目 录 (上 篇)

测试女巫之 Python Unit Test 篇

◆ 作者：王平平

前言:

截止到目前对于 Python 我们已经介绍了 Pywinauto，Pymouse，Pykeyboard，Selenium 这些模块，在博为峰网校我也录制了专门的课程进行了深入的讲解，根据此模块的学习我们可以实现自动化控制运行在 Windows 上的应用程序、可以自动化控制 Browser 上的控件，可以控制鼠标和键盘。但是如何根据我们目前学习的这些模块，针对我们的待测物如何搭建整个测试系统呢？进而如何产生外观漂亮，清晰明朗的测试报告呢？

这次我们学习的就是这方面的知识，可以帮我们搭建测试框架的模块是 Python 自带的一个模块：Unittest，即 Unit test Framework 就可以搭建这个框架。

还会接触到一个模块 HTMLTestRunner，这个模块会帮助我们产出外观漂亮且清晰明朗的测试报告，且不需要学习各种函数，直接调用这个模块即可。

这次 Unittest 相对于之前学习的模块例如 Selenium 或者 Pywinauto，最大的特点

是没有很多的函数需要学习，因为这个模块只是帮助我们建立测试框架，同样 HTMLTestRunner 也不需要了解其中的函数，只需要知道如何调用这个模块即可，且调用模块的方法其实是非常简单的!

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!

第一阶段：工作需求

学习必须要有理由，我们如何也要有充足的工作需求才能说服老板。

所以对于 Unittest 的“工作需求”，女巫总结如下:

1、对一些复杂工作的自动化，需要规划自动化测试的架构，我们通过学习 UnitTest 这个模块可以帮助我们快速建立框架清晰，科学的自动化测试架构

2、我们在完成测试后，希望有一个看起来“高大上”的报告，但是又希望我们不要写太多太复杂的代码，HTMLTestRunner 这个模块可以帮助我们实现这个需求

所以基于上述两个需求，我们可以进行接下来的 Unittest 以及 HTMLTestRunner 的学习。

第二阶段：unittest 基本知识

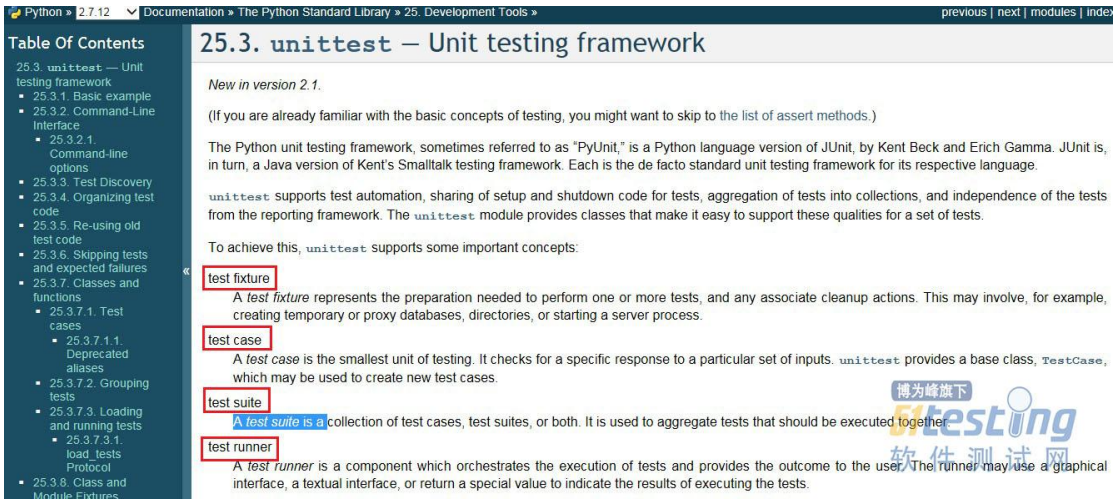
1、unittest 的作用

unittest 支持测试自动化，它是一个测试类来完成对整个软件模块的、测试，它的对象的初始化工作可以在 setUp()方法中完成，而资源的释放则可以在 tearDown()方法中完成，它是将 test case 收集起来一起执行，而产生的报告是针对每个 test Case 对立产生的格式。

2、unittest 官网

https://docs.python.org/2/library/unittest.html

注意：unittest 此模块是 Python 自带的模块，所以不需要另外安装第三方模块。



3、unittest 的重要概念

1) Test fixture(测试治具)

表明一些准备工作需要执行一个或者更多的测试用例，和一些需要清除的动作，它是一个概念，使用 TestCase，setUp 和 tearDown 这些方法来对于测试治具进行初始化，建立测试用例，以及清除。

所以测试治具是被构建的，它是由 setup 方法，TestCase 组件以及 tearDown 方法组成

的。

2) Test case

Test case 是测试的最小单元。根据特殊的一系列的输入产生特定的回应。Unittest 提供一个基本的类：Test case 可以用来创建新的 test cases.

3) Test suite(测试集)

Test suit 是 test case 的集合，它可以用来整合 test cases，使其可以一起进行执行测试。

4) Test runner

Test runner 是执行测试以及提供测试结果给用户的一个组件，它可以使用图形接口，文字接口或者返回一个特殊值来说明执行测试的结果。

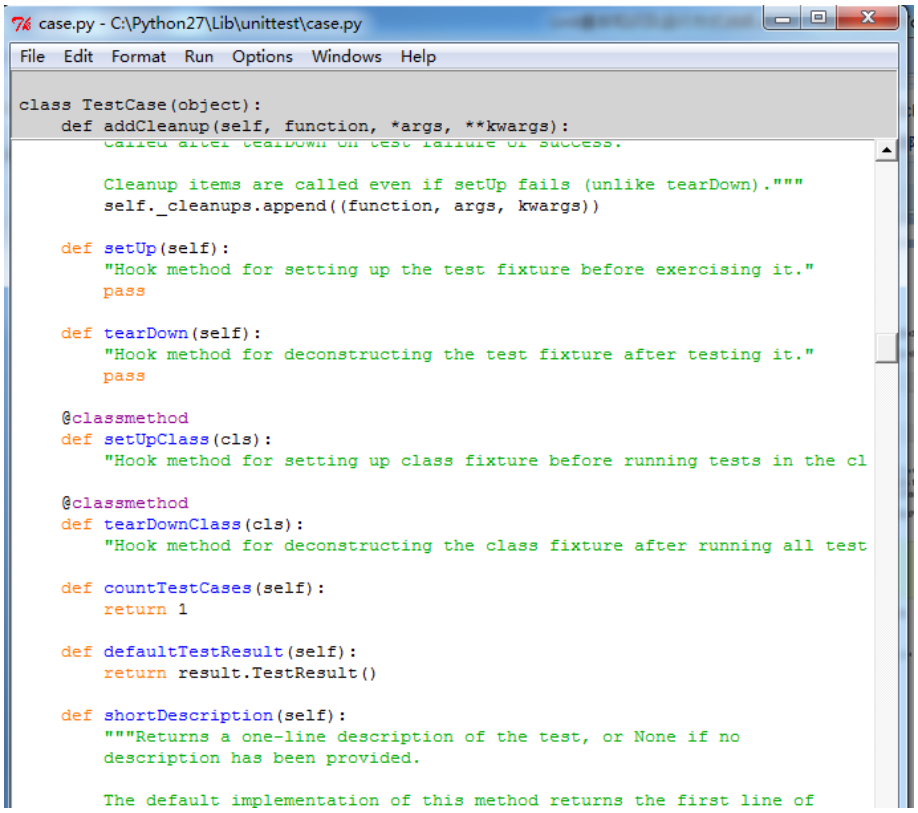
4、具体常用类以及函数介绍说明

1) Test Case 类以及常用函数

a. TestCase 官网说明

进入官网点击左边树形结构的“Classes and Functions”第一个类需要学习的就是 TestCase 类，其中我们用的最多的是 setUp()以及 tearDown()，此两个函数没有任何 input，只要在其中加入 user 自己定义的函数即可。在这两个函数之间加入用户自己定义的函数，这样就建立了自己的 test fixture。

b. TestCase 源代码地址：C:\Python27\Lib\unittest\case.py



c. TestCase 类的作用

主要是建立我们需要测试的若干单元测试用例，也可称为若干个执行函数

d. 框架如下:

✓ setUp 的处理（3 种方式）

a) 测试前的准备工作，可以添加恢复出厂值或者测试前需要设置的参数

b) 直接写 pass

c) 删除本函数

✓ tearDown 的处理（3 种方式）

a) 测试后的清除工作

b) 直接写 pass

c) 删除本函数

✓ 添加调用执行函数的相关的代码

需要测试几个功能就添加几个执行函数，这个执行函数可以理解为单元测试用例。

e. 具体代码例子如下：

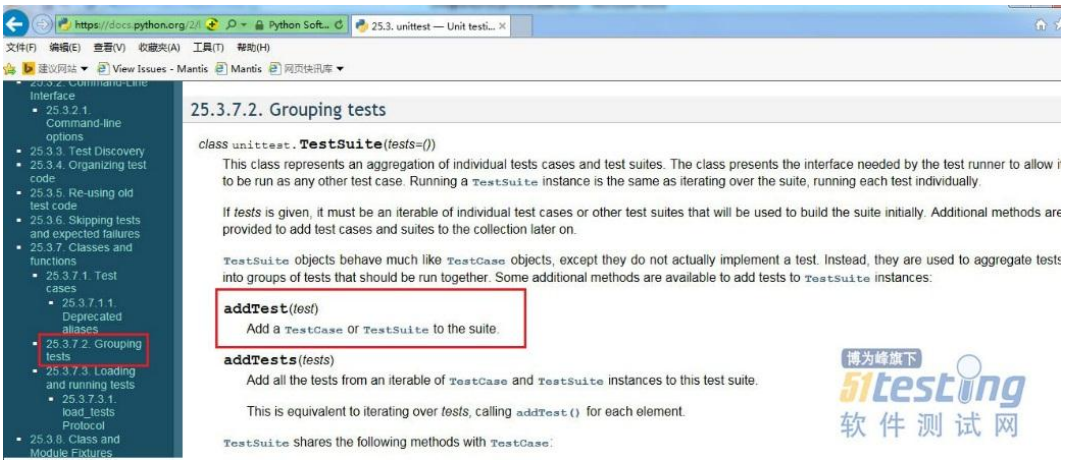
```
#执行测试的类
class test_smoke_test(unittest.TestCase):
    def setUp(self):
        pass


     #System Logs
    def test_Enable_Logging(self):
        Systemlogs.Enable_Logging(self,driver)
    def test_Log_Category_All(self):
        Systemlogs.Log_Category_All(self,driver)
    def test_Log_Category_Network(self):
        Systemlogs.Log_Category_Network(self,driver)
    def test_Log_Category_Security(self):
        Systemlogs.Log_Category_Security(self,driver)


    def tearDown(self):
        pass
```

2) TestSuite 的类以及常用函数

a. TestSuite 官网

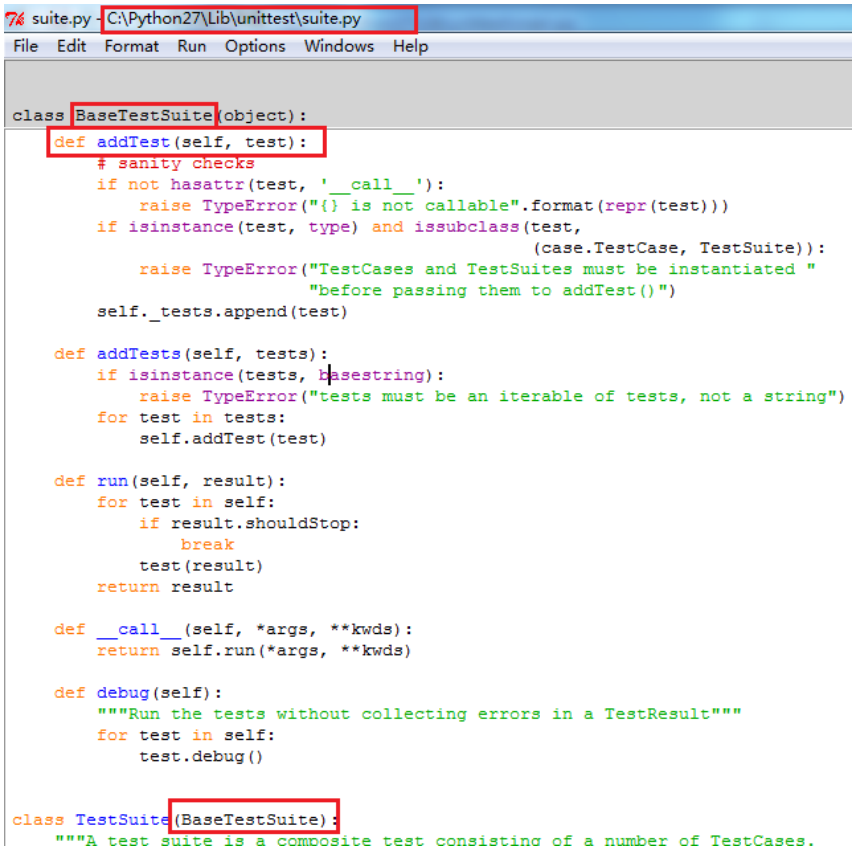
进入官网点击左边树形结构的“Grouping Testing”

此类中我们用的最多的函数是 addTest()



b. TestSuite 源代码地址：C:\Python27\Lib\unittest\suite.py

注意此脚本中有两个重要的类，BasicTestSuite 和 TestSuite，注意 TestSuite 是继承 BasicTestSuite，所以通过 TestSuite 可以同时调用 BasicTestSuite 和 TestSuite 其中的 Function。



c. TestSuit 类的作用

将测试用例聚合成测试套件，即把 test case 整合起来作为一个测试套件，以方便后续运行，所以我们经常使用的一个函数就是 addTest(self, test)，即使用这个函数将我们在上一步建立的 test case 一步步的加入使用 addTest 这个函数将 TestCase 中导入的各个单元测试导入到测试套件中

注意：addTest 这个函数的 input parameter 为 unittest 类名(单元测试函数名)

例如：testsuite.addTest(test_class("test_Data_Roaming"))

```
#执行测试的类
class test_sdk_test(unittest.TestCase):
    def setUp(self):
        pass

    ################################################################################
    #no1.AT Command
    def test_AT_Command(self):
        AT_Command.at_command(self,k)
    #no2.WWAN
    def test_get_wwan_serving_system_provider(self):
        WWAN_System.get_wwan_serving_system_provider(self,k)
    def test_get_wwan_serving_system_radio_mode(self):
        WWAN_System.get_wwan_serving_system_radio_mode(self,k)
    def test_wwan_network_radio_mode_setandget_2Gmode(self):
        WWAN_System.wwan_network_radio_mode_setandget_2Gmode(self,k)
    def test_wwan_network_radio_mode_setandget_3Gmode(self):
        WWAN_System.wwan_network_radio_mode_setandget_3Gmode(self,k)
    def test_wwan_network_radio_mode_setandget_4Gmode(self):
        WWAN_System.wwan_network_radio_mode_setandget_4Gmode(self,k)
    def test_wwan_network_radio_mode_setandget_23Gmode(self):
        WWAN_System.wwan_network_radio_mode_setandget_23Gmode(self,k)
    def tearDown(self):
        pass

if __name__ == "__main__":

    #构造测试集
     testsuite=unittest.TestSuite()
     testsuite.addTest(test_sdk_test("test_AT_Command"))
     testsuite.addTest(test_sdk_test("test_get_wwan_serving_system_provider"))
     testsuite.addTest(test_sdk_test("test_get_wwan_serving_system_radio_mode"))
     testsuite.addTest(test_sdk_test("test_wwan_network_radio_mode_setandget_2Gmode"))
     testsuite.addTest(test_sdk_test("test_wwan_network_radio_mode_setandget_3Gmode"))
     testsuite.addTest(test_sdk_test("test_wwan_network_radio_mode_setandget_4Gmode"))
     testsuite.addTest(test_sdk_test("test_wwan_network_radio_mode_setandget_23Gmode"))
```

第三阶段：运行方式说明

1、直接运行测试用例

1) 弊端：测试用例必须要以 test 开头命名

2) 优点：不需要额外将测试用例导入到 unittest 中

3) 运行方式： if __name__ == "__main__":

unittest.main()

4) 代码举例

```
#执行测试的类
class test_smoke_test(unittest.TestCase):
    def setUp(self):
        pass


     #System Logs
    def test_Enable_Logging(self):
        Systemlogs.Enable_Logging(self,driver)
    def test_Log_Category_All(self):
        Systemlogs.Log_Category_All(self,driver)
    def test_Log_Category_Network(self):
        Systemlogs.Log_Category_Network(self,driver)
    def test_Log_Category_Security(self):
        Systemlogs.Log_Category_Security(self,driver)


    def tearDown(self):
        pass

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

5) 执行顺序：以名字的大小为序

依次为 test_Enable_Logging, test_Log_Category_All,

test_Log_Category_Network

2、将测试用例聚合成测试套件运行方式

1) 弊端：测试用例必须一条一条手动添加，不想运行的用例要一条一条删除或屏蔽掉

2) 优点：执行测试的顺序一目了然，很好控制

3) 运行方式：testsuite=unittest.TestSuite ()

testsuite.addTest(test_smoke_test("Status_3G"))

testsuite.addTest(test_smoke_test("PIN_On"))

4) 代码举例：

```
#执行测试的类
class test_smoke_test(unittest.TestCase):
    def setUp(self):
        pass

     #Status
    def Status_3G(self):
        Status.Status_3G(self,driver)
     #PIN Configuration
    def PIN_On(self):
        PIN_Configuration.PIN_On(self,driver)
    def PIN_Off(self):
        PIN_Configuration.PIN_Off(self,driver)

    def tearDown(self):
        pass

if __name__ == "__main__":

    #构造测试集
     testsuite=unittest.TestSuite()
     testsuite.addTest(test_smoke_test("Status_3G"))
     testsuite.addTest(test_smoke_test("PIN_On"))

     #构建report
     filename="F:\\Smoke test for WLD7\\results.html"
     fp=file(filename,'wb')
     runner=HTMLTestRunner.HTMLTestRunner(stream=fp,title='Result',description='Test_Report')

     #执行测试
     runner.run(testsuite)
```

5) 执行顺序：以添加测试用例的先后为序

依次为 Status_3G, PIN_On

3、嵌套测试套件

1) 弊端：执行测试的顺序不能一目了然，需要通过命名的方式来控制相似名称的测试用例才能使用此方法

2) 优点：测试用例可以批量添加或批量删除

3) 运行方式：testsuite1=unittest.makeSuite()

testsuite.addTest(test_smoke_test,"Time")

testsuite.addTest(test_smoke_test,"PIN")

testsuite=unittest.TestSuite(testsuite1,testsuite2)

4) 代码举例：

