者直接选择 Insert 下的 New Step，直接输入 lr_start_transaction 函数和 lr_end_transaction 函数。

2、添加事务的作用:

1）度量响应时间：响应时间=网络延迟时间+Web server Time+ App sever Time+ Database sever Time，而事务时间=函数自身损耗时间+Think Time+Waste Time+响应时间。

可以为检查点的实现起一定作用。

3、注意事项

事物建立方式可以不同，但是事务必须成对出现，保证事务名一致，一个 start，一个 end;

事务状态 Auto 仅对操作逻辑判断，无法对业务进行判断（函数本身有状态返回，必须所有返回成功，最终才未成功）;

事务开始、结束位置存放应根据实际情况进行设置;

事务时间包括 Waste Time，避免其他操作放在事务之中。

四、设置检查点

1、设置检查点的方式

代码一:

```
web_reg_find("Text=Payment Details"，LAST);
```

代码思路:

1）"Payment Details" 为你要检查的文本;

2）脚本执行到此处，若在页面上找到了这几个字符串，那脚本继续执行下去；若没有找到，脚本将在此报错并且结束。

代码二：

```
web_reg_find("Text=Payment Details"， "SaveCount=para_count"， LAST); //check 的函数
if (atoi(lr_eval_string("{para_count}"))>0) //验证是否找到了页面上的要检查的字符串
lr_output_message("we find the string!");
else
lr_output_message("sorry，don't find the string!");
```

代码思路：

1）"Payment Details"为你要检查的文本；

2）脚本执行到此处，不管页面上是否存在你要检查的字符串，脚本都不会报错，而是执行下去。

3）此段代码将找到的你要检查的字符串的个数，存为一个参数。然后在页面代码的后面，通过检查这个参数的值是否大于 0，来判断是否找到了你所要检查的字符串。

代码三：

```
web_reg_find("Text= Payment Details "，"SaveCount= para_count "，LAST);//check 的函数
lr_start_transaction("Transaction Name ");//必须在检查的事务之前
if(atoi(lr_eval_string("{para_count}"))>0)                    //atoi，将将字符转换为整数类型，验证是否找到了页面上的要检查的字符串
lr_end_transaction("Transaction Name "，LR_FAIL);
else
lr_end_transaction("Transaction Name "，LR_PASS);
```

代码思路：

1）"Payment Details"为你要检查的文本；

2）脚本执行到此处，不管页面上是否存在你要检查的字符串，脚本都不会报错，而是执行下去。

3）此段代码将找到的你要检查的字符串的个数，存为一个参数。然后结合事务通过检查这个参数的值是否大于 0，检查事务的最终状态来判断是否找到了你所要检查的字符串。

2、注意事项

1）web_reg_find 写在要查找内容的请求之前，通常情况下写在如下六个函数之前：Web_castom_request()、web_image()、web_link()、web_submit_data()、web_submit_form()、web_url();

2）Payment Details 必须是前端能返回的文本，否则将始终不能检查到，起不到检查点的效果;

3）Payment Details 的设定可以进入 Tree 模式下的 HTTP View 下的 Response Body 里面，对服务器返回的文本进行检查，当然必须对应相应的页面。

五、插入集合点

1、集合点的含义

用于模拟多用户并发操作的一种技术手段，操作可以是相同任务也可以是不同任务，集合点必须放在事务前。

当通过 controller 虚拟多个用户执行该脚本时，用户的启动或运行步骤不一定都是同步的。集合点是在脚本的某处设置一个标记。当有虚拟用户运行到这个标记处时，停下等待，直到所有的用户都达到这个标记处时，再一同进行下面的步骤，这样能够用最大的用户并发去做下面的操作，主要用于对关键步骤的加压。

2、插入集合点的目的

集合点可以设置多个虚拟用户等待到一个点，同时触发一个事务，以达到模拟真实环境下同时多个用户操作，同时模拟负载，实现性能测试的最终目的。由此可见，插入集合点主要是为了衡量在加重负载的情况下服务器的性能情况，从而找到性能瓶颈。可以把集合点理解成是一种特殊情况下的并发。

3、集合点使用步骤

1）确定并发操作步骤;

2）在并发操作之前，插入集合点;

3）方式：在并发操作前右击鼠标，点击 insert 下的 rendezvous，输入名称则出现 lr_rendezvous("集合点名称")即可;

4、集合点策略（Policy）

1）当百分之多少的用户（受控用户）到达集合点时脚本继续;

2）当百分之多少的运行用户到达集合点时脚本继续;

3）指定数量用户到达集合点时脚本继续。

注意事项:

1）如果脚本中设置了集合点，在 control 的场景中 Scenario/Rendezvous 集合功能点会亮（没有则是灰色）。如果脚本之前没有集合点，后面加进去的，已经加载在场景了要进行刷新，获取最新脚本;

2)集合点一般是跟事务结合使用的，不要把集合点设置到事物里面，因为那样的话脚本等待的时间都计算在事务的时间内，进行压力测试的时候查看响应时间就会有一定的偏差。

使用集合点满足的的基本条件：脚本中有添加集合点函数；场景必须位于组模式(Group Mode)，百分比模式（Percentage Mode）下无法使用集合点;

3) 集合点策略中的运行中用户指的是运行中的所有用户，与是否受控没有关系。

4) 对于不同业务，只需要将集合点的名称设置成相同的名称即可实现不同操作的并发;

5) 用户迭代次数多与一次时，在集合点位置都要参与等待。

六、简单函数的使用

Hello Word 的输出

代码一:

```
lr_output_message("str: %s"，lr_eval_string("Hello Word!"));//直接将字符串转换成变量输出
```

代码二:

```
lr_save_string("Hello Word!"，"Parameter");
//将字符串存入一个新的参数中
lr_output_message("str: %s"，lr_eval_string("{Param}"));//将参数转换成变量输出
```

代码三:

```
Char *word;//借助指针进行存储输出
lr_save_string("Hello Word!"，"Param");
```

```
word=lr_eval_string("{Param}");
lr_output_message("str: %s"，word);
```

代码四：

```
Char str[10]; //借助数组进行存储 输出
strcpy(str，"Hello ");//运用复制函数
strcat(str，"World!");//运用链接函数
lr_output_message("str: %s"，str);
```

数据类型转换函数

1）Sprintf：可以将其它数据类型转换为字符串类型

2）Atoi:将字符转换为整数类型

3）Itoa:将整数转换为字符存储到第二个参数中，可以将整数转换为不同的数据类型字符串

4）Atof：将字符转换为浮点类型（使用时应该首先在文件头声明 double atof(const char *string);），否则结果是错误的

5）Atol：将字符转换为长整数类型

6）Strtol：将一个字符串中的第一个数字字符串部分转换成长整数类型

3、数据类型转换函数的使用

1）Sprintf 的使用

（1）格式化字符串的类别：

%s：格式化字符串

%d：格式化数字类型为字符串

%x：将数字格式化成小写十六进制数的字符串

%X：将数字格式化成大写十六进制数的字符串

（2）格式化结果长度：

%3d(3 位)，%5x（5 位）

（3） 格式化结果对齐方式

%-3d、%-4x、%-5X：左对齐，（没有-表示是右对齐）

（4）实例

```
Int a，b;
float c;
char Str[32];
a=234;
b=-234;
c=234.567;
sprintf(Str，"Str=%s"，"example");
lr_output_message(Str);
sprintf(Str，  "integer(234)=%d"，a);
lr_output_message(Str);
sprintf(Str，"float(234.567)=%3.1f"，c);
lr_output_message(Str);
sprintf(Str，" hex(234)=%x"，a);
lr_output_message(Str);
sprintf(Str，" HEX(234)=%X"，a);
lr_output_message(Str);
sprintf(Str，" integer(-234)=%5d%-5X"，a，a);
lr_output_message(Str);
```

运行结果:

```
Action.c(10): Str=example
Action.c(12): integer(234)=234
Action.c(14): float(234.567)=234.6
Action.c(16): hex(234)=ea
Action.c(18): HEX(234)=EA
Action.c(20): integer(-234)=   234EA
```

2）Atoi 的使用

实例:

```
int i;
i=atoi("1234");//将字符串转换为整数
```

```
lr_output_message("atoi('1234')=%d"，i);
```

运行结果:

```
Action.c(10): atoi('12345')=1234
```

3）Itoa 的使用

实例:

```
int i;
i=234;
char ichange[20];
itoa(i，ichange，2);//将整数转换为二进制的字符串
lr_output_message("ichange=%s"，ichange);
itoa(i，ichange，8);//将整数转换为八进制的字符串
lr_output_message("ichange=%s"，ichange);
itoa(i，ichange，10);//将整数转换为十进制的字符串
lr_output_message("ichange=%s"，ichange);
itoa(i，ichange，16);//将整数转换为十六进制的字符串
lr_output_message("ichange=%s"，ichange);
```

运行结果:

```
Action.c(10): ichange=11101010
Action.c(12): ichange=352
Action.c(14): ichange=234
Action.c(16): ichange=ea
```

4）Atof 的使用

实例:

```
double j;
j=atof("123.45");
lr_output_message("atof('123.45')=%.3f"，j);
```

运行结果:

```
Action.c(10): atof('123.45')=123.450
```

5）Atol 的使用

实例：

```
long k;
k=atol("1234567890123456");//将字符串转换为长整型
lr_output_message("atol('1234567890123456')=%d"，k);
```

运行结果：

```
Action.c(10): atol('1234567890123456')=1234567890123456
```

6）Strtol 的使用

实例：

```
char ichange[20];
k=strtol("123456a7890"，ichange);//将字符串转换为长整型， lr_output_message("strtol()=%d"，k);
```

运行结果：

```
Action.c(10): strtol()=123456
```

注：

遇到如果第一个非空格字符不存在或者不是数字也不是正负号则返回零，否则开始做类型转换，之后检测到非数字(包括结束符 \0) 字符时停止转换，返回整型数

3、日期时间相关函数

1）函数参数介绍

Void lr_save_datetime(const char*format，int offset，const char *name);

其中 const char*format 表明时间的格式，int offset 表示时间的偏移，const char*name 表示参数保存的名称。

时间的偏移量包括：DATE_NOW(现在的日期)、TIME_NOW(现在的时间)、ONE_DAY(一天的时间)、ONE_HOUR(一小时的时间)、ONE_MIN(一分钟的时间)等

注：时间的偏移量可以使用公式，例如：

```
DATE_NOW-ONE_DAY(昨天)
DATE_NOW+ONE_DAY(明天)
```

2）实例：

```
lr_save_datetime("%Y-%m-%d %H:%M:%S"，DATE_NOW，"Time");
```

```
lr_output_message("now=%s"，lr_eval_string("{Time }"));
lr_save_datetime("%Y-%m-%d"，DATE_NOW-(ONE_DAY)，"Time");
lr_output_message("昨天=%s"，lr_eval_string("{Time }"));
lr_save_datetime("%H:%M:%S"，TIME_NOW+ONE_HOUR，"Time");
lr_output_message("1 小时后=%s"，lr_eval_string("{Time}"));
```

运行结果:

```
Action.c(10): now=2016-02-18 17:26:57
Action.c(12): 昨天=2016-02-17
Action.c(14): 1 小时后=18:26:57
```

相关文章:《Loadrunner 学习笔记（一）》

自动化测试中的 Python 基础知识点

◆作者：姜林斌

前 言: 自动化测试的脚本语言重 Java Python Ruby 是比较大众化的，如果项目组是使用的 Java 平台那么自动化测试脚本语言的首选当然是 java，这样可以开发团队无缝对接 更利于深入了解项目中的实现细节和架构。若是非 java 平台的产品，强烈建议使用 Python 作为脚本语言，为嘛呢？Life is short,Just use python。

一、数据类型

1.1 字符串

1.1.1.引号(单/双/三引号)

字符串(string)使用单/双引号时的效果一样;

利用三引号(‘’’ ‘’’)，表示多行的字符串，可以在三引号中自由的使用单引号和双引号。

1.1.2 去空格及特殊字符

使用 ‘’.join(str.split(‘#’))

```
7   string_first = 'This is # # Python'
8   # 去除字符串中的#字符
9   convert_string = ''.join(string_first.split('#'))
10  print convert_string
```

1.1.3 查找字符串

(1)使用 str.index(‘what you want to find’) index<0 时表示未找到

```
string_first = 'This is # # Python'
# 查找是否包含字符
index = string_first.index('Python')
print index
```

(2)使用 str.find(‘what you want to check’) index<0 时表示未找到

```
string_first = 'This is # # Python'
# 查找是否包含字符

print string_first.find('Python')
```

1.1.4 比较字符串

使用 cmp(strA,strB)

```
stringA = 'This is Python'
stringB = 'This is Python'
# 比较字符串
print cmp(stringA, stringB)
```

1.1.5 获得字符串长度

使用 len(str)

1.1.6 将字符串中的大小写转换

str.lower() #小写　str.upper() #大写

str.swapcase() #大小写互换 str.capitalize() #首字母大写

1.1.7 反转字符串

使用 str [::-1]

1.1.8 分割字符串

Str.split('分隔符')--只能指定一个分隔符

Re.split()—可以指定多个分隔符

1.1.9 字符串截取

str = '0123456789'

print str[0:3] #截取第一位到第三位的字符

print str[:] #截取字符串的全部字符

print str[6:] #截取第七个字符到结尾

print str[:-3] #截取从头开始到倒数第三个字符之前

print str[2] #截取第三个字符

print str[-1] #截取倒数第一个字符

print str[::-1] #字符串翻转

print str[-3:-1] #截取倒数第三位与倒数第一位之前的字符

print str[-3:] #截取倒数第三位到结尾

1.1.10 字符串替换

使用 str.replace(old,new,count)

1.1.11 字符串编码和解码

Str.encoding('编码类型')　str.decode('解码类型')

常见的字符编码有 utf-8　gbk　gb2312

1.1.12 字符串的转义

转义字符	描述
\(在行尾时)	续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数 yy 代表的字符，例如：\o12 代表换行
\xyy	十进制数 yy 代表的字符，例如：\x0a 代表换行
\other	其它的字符以普通格式输出

1.1.13 字符串的格式化

符号	描述

%c	格式化字符及其 ASCII 码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数（大写）
%f	格式化浮点数字，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	根据值的大小决定使用%f 活%e
%G	作用同%g，根据值的大小决定使用%f 活%e
%p	用十六进制数格式化变量的地址
符号	功能
*	定义宽度或者小数点精度
-	用做左对齐
+	在正数前面显示加号(+)
<sp>	在正数前面显示空格
#	在八进制数前面显示零('0')，在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X')
0	显示的数字前面填充'0'而不是默认的空格
%	'%%'输出一个单一的'%'
(var)	映射变量(字典参数)
m.n.	m 是显示的最小总宽度,n 是小数点后的位数(如果可用的话)

1.2 列表

列表的数据项不需要具有相同的类型，创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。

1.2.1 访问列表中的值

根据索引值来得到列表中的值

List[0]第一个值

List[-1]列表中最后一个值

1.2.2 更新列表指定项

直接对指定项进行赋值

List[3]=hello

1.2.3 删除列表元素

使用 dellist[index]