```
#执行测试的类
class test_smoke_test(unittest.TestCase):
    def setUp(self):
        pass
     #PIN Configuration
    def PIN_On(self):
        PIN_Configuration.PIN_On(self,driver)
    def PIN_Off(self):
        PIN_Configuration.PIN_Off(self,driver)
    #no3.System Function
     #Time Settings
    def Time_Zone_INDEX_Daylight_0(self):
        Time_Settings_Daylight.Time_Zone_INDEX_Daylight_0(self,driver)
    def Time_Zone_INDEX_Daylight_2(self):
        Time_Settings_Daylight.Time_Zone_INDEX_Daylight_2(self,driver)
    def Time_Zone_INDEX_Daylight_9(self):
        Time_Settings_Daylight.Time_Zone_INDEX_Daylight_9(self,driver)
    def Time_Zone_INDEX_Daylight_10(self):
        Time_Settings_Daylight.Time_Zone_INDEX_Daylight_10(self,driver)
    def Time_Zone_INDEX_Daylight_12(self):
        Time_Settings_Daylight.Time_Zone_INDEX_Daylight_12(self,driver)

    def tearDown(self):
        pass
if __name__ == "__main__":
    #构造测试集
     testsuite1=unittest.makeSuite(test_smoke_test,'Time')
     testsuite2=unittest.makeSuite(test_smoke_test,'PIN')
     testsuite=unittest.TestSuite((testsuite2,testsuite1))
     #构建report
     filename="F:\\Smoke test for WLD7\\results.html"
     fp=file(filename,'wb')
     runner=HTMLTestRunner.HTMLTestRunner(stream=fp,title='Result',description='
     #执行测试
     runner.run(testsuite)
```

5) 执行顺序：

以 unittest.TestSuite(testsuite1,testsuite2)中添加的先后为序，testsuit 中以名字的大小为序。

依次为 PIN_Off, PIN_On, Time_Zone_INDEX_Daylight_0,

Time_Zone_INDEX_Daylight_10, Time_Zone_INDEX_Daylight_12,

Time_Zone_INDEX_Daylight_2, Time_Zone_INDEX_Daylight_9

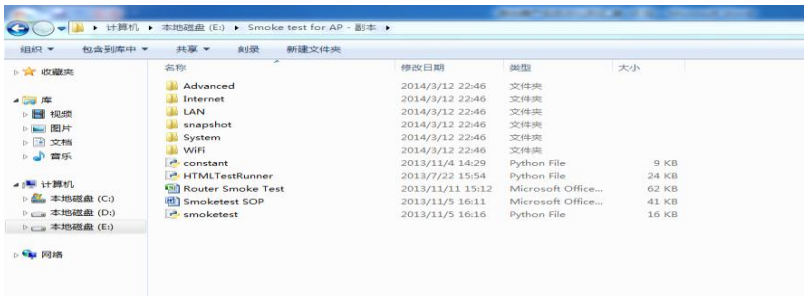
第四阶段：Unittest 的框架

1、目标

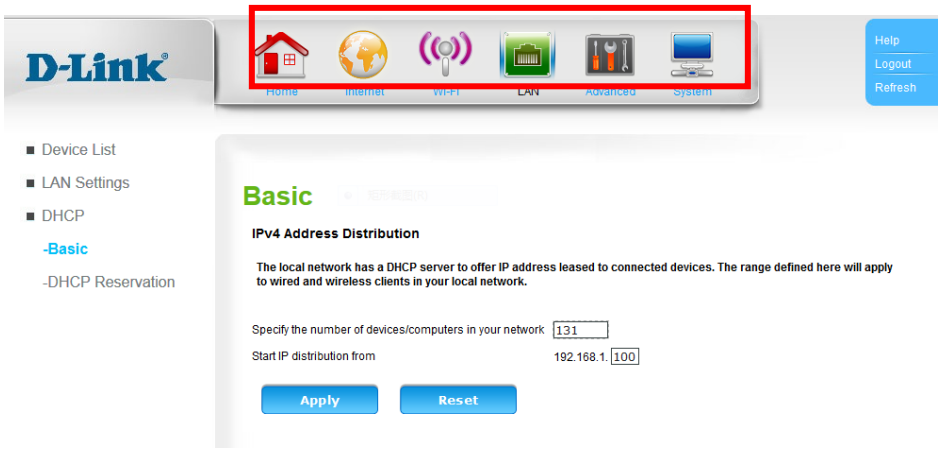
将不同的功能作为独立的模块进行被调用和维护，方便维护，同时也可以使自动化脚本的框架清晰明了。

2、文件夹建立

在某一存储位置下(例如:E 盘)建立一个文件夹：例如 smoke test for AP 在此文件夹下建立有关各个模块的子文件夹(有多少模块就有多少子文件夹)：例如 Advanced 和 Internet 就必须分别建立两个文件夹，如下图

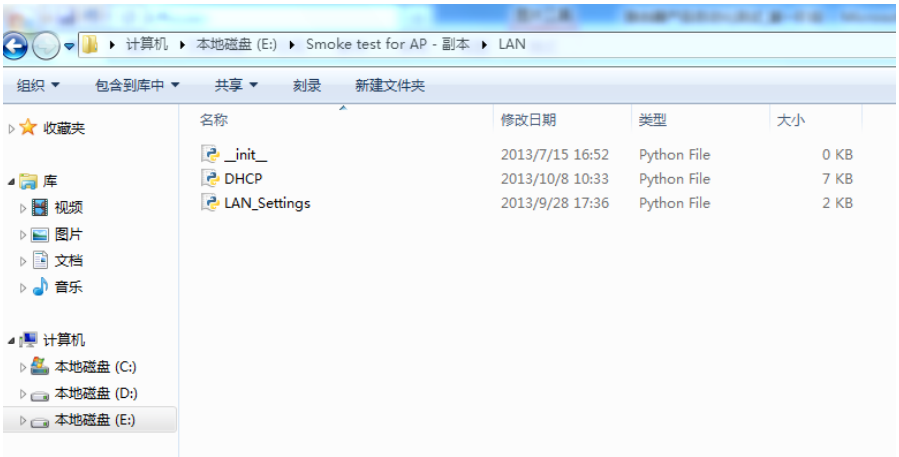


此文件夹是根据下图红框所示的功能块进行建立的



如果 LAN 其中有两个 item 则在 LAN 中建立两个”.py”

文档一个命名为子 function 的名称，如下图:



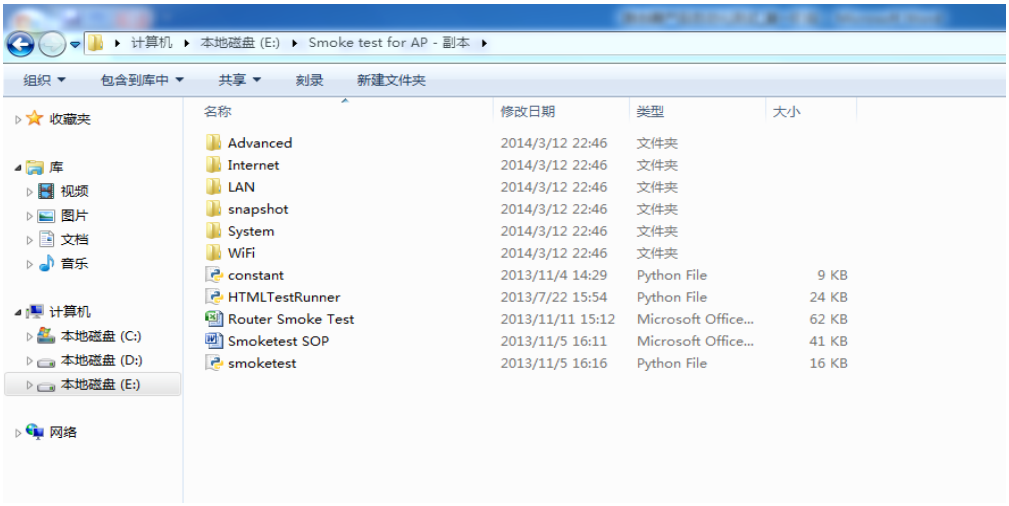
注意：每个文件夹中必须有一个名称为”__init__.py”的内容为空白 py 文档,只有包含此文档则此文件夹中的模块才能被其它的脚本正常调用。

Smoke test for AP 此文件夹中放入调用各个脚本的 smoketest.py 和其中含有各个模块都会调用的函数的 init.py 此两个脚本。

注意：每个模块只能实现一个大的功能模块，如此功能模块是由两个脚本组成：例如 DHCP 和 LAN Setting。

3、脚本的分配

主要分成 3 个方面：smoketest.py ， constant.py ，HTMLTestResult.py 和其它各个功能的子模块(如下图中的各个文件夹)如下图



1) smoketest.py

此脚本的意义是调用各个模块中函数并执行，即执行自动化测试时只要运行此脚本即可，因为它的用法就是调用各个模块并执行各个模块。

如下图:

```
 # coding=utf-8
import HTMLTestRunner,unittest,time,sys,os
from selenium import webdriver
from LAN import DHCP

sys.path.append(os.path.dirname(__file__))
driver = webdriver.Firefox()
driver.implicitly_wait(30)
driver.get("http://192.168.1.1/")
driver.find_element_by_id("txtUserName").clear()
driver.find_element_by_id("txtUserName").send_keys("admin")
driver.find_element_by_id("txtLoginPwd").clear()
driver.find_element_by_id("txtLoginPwd").send_keys("admin")
driver.find_element_by_name("button.login.users.home").click()

#执行测试的类
class test_smoke_test(unittest.TestCase):
    def setUp(self):
        pass
     #DHCP
    def DHCP_Settings(self):
        DHCP.DHCP_Settings(self,driver)
    def DHCP_Function(self):
        DHCP.DHCP_Function(self,driver)
    def DHCP_Reservation_Add(self):
        DHCP.DHCP_Reservation_Add(self,driver)
    def DHCP_Reservation_Edit(self):
        DHCP.DHCP_Reservation_Edit(self,driver)
    def DHCP_Reservation_Delete(self):
        DHCP.DHCP_Reservation_Delete(self,driver)
    def tearDown(self):
        pass


if __name__ == "__main__":
    #构造测试集
    testsuite=unittest.TestSuite()
    testsuite.addTest(test_smoke_test("DHCP_Settings"))
    testsuite.addTest(test_smoke_test("DHCP_Function"))
    testsuite.addTest(test_smoke_test("DHCP_Reservation_Add"))
    testsuite.addTest(test_smoke_test("DHCP_Reservation_Edit"))
    testsuite.addTest(test_smoke_test("DHCP_Reservation_Delete"))


    #构建report
    filename="E:\\Selenium\\Auto test for AP\\results.html"
    fp=file(filename,'wb')
    runner=HTMLTestRunner.HTMLTestRunner(stream=fp,title='Result',description='Test_Report')

    #执行测试
    runner.run(testsuite)
```

smoketest.py 脚本的详细说明:

coding=utf-8

#加入上述语句即可以在此脚本中输入中文，否则会报错

import HTMLTestRunner,unittest,time,sys,os

from selenium import webdriver

from LAN import DHCP

#导入所需要的各个模块

sys.path.append(os.path.dirname(__file__))

driver = webdriver.Firefox()

driver.implicitly_wait(30)

driver.get("http://192.168.1.1/")

driver.find_element_by_id("txtUserName").clear()

driver.find_element_by_id("txtUserName").send_keys("admin")

```
driver.find_element_by_id("txtLoginPwd").clear()
driver.find_element_by_id("txtLoginPwd").send_keys("admin")
driver.find_element_by_name("button.login.users.home").click()
#上述语句是进入路由器的 Web UI 界面，并清空已输的内容，输入正
确的用户名和密码
class test_smoke_test(unittest.TestCase):
def setUp(self):
    pass
#DHCP
    def DHCP_Settings(self):
  DHCP.DHCP_Settings(self,driver)
    def DHCP_Function(self):
 DHCP.DHCP_Function(self,driver)
    def DHCP_Reservation_Add(self):
 DHCP.DHCP_Reservation_Add(self,driver)
    def DHCP_Reservation_Edit(self):
 DHCP.DHCP_Reservation_Edit(self,driver)
    def DHCP_Reservation_Delete(self):
 DHCP.DHCP_Reservation_Delete(self,driver)
    def tearDown(self):
  pass
#上述内容是导入 DHCP 脚本中定义的函数
if __name__ == "__main__":
testsuite=unittest.TestSuite()
testsuite.addTest(test_smoke_test("DHCP_Settings"))
testsuite.addTest(test_smoke_test("DHCP_Function"))
testsuite.addTest(test_smoke_test("DHCP_Reservation_Add"))
testsuite.addTest(test_smoke_test("DHCP_Reservation_Edit"))
testsuite.addTest(test_smoke_test("DHCP_Reservation_Delete"))
    #上述内容是将导入多的函数加到测试集中
  #构建 report
 filename="E:\\Selenium\\Auto test for AP\\results.html"
 fp=file(filename,'wb')
```

```
#上述内容是创建测试报告所在的位置以及创建测试报告
runner=HTMLTestRunner.HTMLTestRunner(stream=fp,title='Result',description='Test_Report')
#上述内容是调用 HTMLTestRunner 脚本，并产生报告
runner.run(testsuite)
#执行测试
```

2) constant.py

将各个模块使用的一些常量，均集中到此脚本中，其它的脚本如需用到这些常量通过调用函数的方式来使用。这样做有一个很大的好处:

即如果这些常量发生变化，只要在此脚本中更改就可以了，其它需要使用这些常量的脚本就不需要再进行更改。

```
import os

##########################################################################
#LAN
DHCP_Reservation_host_name_add_edit='44:37:e6:6b:88:63'
DHCP_Reservation_IP_address_add='192.168.2.10'
DHCP_Reservation_IP_address_edit='192.168.2.22'
DHCP_Reservation_IP_address_delete='192.168.2.26'
DHCP_Reservation_host_name_delete='40:16:9f:95:22:4f'
def get_DHCP_Reservation_host_name_add_edit():
    return DHCP_Reservation_host_name_add_edit
def get_DHCP_Reservation_IP_address_add():
    return DHCP_Reservation_IP_address_add
def get_DHCP_Reservation_IP_address_edit():
    return DHCP_Reservation_IP_address_edit
def get_DHCP_Reservation_IP_address_delete():
    return DHCP_Reservation_IP_address_delete
def get_DHCP_Reservation_host_name_delete():
    return DHCP_Reservation_host_name_delete

Username='admin'
Password='admin'
def get_Username():
    return Username

def get_Password():
    return Password
```

【图 35】

Constant.py 脚本的详细说明如下

```
#LAN
DHCP_Reservation_host_name_add_edit='44:37:e6:6b:88:63'
DHCP_Reservation_IP_address_add='192.168.2.10'
DHCP_Reservation_IP_address_edit='192.168.2.22'
DHCP_Reservation_IP_address_delete='192.168.2.26'
DHCP_Reservation_host_name_delete='40:16:9f:95:22:4f'
#上述代码是对这些常量定义对应的变量
def get_DHCP_Reservation_host_name_add_edit():
return DHCP_Reservation_host_name_add_edit
```

```
def get_DHCP_Reservation_IP_address_add():
return DHCP_Reservation_IP_address_add
def get_DHCP_Reservation_IP_address_edit():
return DHCP_Reservation_IP_address_edit
def get_DHCP_Reservation_IP_address_delete():
return DHCP_Reservation_IP_address_delete
def get_DHCP_Reservation_host_name_delete():
return DHCP_Reservation_host_name_delete
#为了其它脚本便于调用,将这些变量定义为函数
Username='admin'
Password='admin'
def get_Username():
return Username

def get_Password():
return Password
```

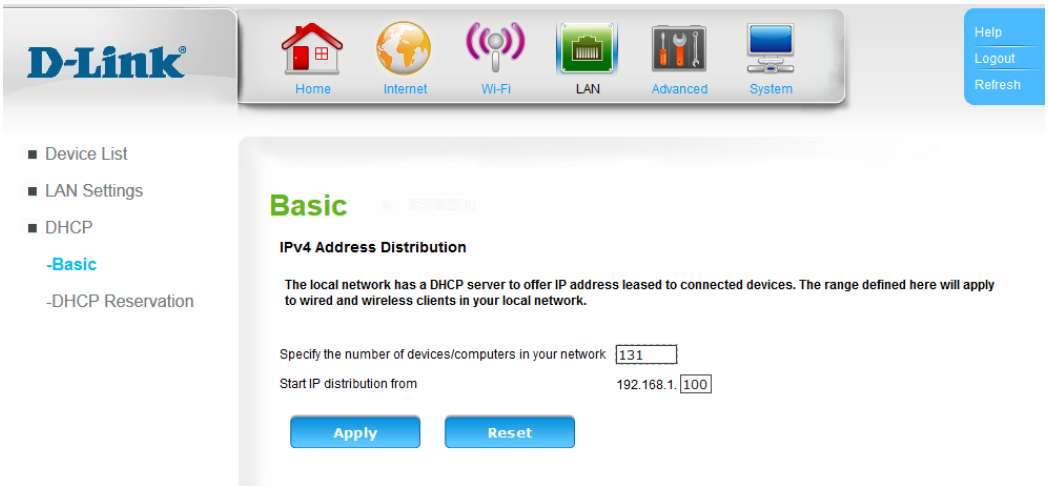
3)各个子模块

这些脚本的作用是：具体操作路由器各个功能。

例如：实现的功能是在 DHCP 中”specify the number of devices/computers in your network”自己设置一个值，点击 Apply 后确认设置的数值正确。如下图

```
from selenium.webdriver.support.ui import Select
import time
testcase=__name__.encode('utf-8')
#DHCP
def DHCP_Settings(self,driver):
driver.find_element_by_id("lan_menu_img").click()
 #点击 LAN 进入 LAN 的设置界面
driver.find_element_by_link_text("DHCP").click()
#点击 DHCP 进入 DHCP 设置界面
driver.find_element_by_id("num_devices").clear()
 #选择文本框 ""Specify the number of devices/computers in your network"，并清除文本框默认的值
```

```
    driver.find_element_by_id("num_devices").send_keys("18")
  #设置文本框的值为 18
    driver.find_element_by_name("button.apply").click()
  #点击 apply,保存此设置
after_set1=driver.find_element_by_id
("number_devices").get_attribute("value")
self.assertEqual(after_set1,"18")
#判断保存成功后的值与设置的值是否相同
```



第五阶段：自动产生 HTML 报告

1、下载 HTMLTestRunner.py 此脚本，下载地址：

http://www.51testing.com/html/12/n-3715212.html

2、放置位置

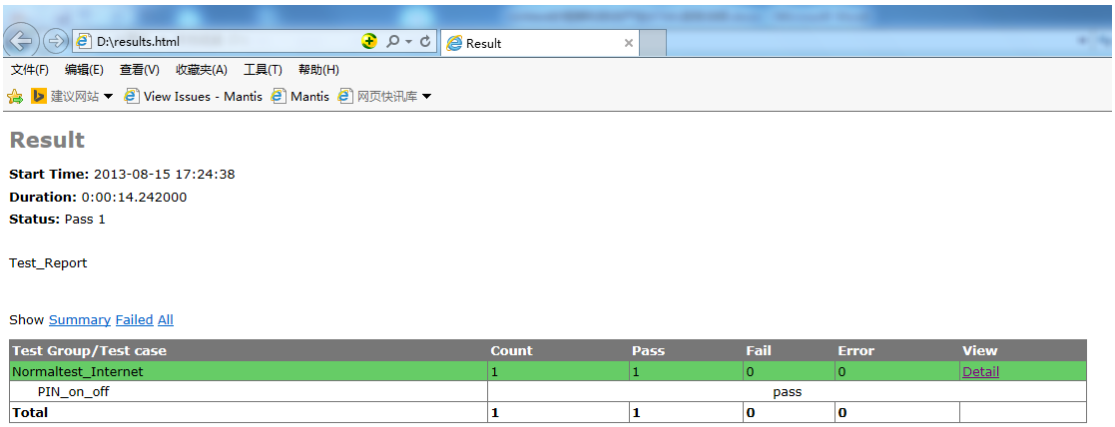
将此脚本放到 c:\python27\lib 此文件夹中，或者放置到与 smoketest.py 在同一个文件夹中

3、执行脚本中添加语句 import HTMLTestRunner

4、执行脚本中删除语句 unittest.main()，添加语句