1.2.4 截取列表

List[0:]列表中所有的值

List[2:4]列表中第 45 两个值

1.2.5 插入列表

List.append(object)在列表末尾加入值

List.insert(index,object)在指定位置插入值

1.2.6 列表的其他操作

Cmp(list1,list2)比较两个列表相同返回 0

Max(lsit)返回列表中最大的值

Min(list)返回列表中最小的值

Len(lsit)返回列表长度

List.count(object)返回值在列表中出现的次数

List.remove(object)移除列表中第一个匹配项

1.3 字典

字典是另一种可变容器模型，且可存储任意类型对象。

字典的每个键值(key=>value)对用冒号(:)分割，每个对之间用逗号(，)分割，整个字典包括在花括号({})中。

1.3.1 访问字典的值

根据键名称来得到键对应的值

```
dict_demo={'Name':'Kevin','Age':'29','Job':'QA'}
printdict_demo['Name']
```

1.3.2 修改字典

修改现有键值直接对指定键进行赋值 dict_demo[‘Age’]=’30’

添加新的键值

```
Dict_demo[‘School’]=’ZhengZhouUniversity’
```

1.3.3 删除字典元素

删除指定键

```
Deldict_demo[‘School’]
```

清除字典

```
Dict.clear()
```

不允许同一个键出现两次。创建时如果同一个键被赋值两次，后一个值会被记住。

键必须不可变，所以可以用数字，字符串或元组充当，所以用列表就不行。

1.3.4 字典内置函数

cmp(dict1，dict2)比较两个字典是否相同

len(dict)计算字典键的总和

dict.has_key()判断键是否在字典中

1.4 日期

日期的格式化:

```
#%a 周几的英文缩写
#%A 周几的全拼
#%b 月份的缩写
#%B 月份的全拼
#%c 以本地时间显示
#%d 显示 1-31 之间的数，每月的第几天
%H 以 24 小时制显示小时
```

%I 以 12 小时制的方式显示当前小时

#%j 显示当前日期为一年中的第几天

%m 显示 1-12 之间的月份

%M 显示 00-59 之间的分钟数

#%p 以 A.M./P.M.方式显示是上午还是下午

%S 显示 0-59 之间的秒数

#%U 显示一年中的第几周

#%w 显示一周中的第几天

#%y 显示(00-99)之间的年份

%Y 显示完整年份

二、变量

变量存储在内存中的值。这就意味着在创建变量时会在内存中开辟一个空间。

基于变量的数据类型，解释器会分配指定内存，并决定什么数据可以被存储在内存中。

因此，变量可以指定不同的数据类型，这些变量可以存储整数，小数或字符。

同时为多个变量赋值。例如:

a=b=c=1

创建一个整型对象，值为 1，三个变量被分配到相同的内存空间上。

也可以为多个对象指定多个变量。例如:

a,b,c=1,2,"john"

两个整型对象 1 和 2 的分配给变量 a 和 b，字符串对象"john"分配给变量 c

Python 中变量的命名规则:

首字母为英文和下划线，其它部分则可以是英文、数字和下划线(即：_)，

而变量名称是区分大小写，即变量 temp 与 Temp 为不同变量。

三、条件和循环

3.1 条件语句

if 语句的判断条件可以用>（大于）、<(小于)、==（等于）、>=（大于等于）、<=（小于等于）来表示其关系。
当判断条件为多个值是，可以使用以下形式：

```
if 判断条件1:
    执行语句1……
elif 判断条件2:
    执行语句2……
elif 判断条件3:
    执行语句3……
else:
    执行语句4……
```

由于 python 并不支持 switch 语句，所以多个条件判断，只能用 elif 来实现，

如果判断需要多个条件需同时判断时，可以使用 or（或），表示两个条件有一个成立时判断条件成功;

使用 and（与）时，表示只有两个条件同时成立的情况下，判断条件才成功。

3.2 循环语句

Python 中的循环语句只有 for 和 while 两种。

while 循环	在给定的判断条件为 true 时执行循环体，否则退出循环体。
for 循环	重复执行语句
break 语句	在语句块执行过程中终止循环，并且跳出整个循环
continue 语句	在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。
pass 语句	pass 是空语句，是为了保持程序结构的完整性。

四、函数及其调用

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。

函数能提高应用的模块性，和代码的重复利用率。

4.1 Python 中定义函数的规则：

函数代码块以 def 关键词开头，后接函数标识符名称和圆括号()。

任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数。

函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。

函数内容以冒号起始，并且缩进。

Return[expression]结束函数，选择性地返回一个值给调用方。不带表达式的 return 相当于返回 None。

4.2 Python 函数中的参数:

所有参数（自变量）在 Python 里都是按引用传递。

如果你在函数里修改了参数，那么在调用这个函数的函数里，原始的参数也被改变了。

当函数的参数不确定时可以使用*args 和**kwargs。

*args 没有 key 值，**kwargs 有 key 值

同时使用*args 和**kwargs 时，*args 参数列必须要在**kwargs 前

4.2.1Python 中的不定长参数*args

```
Def unkonw_parameters(*args):
parameter_list=args
printu'当前参数总数是：',len(parameter_list)
printu'当前函数的参数分别是：',parameter_list
unkonw_parameters('Python','Say:','Hello')
unkonw_parameters('Life','is','short','just','use','python!')
```

输出如下:

当前参数总数是：3

当前函数的参数分别是：('Python','Say:','Hello')

当前参数总数是：6

当前函数的参数分别是：('Life','is','short','just','use','python!')

4.2.1 Python 中的不定长参数kwargs**

```
Def unkonw_parameters(**kwargs):
parameter_dict=kwargs
```

printu'当前参数总数是：',len(parameter_dict)

printu'当前函数的参数分别是：',parameter_dict

```
unkonw_parameters(Name='Python')
unkonw_parameters(Name='Java',Depend='JVM',To='Run',What='AnyWhere')
```

输出如下：

当前参数总数是：1

当前函数的参数分别是：{'Name':'Python'}

当前参数总数是：4

当前函数的参数分别是：{'To':'Run','What':'AnyWhere','Depend':'JVM','Name':'Java'}

4.3 Python 中的匿名函数 lambda

python 使用 lambda 来创建匿名函数。

lambda 只是一个表达式，函数体比 def 简单很多。

lambda 的主体是一个表达式，而不是一个代码块。仅仅能在 lambda 表达式中封装有限的逻辑进去。

lambda 函数拥有自己的名字空间，且不能访问自有参数列表之外或全局名字空间里的参数。

虽然 lambda 函数看起来只能写一行，却不等同于 C 或 C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

lambda 函数的语法只包含一个语句，如下：

```
lambda[arg1[,arg2,.....argn]]:expression
```

如：

```
count=lambdaarg1,arg2:arg1+arg2
print count('Hello','\tWorld')
```

输出结果：

```
Hello  World
```

五：文件操作

5.1 操作文件的函数

打开文件 open()

File object=open(file_name[,access_mode][,buffering])

各个参数的细节如下:

file_name: file_name 变量是一个包含了你要访问的文件名称的字符串值。

access_mode: access_mode 决定了打开文件的模式: 只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。

buffering: 如果 buffering 的值被设为 0，就不会有寄存。如果 buffering 的值取 1，访问文件时会寄存行。如果将 buffering 的值设为大于 1 的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。

content=open('D:\python_demo.txt','r+')

不同模式打开文件的完全列表：

模式	描述
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。
rb	以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。
r+	打开一个文件用于读写。文件指针将会放在文件的开头。
rb+	以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。
w	打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
wb	以二进制格式打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
w+	打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
wb+	以二进制格式打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
a	打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
ab	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
a+	打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。
ab+	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

file.close()

关闭文件。关闭后文件不能再进行读写操作。

file.flush()

刷新文件内部缓冲，直接把内部缓冲区的数据立刻写入文件，而不是被动的等待输出缓冲区写入。

file.fileno()

返回一个整型的文件描述符(filedescriptorFD 整型)，可以用在如 os 模块的 read 方法等一些底层操作上。

file.isatty()

如果文件连接到一个终端设备返回 True，否则返回 False。

file.next()

返回文件下一行。

file.read([size])

从文件读取指定的字节数，如果未给定或为负则读取所有。

file.readline([size])

读取整行，包括"\n"字符。

file.readlines([sizehint])

读取所有行并返回列表，若给定 sizeint>0，返回总和大约为 sizeint 字节的行,实际读取值可能比 sizhint 较大,因为需要填充缓冲区。

file.seek(offset[,whence])

设置文件当前位置

file.tell()

返回文件当前位置。

file.truncate([size])

截取文件，截取的字节通过 size 指定，默认为当前文件位置。

file.write(str)

将字符串写入文件，没有返回值。

file.writelines(sequence)

向文件写入一个序列字符串列表，如果需要换行则要自己加入每行的换行符。

重命名文件 rename()

os.rename(current_file_name,new_file_name)

删除文件 remove()

Os.remove(filename)

5.2 操作目录的函数

创建目录 os.mkdir()方法

可以使用 os 模块的 mkdir()方法在当前目录下创建新的目录们。你需要提供一个包含了要创建的目录名称的参数。

改变当前工作路径 os.chdir()方法

可以用 chdir()方法来改变当前的目录。chdir()方法需要的一个参数是你想设成当前目录的目录名称。

显示当前工作路径 os.getcwd()方法:

getcwd()方法显示当前的工作目录。

删除目录 os.rmdir()

在删除这个目录之前，它的所有内容应该先被清除

六：数据库操作

6.1：MySQL 数据库操作

```
Import MySQLdb
#Windows 下安装 MySQLdb 时需要选对安装包若不然坑大大滴
conn_str=MySQLdb.connect(host='localhost',user='root',passwd='',db='mysql',port=3306)
session=conn_str.cursor()
session.execute("showstatuslike'Table_locks_immediate'")
data=session.fetchone()
print data
session.close()
```

6.2：SqlServer 数据操作

```
Import pyodbc
defget_select_result_in_sql_server(sql_select):
```

```
    conn=pyodbc.connect(DRIVER='{SQLServer}',SERVER='FARIDAH-
PC',DATABASE='GeAutoTestDB',UID='sa',PWD='sa')
    cursor=conn.execute(sql_select)
    row=cursor.fetchone()
    conn.commit()
    select_count=row[0]
    conn.close()
    return select_count
```

6.3: Sqlite 数据库操作

```
    Import sqlite3
    #update table set something with sqlite
    defsqlite_update(sql_update):
    db=sqlite3.connect("E:\Environment For
Test\Training\Dicom\UID20160223092840024 7\Database\somostorage.db")
    cousor=db.cursor()
    cousor.execute(str(sql_update))
    db.commit()
    db.close()
```

6.4: Oracle 数据库操作

使用 cx_Oracle 库

```
    Import cx_Oracle
    conn=cx_Oracle.connect('kevin/123456@192.168.10.222/Orcl')
    cursor=conn.cursor()
    sql_str="select'this is cx_oracle for python'from dual"
    cursor.execute(sql_str)
    rows=cursor.fetchone()
    print rows
    conn.commit()
    cursor.close()
    conn.close()
```

七：Http 和 Socket 常用的库

7.1: Http 协议

推荐使用 Requests 库（可用作接口自动化测试）

```
import requests
# get请求
get = requests.get("http://ws.webxml.com.cn/WebServices/WeatherWS.asmx/getRegionCountry")

print u'Http报文头', get.headers
print u'Http状态码:', get.status_code
print u'服务器返回的Cookie:', get.cookies
print u'服务器返回的内容:', get.text

# post请求
json_data = {'theCityCode':'2057', 'theUserID':''}
post = requests.post("http://ws.webxml.com.cn/WebServices/WeatherWS.asmx/getWeather",data=json_data)

print u'Http报文头:', post.headers
print u'Http状态码:', post.status_code
print u'服务器返回的内容:', post.text
```

7.2: Socket 协议

服务器端:

```
#服务器端
HOST='127.0.0.1'
PORT=10022
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((HOST,PORT))
s.listen(10)
conn,addr=s.accept()
whileTrue:
data=conn.recv(1024)
if not data:
break
conn.sendall(data)
```

客户端:

```
HOST='localhost'
PORT=10022
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((HOST,PORT))
```

```
s.sendall(b'尝试发送 socket 报文 1')
data=s.recv(1024)
print('Received',repr(data))
s.close()
```

八、操作常用文件

8.1 操作 Excel

使用 **openpyxl** 库

```
From openpyxl.reader.excel import load_workbook
#parent directory of current working directory
parentdir=os.path.split(os.getcwd())[0]
#common directory
autocase=str(os.path.split(parentdir)[0])+'\\AutoCase\\AutoTestCases.xlsx'

#在 Excel 中记录结果
#注意使用了相对路径保持整个项目的完整性
Def WriteExcel(result,locator,sheetname):
book=load_workbook(autocase)
sheet=book.get_sheet_by_name(sheetname)
if result==True:
sheet.cell(locator).value='Pass'
elif result==False:
sheet.cell(locator).value='Fail'
book.save(autocase)
```

注意：excel 的单元格命名是从 1 开始的~

8.2 操作 xml

Python 自带的 xml 库

以下例子为个人写的读取接口用例(xml 格式)执行测试并保持测试结果到 excel 文件中的脚步。

使用示例：

```
#-*-coding=utf-8-*-
```

```
#CreateBy:晴空-姜林斌
Import os
Import sys
import requests
from xml.dom.minidom import parse
from openpyxl.reader.excel import load_workbook

#Window 下必须加载 Commons 库的目录否则在 Dos 窗口中执行时会找不到库文件
parent_dir=os.path.split(os.getcwd())[0]
comm_dir=parent_dir+str("\BaseFunction")
sys.path.append(comm_dir)
printparent_dir
AUTO_RESULT=u"E:\PyScripts\MdServices\TestCases\WebService Cases.xlsx"

#自动化测试用例 Excel 文件
case_file=open(u'E:\PyScripts\MdServices\TestCases\casedata.xml','rb')

#加载 XML 文件
tree=parse(file=case_file)
collection=tree.document Element

#读取所有接口方法信息
operation_list=collection.get Elements By Tag Name('Operation')

#读取保存测试结果的文件
book=load_workbook(AUTO_RESULT)
#获取 Excel 文件中的表单 sheet
sheet_names=book.get_sheet_names()
#指定工作表单--根据 sheet 名查找
working_sheet=book.get_sheet_by_name(sheet_names[0])
#Excel 中测试用例开始的索引
start_index=2
```

```
#循环每个 webservice 方法
For operation in operation_list:
#定义 operation 的 UR 地址
uri=operation.get Attribute('url')
#读取接口中方法的调用方式 getposthead 等等
action_type=operation.get Attribute('action')
#接口方法名
function_name=operation.get Attribute('name')
#读取所有测试用例
case_list=operation.get Elements By TagName("case")
#循环执行所有测试用例
For case in case_list:
#得到测试用例中所有参数的名称和值
parameter_list=case.get Elements By Tag Name("Parameter")
length=len(parameter_list)-1
json_data={}
forindexin(0,length):
#得到参数名称
name=parameter_list[index].getAttribute('name')
#得到参数值
value=parameter_list[index].getAttribute('value')
#将每个参数和值存为字典形式即 JSON 格式
json_data[name]=value
#接口调用方式为 post
ifaction_type=='post':
#接口中方法的调用方式为 post
response=requests.post(uri,data=json_data)

#测试用例编号
case_id_locator='A'+str(start_index)
#保存 caseid
working_sheet.cell(case_id_locator).value=start_index
#保存 Excel 文件的更改信息
```

```
book.save(AUTO_RESULT)

#service 接口 url 地址
service_url_locator='B'+str(start_index)
#保存 caseid
working_sheet.cell(service_url_locator).value=uri
#保存 Excel 文件的更改信息
book.save(AUTO_RESULT)

#接口方法名称
function_name_locator='C'+str(start_index)
#保存 caseid
working_sheet.cell(function_name_locator).value=function_name
#保存 Excel 文件的更改信息
book.save(AUTO_RESULT)

#实际返回结果
actual_result_locator='G'+str(start_index)
#保存 caseid
working_sheet.cell(actual_result_locator).value=response.content
#保存 Excel 文件的更改信息
book.save(AUTO_RESULT)

#预期返回结果
expect_result_locator='F'+str(start_index)
expected_content=working_sheet.cell(expect_result_locator).value
#runresult 测试结果
run_result_locator='H'+str(start_index)

#对比预期结果和实际结果
If expected_content==actual_result_locator:
working_sheet.cell(run_result_locator).value='Pass'
else:
```

```
working_sheet.cell(run_result_locator).value='Fail'
#保存 Excel 文件的更改信息
book.save(AUTO_RESULT)

#接口调用方式为 get
elifaction_type=='get':
response=requests.get(uri,data=json_data)
print response.text
#Excel 中循环
start_index+=1
print'startindex',start_index
```