```
testsuite=unittest.TestSuite()
testsuite.addTest(test_class("test_Data_Roaming"))
filename="E:\\re\\results5.html"
fp=file(filename,'wb') runner=HTMLTestRunner.HTMLTestRunner(stream=fp,title='Result',desc
ription='Test_Report')
runner.run(testsuite)
```

注：黄色标记为自己定义的内容



总结

我们这一期介绍的是 Unittest 以及 HTMLTestRunner 这两个模块的学习，这两个模块学习起来比较简单，因为并不需要学习模块中每个函数的用法，重点是学习 Unittest 的框架以及其中的每个 test case 的运行方式，把框架和运行方式搞懂了，就可以根据我们自己的测试需求搭建属于自己的自动化测试框架。HTMLTestRunner 实际上是一个开源的脚本，目前这个脚本在官网上也可以直接下载，这个脚本大家不需要查看其中的内容，只要快速了解如何调用这个脚本就可以了。所以我们这一期的学习还是非常轻松的，在积累的前几期的知识的基础上，我们可以轻松搭建自动化测试框架，以及会非常轻松的产出看起来非常“高大上”的报告。如果觉得这几期讲解的模块知识，光看文章学会运用还是

有些难度的话，可以看看我录制的课程《Python 系列课程之玩转自动化》（http://www.atstudy.com/course/158），配合这几期的文章知识，学习掌握会更形象透彻一些。

还是那句话大家在解决工作中的具体问题时，思路一定要灵活，对于具体问题一定要先考虑这个模块的作用是什么，我如何快速学习这个模块，如何学习才会比较节省人力物力，这样你才能与老板真的有共同语言：即一定要切记：一定要用最少的人力物力，实现你的自动化！

参考文献：

1) Python 官网：http://www.python.org/getit/

2) UnitTest 的官网：https://docs.python.org/2/library/unittest.html#
(此网站包括如何使用，以及其中的一些常用函数的介绍)

3) HTMLTestRunner 下载官网：https://pypi.python.org/pypi/HTMLTestRunner

时间管理方法

◆搜狗测试：陈　俭

一、思考

1、工作是否忙碌？盲目？

盲目的原因:

a、人生关键时期，被迫要求角色转变

b.我们还来不及构建自己的职业规划和人生目标

c.我们不具备平衡生活和工作的能力

d.没有养成良好的习惯

确实很忙分两种人：一种是具备时间管理能力的人，另一种是不具备的人；前者用20%的时间完成后者 80%的工作。

时间黑洞：它会永无止境的吞噬你宝贵的时间

2、价值观是什么？

成熟的标志是明确自己的价值观，价值观是指一个人对周围的客观事物（人、事、物）的意义、重要性的总评价和总看法。职业价值观直接关系到我们判断工作任务的轻重、缓急。

价值观没有对错。

比如有三种人

1) 拿着地图走路，做事之前已经做好详细的加护，按照计划执行

2) 看着路牌走路，走一步看一步，每到十字路口都要重新选择一次，最终到哪也不知道

3) 顺着方向走，选择一个大的方向，朝着这个方向努力，靠信念往前走。

第一种是建筑师、第二种是科研人员、第三种是创业者。没有对错，只是看你愿意做哪种人。

3、找到自己搞笑的时间段，如何寻找

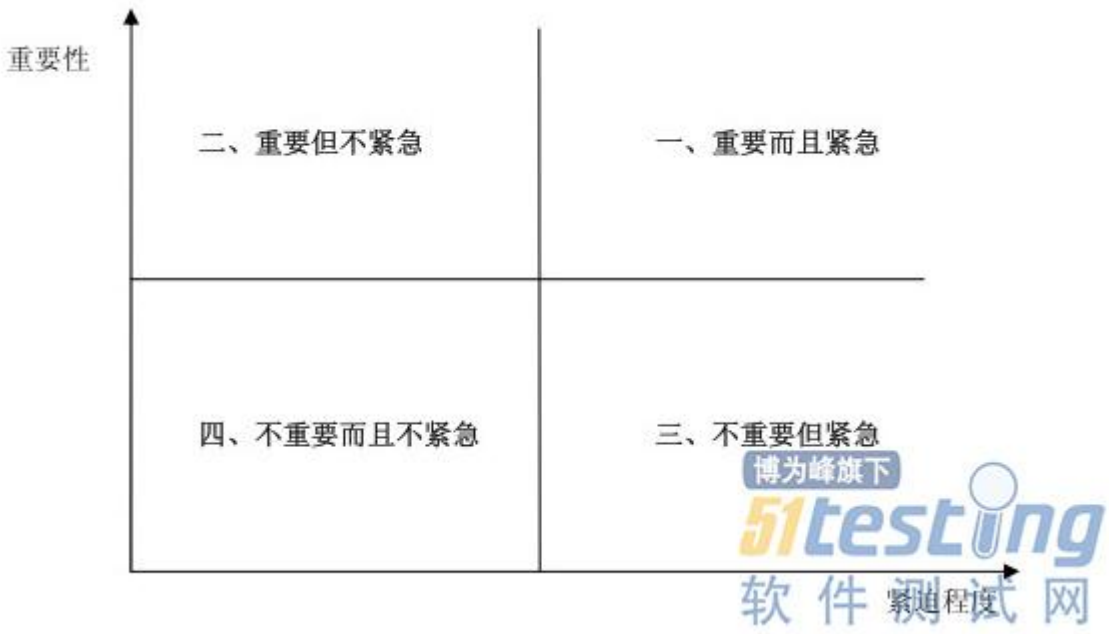
坚持两天，每天每隔一小时计划下一个小时的任务，连续两天后，可以统计出自己那个时段的效率最高，这个时段用来处理最重要的事

4、你是如何管理时间的？

讨论并记录

二、时间管理的方法

1、四象限法则



第一象限：重要紧急

比如危机公关、小孩填报志愿、紧急上线包。这类事情必须马上处理，否则有严重后果。

思考：是否真的有那么多紧急重要的事？

第二象限：重要不紧急

如制定 2017 年团队目标、制定明年家庭旅游计划、考驾照。这些事看起来不紧急，但是我们不可能置之不理，如果不重视将来会发展为重要紧急的事。

思考如何避免更多的事进入到第一象限？

第三象限：紧急不重要

如突然来电、临时会议。

思考：如何减少这个象限的事务

第四象限：不重要不紧急

如追剧、闲聊、嗑瓜子儿等

思考：工作中应该避免这个象限的事

四象限使用方法：

a.先轻重

b.再缓急

c.对每个象限任务进行高、中、低优先级排列

d.优先解决重要且紧急的工作，制定计划去做重要但不紧急的工作

2、衣柜整理法

衣柜整理法的五个流程是：收集、整理、组织、执行、回顾

收集：通过尽量少的工具（手机、便签等）收集一切未完成的工作。

整理：将收集的条目进行分类，分为可以执行和不能执行两种。可执行的任务分为：

- 两分钟之内可以搞定的事——立即去做
- 需要多个步骤搞定的项目——分解成任务-下一步行动
- 可以直接做的事——有空的时候搞定
- 特定时间做的事——写在日程表里，设置到时提醒

组织：将分类的条目重新储存，形成一个“收集篮”和三个清单。

- “收集篮”：一切引起我们注意的东西，所有接收到的任务列表

- 将来清单：将来或可能的清单；
- 项目清单：需要多个步骤才能完成的项目；
- 待办事项清单：今日可以立刻做的清单。

回顾：计划和目标并不是一成不变的，避免盲目，需要定期回顾，可以对计划进行更新和完善。

- 回顾“将来清单”，可以将一些不感兴趣的事情和一些时机和条件不成熟的删去。将一些可实现的事放到项目清单。
- 回顾“项目清单”，了解项目进展的进度，考虑各个项目中需要增加或者删除哪些流程，确保项目的高效的执行。
- 回顾“待办事项清单”，了解哪些事情是否及时处理。

建议回顾频率：每天下班或每周一次

行动：执行我们的行动。

3、脑袋里只装一件事

假设有 100 件事待办，收集篮把所有问题都装起来，清空大脑，每次挑出一件事处理，处理完一件事，扔掉，再挑下一件事处理，大脑只需要同时处理一件事，就不会有那么大的压力了。

举例：车站咨询员，无论有多少旅客，只同时回答一个人的问题，这样就能有条不紊

好处：专注、成就感、摆脱压力

脑袋里只处理一件事，我们只需要思考，下一个行动是什么。

4、区分“项目”和“行动”

项目由行动组成，行动是可执行的步骤

如何找到下一步行动？

1) 动词开头

2) 内容清晰

3) 描述结果

4) 设定开始时间、周期、最后期限

用 5W 正确描述“下一步行动”

为什么？ 谁？ 什么时候？ 在哪里？ 做什么？

项目举例：制作产品销售策略、计划周末旅行、建立 XX 制度、准备考研究生、学习 python

任务举例：挑选一只鱼竿、跟进老大反馈的 bug

5、番茄工作法

使用番茄工作法，选择一个待完成的任务，将番茄时间设为 25 分钟，专注工作，中途不允许做任何与该任务无关的事，直到番茄时钟响起，然后在纸上画一个 X 短暂休息一下（5 分钟就行），每 4 个番茄时段多休息一会儿。

亲身体验：执行用例时使用番茄工作法，效率很高

6、时间管理的三大杀手

1) 拖延
2) 犹豫不决
3) 目标不明确

解决方案:

- 细化目标，对每个任务进行分解，明确下一步动作
- 设定期限，并给自己设立一些奖惩措施
- 优先解决难啃的任务，举例：一半沙子和一般石头，如何装进一个瓶子里，装满。两种方式，先沙子，后石头；先石头，后沙子，第二种会装满。沙子是琐事、小事；石头是重要的事。

三、时间管理工具推荐

- ✓ 奇妙清单 app
- ✓ iphone 的提醒事项、日历
- ✓ Onenote
- ✓ outlook 约会

需求评审前的测试准备流程规范

◆ 搜狗测试：李　越

目的：对于测试人员来说，要做好测试工作、让产品质量更高，用户体验更好，那就需要尽早介入，通常在需求文档形成后需求评审前，分析需求文档，做好讲解前的测试准备工作，这样就可以帮助完善需求文档，并为后续的测试工作铺平道路，节省沟通成本、避免后续过多的需求变更和实现变更带来的各种麻烦。

一、分析需求目的合理性

分析为什么会提出这个需求，该需求的目的对用户来说是否可接受。即这个需求能给用户带来什么，用户需要通过什么路径来得到需求带来的结果。

例：

浏览器四周年时的刮刮乐扩展，此需求提出是为了借助四周年的契机，向沉默用户推送安装刮刮乐扩展，以此唤起沉默用户。扩展的功能是登录通行证后可以参与刮刮乐活动，此活动的奖品是几个苹果产品，中奖率很低。看到这里，你是否开始否定这个需求了呢？我们可以分析一下，首先要明确一点，沉默用户是很久没有使用搜狗浏览器的了，对于这些人，面对参与刮奖还需要登录通行证的繁琐操作和中奖率极低的奖品诱惑时，是否会去参与活动呢？上线后该活动的参与用户为几百人。

二、细看需求描述，画出流程图

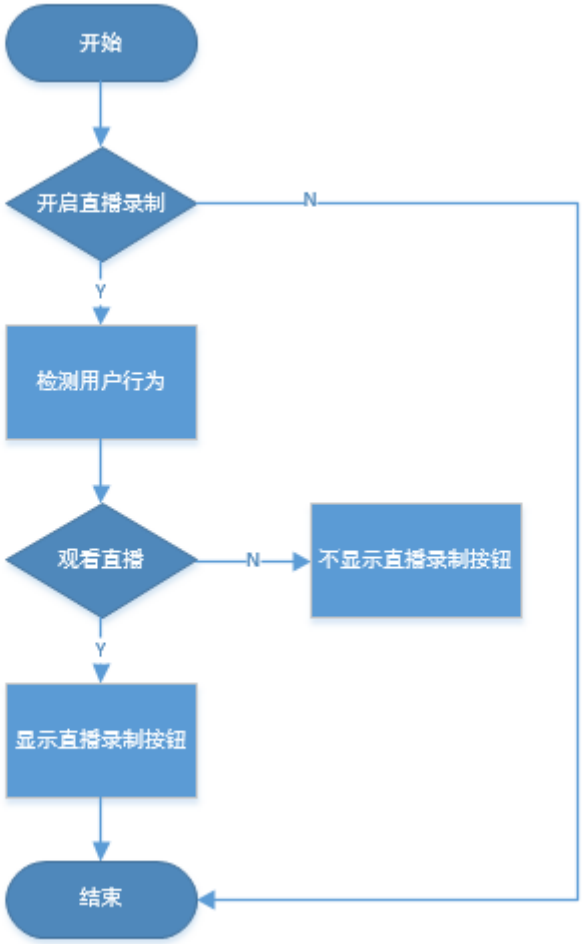
这里的流程图是需求流程图，也就是完全基于需求文档的流程图，不涉及到开发实现。

例：

需求描述：用户可以在选项中打开或者关闭直播录制功能，关闭直播录制功能后直

播录制按钮不再出现。当检测到用户在浏览器页面中观看直播时，显示直播录制功能按钮。

需求流程:



三、根据流程，提出问题

根据需求流程图，争取在每一环节都提出问题，然后记录下来。

例:

1.开启直播录制

（1）怎样开启直播录制?

（2）都支持哪些直播网站?

2.检测用户行为

（3）怎样检测用户行为？

3.观看直播

（4）怎样才算观看直播呢？

4.显示/不显示直播录制按钮

（5）什么时机显示直播录制按钮？

（6）在哪里显示直播录制按钮？

（7）直播录制按钮长什么样子？

（8）滚动页面，直播录制按钮还显示么？

（9）什么时机直播录制按钮会消失呢？

四、根据需求描述，补充问题

例：

上面需求描述是：用户可以在选项中打开或者关闭直播录制功能，关闭直播录制功能后直播录制按钮不再出现。当检测到用户在浏览器页面中观看直播时，显示直播录制功能按钮。

补充问题：

在选项中哪里打开直播录制？

既然在选项中有开关，那么该项纪录在 config.xml 还是 commcfg.xml？

这一项的字段是什么？

这一项是否跟随通行证同步？

五、问题回顾，增删问题

将文档全部看完后 ，回顾记录的问题，有的可能已经被解答，有的联系上下文可能还说不通，产生新的问题，再次记录。

例：

第一个问题问怎样开启直播录制，而在需求描述中已经说了在选项中打开或者关闭直播录制功能，所以第一个问题已经解答，可以删掉。

六、评估已有需求是否符合需求目的

主要是从已有需求是否符合需求目的、本需求是否符合需求目的、是否有更简单便捷的路径满足需求目的三方面考虑。

例:

追剧需求中的弹泡提醒时间，分为连续观看和固定星期观看。最近三天，每天都有同一剧集的观影记录，对于观看日期连续的剧集，需要每日提醒。同一剧集，连续两周的观看星期时间相同，则成为固定星期时间观看，需要在相应的星期时间提醒。

其实对于用户来说，分的这么仔细更容易让用户混淆，还不如直接采用最近一次观看剧集的时间进行提醒更简单，更容易让用户理解和接受。

七、找寻缺陷，完善需求

找寻缺陷，主要指的是在需求中没有说明的但是又是必不可少的一项。

例:

需求统计 pingback，隐性需求。

八、提出建议，锦上添花

可以从系统规范、产品 UI、需求细节合理性以及产品易用性几个方面入手，站在用户角度进行思考。

1、符合系统规范

产品风格尽量与系统风格保持一致，这样使得产品更专业，同时增强用户易用性，更容易让用户理解。

例:

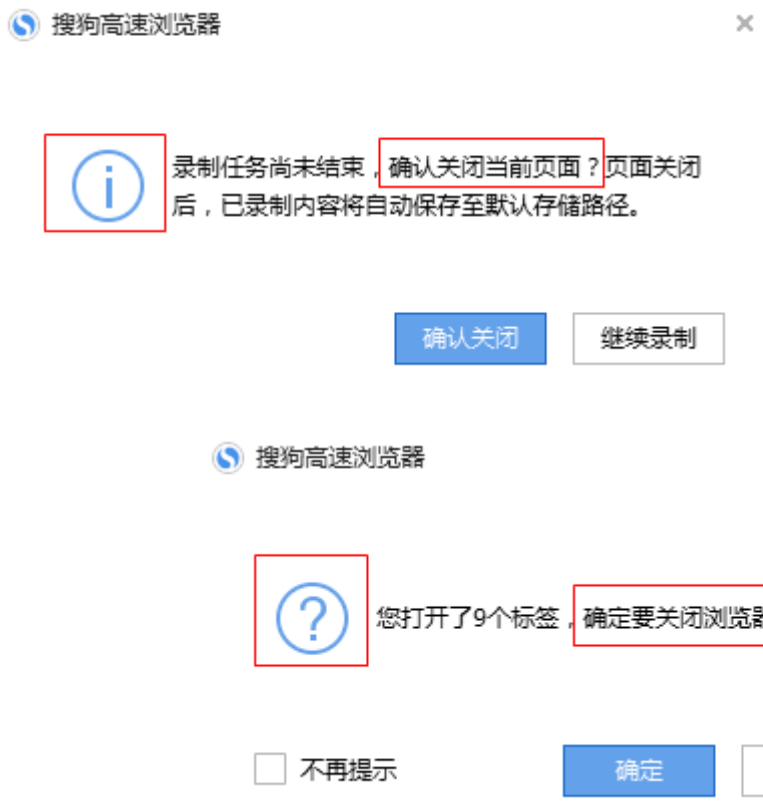
Window + M 是系统快捷键，那么在浏览器的快捷键中，就不要设置它为浏览器快捷键，以免与系统快捷键冲突。

2、同一产品类似功能要界面风格一致

同一产品的类似功能界面风格保持一致，既规范产品，又减少用户混淆度，提高用户体验。

例:

直播录制过程中关闭标签页后弹出的提示框，都是在询问用户是否关闭，但是前面的 icon 一个是叹号，一个是问号。



3、借鉴竞品类似功能逻辑

对于类似功能，可以对比竞品，如果竞品优于我们，我们可以借鉴或与其保持一致。

例:

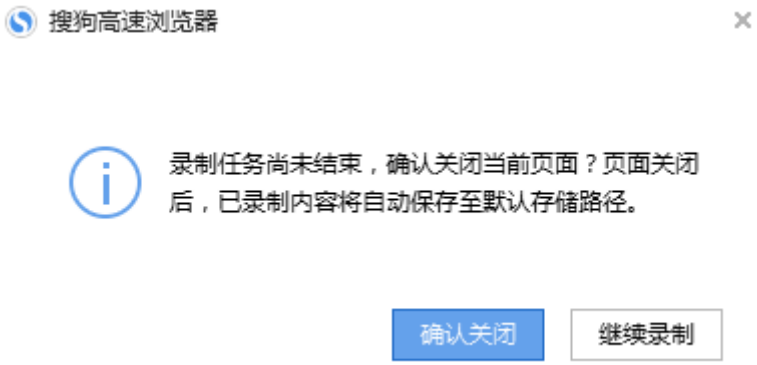
之前我的最爱搜索框中没有热词显示，对比竞品发现，他们类似的搜索框中有显示热词，可以增加用户对实事的关注，提高用户体验，所以在我的最爱的搜索框中引入了热词。

4、文案简洁通俗易懂

在产品的需求文档中，可能某些文案太长或者难于理解，都可以向产品提出建议。

例:

直播录制过程中关闭标签页后弹出的提示框，文案就很长，可以向产品提出建议。



九、备注

隐性需求：没有被明确规定，但是有可能或不应该拥有的功能或特性。

测试人生之谁的青春不迷茫

◆作者：谢 鹏

匆匆一年又将过去，除了年龄上增长了一岁；我还在想在一年里我成长了多少。

不变的其实是我们一直在变化。在测试的职场中，我已经忘了有多少成功和失败，或许这些都不重要。重要的是我现在的心态，总结起来也就两个字：坚持。

我不曾一次想放弃测试这条路；在公司里的整个技术团队中，测试的团队是永远不可能和开发成比例的，有的还是一个测试负责整个公司的项目，其实我就是这样。在每次需求提出到产品原型再到开发，测试几乎没有能够完整的了解需求的走向和变更，到最后开发完成，拿给了你与产品效果图完全不同的产品给你测时，你发现之前的测试用例完全白写了，接下来的测试就根据自己的经验来吧；再或者你搭建起一个缺陷平台起来后，发现一个月之后再也没人用了，领导不重视，流程不完整，缺陷交流繁杂。但是说到底，一个公司还是不能缺少测试，就算它的流程在烂，没有测试的产品终究没有好的用户体验，没有好的用户体验终究会导致公司倒闭。

自从来到新的公司，我就一直在做 APP 的测试。这是我之前没接触的模块，之前做的都偏向 pc 端测试、web 测试。我自以为做 APP 测试能学到更多，我渐渐的发现一个小的创业公司根本没有所谓的软件开发流程。我尝试改变，我搭建缺陷管理平台，用了两个星期就没人用了，领导不重视，没有流程监督，就此作罢。但我没放弃，我在想做 APP 如何能做好，如何实现自动化测试，如何对 APP 进行性能测试。通过查看论坛、网页；安装各种工具，不断去发现新的测试途径，慢慢的发现这些自动化工具需要编程语言的支持，需要编写脚本代码和对工具的灵活使用；我作为测试，编程能力是薄弱的；我想尽力去弥补自己的短板，在网上找公开课看，边看边学，想通过编写很小的程序来发现乐趣；我发现我不能坚持下来，我想放弃，我想每天打打酱油多爽，何必搞得自己那么累。是呀，我到现在都在不停的心里斗争中，但是我只要有那么点想学习的热情，我还是会去看看，还是想提升自己，还是不能安心的打酱油，还是想做出点有用的东西。我

有去书店买过《谁的青春不迷茫》这本书，说到底这本书其实就是作者成长的自述。看完我就觉得书名是最有力的，我无力反驳，甚至可以说迷茫伴随了一生，但在迷茫中，必须有自己的坚持；虽不见远方，亦路在脚下。

当然一个人的测试，负责整个公司的项目也有很多。网站测试，app 测试，微信测试...等等。没有那么多人测试，我却可以借助工具。就比如手机的兼容性测试，一款 app 需要兼容上千款手机，我如何去实现；这可以用到云端测试工具，国内有很多这种平台，通过上传 APK，检测漏洞和机型兼容性，能即时给出测试结果，一目了然。再说到网页测试，可以通过 webpage 测试平台，通过输入 url，图形化展示测试结果，方便迅速的看到网页错误。当然还有很多工具，不胜枚举。

说到底，坚持还是会带来收获的，只是有的时候表现的不是那么明显。当我在一条路上坚持不断的探索时，你会发现其实你的努力没有白费，即使你没看到任何结果，但你却拥有了一颗坚定的心。

如何处理大数据性能问题？

◆ 译 者：yonger

下一代开源大数据框架如 spark 和 Flink 的目标是达到大规模并行处理数据库如 Netezza 那样的性能和良好的用户体验

由于所有主流的大数据框架(Hadoop，spark，Flink)使用 JVM，所有他们都有着两个 JVM 限制:

1、Java 对象消耗的内存远大于对象中所有属性的大小

2、Major GC 的停顿会使得性能下降，尤其是当你尝试缓存更多的数据(Java 对象)以便后期对数据进行纯内存处理

所有 Flink，Spark，Hadoop 为高性能引入的工程技术主要目的就是为了尽可能的减少这两方面的影响.本文将为你展示这些技术。

在所有性能改进的之外，Alexey Grishchenko 所写的如下 blog 给了我们一个清晰的提醒，虽然基于 Java 的大数据框架在性能方面已经有很大提升，但是要追上大规模并行处理系统的性能还有很长一段路要走

性能的 80/20 原则

不是所有操作所需要的开销都是一样的，极少数的操作产生了很高的性能开销，许多大数据框架常用操作组合是:

1、压缩 vs 不压缩–这直接影响到数据被读出和写入的速率，不同的压缩方式/算法消耗的 cpu 资源差异很大

2、排序-大数据处理基本都会有“reduce”操作，而排序在 map-reduce 处理方式中则是

“reduce”阶段的关键。由于“join”操作会依赖于“reduce”，所以对 key 的排序也会关系到“join”的操作上

3、清洗-“reduce”阶段第二个最重要的操作就是清洗了，清洗操作也是关联操作的关键

4、关联-只有 map 阶段的 job 通常有很好的性能.但当有 reduce 阶段的时候性能会变成一个需要考虑的重要因素。一个 reduce 操作包含了关联，在只有 map 阶段的任务中可以使用广播关联，但广播关联通常在关联的一端数据量足够小的时候才用，这个很小的数据集被广播到所有节点，由一个 hashmap 的数据结构来维护同时由关联键来索引。另外一端大一点的数据集通过使用关联键在这个内存中的 hashmap 中查找相应的记录。当只有 map 阶段时，即使两端数据集都比较大也可以按照关联键来排序，Pig 框架就是充分利用了这一点，被称为“merge-joins”技术，它会把两边的数据集都在 lock-step 中遍历，然后在 map 阶段中关联起来(这里实际上有一个预先索引的过程)。当然有关联键进行自然排序对数据集是很少见的，在关联大数据集的时候 reduce 操作还是必不可少的。

搞定上面的这些问题是使得大数据处理效率变得更快的关键。

性能约束和 JVM 特征

就像前面提到的，Java 在性能方面引入的两个约束因素:

1、Java 对象的存储开销不容小觑

2、使用大量生存周期很短的对象会使得吞吐量性能下降，因为大量的时间会花在 Major GC 上

只在 JVM 堆栈上操作处理大数据很容易导致内存溢出错误，这会使得 JVM 进程被杀死退出(不像其他错误可以被处理).这是大数据应用中最常见的一个性能问题.

比如，在一个有许多记录但是只有少数键值对能够发生关联的不均衡数据集中，当所有的键值都保存在内存中时 reduce 操作会失败

从索引技术可以避免在 reduce 阶段时把所有的键值放到内存中，图像应用就是属于这类应用中的一种.

这就是大数据框架在使用内存时小心谨慎的重要原因了，比如磁盘溢出技术.这虽然会影响性能，但是会使得框架足够健壮来处理不同的负载.

比如，Pig 框架会把数据保存在磁盘上，DataBag 接口的实现及其广泛的应用在 Pig 中，当 tuples 的数量达到一个预先定义的值时会把数据刷到磁盘上(默认是 20000)

JVM 存储开销

在 Java 堆栈中一个典型的对象存储由如下组成-

1.对象头-只有几个字节用作维护信息.Hotspot 版本的 VM 使用 8 个字节来表示一个普通对象的头，另外再使用 4 个字节来表示数组对象的长度

2.原始类型-原始类型也需要字节来存储.比如 boolean 类型需要 1 个字节，char 需要两个字节，int 需要 4 个。

3.引用类型存储-引用类型占用 4 个字节

4.空占位符-每个对象存放时总字节数是 8 的倍数，所有这中间总是有空的占位符

如图 1 所示，一个 Java 类对象的实例包含了一个原始类型 boolean 实例所占用的 8 个字节用作头,另外一个字节表示 boolean 实例本身,剩余 7 个字节就会被空占位符填满。

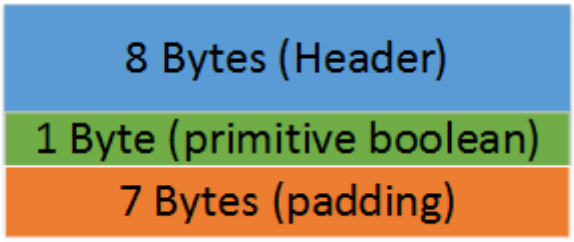


图 1:一个只有一个 boolean 实例变量的 java 类在内存中的占用情况

这个例子证实了一个普通 java 对象很大的存储开销，并且随着如 hashmap 和 list 这样复杂对象变得非常大

JVM 存储开销影响序列化和反序列化

在大数据范围内发生如下情形时你肯定需要序列化和反序列化:

1、Mapper 输出结果到本地磁盘

2、Mappers 从本地磁盘读出数据来做 map 端的排序

3、Reducers 从 mapper 节点上获取排序和清洗好的结果

在上面的每种情形中，实际的数据量都比原始字节表示的要多很多

Java 对象的存储开销影响到内存如何有效的使用分级缓存。即使所有你想要的数据在内存中，L1/L2/L3 级别的缓存也不可能大到能装下这些所有的数据，这也会使得缓存在过渡的交换之中从而降低性能。

JVM 存储开销影响 GC

JavaGC 最关键的是要更多的对象在年轻代时就被收集. Java 堆栈被划分为两个区域，年轻代和年老代。一般来说年轻代占用整个堆栈空间的 20%，而年老代占用了剩余的 80%。当一个对象第一次被创建时，他们被分配在年轻代，当年轻代满了的时候，minorGC 就会执行。在这个过程中不再被引用的对象就被清除掉，剩余的还在被引用的就移到年老代。当年老代被填满时，major GC 最终就会发生。Major 收集算法会遍历在图中的每个对象，从老年代中回收内存。注意这只是非常简单的描述了 GC 的过程，更加详细的了解可以上官方网站查找。

JVM 的这些额外开销降低了 OOM 的可能性，但使得你真正可用的内存空间减少。

大数据框架一直朝着全内存存储和计算的方向进行。 Map-Reduce 过去一直主要是 I/O 资源受限型的做法。其早期实现的迭代算法使得在每一次迭代都需要从磁盘文件系统中读取结果进行下次迭代，并且还要把结果写回磁盘上。这种做法使得整个 MR 的任务运行非常慢。

Spark 提出以数据能被以分布式方式缓存在内存中的 RDD 概念，这种方式没有了过渡的 I/O 开销，在启动和停止任务的开销方面也更小。然而，这种折中把风险转移到了 GC 上，原因是，当 Spark 使用磁盘溢出技术来控制对内存的过渡使用时，用户应用程序能利用的内存也越来越少了

在 Java 中序列化方案的演变

Java 的序列化方案一直在进化，默认的 java.io.Serializable 性能消耗非常高，因为每个实例都存储了类的 metadata 以及引用到的类型的 meta-data 信息。这种方案只适合当你想要序列化字节里面包含所有信息且这些都会反序列化为一个 Java 实例的情况。

Java.io.Externalizable 接口已经能够处理解决部分问题了，当使用这个接口时，程序员可以控制类中的属性如何序列化和反序列化，因此类属性的元信息不再需要存放到序列化字节流中。但是当 java.io.Externaizable 接口需要调用包含没有参数的构造函数类反序列化时，类的这些元信息还是需要存放到序列化字节流中的。

Java.io.Serializable 和 java.io.Externaizable 都有同样的限制，对每一个反序列化的实例都会创建一个单独的 java 对象，这就意味着当有非常多的实例反序列化时就会生成大量的对象，这增加了 GC 的压力。

Kryo 类库改进了这些问题。Spark 支持默认的 Java 序列化方式，但它推荐你使用 Kryo 方案以提升性能。Kryo 通过如下的方式改建了以上的两个问题:

首先，Kryo 提供了一种选择来存储整型索引以代表类被序列化，因为一个完整的类名称包含了包名称，这在序列化一个类的时候严重的降低了性能。当然缺点是，你必须舍弃可移植性，因为当你反序列化一个实例时，你必须使用同一个整型索引来指向这个类。在实际应用中，这确实是个问题。

第二，Kryo 支持在反序列化过程中池化实例，意思就是，即使你有数以万计的实例在某个时刻被反序列化也只有非常少的实例占用内存

然而，这个方案仍然会存储每一个实例，如果一个流中只包含具体类的实例，这就可以被进一步改进， 这个类的元数据需要在流的开始被存放一次，后面跟着这个流中所有实例的序列化过的属性字节流。在反序列化的过程中，只有一个实例需要被创建，字节流需要被一次读入到一个实例。在处理完当前反序列化实例后，下一个实例字节流就会被读入在反序列化一开始就创建的那个相同实例中。 Hadoop org.apache.hadoop.io.Writable 就是使用这个方法，因为这是大数据处理模式中的常规方法，同样也被 Spark 和 Flink 引入。

当你使用最后这种方案时，需要强调的是，你必须使用具体的类. 多态在大数据不能很好的被支持。如果一个实例中的类只有在运行时才能确定，这个类的元数据需要被包含在序列化字节流中。 这个解释应该很清晰的指出了为什么 java.io.Serializable 和 java.io.Externalizable 如此臃肿了，因为他们都支持多态，这也是为什么 setMapperOutputKeyClass，setMapperOutputValueClass，setOutputKeyClass 和 setOutputValueClass 方法在使用 Hadoop MapReduce 的任务实例时需要被精确调用的主要原因。这些调用在程序员看来非常烦人，但是这是实现更高的性能的必要手段，这些调用允许实例在没有概要定义时被序列化，图 2 展示了不同方案在存储开销和 GC 性能方面的影响

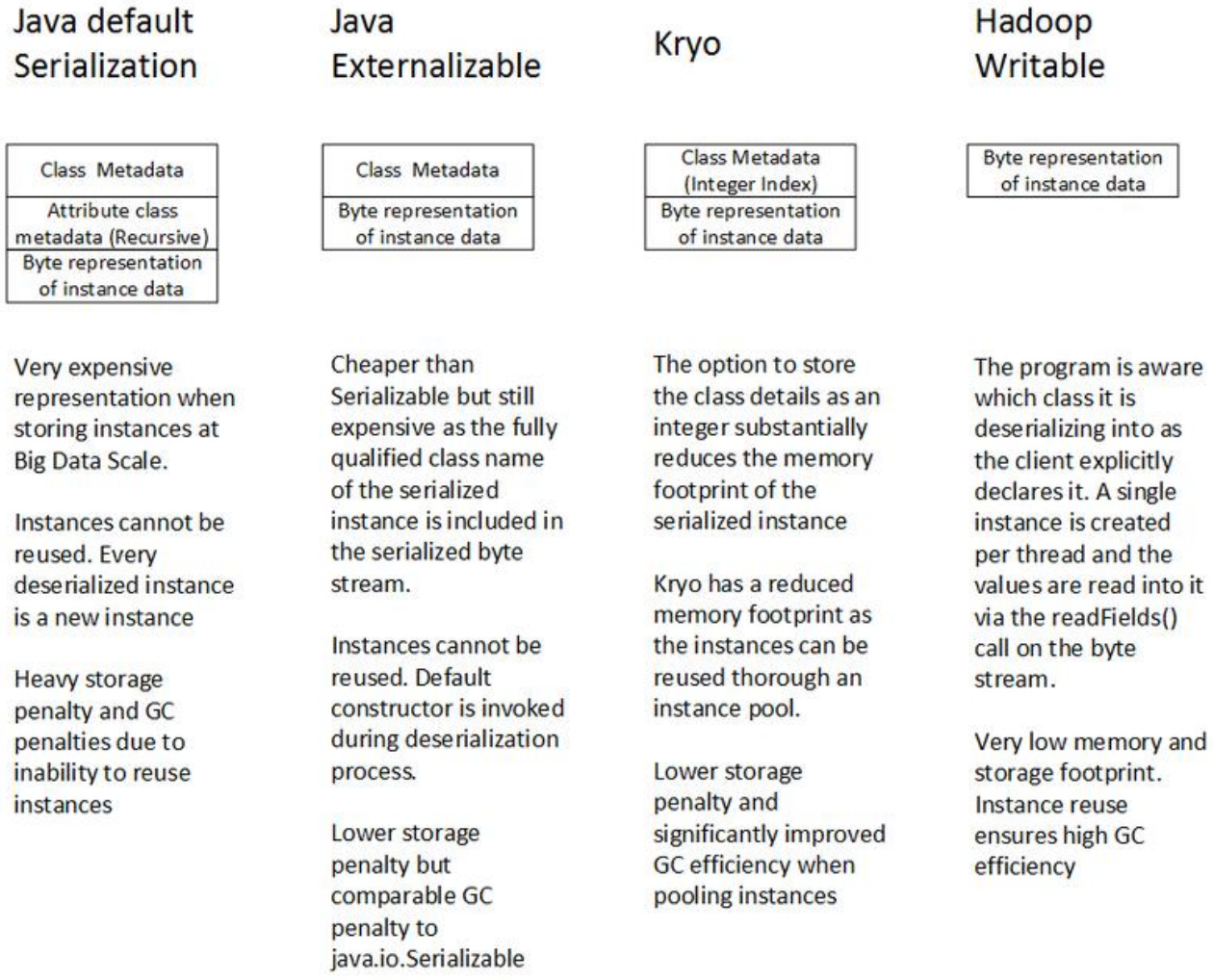


图 2: Java 序列化方案的进化

高性能工程-框架如何实现

一个高性能的分布式平台包含如下的技术:

1、特殊的序列化方案以减少序列化对象占用的内存空间，同样使得内存的分级缓存利用更高效。新框架(Spark，Flink)由于这个目的积极使用 sun.misc.Unsafe 程序包

2、接管内存-最通用的技术是分配和释放大的内存块。首先，这避免了堆栈碎片化，从而更高效的使用内存。第二，它能减少对象进入老年代使得 Major GC 能更快的完成。Hbase 就是利用这种技术，Flink 甚至通过尽量避免 major GC 来把这项技术带入一个更高的层面。