8.3: 操作 json

可以使用自带的 json 库，也可以使用 demjson 库。

```
#-*-coding=utf-8-*-
#CreatedBy:晴空-姜林斌
Importdemjson
Importjson
dataA=[{'a':1,'b':2,'c':3,'d':4,'e':5}]
print demjson.encode(dataA)
dataB='{"a":1,"b":2,"c":3,"d":4,"e":5}'
print demjson.decode(dataB)
```

九、相见恨晚的三方库

9.1 Requests 库

Get/post/head/options/delete

1) get 请求(url 无参数，无定制 header)

```
get=requests.get("http://www.baidu.com")
printget.status_code
printget._content
```

2) get 请求(url 有参数，定制 header)

如果你想为请求添加 HTTP 头部，只要简单地传递一个 dict 给 headers 参数就可以

了。

```
head={'apikey':'5218cc6fa3ea3966a3590437996579dd'}
parameters={'id':'412724198807184917'}
get=requests.get("http://apis.baidu.com/apistore/idservice/id",params=parameters,headers=head)
printget.status_code
#将 unicode 进行转义
printstr(get.content).decode('unicode_escape')
```

在 head 中定制指定 content-type:

```
url='https://api.github.com/some/endpoint'
payload={'some':'data'}
headers={'content-type':'application/json'}
r=requests.post(url,data=json.dumps(payload),headers=headers)
```

3) 定义响应报文的编码格式

```
head={'apikey':'5218cc6fa3ea3966a3590437996579dd'}
parameters={'id':'412724198807184917'}
get=requests.get("http://apis.baidu.com/apistore/idservice/id",params=parameters,headers=head)
#报文返回的内容默认格式是 GBK
printget.encoding
get.encoding='utf-8'
#改变编码格式后
printget.encoding
```

4) 二进制返回的内容

以请求返回的二进制数据创建一张图片，你可以使用如下代码:

```
From PIL import Image
From StringIO import StringIO
i=Image.open(StringIO(r.content))
```

5) json 响应内容

```
import requests
r=requests.get('https://github.com/timeline.json')
r.json()
```

6) 发送 cookies 到服务器

url='http://httpbin.org/cookies'

cookies=dict(cookies_are='working')

r=requests.get(url,cookies=cookies)

7) session 对象

```
s = requests.Session()

s.get('http://httpbin.org/cookies/set/sessioncookie/123456789')
r = s.get("http://httpbin.org/cookies")

print(r.text)
```

8) SSL 安全证书

SSL证书验证

Requests可以为HTTPS请求验证SSL证书，就像web浏览器一样。要想检查某个主机的SSL证书，你可以使用 verify 参数:

```
>>> requests.get('https://kennethreitz.com', verify=True)
requests.exceptions.SSLError: hostname 'kennethreitz.com' doesn't match either
```

在该域名上我没有设置SSL，所以失败了。但Github设置了SSL:

```
>>> requests.get('https://github.com', verify=True)
<Response [200]>
```

对于私有证书，你也可以传递一个CA_BUNDLE文件的路径给 verify 。你也可以设置 REQUEST_CA_BUNDLE 环境变量。

如果你将 verify 设置为False，Requests也能忽略对SSL证书的验证。

```
>>> requests.get('https://kennethreitz.com', verify=False)
<Response [200]>
```

默认情况下， verify 是设置为True的。选项 verify 仅应用于主机证书。

你也可以指定一个本地证书用作客户端证书，可以是单个文件（包含密钥和证书）或一个包含两个文件路径的元组:

```
>>> requests.get('https://kennethreitz.com', cert=('/path/server.crt', '/path/
<Response [200]>
```

如果你指定了一个错误路径或一个无效的证书:

```
>>> requests.get('https://kennethreitz.com', cert='/wrong_path/server.pem')
SSLError: [Errno 336265225] _ssl.c:347: error:140B0009:SSL routines:SSL_CTX_us
```

9.2 openpyxl 库

From openpyxl import Workbook

From openpyxl import load_workbook

From openpyxl.writer.excel import ExcelWriter

1) 打开已经存在的 excel 文件

work_book=load_workbook(filename='D:\PyScripts\TestData.xlsx')

2) 定位到指定 sheet 表单

Work_sheet=work_book.get_sheet_by_name('sheet 名称')

3) 给单元格赋值

```
Work_sheet.cell('A3').value='HelloPython'
```

或者

```
Work_sheet.cell(row=i,column=j)='Thisisassigned'
```

4) 新建 excel 文件

```
Wb=Workbook()
wb.save(filename='test.xlsx')
```

5) 新建 sheet 表单

```
#insert at the end(default)
ws1=wb.create_sheet()
#or
#insert at first position
ws2=wb.create_sheet(0)
```

6) 给 sheet 表单命名

```
ws.title="NewTitle"
```

9.3: nose 库

- Nosetests 常用参数

nosetests–v: debug 模式，看到具体执行情况，推荐大家执行时用这个选项

nose 会捕获标准输出，调试的 print 代码默认不会打印。nosetest–s 可打开 output 输出，否则全部通过时不打印 stdout。

默认 nosetests 会执行所有的 case，若想单独只执行一个 case，执行 nosetest--tests 后跟要测试的文件(nosetests 后面直接跟文件名，其实也可以直接运行该 case)。

nosetest--pdb-failures: 失败时，立马调试。这选项很赞，可看到失败时的即时环境。

nosetests--collect-only-v: 不运行程序，只是搜集并输出各个 case 的名称

nosetests-x: 一旦 case 失败立即停止，不执行后续 case

nosetests–failed: 只执行上一轮失败的 case

- 该框架可用的函数

以下函数都可以直接使用，无需导入头文件等

Safe 封装 Assert 语句

AssertEqual

AssertEqual(s1,s2,msg=''):

@note: 判断 s1 和 s2 是否相等，两者类型必须一致

[例子]: \r\n

AssertEqual("鲜花","鲜花","检查前面两个字符串是否相等");\r\n

@param: s1,s2 待比较的对象，对象类型不定，但是两个对象必须相同类型

@rtype: int

@rvalue: 相等返回 0，不相等抛异常

AssertNotEqual

AssertNotEqual(s1,s2,msg=''):