3、通过最大化使用多级缓存更高效的利用内存-举个例子，排序操作只需要排序的属性，如果整个对象都加载到内存会很有可能导致缓存被频繁的清除，这使得 CPU 变为瓶

颈，因为 CPU 会需要更多的时间片来等待数据从主内存中读出而不是高速缓存。如果我们只加载需要的后面操作中使用到的必要的属性，这就会提高 CPU 的利用率。大数据里面的一个常规操作是“比较两个实例”(在排序阶段中需要的，也是关联 Reduce 中最关键的操作)，如果这种比较能通过二进制数据来实现的话，就不需要把字节流序列化成一个单独的实例。所有的大数据框架基本都是这样做的，最常用的 Hadoop 数据类型 org.apache.hadoop.io.Text 就是通过其父类 org.apache.hadoop.io.BinaryComparable 来利用这个技术的。

4、利用堆外内存- 通过 Java NIO 或者 基于 sun.misc.Unsafe 包自定义实现来利用 Java 堆栈之外的内存。目的是提供一个接口出来用以访问不被 Java 堆栈所有管理的内存，这种内存是不需要 GC 的，因此会是的 Java 堆内的 GC 运行的非常快。缺点是，编程者必须自己来管理这些内存，HBase 为块缓存提供了堆外内存的选项，Spark 在清洗和块缓存过程中也利用了堆外内存。Flink 也正在实现堆外内存的选项来管理内存子系统。

框架本身特定的序列化方案

前面我们已经简单的讨论过这个问题，现在让我们讨论下更细节地方。

在大数据范畴中需要专门的序列化技术是因为流式处理的数据通常都有同样的数据类型，我们可以使用一个自己实现的序列化方案来达成下面的优化-

1、每个数据集只需要保存一个次额外的 schema(不是多态)

2、为每个元组类型保存字节流，利用偏移量来指向实例属性(看如下 Spark 的例子)

通常当你需要读取一个实例的属性时，我们需要先反序列化整个实例然后写一个 getXXX 的调用来获取实例的属性. 自定义的序列化方案允许我们仅抽取出需要的属性的字节，然后反序列化. 当一个实例拥有许多属性时，这种方法允许我们抽取只想要的那个，它通过重要性排序减少很多额外实例对象的生成，从而更高效的利用高速多级缓存。因为排序，清洗，关联操作就是这种只需要少量属性的操作过程，这使得内存和 CPU 的利用率非常高。

Hadoop 方案-org.apache.hadoop.io.Writable

这个话题在前面已经讲过，可以参考图 2

Spark 方案-Project Tungsten

Spark 鼓励你在 Java 序列化的时候使用 Kryo，它对 Spark 的数据类型有着非常成熟的面向对象模型支持，当然序列化后也会占用更多的内存(Kryo 能够使得数据类型转化为一个整型，但是每个实例需要额外的 4 个字节的开销来关联这个整型)

Project Tungsten 最终认为上面这个方法做的太纯粹了，为了性能必须牺牲一定的这种纯粹性。

如下的展示提供了一个例子来了解 Project Tungsten 的全貌:

最关键的主意是利用了在 Spark 中一个具有如下特征的典型数据集-

1、数据集通常使用唯一的 schema 来重组(如果你想象这是一个基于关系型数据库的结果集，这就更加明显了)

2、为每个元组存储类型信息非常浪费，为元组的每个部分存储数据类型信息当然更加浪费，最有效的方式是只存储 schema 一次，使用多次

在这个 slide 中最关键的点是:

通用化是有成本的，所以我们应该使用语义和概要来利用特性代替

Tungsten 也通过自定义的序列化方案来提高清洗阶段的性能.序列化方案的效率比起通过网络传送对清洗阶段影响更大。

压缩自定义的序列化概要会通过减少字节在网络上传输的数量从而使得序列化过程更快来节省时间。可以通过如下 Blog 的代码生成片段获取更加详细的信息，那段代码展示了基于清洗自定义序列化实现比 Kryo 快两倍

如果你想要进一步提升你大数据应用的性能，你就需要充分了解利用你应用的特征(数据如何到达以及数据结构如何)

Flink 方案

Flink 设计了一套从更本上更高效的序列化方案.首先，跟 Spark 一样，Flink 也支持任意的 Java 和 Scala 类型，随着数据类型必须像 hadoop 一样实现特定接口，这是一个很大的进步，Flink 利用反射来确定用户自定义函数的返回类型，Scala 的程序是使用 Scala 编译器来分析的，Flink 利用了一个叫 TypeInformation 的自定义类来表示任意的数据类型，这个类支持如下的类型:

- 任何 Java 原生类型或者 java.lang.string

- 任何 Java 原生数组类型和 java.lang.String
- Hadoop Writable 接口的任何实现类
- 任意的 Flink 元组类型，Flink 元组是有着类型域的固定长度元组的 Java 表示
- 任意的 Scala CaseClass 类型(包含 Scala 元组类型)
- 任意的 POJO(Java or Scala)，比如，所有域都是 public 的对象，或者遵循通用命名习惯的可以通过 getter 和 setter 方法访问的
- 任何不能被另外一种类型确定的数据类型

上面的每种 TypeInformation 类型实现提供了一个自定义的序列化器，GenericTypeInfo 是 TypeInformation 后面的重排序和对 Kryo 的委托，更多的细节请阅读这段-Flink 是如何序列化对象的？

TypeInformation 也提供了一个在二进制格式数据类型比较方面更高效的自定义类型 TypeComparators，它通过把数据类型用作 key 以便快速比较和排序. TypeInformation 在 Flink 中是一个公共的 API，能够被用来扩展支持自定义的 TypeInformation，序列化器和比较器则在处理自定义类型上更高效.

TypeInformation 也集成 Flink 的自定义内存管理(这就是我们会在稍后说的 MemorySegment)，这利用了 JavaUnsafe 操作的高性能.

Flink 完美的结合一个高效可扩展的序列化机制，对自定义数据类型可扩展，并且集成了为支持尽量避免 majorGC，提高吞吐量而设计的更高效的内存管理模块.

接管内存管理

Major GC 是达到高性能最大的一个障碍，它跟一个通用技术-缓存，在实现高性能之路上是冲突的，缓存太多会导致更频繁的 major GC，从而影响吞吐量. 在最差的情况下它也会造成内存溢出. 过渡使用内存也会导致更多的内存碎片，从而导致出现总的可用内存超过程序请求的内存时发生的可拍的内存溢出现象

HBase 中的内存管理

接管内存管理对于 Spark 或者 Flink 来说不是个新鲜的事情，HBase 同样也已经做的很好，这段将说说在大数据应用中早期的内存管理系统是怎么样的，同时也会讲述 Spark 和 Flink 他们是如何为了达到更高的性能对内存管理技术进行改进的.

HBase 使用 Memstore 作为它的内存数据存放，然后定期把这些数据刷到磁盘，在 HBase MemStore 中是以 2MB 的内存块作为分配单元，这项技术在由 Cloudera 发布的系列文章- 在 Apache HBase MemStore 中如何避免 Full GCs-本地缓存 中详细的讲解了，这些文章对 Java 垃圾回收方面讲述的相当详细，以及 HBase 是如何在处理由 Java 垃圾回收所产生的这些限制的。

HBase 所使用的方法可以总结如下:

1) HBase 是一个 key-value 的存储方式，它把 key-value 对作为字节数组方式存储
2) 很多的 key-value 对都是很小的，这主要是为了避免每个 key-value 对都会创建单独的字节数组.
3) 当一个 key-value 对第一次存储到 HBase 中，它就是存放在一个叫 MemStore 中的，这个 MemStore 对它所使用的内存有一个预算阀值，当这个阀值超过了的时候，数据会从 Memstore 刷到磁盘中，更多细节请阅读这个 blog
4) 在这个阀值之内，Memstore 会以 2MB 的块分配内存以存储内存中的 key-value 对，当一个 2MB 的块存满时就会分配一个新的块以继续存储新的 key-value 对这项技术对内存的使用很高效，由于以 2MB 分配为单元，所产生的空内存碎片也会少很多。对于 GC 来说，比起更小的内存块，2MB 的内存块在垃圾回收的时候也会更快，另外 2MB 的大内存块在回收后也更容易被重新分配，一个 2000 个单独的 1KB 的片段在内存中比分配一个 2MB 的片段要困难很多.

Flink 中的内存管理

Flink 做得跟 HBase 差不多，只不过范围更大而已，Flink 把 Java 堆栈内存划分为 3 个分区-

- 网络缓存-默认只有 2048 个字节，Flink 在启动阶段分配了 32KB 缓存作为启动缓存，而且这个缓存还可以通过参数 taskmanager.network.numberOfBuffers 来调整
- 内存管理池-这是一个由 MemoryManager 实例管理的 32KB 缓存大集合. 每个缓存都是 MemorySegment 的一个实例，MemoryManager 面向很多算法池化 MemorySegments，使用单实例 MemoryManager 来分配和释放 MemorySegment，在内存管理池和网络缓存中将近 70%的堆栈内存空间是共享的

- 用户代码的内存-剩余的内存主要是留给用户代码的，从 GC 的角度看这一部分主要是年轻代区域，意味着主要用来存放用户代码所产生的短生命周期的实例.

上面这些方法的好处是-

1) 短生命周期实例放在最后的片段，这样收集会非常快

2) 算法申请的内存是以 32KB 为单位的片段，释放的时候也是 32KB，但是从 GC 的角度看，内存从来不会释放，MemoryManager 实例仍然会持有应用分配和释放的 MemorySegment 实例的引用

3) 算法对他们使用的内存片段个数有严格的预算阀值，如果一个算法超过它的这个预算阀值，它需要把未使用到的片段存放到磁盘，并且在需要的时候再把这些数据片段从磁盘读出来. 如果这个算法需要比它允许的预算更多的内存，它就需要把一些片段放到磁盘上并且重用 MemorySegment

我们再一次看到“为了高性能需要特征性而牺牲通用性”，为了达到更好的吞吐量和性能我们必须使得我们的代码更复杂。图 3 表明了 Flink 方式的内存管理

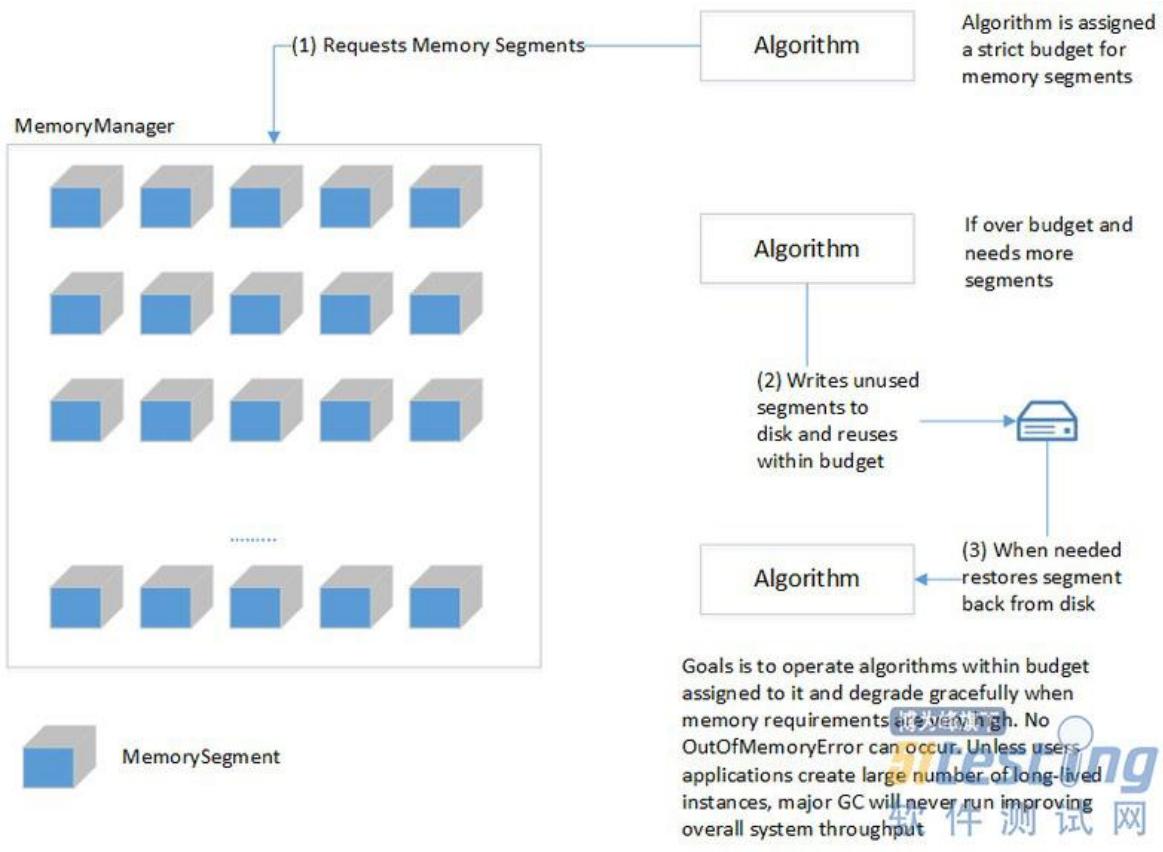


图 3: Flink 内存管理

算法都是以 32KB 的片段申请和释放内存，但是从 GC 的角度看，内存从来不会被真正释放，很多用户自定义的短生命周期对象在 major GC 的时候就被回收，只要应用程序不尝试为他们自己创建缓存，在老年代中的内存就不会变，major GC 也不会被触发.这极大的提升了 Flink 系统的整体吞吐量. 关键的地方在于-一个编程者不应该尝试为长生命周期的实例自己创建缓存，这会使得年老代会被填满，从而触发 majorGC，甚至内存溢出，如果编程者必须要这么做的话，它需要使用 MemoryMnager 来自行管理内存。

使用分级缓存实现更高性能

随着网络速度越来越快，CPU 开始在大数据范畴内慢慢成为瓶颈，从 L1/L2/L3 高速缓存中读取数比起直接从内存中要快，已经分析深入分析程序执行，我们发现在从主内存中读取数据时 CPU 等了相当长的一段时间比起整个时间来说，如果这些数据从分级高速缓存中读取的话，这个等待时间要大大缩短，整个算法的性能也会有很大的提升

上面，我们讨论过直接使用二进制数据的自定义序列化方案来尽量避免对整个实例反序列化，这种自定义的序列化方案提供了一个实例的字节是如何存放在内存中的细颗粒度控制，这就允许直接指向一个实例的特性属性而不需要在内存中反序列化整个实例，比如排序操作，它就让算法仅以 sort-key+pointer 的方式来指向实例，从根本上提升了 CPU 的效率

Flink 方案

考虑 Flink 是如何执行实例的排序，这是一个非常常见的操作，假设 Flink 需要一个只有由单个整型属性组成的数据集进行排序，Flink 在为了实现更高性能方面做了这样一些事情:

1、数据集被分成两个内存片段存放

代表所有数据的字节存放在其中一个内存段中，许多的数据集有相同的概要，因此 Flink 像”框架特有的序列化方案”那段那样对存储专门进行了优化.

另外一个内存段用来存放数据集中每个元素的两个属性-指向数据集元素的指针按照排序关键字来排序

2、排序现在可以按照指针和排序属性和组合来执行了，这样由于不是所有的数据都需要完全排序，充分利用了内存中的高速缓存，效率相当快。

上面这些方法的其他优点-

Flink 可以直接在二进制数据上进行操作，如果实在一个字节数组上操作就完全不需要反序列化排序的属性，记得 org.apache.hadoop.ioText 两个实例使用仅使用一个字节数组

对于由内存片段维护的数据集元素也不需要反序列化，这个数据集可以安全的被存放在磁盘上.

Spark 方案-Project Tungsten

Spark 在排序算法方面跟 Flink 基本是一样的，更多细节可以看这个 blog

利用堆外内存

利用堆外内存是让 Java 性能最大化的最后一种方法。Java 提供了这样的库，可以让程序非常明了的管理内存。托管堆是 JAVA 编程语言的特征，它让程序员在执行内存管理任务的时候可以减少复杂度及避免容易犯错的地方。然而，它也有缺点。首先，作为一个程序员，你无法控制垃圾回收的运行频率以及运行多久。其次，如果你不加鉴别地缓存，你就会碰到“OutOfMemoryError”。就跟之前谈到的，尽管机器上由于内存碎片还有一些可用的内存空间，但托管堆也会占尽空间。

没人逼迫你只能使用 Java 的内存管理。通过利用 java.nio 或者 sun.misc.misc.Unsafe 程序包，程序员可以进入到非托管内存。缺点是：程序员需要负责分配和释放内存。优点是：内存管理不会收到垃圾回收进程的限制，大大减少了迅速返回的垃圾回收上的负载。平衡点就是：你需要采用更高的更复杂开发程序达到预期或者更高的吞吐量。

虽然在构建自定义应用时不常使用，但高性能计算框架一直并会长久使用堆外内存。Terracotta 在 BigMemory 产品上利用非托管内存。BigMemory 可以使用缓存达到 1 个 G 的水平。

OpenHFT 提供了 Chronicle Library，它可以支持高性能，低延迟及被动的处理框架。这个框架大量使用堆外内存。

HBase 对于 Block Cache 支持堆外内存选项.

Flink 正在开发 MemoryManager，它将利用堆外内存。更多细节详见这里:

Spark 通过 Tachyon 抽样缓存支持堆外存储（实验性）。

我开发一个叫 LargeCollections 的库，它采用 LevelDB 去支持大量的堆外 Hashmap。

这个项目在 LevelDB C++实现上也使用了 JNI 包装。

最终所有框架的目标是实现 MPP 的性能水平。堆的使用让 JAVA 程序员更进一步的使用 JAVA 像 C 编程语言。

将这些经验应用到你自己的大数据应用中

尽管以上谈到的工程方案可能看上去很独特，但这些经验可以运用到你自己的大数据应用中。我所应用的技术如下:

1、开发你自己的简洁的序列化方案。这个减少了任务之间的输入/输出。

2、尽可能使用二进制数据。尽可能避免从二进制到对象，然后又返回到二进制。CPU 确实是大数据的瓶颈。每次你开始 org.apache.hadoop.io.Text 的时候，你已经在做了。找到机会采用 org.apache.hadoop.io.Text 用到的技术来比较你自己的键值类。

3、总是处理最少的数据。假如只有一小部分数据是在你处理的那部分中，而大部分数据不是，你需要在最后执行一个开销很大的关联，但这样你不用通过网络或者反序列大的对象发送太多不必要的数据到内存。

4、不要回避使用堆外内存。有时候使用它可以帮助高性能技术如 Map-Side 的加入。如果你要关联的数据集一端数据量不是很大，就可以把这部分数据存放在堆外内存中，这样就可以通过关联键来进行查找，而且在 Mapper 阶节点上没有很大的 Reduce 操作的开销(需要清洗和排序)，这是在本文谈到的广播关联的变化。我已经开发了 LargeCollections 提供简单的界面去使用堆外内存。它剔除了在使用 Usafe 或者 NIO 库时的复杂性，给程序员提供了他们熟悉的简单的 Hashmap 的抽象概念。它支持像 org.apache.hadoop.io.Writable types 高效的数据类型。它也支持开发定制的 Serializer/De-Serializer (SerDe)去支持自定义序列化方案。因为它使用的是 Java 的堆外内存，所以大大的减少了对 Java 内存管理过程的压力。

后记

写此文的目的是能够收集在大数据框架下使用的性能工程技术的演变从而实现 C 级水平的性能。

测试公主之为人鱼公主找到真爱

◆作者：测试小公主

当 APP 开发出来后，Lisiya 就和天蝎星的居民们一起愉快地连接 wifi，并使用 wifi 愉快地上网和发送信息。

Lisiya 在天蝎星上使用 APP 已经很熟练了，同时 APP 的联网功能也已经日趋稳定。在和天蝎星上的国民一起交流的时候，Lisiya 也为 APP 研制了各种新的需求，使得 APP 更加易于使用，功能多样。

由于水星宝宝们给天平星带来了和平和欢乐，天平星的所有国民都非常感激。

不久，他们便收到了来自天平星和天蝎星的来信，信中附了一份绝密的，宇宙飞船操作手册！

并说让他们在一个月内将这个操作手册背熟，然后就能驾驶宇宙飞船了！而当他们学会了驾驶宇宙飞船之后，天平星的国王和王后邀请 Lisiya 参观天平星，作为奖赏。

这时候，Lucherian 来了，他说，晚上带她们去看看宇宙飞船！

然后 Lucherian 将宇宙飞船的操作要领都全部教了一遍给 Lisiya，她马上都能倒背如流了。

操作手册的主要内容有:

Test case:

1: 分别使用宇宙飞船的计算机和手机，连接宇宙星际 wifi

2: 确保宇宙飞船和手机都是有电的

3: 拥有足够的上网时长

4: 在宇宙飞船的计算机中认证上网，在手机中认证上网

5: 在宇宙飞船的计算机中的上网页面点击-“去往天平星”-”视频“

6: 在手机的上网页面点击-“去往天平星”-”视频“

7: 在宇宙飞船上点击“去往天平星”按钮，宇宙飞船就可以自动往天平星的方向前进

(Lisiya 写着宇宙飞船的测试用例，运用等价类，边界值，流程图的方法，进行分析和写作，并在一边研究，一边还和 Christina 进行交流，确保在遇到各种问题的时候，都有具体的应对方案)

同理，点击“去往天蝎星”按钮，宇宙飞船就可以自动往天蝎星的方向前进

手机不仅拥有和宇宙飞船的计算机一样的功能，更能方便携带。

Lisiya 按照测试用例所写的，在宇宙飞船中，发现显示屏上面的人竟然是外婆 Christina，外婆会一路陪伴和保护 Lisiya 的，这是个秘密，连 Victoria 女王都不知道！ 因为 Victoria 并不欢迎 Lisiya，她如果知道 Lisiya 旅行的途中，一直都由 Christina 保护着，她会生气的。

Christina: 亲爱的孩子，你好吗？

Lisiya: 我非常好，外婆！

Lisiya: 外婆，您在显示屏中看起来好年轻，和我妈妈长得非常的像！

Lisiya: 真希望，外公能进来看到你的样子啊！

Christina: 我还没做好心理准备让他看呢！我们分别太久了。

Christina: 但是我们一直通信。这就够了。

Christina: 这次旅行是特别为你设计安排的，你要感谢天平星，当然天平星更要感谢你!

一路上他们就这么聊着聊着，都忘了跟 liyiya 和 Lucherian 说道别了，

一路上非常的平稳，安全。

如果遇到任何情况，外婆都及时指点，很快就离开了大气层，到达了宇宙中！

浩瀚的宇宙！美丽的宇宙！让人流连忘返，沉醉其中，一路上他们遇到好多颗美丽的行星，好像上面都是有生命的，多么让人遐想连篇……

虽然去的路上他们都很安全，但是回来的时候， 飞船经历了强烈的震荡和牵引，这出自于一颗路过的具有强烈引力的星球，仿佛在这颗星球上面有着某种异于和谐的磁场，

那是混乱无章的磁场，是一种令人不安的磁场。

当 Lisiya 醒来的时候，她躺在了在百花丛中，阳光明媚，这花非常的大，因为花非常的大，她才能非常安全的着陆，就像是落在了弹簧垫子上一般。

虽然宇宙飞船不知了踪影，但是幸好她的手机还在身边。

查看了手机，Lisiya 发现幸好在这颗星球上，是有星际宇宙 wifi 的。

并且手机还有电，依靠连上的星际宇宙 wifi，以及手机中的指引，她不断地走着，

终于，Lisiya 看到前面有一条河，河水非常纯净， 在河边，她发现了破损了的宇宙飞船。

Lisiya 利用宇宙飞船的电给手机充满了电。

她记得背过的操作手册中，写到如何联系 Lucherian 的方法，有一个秘密按钮的，她找到了这个秘密按钮，于是，不一会，Lucherian 的影像就出现在她面前，他们都激动极了……

Lucherian 说，他们一时不会那么快有宇宙飞船飞过来的，至少要等到 24 小时，才能有宇宙飞船飞过来救她……

这时候 Lisiya 看到在河中有美人鱼在唱歌。原来这美人鱼是当初海的女儿的投胎的结果，她现在是她的水中的女王了，掌管着整个海域，她过着她认为开心的生活。他们生活的星球叫做--转世星。但是她还是想念她前世的那个王子，她是多么的爱他 ，即便是转世了，还是忘不了他！她知道他是地球人，但是她永远在这遥远的海域的水中，去不了地球。300 年过去了，他到底去了哪里？

她告诉 Lisiya 到了晚上她必须到海底过夜，因为这个星球的晚上是非常恐怖的。所以她让 Lisiya 坐到一个潜水艇中，然后下了海，顺便欣赏这美丽的水下世界！

人鱼公主在海底很好地招待了 Lisiya，Lisiya 感到无比的快乐。

人鱼公主告诉 Lisiya，转世星的夜晚之所以非常恐怖，是因为很多的妖魔鬼怪在夜晚都会出现，当这一刻到来的时候，这些妖魔鬼怪有些会散落到外太空中，在外太空中，如果他们遇上了一颗有人类的星球，比如地球，那么他们就会附着到地球上，将人类的邪恶力量增强。

人鱼公主告诉 Lisiya，正是因为这个因素，妖魔鬼怪才会源源不断的去往地球等星球

中，形成强大的恶势力，一旦白天到来，这些妖怪就会躲起来，到那些阴暗的地方，完全没有光照的地方，一旦有人路过那些暗处，他们照样会兴风做恨的。到了夜晚，这些妖魔鬼怪更是无法无天，残害无辜。

Lisiya 说：那我们该如何改变这种境况呢？难道就任由这些恶魔无法无天，伤天害理吗？

人鱼公主说，按理说，这些妖魔鬼怪是被严格的控制住的，他们会被地狱或者地牢所收纳，地狱中有着严格的制度，是不会让妖魔鬼怪出来作恶的。本来地狱 wifi 只有地狱的妖魔鬼怪可以使用，并且认证上网，且使用 Lisiya 他们设计的 app 进行账号管理，且每一个妖魔鬼怪只有一个账号。

可是 Lisiya 他们的 APP 不能识别这个账号是否就是本人的，以及一个人是否只有一个账号。

也就是说，一个恶魔可能拥有多个账号，而一个账号也可能被多个恶魔同时使用，这时候地狱 wifi 和地牢 wifi 就无法控制和管理这些恶魔了。因为他们成为了游魂。 Wifi 的管理机制彻底崩溃。

而恶魔级别低的应该使用地狱 wifi，级别高的使用地牢 wifi，而现在他们甚至可以使用宇宙星际 wifi（只有非常正义和善良的人才使用的 wifi），可见他们的破坏程度是如此之大。

此刻 Lisiya 知道了，一旦能确保哪些妖怪是地狱的，哪些是地牢的，并且能让他们每个妖怪都只有一个账号，每个账号同时只能使用在一个设备上，就可以很好的管束他们的行为，即使是在晚上，转世星也会很安全，同时，地球，宇宙和其他具有人类的星球也不会因此而成为妖魔聚集地。

于是 Lisiya 想了方法，每次登陆 wifi 都必须经过审核，每个妖魔登陆时都要有记录，等一系列的举措，同时还要狠仔细地统计妖魔鬼怪的总数，不能遗漏任何一个妖魔鬼怪，在 APP 中增加每天妖魔鬼怪需要学习的向善的课程。

24 小时之后新的宇宙飞船来解救 Lisiya 去了天蝎星，来到了测试团队中进行工作。

经过 Lisiya 团队的讨论，设定了新的需求，而需求是：

Lisiya 测试团队将 wifi 的使用者分为三个类型，他们分别是：

- 地狱恶魔
- 地牢恶魔
- 正义生命

同时也开发了三种不同类型的 wifi:

- 地狱 wifi
- 地牢 wifi
- 宇宙星际 WIFI

1) 给每一个连接的用户设置一个唯一的用户号码。
2) 用户连接 wifi 后进行面部识别，确保是本人。
3) 用户登陆时确定每个用户只能登陆在一个手机上，用两个手机同时登陆，则之前一个手机登陆无法上网。
4) 地狱恶魔只能连接地狱 wifi 并上网，地牢恶魔只能连接地牢 wifi 并上网，正义生命只能连接宇宙星际 WIFI 并上网，否则无法连接上网。
5) 每次登陆 wifi 需要输入密码，且密码有一定的复杂度，比如必须要有数字，英文和特殊符号。
6) 同一个生物使用宇宙身份证只能注册一个 wifi 账号。
7) 登陆密码输错多次将被锁定，解锁需要回答问题。
8) 登陆错误次数多的用户将进行统计和分析。

相应的测试工作紧张而忙碌的进行着并很快的完成了。

此刻，宇宙，地球，转世星，天蝎星得到了安宁。地狱 WIFI，地牢 WIFI，宇宙星际 WIFI 都有规律地使用着。

为了表达对于美人鱼家族的感谢，Lisiya 赠送给美人鱼家族每位美人鱼一个星际 WIFI 的账号，每位美人鱼 300 年的上网时长！

定时派遣天蝎星的专家去转世星视察 WIFI 的使用情况，而这个 WIFI 所连接的 APP 还可以连接转世星和天蝎星最最先进的科学技术！

这里还想说明一点，虽然转世星上面的美人鱼只是地球上美人鱼的灵魂，但是所连的 WIFI 并非地狱 WIFI，地牢 WIFI，而是宇宙星际 WIFI，这是因为这个美人鱼家族历来都是善良的群体，挽救过无数地球上船员的生命，自始至终对于地球有着敬仰和爱慕之情。

那种爱慕，自从 300 年前的小美人鱼那一代就有了，她们没有因此而成为人类，但是她们也没有伤害过任何一个地球人。

所以他们是不会去地狱的，更不会去地牢的，他们理应获得使用宇宙星际 WIFI 的权利。

Lisiya 深深的理解，人鱼公主非常想要再次去一次地球，去寻找那个王子，不论那个王子是否爱她，她都无怨无悔。

不出所料，人鱼公主非常愉快的答应去一次地球！去寻找那个当年她变为泡沫的地方。

“可是，我的腿还没有啊。。我怎么能离开这大海呢?” 人鱼公主说。