@note: 判断 s1 和 s2 是否不相等，两者类型必须一致

[例子]: \r\n

AssertNotEqual("鲜花","鲜花 a","如果前面两个字符串相等，则输出该信息");\r\n

@param: s1,s2 待比较的对象，对象类型不定，但是两个对象必须相同类型

@rtype: int

@rvalue: 相等返回 0，不相等返回抛异常

AssertTrue

AssertTrue(s,msg=''):

@note: 断言 s 参数为 True

[例子]: \r\n

AssertTrue(1,"如果前面一个参数为假，则输出该信息");\r\n

@param: s,待判断是否为 True 的对象

@rtype: int

@rvalue: s 对象是真则返回 0，s 对象是假，则抛出异常

AssertFalse

AssertFalse(s,msg=''):

'''

@note: 断言 s 参数为 False

[例子]: \r\n

AssertFalse(0,"如果前面参数的值为真，则输出该信息");\r\n

@param: s,带判断是否为 False 的对象

@rtype: int

@rvalue: s 对象为假则返回 0，s 对象为真则抛出异常

AssertIn

AssertIn(s,l,msg=''):

@note: 判断 s 是否属于 l

[例子]: \r\n

AssertIn(1,[2,3,1,5],

"如果第一个参数不属于第二个参数对应的集合，则输出该信息");\r\n

@param: s 表示元素，l 表示集合,msg 表示出错是打印的消息

@rvalue: int

@rtype: 如果 l 中包含 s，返回 0，如果 l 中不包含 s，则抛出异常

AssertNotIn

AssertNotIn(s，l，msg=''):

@note: 判断 s 是否不属于 l

[例子]: \r\n

AssertNotIn(2，[3，4，5，6]，

"如果第一个参数属于第二个参数对应的集合，则输出该参数");\r\n

@param: s 表示元素，l 表示集合，msg 表示出错是打印的消息

@rvalue: int

@rtype: 如果 l 中不包含 s，返回 0，如果 l 中包含 s，则抛出异常

AssertInclude

AssertInclude(s，t，msg=''):

@note: 判断 s 是否包含 t，

[例子]: \r\n

AssertInclude([1，2]，[1，2，3，4]，

"如果第二个参数对应的列表不包含第一个参数对应的列表，则输出该信息");\r\n

@param: s 和 t 都是列表类型

@rvalue: int

@rtype: 如果 s 中包含 t，返回 0，如果 s 中不包含 t，则抛出异常

AssertExclude

AssertExclude

@note: 判断 s 是否不包含 t

[例子]: \r\n

AssertExclude([1，2]，[3，4，6]，

"如果第二个参数对应的列表包含第一个参数对应的列表，则输出该信息")

@param: s 和 t 都是列表类型

@rvalue: int

@rtype: 如果 s 中不包含 t，返回 0，如果 s 中包含 t，则抛出异常

'''

AssertGreater

AssertGreater(s1，s2，msg=''):

@note：判断 s1 是否大于 s2，两者类型必须一致

[例子]：\r\n

AssertEqual(3，2，"检查第一个对象是否大于第二个对象");\r\n

@param：s1，s2 待比较的对象，对象类型不定，但是两个对象必须相同类型

@rtype：int

@rvalue：相等返回 0，不相等抛异常

Nose 自带 assert 语句

nose.tools

assert_almost_equal(first，second，places=7，msg=None)

assert_almost_equals(first，second，places=7，msg=None)

assert_equal(first，second，msg=None)

assert_equals(first，second，msg=None)

assert_false(expr，msg=None)

assert_not_almost_equal(first，second，places=7，msg=None)

assert_not_almost_equals(first，second，places=7，msg=None)

assert_not_equal(first，second，msg=None)

assert_not_equals(first，second，msg=None)

assert_true(expr，msg=None)

eq_(a，b，msg=None)

ok_(expr，msg=None)

Python 脚本测试注意:

Python 脚本的测试框架生成目前有自己书写，python 代码写 case 需要注意以下几点

1、python 测试文件（test**.py）及函数的名字(test**)需要以 testorTest 开头

Nose runs functional tests in the order in which they appear in the module file.

2、python 测试文件(test**.py)需要将模块的 output 文件夹导入到系统变量中才能够使用 output 下的 python 文件

如：sys.path.append("../../../output")

setUp 和 tearDown

框架提供 setUp 和 tearDown 函数，但与 btest 的同名 setUp 和 tearDown 不同，nose 的 setUp 和 tearDown 函数分别在所有 case 执行前和执行后触发，而不是在每一个 case 前后执行。因此，nose 的 setUp 中可以做诸如拷贝或移动全部 case 必需文件，打印 case 入口提示信息等工作，不能做仅与某个 case 单独相关的工作。

9.4：simplejson 库

1) simplejson.dump(json_data，file)

使用 dump 将 python 数组对象保存在一个包含 JSON 格式的文件中

```
Import simplejson
File=open(‘file_path’，’w+’)
Json_data={‘A’:’dataA’，’B’:’dataB’}
Simplejson.dump(json_data，file)
File.close()
```

2) simplejson.dumps(json_data)

使用 dumps 将 python 字典对象转换为一个包含 JSON 格式的字符串

```
Json_data={‘C’:’dataC’，‘D’:’dataD’}
Str_json=simple json.dumps(json_data)
```

3) simple.load(json_object)

使用 load 读取一个包含 JSON 数组格式的文件后，得到一个 python 对象

```
File=open(‘file_path’，‘r’)
Json_obj=simple.load(file)
```

1) simple.loads(json_str)

使用 loads 读取一个包含 JSON 字典格式的字符串后，得到一个 python 对象

```
Str_obj=‘’’{‘E’:’datae，‘F’:’dataf’}’’’
Json_obj=simple.loads()
```

9.5: psutil 库

根据进程名杀掉进程

```
Import psutil
Import os
Import datetime
current_year=datetime.datetime.today().strftime('%Y')
today=datetime.datetime.today().strftime("%Y-%m-%d")
tomorrow_str=str(datetime.datetime.now()+datetime.timedelta(days=2))
tomorrow_time=tomorrow_str[:10]
#输出进程 pid
Def kill_process_by_name(process_name):
process_list=psutil.process_iter()
for process in process_list:
if process_nameinstr(process):
print process.pid，process
os.kill(process.pid，9)
print u'当前日期'，today，u'\t 已 kill'，process_name，u'所有进程'
kill_process_by_name("chromedriver.exe")
def monitor_win():
#监控 CPU 使用
cpu_useage_info=str(str(psutil.cpu_times()).split('=')).split('，')
user_cpu=float(str(str(cpu_useage_info[1])).split('.')[0])[2:])
sys_cpu=float(str(str(cpu_useage_info[3])).split('.')[0])[2:])
idle_cpu=float(str(str(cpu_useage_info[5])).split('.')[0])[2:])
cpu_use_count=user_cpu+sys_cpu+idle_cpu
user_usage=str((user_cpu/cpu_use_count)*100)[0:5]
sys_usage=str((sys_cpu/cpu_use_count)*100)[0:5]
idle_usage=str((idle_cpu/cpu_use_count)*100)[0:5]
print u"User 态 Cpu:%s"%user_usage，'%'
print u"Sys 态 Cpu:%s"%sys_usage，'%'
print u"Idle 态 Cpu:%s"%idle_usage，'%'
#监控内存使用率
```

```
memory_usage=str(str(psutil.virtual_memory()).split(', ')[2]).split('=')[1]
print u'内存使用率:%s'%memory_usage
swap_mem_usage=str(str(psutil.swap_memory()).split(', ')[3]).split('=')[1]
print u'内存 swap 分区使用率:%s'%swap_mem_usage
```