“明天让 Lucherian 和我的外婆给你想办法，今晚要好好休息!”Lisiya 说。

“好的”人鱼公主说。

于是 Lisiya 也好好的睡了一觉，因为白天实在是太累了!

白天到了，一切都恢复美好!

Lisiya 他们准备了一个大水缸放入了宇宙飞船，而人鱼公主就被放在这个大水缸里面。飞船安全的来到了地球。

地球上的男子铺天盖地，到底谁才是，小人鱼所救的王子转世?

对了，应该去你救王子的地方， 你还记得吗?

记得的!

于是他们在人鱼公主的指引下，飞去了那个地方。

小人鱼又跳入了大海……寻找她的记忆!

大海还是以前那个样子，只是小人鱼的家人已经消失了，小人鱼这么想的，

她继续往深处游着，游着……

突然她，竟然发现了那个皇官，还有她的画像！

当然，还有她的故事。

甚至，她的家人，后代！

300 年过去了，现在的人鱼们，已经完全是他们的后代，

他们认出了她！

她表明了来意，人鱼公主问，那个王子后来怎么样了，那个王子，他娶了新娘后，过得并不幸福，因为那不是真爱！

那后来呢？

后来他还是，过世了，埋葬在海边。

因为他总是梦到美人鱼。

“我想去看看他的墓。”

于是她游过去了。

但是她看到的竟然就是王子本人，

为什么呢？

因为，既然他结婚的时候，小人鱼必须死，那么，当他离婚的时候，他就必须死， 而且他再次见到小人鱼的一天 ，就会是他重生的日子了。

奇迹般的，小人鱼的鱼尾变成了两双完好的腿脚，而且这次一点都不疼了，而且她的舌头也都还在，她多么美丽！他们结婚了！在这个新世纪，结婚了。Lisiya 和 Liyiya 参加了他们的婚礼。如此的隆重和美好！包括王子当年的后代的后代都参加了，婚礼结束之后，接新娘和“王子”回到皇官，作为皇族的新成员，住在了皇官中。

从此小美人鱼成为了一个真正的人，而且还愉快地生活在了地球中。

参加完了地球上的人鱼公主的婚礼，Lisiya 他们立即乘坐宇宙飞船飞往转世星，将这个人鱼公主寻找到真爱的故事告诉转世星上面的人鱼们，她们把地球婚礼图片和故事传给了，人鱼世界的接班人。 另一条美丽的美人鱼。未来的人鱼海域掌管人，她是小人鱼的亲姐姐的灵魂。

随即举行了一个庄重的仪式，让这位新美人鱼领导人能够牢牢记住自己的使命。 而

这样她们的使命就完成了。

消失的人鱼和加固的防线

转世星的人鱼女王奉命管理好转世星的地狱和地牢，不让转世星有任何不安因素，一旦有不安因素，立即通知 Lisiya。

有一天，人鱼女王告诉 Lisiya，有 5 条幼年的美人鱼消失了，不知了去向。

同时有一条尊贵的美人鱼皇室成员报告说她的手机被偷了。

而消失的时间，正好是她手机失踪后的 5 分钟内。

经过研究 Lisiya 发现，当手机连接了星际 wifi 并成功上网后，如果同一个手机被地牢恶魔使用，地牢恶魔还可以使用星际 wifi 成功上网 5 分钟，5 分钟后，地牢恶魔无法再次使用星际 wifi，只能使用地牢 wifi 上网。

一定是有一个地牢恶魔趁这 5 分钟的星际 wifi 的缓存时间,将 5 条美人鱼给带走的!

通过仔细的排查，他们发现，的确是这 5 条美人鱼被地牢恶魔带去了地牢，因为地牢恶魔对于人鱼家族的嫉妒，他们趁着这 5 分钟的缓存时间，打破了只能连接地牢 wifi 的限制，连上了星际 wifi，逃出了地牢，来到了美人鱼水域，将 5 条还在睡觉的人鱼带入了残忍的地牢中，并且对她们进行恶语相向，残忍虐待。

由于地牢水域非常凶险，Lisiya 找到了擅长游泳的测试工程师小暖暖，潜水将 5 条美人鱼救出，并让开发修复了 bug。

然后测试组对缓存问题进行了测试，确保任何 APP 的改变带来的缓存时间尽可能的小。

虽然 5 条小人鱼被找到了，但是她们得了抑郁症，解救她们唯一的方法就是让他们加入测试团队!

新的团队和新的任务建立了起来!

现在天蝎星测试团队的测试工程师一共有测试公主，测试暖暖和 5 条美人鱼。而测试暖暖的妻子是另外一个星球的公主，在另外一个测试团队中从事测试工作。

为了更好的帮助 5 条美人鱼进行测试工作，测试团队决定利用 5 天时间，为测试团队中的 7 个人进行测试培训。

Lisiya，小暖暖和 5 条美人鱼在一起讨论测试方法，同时也治疗 5 条美人鱼的抑郁症。

Lisiya 说，现在我们来讨论一下性能测试和功能测试理论吧！

“好的”，大家说。

Lisiya 问，平时我们测试时，要考虑哪些测试方法呢？

小暖暖说：如果是测试网页的话，要考虑 cookie 的测试，要测试 cookie 的保存信息是否正确，cookie 的有效期等等。

Lisiya 说，对，如果有多种设备，还需要测试兼容性测试，比如不同浏览器，不同设备，比如手机（考虑到常用手机，比如 android 的各种手机品牌，ios 的手机），电脑（windows，linux 等等），宇宙飞船的系统是否能支持所开发的软件。

小暖暖说，还有安全性测试，为了防止被刷，需要增加图形验证码，密码的规则增加，安全提问，登陆次数限制，某个任务每日限制次数，为输入框限制输入内容，防止 SQL 注入等等。

Lisiya 说，我们讨论的非常好，请小美人鱼们仔细记下笔记。

Lisiya 说，常用的性能指标有：

1、时间--响应时间

响应时间就像是真爱，真爱是不会随着时间的流逝，而有一丝一毫的改变的，响应时间的 x 轴是时间，就像是人所经过的时间，而 y 轴是响应时间，虽然响应时间会有一定的波动，但是真爱是不会因为时间的不断推进而有大幅度的变化，允许有一定的变化，但是绝对不能变为无穷大，否则那就不是真爱。

2、虚拟用户数量—响应时间

是不同虚拟用户数量下的响应时间曲线。

Lisiya 说，现在让我们来温习一遍有关性能测试数据的统计学方法吧！

1、平均值：

1）相对于整个数轴来分析时，平均的响应时间的值，可以看出响应时间的主要在什么数值上，但是它有个缺点，当最长的响应时间和最短的响应时间相差特别大的时候，这个平均值就显得没什么意义了，只有当最长的响应时间和最短的响应时间相差不大的

时候，它才会有意义，可以说明响应时间的主要数值。

相对于某个时间点上来说，平均响应时间是有意义的，说明的是某个时间点上所有并发用户的平均响应时间，如果在某个时间点上，平均响应时间特别长，说明有两种可能性 1*所有用户的响应时间都特别长 2*有个别用户的响应时间特别长，就可以从这个时间点来分析这个时间点的特征，从而得出相关的结论。

相关公式如下:

1: 已知方差，检验一组数据的平均值

使用统计量: $$z=\frac{\bar{x}-u0}{\sigma}\sqrt{n}$$

2: 未知方差，检验一组数据的平均值

$$T=\frac{\bar{x}-u0}{s}\sqrt{n}$$

使用统计量:

3: 已知方差，检验两组数据的平均值

$$z=\frac{\bar{x}-\bar{y}}{\sqrt{\frac{sigma1^2}{n1}+\frac{sigma2^2}{n2}}}$$

使用统计量:

4: 未知两组数据的方差，但是知道两组数据的方差相等，检验两组数据的平均值

使用统计量:

$$T=\frac{\bar{x}-\bar{y}}{\sqrt{(n1-1)s1^2+(n2-1)s2^2}}\sqrt{\frac{n1n2(n1+n2-2)}{n1+n2}}$$

5: 平均值的置信区间 （sigma 已知）

$$(\bar{x}-Z_{(1-\frac{\sigma}{2})}\frac{\sigma}{\sqrt{n}},\bar{x}+Z_{(1+\frac{\sigma}{2})}\frac{\sigma}{\sqrt{n}})$$

2、方差:

说明所检测的数值的离散程度的指标。

$$\sigma^2=\frac{1}{N}\left(\sum_{i=1\to n}(x_i-\bar{x})^2\right)$$

公式为:

如果方差越大，就说明系统响应时间波动越大，因此不稳定。同时可以通过某个时间方差非

常大，可以得出相关结论。

对于响应时间，如果有两组数据，一组方差大一组方差小，并不一定说明方差大的数据没有方差小的数据好，如果方差大的数据的平均值小于方差小的数据的平均值，有可能说明方差大的数据是在响应时间较小的范围内波动，而方差小的数据则可能在响应时间大的范围内波动。

相关公式如下:

1: 检验一组数据的方差为某一个数值。

使用统计量:

$$\chi^2=\frac{\sum_{i=1\to n}(x_i-\bar{x})^2}{\sigma_0^2}$$

2: 检验两组数据的方差相等。

$$F=\frac{S_1^2}{S_2^2}$$

3: 两种统计表合并成为一张表。

使用范围，这两张表拥有同一种横坐标，比如时间为横坐标，纵坐标各不相同。

它的理念是: 当时间刻度变化的时候，这两张表的纵坐标分别对应不同的值，将这两个值分别作为横坐标和纵坐标，这样就能更好的体现，这两张表中的纵坐标互相之间的关系了。

美人鱼和小暖暖都在认真的听着，记着笔记。

现在我们需要实际操作一下，这样可以加深印象， Lisiya 说。

有关性能测试的工具有很多种，比如 LoadRunner，Jmeter。

Lisiya 说：**现在我们讲一下 Jmeter 的使用吧！**因为 Jmeter 的深入研究要有待时日，我们就先尝试着最简单的 Jmeter 的测试吧！万事都要一步一步来。

小暖暖和其他 5 条美人鱼说：好的。

1：安装 JMETER

2：设置 ...../jmeter/bin/user.properties 文件

增加语句:

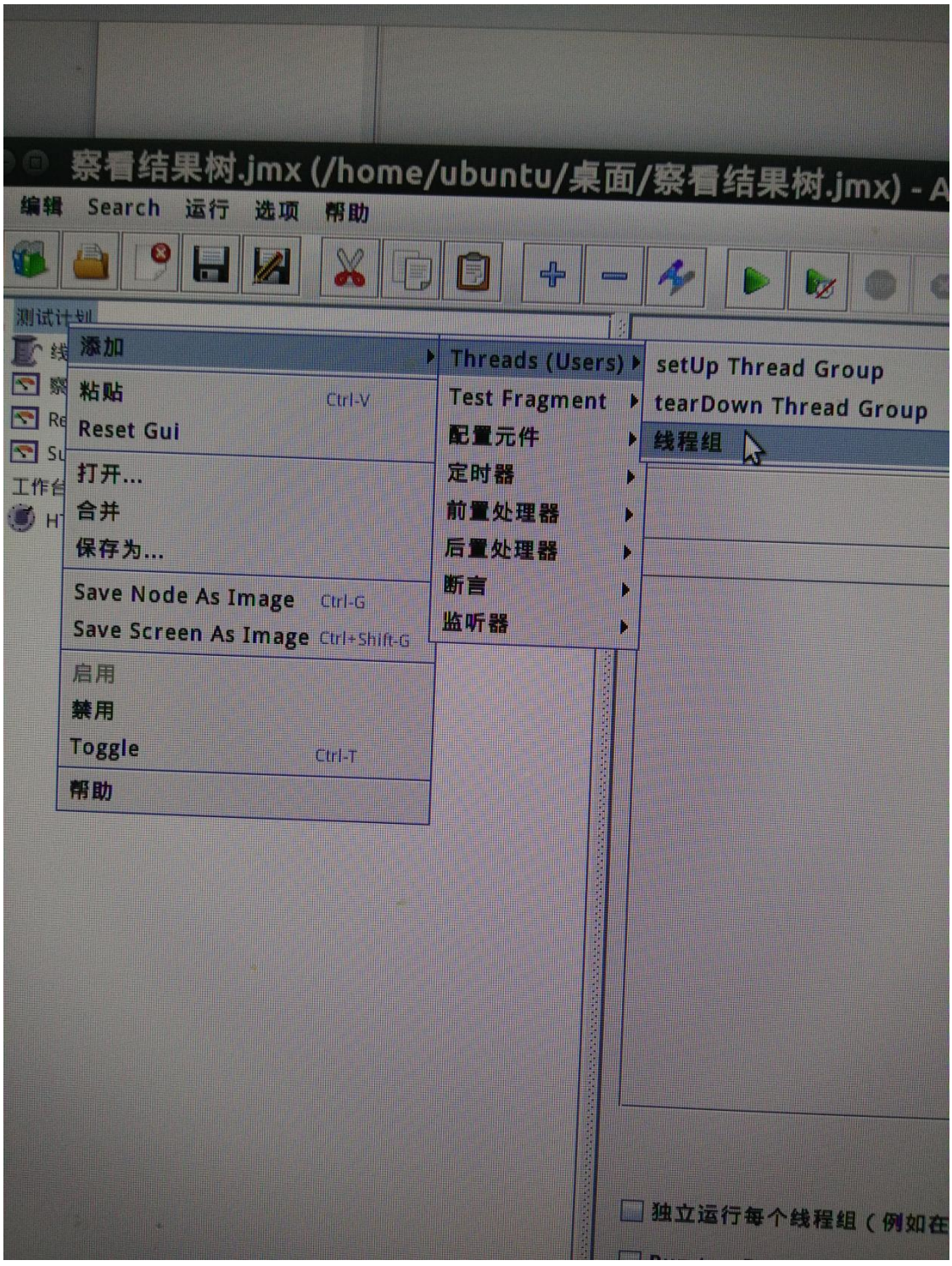
proxy.cert.alias=anything

只有增加了这句语句才能使用代理服务器进行录制脚本

以下是普通的 Jmeter 使用方法

1、添加线程组

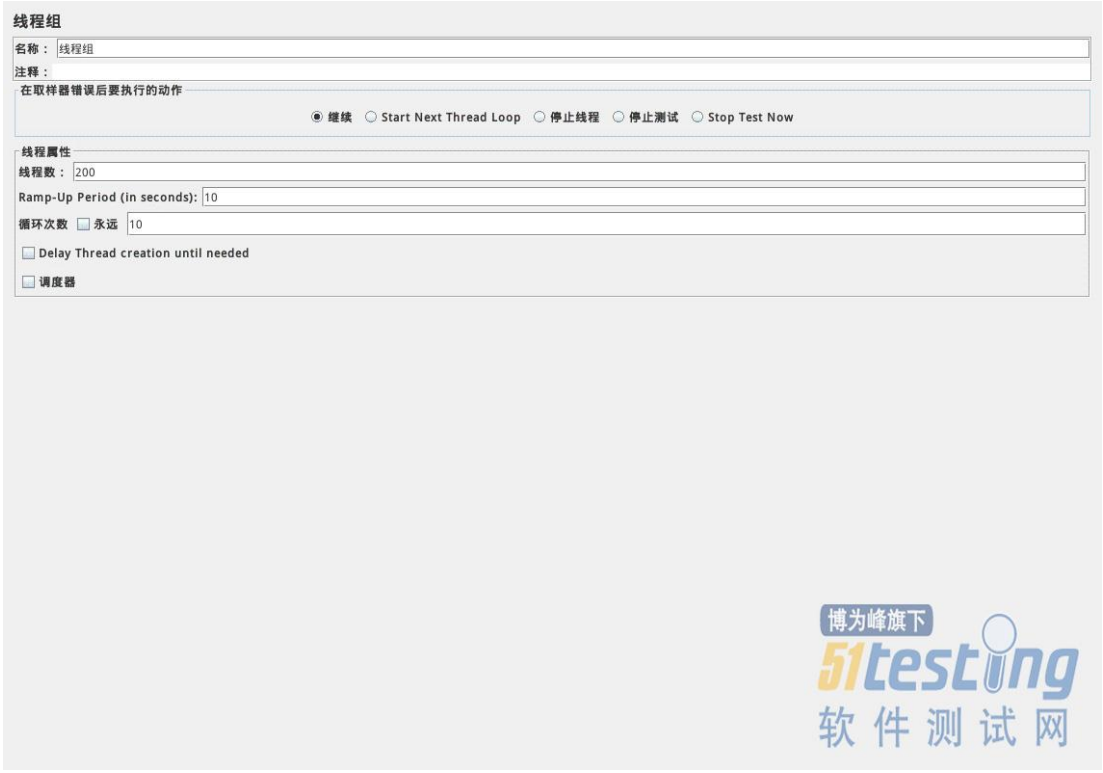
测试计划->添加->Threads(Users)->线程组



2、配置线程组的数值

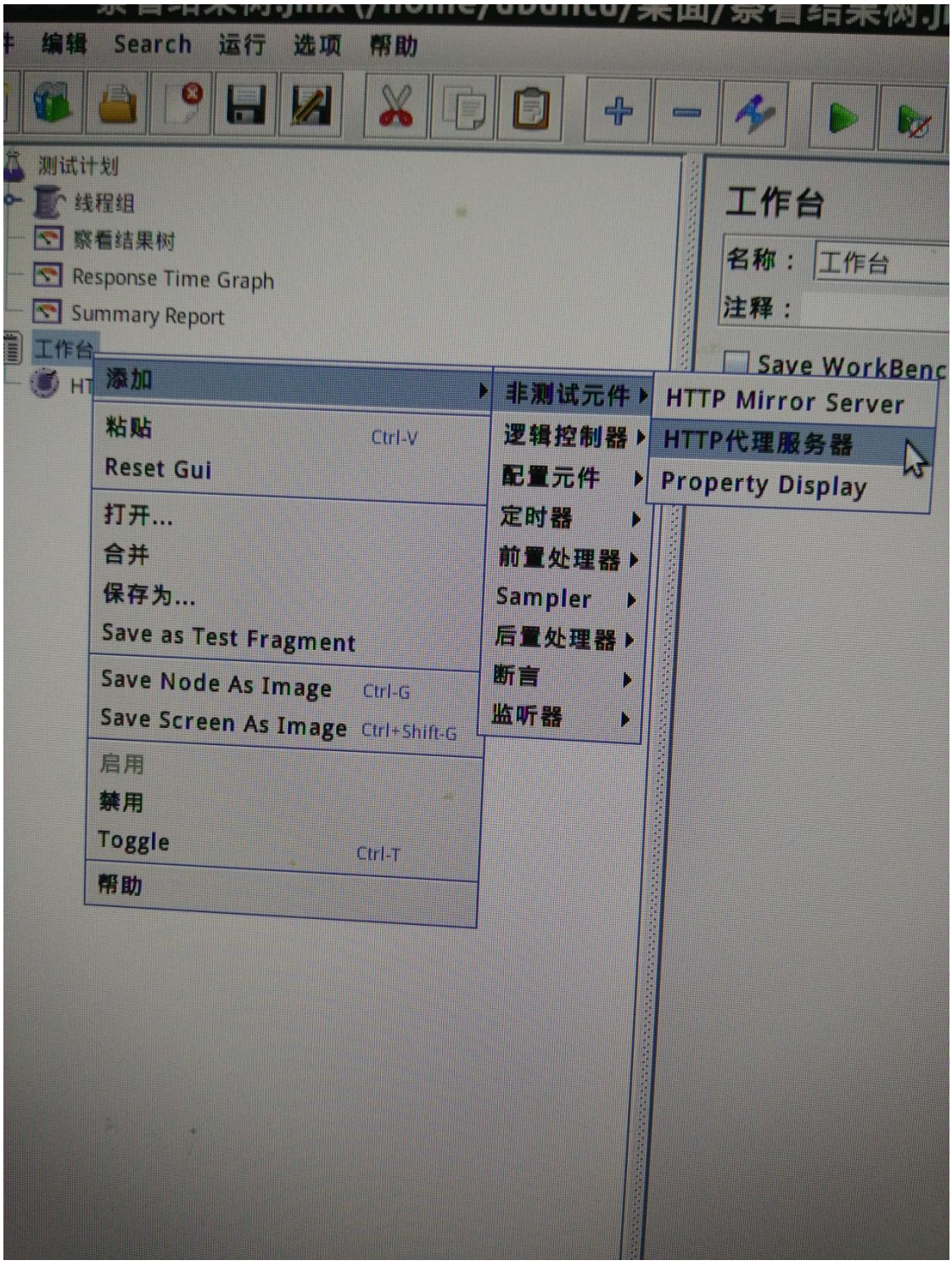
线程数：200，Ramp-up period(in second):10

循环次数：10

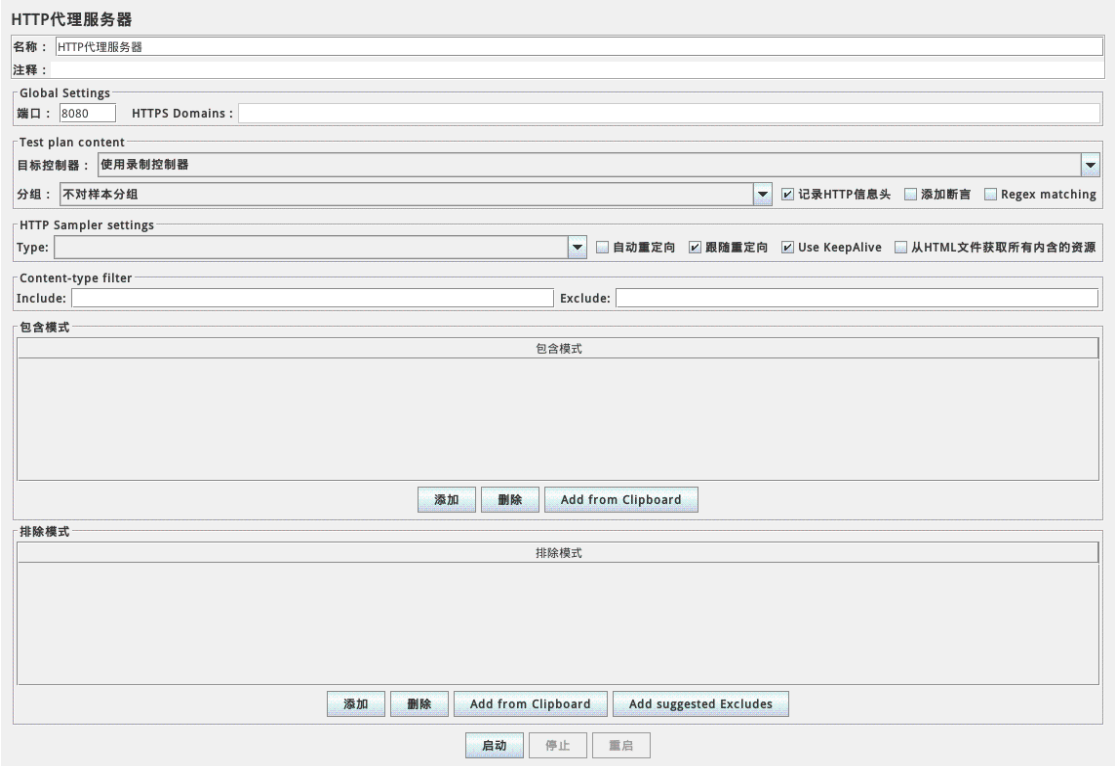


3、添加 HTTP 代理服务器

工作台->添加->非测试元件->HTTP 代理服务器



4、设置 HTTP 代理服务器的端口为 8080，这样就可以使用代理服务器端口号为 8080 的浏览器进行录制所需要的性能测试的部分



5、添加响应时间图表

测试计划->添加->监听器->Response Time Graph

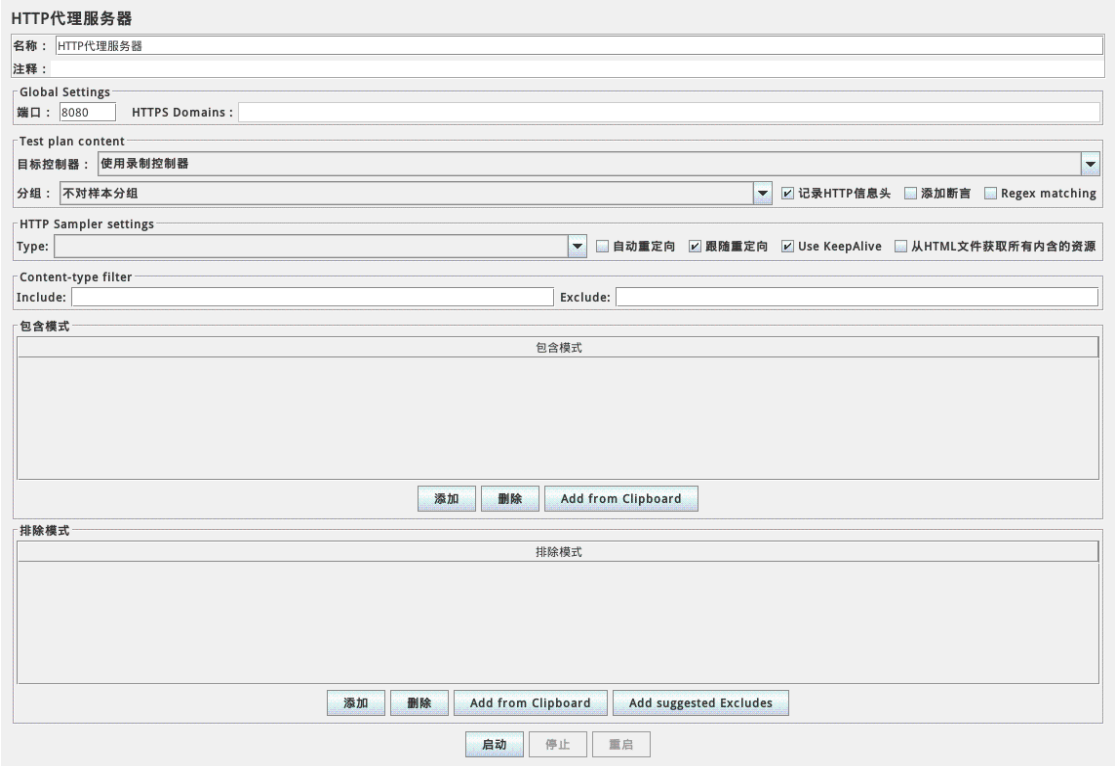
6、添加 Summary Report 图表

测试计划->添加->监听器->Summary Report

7、设置浏览器的代理服务器端口号为 8080

这里我们使用 Firefox 浏览器

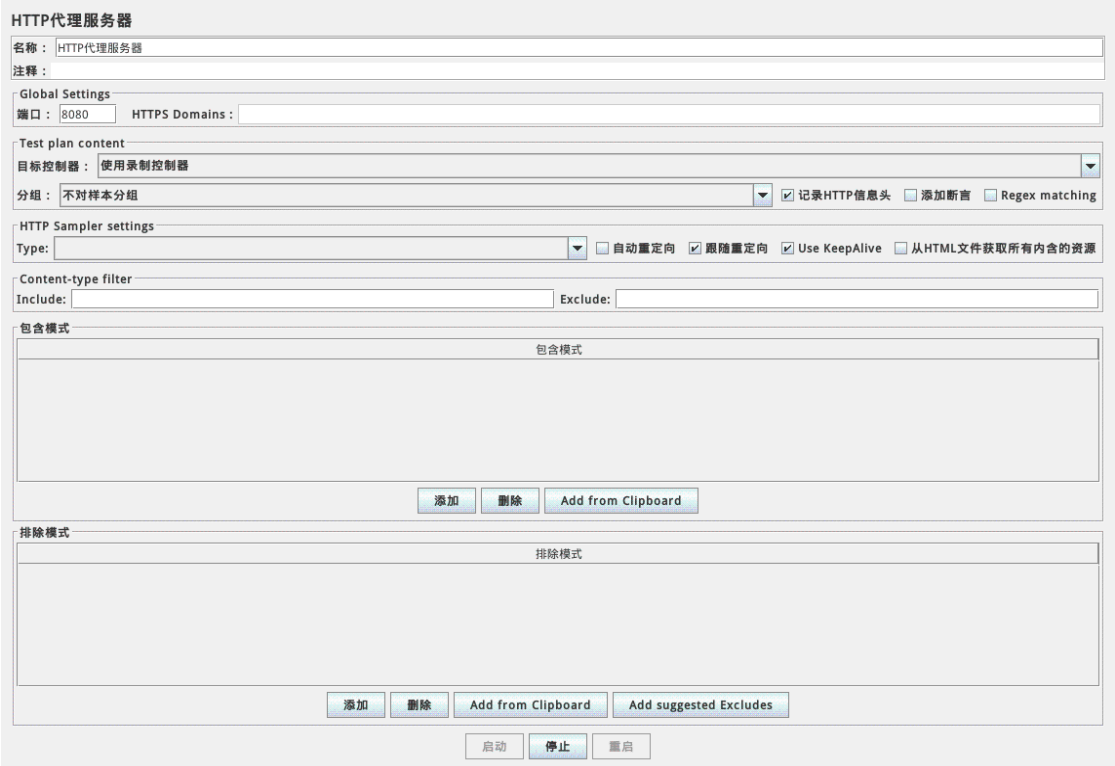
8、当以上都准备好之后，点击“工作台”->“HTTP 代理服务器”->“启动”按钮



9、打开 Firefox 浏览器，输入需要测试的网址，比如“http://bbs.****.com/search.php”，

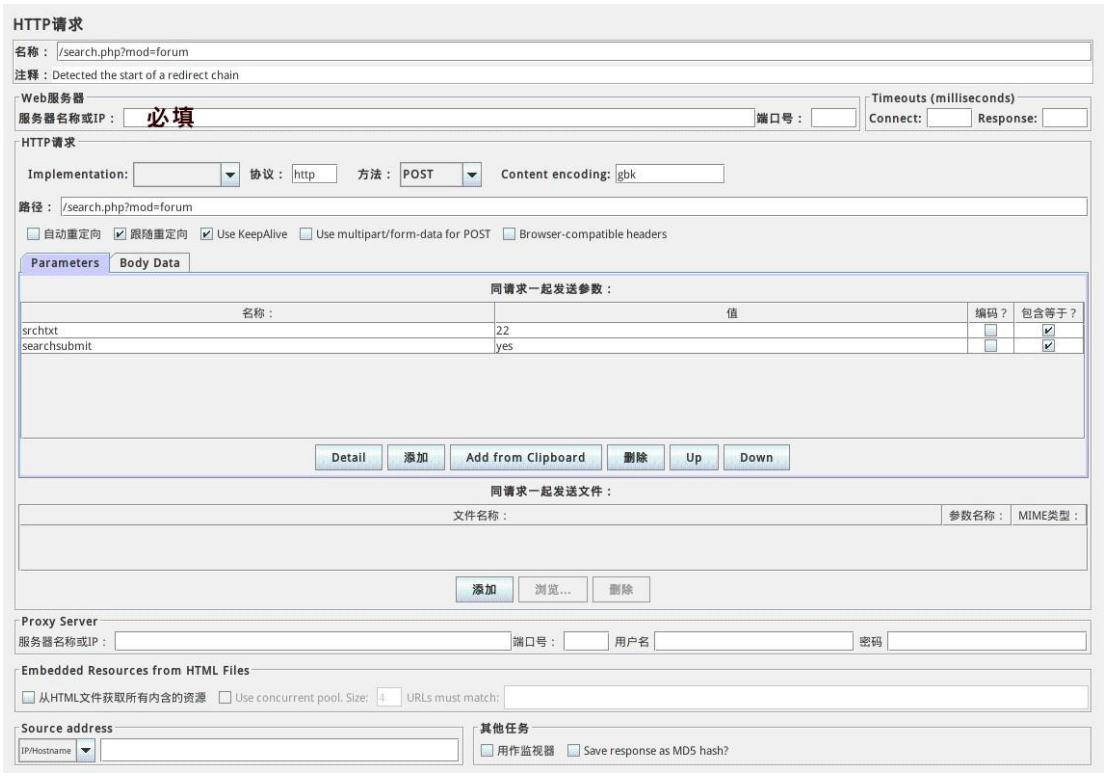
然后输入要搜索的内容，比如“22”，点击“搜索”按钮。

10、停止 HTTP 代理服务器。



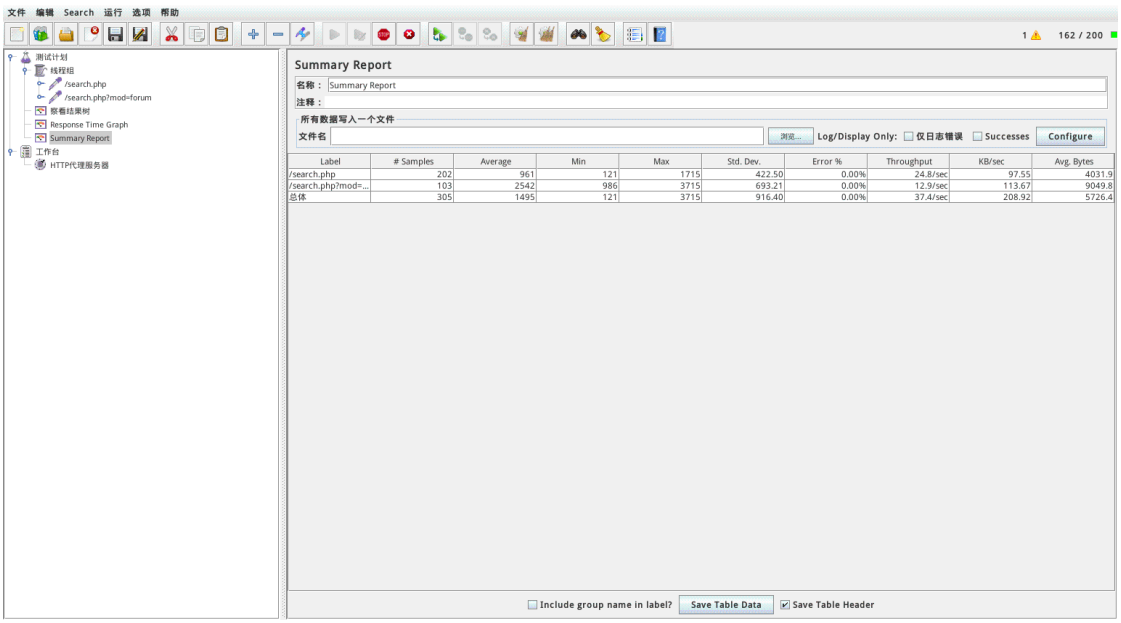
11、在 Jmeter 中删除多于的录制内容和多于的参数。直到最简化。

在这里，我们发现/search.php?mod=forum 的 http 请求中，所需要的参数只需要两个即可。

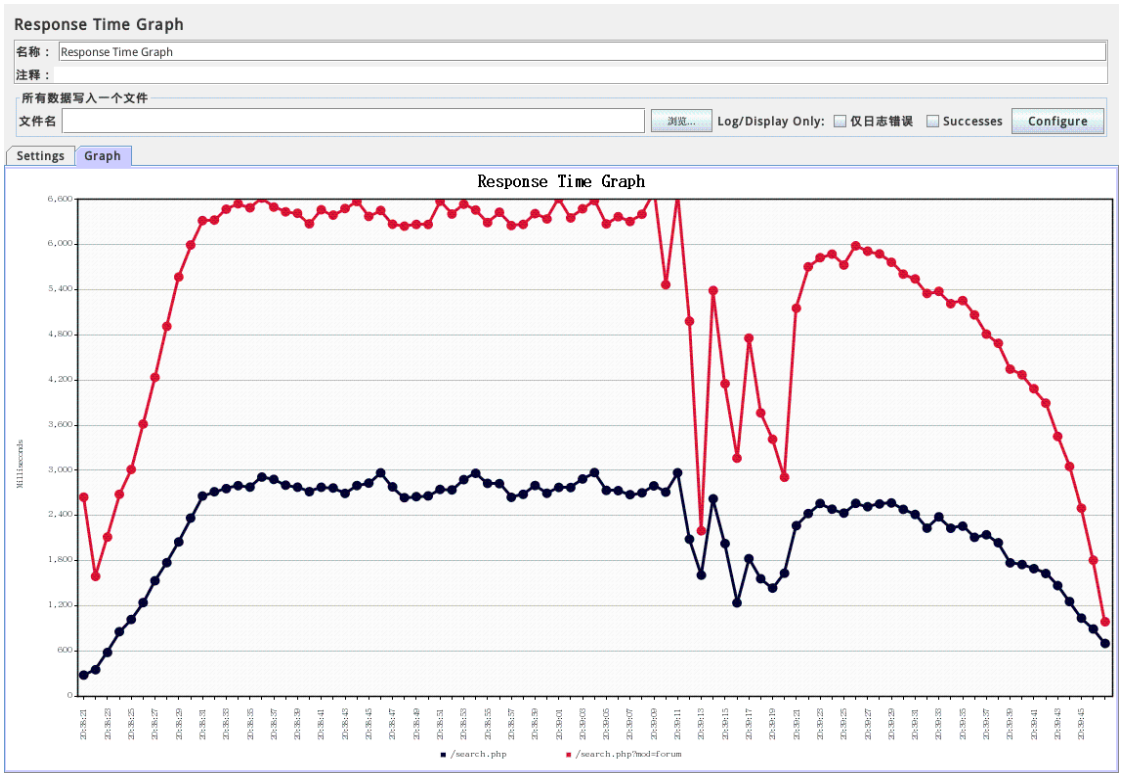


12、点击任意一个 report，然后点击“启动”按钮就可以看到 report 的具体数值变化过程。

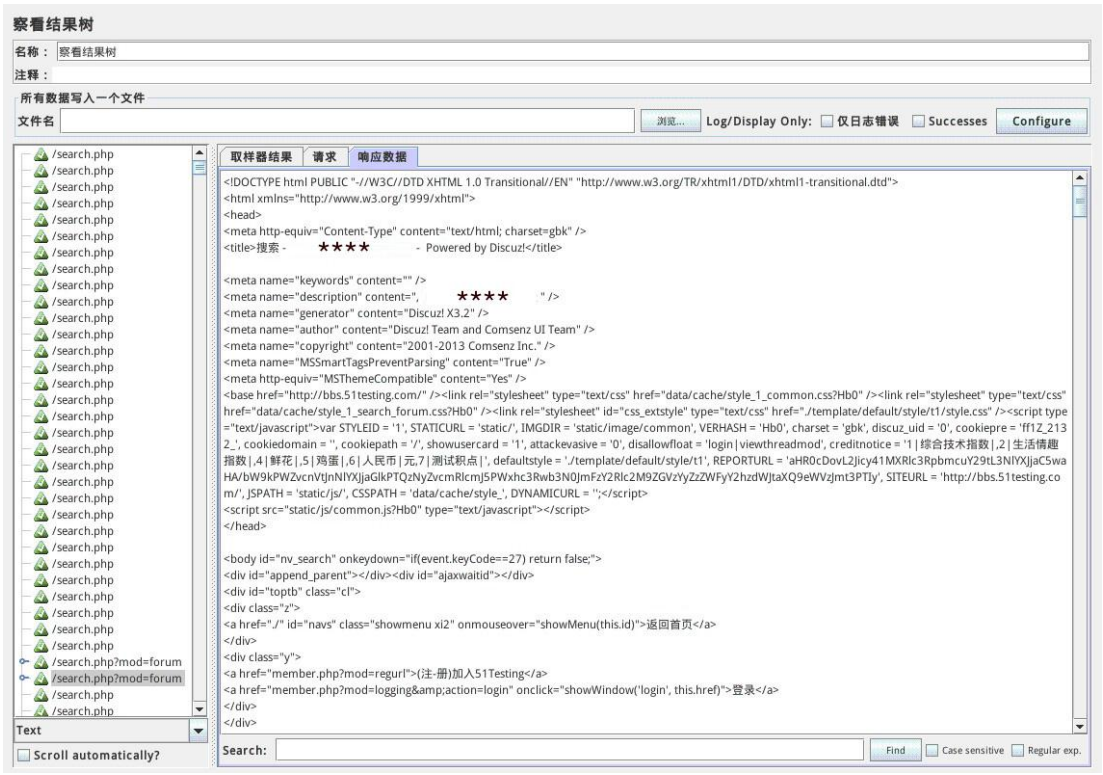
比如，我点击 Summary report，然后点击“启动”的情况如下



13、下图是 Response Time 的图表样式



14、查看响应内容，判断性能测试运行是否真正的能搜索出内容，如果返回结果和真实情况的搜索的返回结果一致，则代表性能测试脚本录制正确。此时，我们可以在“查看结果树”->“响应数据”中查看到返回的 html 的情况，因为查看到的 html 内容符合真实情况，因此判定为是正确的。



Lisiya 说，以上只是对于 Jmeter 的最基础的介绍，我们以后要多多学习 Jmeter

大家回答说，好的，这真的是非常有趣的学科！

拓展学习： http://www.atstudy.com/course/183
《Jmeter 性能测试》手把手实例教你如何运用 Jmeter 应对各种性能测试任务

测试人员例会会议议题制定规范

◆ 作者：陈 俭

一、例会的目的:

1) 加强组员分享意识

2) 同步项目整体情况，解决组内问题

3) 传达目标--打造一支测试精英队伍

二、内容甄选参考方法步骤:

1) 组员每个人写出自己想了解的议题，如限制至少 5 个

2) 统计出议题的票数