9.6: xml 库(beautifulsoup 替代之)

```
#从 xml.dom.ninidom 中导入 parse
From xml.dom.minidom import parse
#指定要操作的文件
xml_file=open("D:\PyScripts\RobotWithPython\casedata.xml", 'rb')
#转义要操作的文件
tree=parse(file=xml_file)
#得到 DOM 对象的文档元素
dom=tree.documentElement
#查找 tagname 是 case 的所有元素
nodes=dom.getElementsByTagName('case')
for node in nodes:
#获得节点的名称
Print node.nodeName
#获得节点的值只对文本节点有效
Print node.nodeValue
#获得节点的
print node.firstChild.data
#获得子标签
Node[0].getElementByTagName()
#获得标签属性
item.getAttribute("passwd")
#获得标签对之间的值
node.firstChild.data
```

完整实例:

```
From xml.etree.ElementTree import ElementTree, Element
```

defread_xml(in_path):

```
    '''读取并解析 xml 文件 in_path:xml 路径 return:ElementTree'''
    tree=ElementTree()
    tree.parse(in_path)
    return tree
    defwrite_xml(tree，out_path):
    '''将 xml 文件写出 tree:xml 树 out_path:写出路径'''
    tree.write(out_path，encoding="utf-8"，xml_declaration=True)
    defif_match(node，kv_map):
    '''判断某个节点是否包含所有传入参数属性 node:节点 kv_map:属性及属性值组成的 map'''
    forkeyinkv_map:
    if node.get(key)!=kv_map.get(key):
    return False
    else
    return True
    #---------------search-----
    deffind_nodes(tree，path):
    '''查找某个路径匹配的所有节点 tree:xml 树 path:节点路径'''
    Return tree.findall(path)
    defget_node_by_keyvalue(nodelist，kv_map):
    '''根据属性及属性值定位符合的节点，返回节点 nodelist:节点列表 kv_map:匹配属性及属性值
map'''
    result_nodes=[]
    for node in nodelist:
    if if_match(node，kv_map):
    result_nodes.append(node)
    return result_nodes

    #---------------change-----
    defchange_node_properties(nodelist，kv_map，is_delete=False):
    '''修改/增加/删除节点的属性及属性值 nodelist:节点列表 kv_map:属性及属性值 map'''
    For node in nodelist:
```

```
For key in kv_map:
If is_delete:
If key in node.attrib:
Del node.attrib[key]
else:
node.set(key，kv_map.get(key))
```

def change_node_text(nodelist，text，is_add=False，is_delete=False):

```
'''改变/增加/删除一个节点的文本 nodelist:节点列表 text:更新后的文本'''
For node in nodelist:
If is_add:
node.text+=text
elifis_delete:
node.text=""
else:
node.text=text
```

def create_node(tag，property_map，content):

```
'''新造一个节点 tag:节点标签 property_map:属性及属性值 mapcontent:节点闭合标签里的文本内容
return 新节点'''
element=Element(tag，property_map)
element.text=content
return element
```

def add_child_node(nodelist，element):

```
'''给一个节点添加子节点 nodelist:节点列表 element:子节点'''
For node in nodelist:
node.append(element)
```

defdel_node_by_tagkeyvalue(nodelist，tag，kv_map):

```
'''同过属性及属性值定位一个节点，并删除之 nodelist:父节点列表 tag:子节点标签 kv_map:属性及
属性值列表'''
For parent_node in nodelist:
children=parent_node.getchildren()
for child in children:
if child.tag==tagandif_match(child，kv_map):
```

```
parent_node.remove(child)
if__name__=="__main__":
#1.读取 xml 文件
tree=read_xml("./test.xml")
#2.属性修改
#A.找到父节点
nodes=find_nodes(tree，"processers/processer")
#B.通过属性准确定位子节点
result_nodes=get_node_by_keyvalue(nodes，{"name":"BProcesser"})
#C.修改节点属性
change_node_properties(result_nodes，{"age":"1"})
#D.删除节点属性
change_node_properties(result_nodes，{"value":""}，True)
#3.节点修改
#A.新建节点
a=create_node("person"，{"age":"15"，"money":"200000"}，"thisisthefirestcontent")
#B.插入到父节点之下
add_child_node(result_nodes，a)
#4.删除节点
#定位父节点
del_parent_nodes=find_nodes(tree，"processers/services/service")
#准确定位子节点并删除之
target_del_node=del_node_by_tagkeyvalue(del_parent_nodes，"chain"，{"sequency":"chain1"})
#5.修改节点文本
#定位节点
text_nodes=get_node_by_keyvalue(find_nodes(tree，"processers/services/service/chain"),
{"sequency":"chain3"})
change_node_text(text_nodes，"newtext")
#6.输出到结果文件
write_xml(tree，"./out.xml")
```

9.7 Selenium 库

Selenium2，又名 WebDriver，它的主要新功能是集成了 Selenium1.0 以及 WebDriver

（WebDriver 曾经是 Selenium 的竞争对手）。也就是说 Selenium2 是 Selenium 和 WebDriver 两个项目的合并，即 Selenium2 兼容 Selenium，它既支持 SeleniumAPI 也支持 WebDriverAPI。

```
From selenium import webdriver
From selenium.webdriver.common.action_chain simport Action Chains
From selenium.webdriver.support.uiimport WebDriver Wait
From selenium.common.exception simport No Such Element Exception
driver=webdriver.Chrome()
```

对 UI 自动化测试的同学这个是必须的，这里就不做介绍了。

9.8 pywin32 库

WindowsPywin32 允许你像 VC 一样的形式来使用 PYTHON 开发 win32 应用。代码风格可以类似 win32sdk，也可以类似 MFC，由你选择。如果你仍不放弃 vc 一样的代码过程在 python 下，那么这就是一个不错的选择。

在 UI 自动化测试中，有时候需要模拟鼠标实际滑动和滚轮滑动，pywin32 此时就会大显神威。

```
From selenium.webdriver.common.action_chains import ActionChains
From selenium.webdriver.support.uiimport WebDriverWait
Import time
Import win32api
Import win32con
#拖动 canvas 使用 PyWin32 库辅助实现
#只移动第一个 canvas 中的图片
defmove_view(layout_name):
WebDriverWait(driver10).until(lambda
the_driver:the_driver.find_element_by_id('canvasDivContainer0').is_displayed())
If layout_name=='CC':
switch_view_to_cc_layout(0)
#移动鼠标到指定坐标点
win32api.SetCursorPos([300，300])
#按下鼠标右键
win32 api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN，0，0，0，0)
```

```
time.sleep(1)
#向右下方移动鼠标
win32api.mouse_event(win32con.MOUSEEVENTF_MOVE，10，10，0，0)
time.sleep(1)
#在鼠标当前位置松开右键
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP，0，0，0，0)
time.sleep(2)
driver.get_screenshot_as_file("D:\Move_image_On_CC_Layout.png")
elif layout_name=='TC':
switch_view_to_tc_layout(0)
#移动鼠标到指定坐标点
win32api.SetCursorPos([200，200])
#按下鼠标右键
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN，0，0，0，0)
time.sleep(1)
#向右下方移动鼠标
win32api.mouse_event(win32con.MOUSEEVENTF_MOVE，10，10，0，0)
time.sleep(1)
#在鼠标当前位置松开右键
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP，0，0，0，0)
time.sleep(2)
driver.get_screenshot_as_file("D:\Move_image_On_TC_Layout.png")
elif layout_name=='MT':
switch_view_to_mt_layout(0)
#移动鼠标到指定坐标点
win32api.SetCursorPos([130，130])
#按下鼠标右键
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN，0，0，0，0)
time.sleep(1)
#向右下方移动鼠标
win32api.mouse_event(win32con.MOUSEEVENTF_MOVE，10，10，0，0)
time.sleep(1)
#在鼠标当前位置松开右键
```

```
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP，0，0，0，0)
time.sleep(2)
driver.get_screenshot_as_file("D:\Move_image_On_MT_Layout.png")

#trainee check note info
Def trainee_check_note_info():
try:
if driver.find_element_by_class_name(note_save_class)isNone:
print'note info for trainee default foldering'
else:
print'trainee complete to check note information'
except:
SomoCS.write_exception_info()

#关闭浏览器非预期提示
defclose_tips_of_dialog():
win32api.SetCursorPos([300，165])
#按下鼠标右键
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN，0，0，0，0)
time.sleep(1)
#向右下方移动鼠标
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP，10，10，0，0)

#模拟 WindowLevel 操作 WL 值分别改变 10
#只在第一个 canvas 上操作
defchange_window_level(layout_name):
WebDriverWait(driver,10).until(lambda
the_driver:the_driver.find_element_by_id('canvasDivContainer0').is_displayed())
iflayout_name=='CC':
switch_view_to_cc_layout(0)
#移动鼠标到指定坐标点
win32api.SetCursorPos([300，300])
#按下鼠标右键
```

```
win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTDOWN，0，0，0，0)
time.sleep(1)
#向右下方移动鼠标
win32api.mouse_event(win32con.MOUSEEVENTF_MOVE，10，10，0，0)
time.sleep(1)
#在鼠标当前位置松开右键
win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTUP，0，0，0，0)
time.sleep(2)
driver.get_screenshot_as_file("D:\ChangeWindowandLevel_On_CC_Layout.png")
elif layout_name=='TC':
switch_view_to_tc_layout(0)
#移动鼠标到指定坐标点
win32api.SetCursorPos([200，200])
#按下鼠标右键
win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTDOWN，0，0，0，0)
time.sleep(1)
#向右下方移动鼠标
win32api.mouse_event(win32con.MOUSEEVENTF_MOVE，10，10，0，0)
time.sleep(1)
#在鼠标当前位置松开右键
win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTUP，0，0，0，0)
time.sleep(2)
driver.get_screenshot_as_file("D:\ChangeWindowandLevel_On_TC_Layout.png")
elif layout_name=='MT':
switch_view_to_mt_layout(0)
#移动鼠标到指定坐标点
win32api.SetCursorPos([130，130])
#按下鼠标右键
win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTDOWN，0，0，0，0)
time.sleep(1)
#向右下方移动鼠标
win32api.mouse_event(win32con.MOUSEEVENTF_MOVE，10，10，0，0)
time.sleep(1)
```

```
        #在鼠标当前位置松开右键
        win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTUP，0，0，0，0)
        time.sleep(2)
        driver.get_screenshot_as_file("D:\ChangeWindowandLevel_On_MT_Layout.png")
        #模拟图片的放大缩小
        Def larger_the_view(layout_name):
        if layout_name=='CC':
        switch_view_to_cc_layout(0)
        WebDriverWait(driver,10).until(lambda
the_driver:the_driver.find_element_by_id('canvasDivContainer0').is_displayed())
        #移动鼠标到指定位置
        win32api.SetCursorPos([300，300])
        #按下鼠标中间键
        win32api.mouse_event(win32con.MOUSEEVENTF_MIDDLEDOWN，0，0，0，0)
        time.sleep(1)
        #移动鼠标
        win32api.mouse_event(win32con.MOUSEEVENTF_MOVE，10，10，0，0)
        time.sleep(1)
        #松开鼠标中间键
        win32api.mouse_event(win32con.MOUSEEVENTF_MIDDLEUP，0，0，0，0)
        time.sleep(2)
        driver.get_screenshot_as_file("D:\LargerView_On_CC_Layout.png")
        elif layout_name=='TC':
        switch_view_to_tc_layout(0)
        WebDriverWait(driver,10).until(lambda
the_driver:the_driver.find_element_by_id('canvasDivContainer0').is_displayed())
        #移动鼠标到指定位置
        win32api.SetCursorPos([200，200])
        #按下鼠标中间键
        win32api.mouse_event(win32con.MOUSEEVENTF_MIDDLEDOWN，0，0，0，0)
        time.sleep(1)
        #移动鼠标
        win32api.mouse_event(win32con.MOUSEEVENTF_MOVE，10，10，0，0)
```

```
        time.sleep(1)
        #松开鼠标中间键
        win32api.mouse_event(win32con.MOUSEEVENTF_MIDDLEUP，0，0，0，0)
        time.sleep(2)
        driver.get_screenshot_as_file("D:\LargerView_On_TC_Layout.png")
        elif layout_name=='MT':
        switch_view_to_mt_layout(0)
        WebDriverWait(driver,10).until(lambda
the_driver:the_driver.find_element_by_id('canvasDivContainer0').is_displayed())
        #移动鼠标到指定位置
        win32api.SetCursorPos([130，130])
        #按下鼠标中间键
        win32api.mouse_event(win32con.MOUSEEVENTF_MIDDLEDOWN，0，0，0，0)
        time.sleep(1)
        #移动鼠标
        win32api.mouse_event(win32con.MOUSEEVENTF_MOVE，10，10，0，0)
        time.sleep(1)
        #松开鼠标中间键
        win32api.mouse_event(win32con.MOUSEEVENTF_MIDDLEUP，0，0，0，0)
        time.sleep(2)
        driver.get_screenshot_as_file("D:\LargerView_On_MT_Layout.png")
        #模拟深度层切换
        Def step_skin(layout_name):
        if layout_name=='CC':
        WebDriverWait(driver,10).until(lambda
the_driver:the_driver.find_element_by_id('canvasDivContainer0').is_displayed())
        time.sleep(2)
        win32api.SetCursorPos([100，100])
        time.sleep(1)
        #滚动鼠标中间滚轮一个滚轮单位的值为 120
        For step in range(1，4):
        #if step<skin_steps:
        win32api.mouse_event(win32con.MOUSEEVENTF_WHEEL，0，0，120，0)
```

```
        time.sleep(1)
        driver.get_screenshot_as_file("D:\step_skin_on_CC_Layout.png")
        time.sleep(2)
        elif layout_name=='TC':
        switch_view_to_tc_layout(0)
        WebDriverWait(driver,10).until(lambda
the_driver:the_driver.find_element_by_id('canvasDivContainer0').is_displayed())
        time.sleep(2)
        win32api.SetCursorPos([100，100])
        time.sleep(1)
        #滚动鼠标中间滚轮一个滚轮单位的值为 120
        For step in range(1，4):
        #if step<skin_steps:
        win32api.mouse_event(win32con.MOUSEEVENTF_WHEEL，0，0，120，0)
        time.sleep(1)
        driver.get_screenshot_as_file("D:\step_skin_on_TC_Layout.png")
        time.sleep(2)
        elif layout_name=='MT':
        switch_view_to_mt_layout(0)
        WebDriverWait(driver,10).until(lambda
the_driver:the_driver.find_element_by_id('canvasDivContainer0').is_displayed())
        time.sleep(2)
        win32api.SetCursorPos([100，100])
        time.sleep(1)
        #滚动鼠标中间滚轮一个滚轮单位的值为 120
        For step in range(1，4):
        #if step<skin_steps:
        win32api.mouse_event(win32con.MOUSEEVENTF_WHEEL，0，0，120，0)
        time.sleep(1)
        driver.get_screenshot_as_file("D:\step_skin_on_MT_Layout.png")
        time.sleep(2)
```

更多自动化测试中 Python 运用技巧请学习我的课程《分层自动化测试—从入门到放弃》