3) 深入了解每个人想了解议题的原因

4) 细化每个议题，直至内容可分享

5) 挑选出可分享的、对组员意义较大，且排名靠前的议题

6) 指派议题分享负责人

7) 分享排期

三、参考例会内容:

1) 技能分享

目的：相当于技能培训，至少每两周有一次本类分享，

- 服务端测试（单测、接口测试）
- 性能测试系列

- 自动化测试系列
- 随机测试思想
- 黑盒用例设计方法
- Appium 安卓自动化框架
- 浏览器性能评测方法
- Android 单测（框架、用例代码）
- 浏览器稳定性框架
- 浏览器白盒测试方法

2) 工作方法

目的：培养组员职业素养，本部分分享比较容易偏理论，要求必须与工作实际结合，有实际的例子，对测试工作有帮助才能分享。

内容：

- 时间管理
- todo 管理
- 情绪管理
- 人际关系
- 逆商
- 提问技巧

3) 工具分享

目的：包括常用业界测试用具和自己写的工具，提升组员的工具意识。要求分享的工具对测试工作有帮助。

内容：

- windbg 工具常用分析方法
- PM 工具使用方法

- fiddler 工具

4) 读书分享

目的：培养组员读书习惯，这部分分享容易偏理论。要求包含以下几部分：书名、内容简介、读后感、对大家的帮助。

5) 【例行】Bug 总结

目的：bug 总结分享，脑暴原因，避免以后大家犯同样的错误

6) 【例行】项目复盘结论公示

目的：不在例会上做复盘，但是复盘的有用的结论可以在例会上公示分享

7) 【例行】项目进度公示

目的：同步项目组目前的进度和未来的安排，提升组员的项目关心度

8) 【例行】组内新流程公示

目的：新流程同步给大家

9) 【例行】组内颁奖

目的：增强个人荣誉感。

四、可能存在的问题及解决方案

问题 1: 议题内容过于庞大，模糊，不易分享，如“python 学习分享”

解决方案:

了解组员意愿集中程度，如果大部分人有需求，可以细化议题，排期分享；如果只有一到两人想了解，就不用排到例会分享议题中，建议自学。

问题 2: 分享内容不符合预期，较浅薄。

解决方案：可以在组员间做调查，如果一半以上人觉得没有收获，明确告知该分享人，以后不用参与到知识分享。

问题 3: 与项目任务冲突，导致无法按期分享

解决方案：可以与其他分享议题的同学协调排期，如果不能协调，顺延到下周分享

性能测试流程

◆作者：曹承臻

一、背景:

当前所有同学做性能测试方法不一，没有统一的规范指导文档可以参考。

二、目的:

形成性能测试整个流程的指导规范，以后做性能测试的同学可以依据此规范做性能测试。

三、具体流程:

（1）需求评估

目的: 评估是否需要做性能测试。

- 需要做性能测试

新产品要上线，预估单台机器 QPS 峰值超过 100。

已经上线过的产品，由于接入了新的业务或者用户量增加，预估单台机器 QPS 峰值超过 100。

- 不需要做性能测试

单台机器 QPS 峰值低于 50 的需求。

有相同产品实现逻辑的产品，且已经做过性能测试。

例如: 假如一个请求，每次用户开启应用时都会发送到服务器，服务器则会返回给客户端本账号在好友中的积分排名情况。从产品的角度认为，每次应用启动，都会触发服务器查询一次数据库。这样会数据库会造成很大压力。而测试再了解了具体实现后发现: 针对每个用户机器码的排序数据是从 redis 服务器返回的，而 redis 服务器会每隔一小时请求存储的 mysql 服务器来更新账号排名信息，这样看来 mysql 服务器请求频率很低，

没有任何压力。由于 redis 服务器的性能之前已经测试过类似的，没有性能问题，所以这次并不需要对 mysql 服务器做压力测试。

- QPS 评估方法:

产品已经灰度或者上过线，可直接参考灰度数据来推算全量用户的 QPS 峰值。

产品未上过线，可通过类似已上线的产品来评估线上 QPS

以上两种都不符合，可通过通用算法来推算 QPS。

计算模型:

每台服务器每秒处理请求的数量=((80%*总 PV 量)/(24 小时*60 分*60 秒*40%)) / 服务器数量。

其中关键的参数是 80%、40%。表示一天中有 80%的请求发生在一天的 40%的时间内。24 小时的 40%是 9.6 小时，有 80%的请求发生一天的 9.6 个小时当中（很适合互联网的应用，白天请求多，晚上请求少）。

简单计算的结果:

((80%*500 万)/(24 小时*60 分*60 秒*40%))/1 = 115.7 个请求/秒

（2）逻辑了解:

目的: 产品、开发、测试相互认识，便于后续的沟通。

方法: 在逻辑了解时，组织产品、测试、开发做需求评审及产品实现讲解会。

讲解会需要了解的点:

- 了解整个性能测试的业务逻辑。一般需要了解请求个数，请求参数含义，请求参数可取值等。
- 了解服务器部署情况，主要要和线上服务器做明确隔离。不要简单认为所有的请求都是指向测试服务器，就认为是只向测试服务器打压。主要分为三种情况:1、测试请求会经过跳转链接到线上服务器。2、测试请求的 js 会异步请求线上资源。3、测试服务器会经过逻辑处理后返回给客户端，而这里的逻辑处理可能包括到线上服务器处理或查询数据。
- 了解本次性能测试的测试目标 QPS 及推断方法。

- 了解本次打压的请求间和参数间比例关系，这样便于后续写脚本时能够尽量模拟线上用户行为。

（3）测试方案:

主要用来评估本次性能测试的排期。并以邮件通知到各方。

发送时机：开发实现讲解之后，用例设计之前。

模板参考:（见附录一）

（4）环境搭建:

1) 测试服务器的搭建的搭建。测试环境可以有开发来搭建。原则上测试服务器配置不能高于线上服务器的配置，且测试服务器部署的服务要尽量接近线上服务器，比如说线上服务器上运行了 3 个服务，峰值时会占用 30%的 cpu，那么测试服务器也要运行同样类似，且打压时 cpu 最高只能打到 70%。这样得出的结果才具有指导意义。

2) 打压环境的搭建：主要指 linux 打压机的部署。具体部署方法就不在此赘述了，具体可以参照：http://www.51testing.com/html/13/n-3715213.html

3) 如果没有条件来搭建测试服务器，可否直接用线上的？

- 如果线上的服务器之前打过压力，可以直接用线上的压力指标来衡量是否满足本次需求。
- 如果之前没有打过压力，则在确认线上使用负载均衡的前提下，保证多台服务器中有一台挂掉后，其他服务器可以做到负载均衡压力，使线上用户不受影响时。可以对其中一台进行打压，在打压时需挑选线上用户非高峰时间段进行，且密切关注打压服务器情况，一旦挂掉，迅速启动起来。

（5）数据准备:

主要指性能测试有效数据的准备。请注意是有效数据？

举例：加入你手动写完脚本后，跑一下脚本，发现服务器返回没有报错。都是 200 的 response。这是否就说明是有效的打压呢？未必！应该回放脚本时，通过抓包工具抓包看下，对比真正使用场景中返回的 response。看下内容格式是否一致。如果你的打压脚本返回的是空的 200 请求或者仅仅有关键字，但是内容都是空的，而真正场景返回的是大

小为 15K 的 json。这说明你的打压场景是有问题的。

如果出现问题：应该和开发沟通原因，最后让打压请求返回正确的数据。

准备测试机的查看权限。服务器 log 位置及信息。主要信息包括服务器响应时间，出错 log 标示，具体客户端请求信息。

（6）脚本开发：

根据上面获取到的逻辑和数据进行脚本开发。视个人习惯，可以使用录制方式和手动写脚本。但是需要注意一点，在保证尽量模拟用户行为的前提下，尽量使脚本简单。如 if 语句和 for 循环。这类语句在高并发下，本身就可能导致压力机性能问题。

（7）打压过程：

打压过程并不是放那里不管，通过 linux 系统提供的命令，需要时刻关注被测服务器的性能指标，结合 LoadRunner 场景的曲线来动态判断是否存在瓶颈。LoadRunner 场景曲线主要关注 HPS、TPS、responsetime。服务端主要监控 cpu、内存、磁盘 IO、网络 IO。然后从这几方面再层层深入查找问题。另外，值得强调的是，打压过程不仅需要关注被测服务器的指标，也需要关注打压 agent 的性能指标是否正常。比如说如果并发数一直上不去，可能是打压机本上连接数受限导致的。也可能是打压机自己出口带宽满了导致的。

（8）问题定位&修改优化&回归：

问题定位是一个比较考验个人综合技术能力的地方。也是整个性能测试的核心所在。需要强调的是，不是看到某个参数遇到问题了，就直接可以断定服务区的瓶颈就在这个地方。比如：

- 大量的页调入请求导致内存队列的拥塞
- 网卡的大吞吐量可能导致更多的 CPU 开销
- 大量的 CPU 开销又会尝试更多的内存使用请求
- 大量来自内存的磁盘写请求可能导致更多的 CPU 以及 IO 问题

具体是哪个部分的问题，需要再根据具体的逻辑实现和推断来逐步缩小范围。主要的过程可以参考一下步骤层层深入。服务器硬件瓶颈-〉网络瓶颈（对局域网，可以不考虑）-〉服务器操作系统瓶颈（参数配置）-〉中间件瓶颈（参数配置，数据库，web 服务器等）-〉应用瓶颈（SQL 语句、数据库设计、业务逻辑、算法等）注：以上过程并不是

每个分析中都需要的，要根据测试目的和要求来确定分析的深度。对一些要求低的，我们分析到应用系统在将来大的负载压力（并发用户数、数据量）下，系统的硬件瓶颈在哪儿就够了。

最后就是开发优化，测试回归，这里需要强调一点：测试回归时，如果开发优化时修改了功能逻辑，则需要根据改动回归功能逻辑是否正常。不能只盯在性能指标满足了就 OK 了。

(9)性能报告:

性能测试报告没有固定的格式，但是大体需要包括以下几个方面:

- 整体结论：本次性能测试的整体结论。是否符合预期，是否可以上线。
- 具体推算方式：主要指给出整体结论的参考数据、计算公式等。
- 具体性能图标：服务器 cpu、内存、网络 IO、磁盘 IO、TPS、responsetime 等指标，涉及到数据库的还需要给出与数据库相关的指标。

（附录一）：性能测试方案模板

邮件名称:【性能测试方案】+本次项目名
发件人: 本次需求的开发、测试、产品
抄送人: 测试 leader、产品 leader、开发 leader、PM 及本次需求的测试邮件组。
正文:

一、测试目的

本次性能测试目标

二、测试任务

（1）测试阶段分布:

测试准备时间:

所有服务器提测时间：XXX

测试开始时间：XXX（一般需要在功能稳定之后，避免由于功能改动引起服务器框

架改变）

测试完成时间：XXX

（2）测试任务:

产品需求	测试需求		测试范围	开发负责人	测试时间预估	测试负责人	备忘
泛灵犀&新闻服务器性能测试	服务器	打压机部署	已有可用打压机	XXX	X	XXX	\
		服务器逻辑了解	接口参数、服务器逻辑	XXX	X	XXX	
		打压脚本编写	用户场景模拟	XXX	X	XXX	
		测试场景部署	用户场景模拟	XXX	X	XXX	
		结果分析调优	瓶颈定位、调优，结果产出	XXX	X	XXX	

（3）测试条件:

- 前端泛灵犀服务器与后端新闻服务器均为测试服务器，并与线上服务器数据隔离。
- 测试服务器可以正常提供服务并保持性能稳定。
- 测试数据要尽量模拟真实的用户操作。

三、详细计划安排:

日期	姓名
XXX	服务器逻辑了解&脚本编写
XXX	场景打压&结果分析&调优
XXX	结果分析&结果产出

四、测试环境说明

- 硬件/系统配置：服务器配置
- 程序配置：无

五、风险备忘

本次测试的已知风险。

六、测试分组

测试分组一

- 测试目的:

并发较多泛灵犀&新闻服务器请求时，查看泛灵犀服务器及新闻服务器各项性能。

- 持续时间：1h

- 测试数据

搜索词由开发提供

机器码随机、经纬度随机

平台随机选择

输入法版本 8.5，8.6 随机选择

- 测试方法:

a) 模拟用户操作过程，组成打压请求脚本。

b) 使用 Loadrunner 向泛灵犀服务器打压

c) 每秒请求数为 20 个，每 2 分钟增加 20 个/秒，一直增加到 1000/秒并持续 30 分钟

d) 重点关注同时并发用户数，TPS，响应时间，server 端的 CPU 和内存占用情况

e) 对比得到的性能结果数据，尝试找出性能瓶颈。

- 测试脚本逻辑:

整组打压请求的序列为：input1

（附录二）：性能测试报告模板

邮件名称:【性能测试结果】+本次项目名
发件人: 本次需求的开发、测试、产品
抄送人: 测试 leader、产品 leader、开发 leader、PM 及本次需求的测试邮件组。
正文:
一、测试结论:（给出客观的测试结果及产品建议、评估方法）

一元夺宝服务器单台可以保持整组请求 300QPS 并保持稳定，其中 charge/list.html 接口是系统的主要瓶颈，请产品看下 QPS 是否符合预期，如果不符合预期，建议从这个接口着手继续优化。

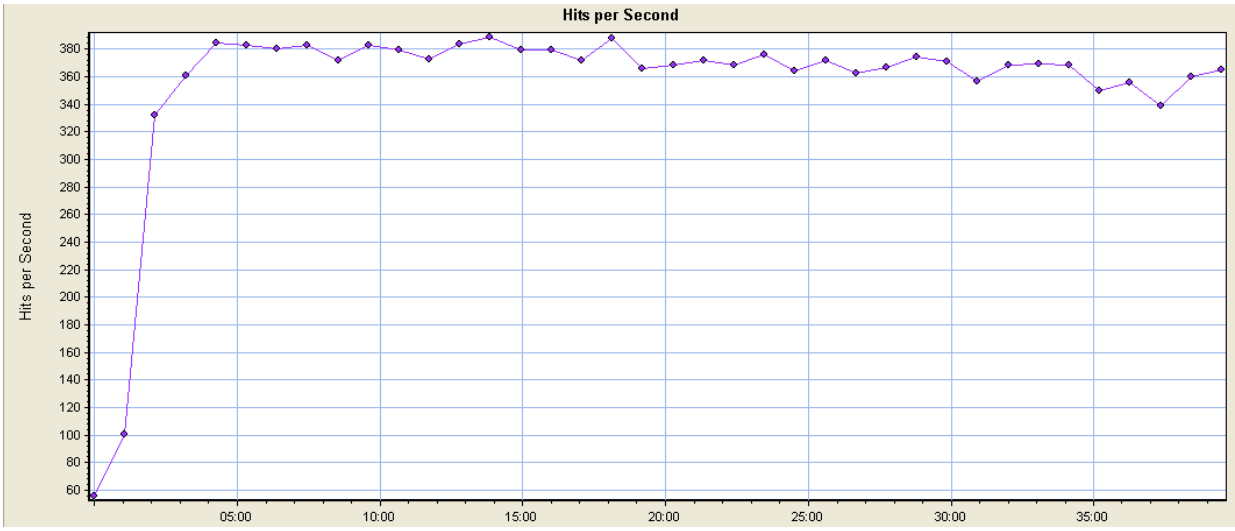
具体接口 QPS 情况:

接口名称	QPS
Pay	550
Charge	65
Detail/index	2000
Usercenter	300
其他	1500

Charge 接口主要包括：charge/index.html、do.html、list.html，其中 QPS 达到 60 时，charge/list.html 接口的响应时间超过 2s。

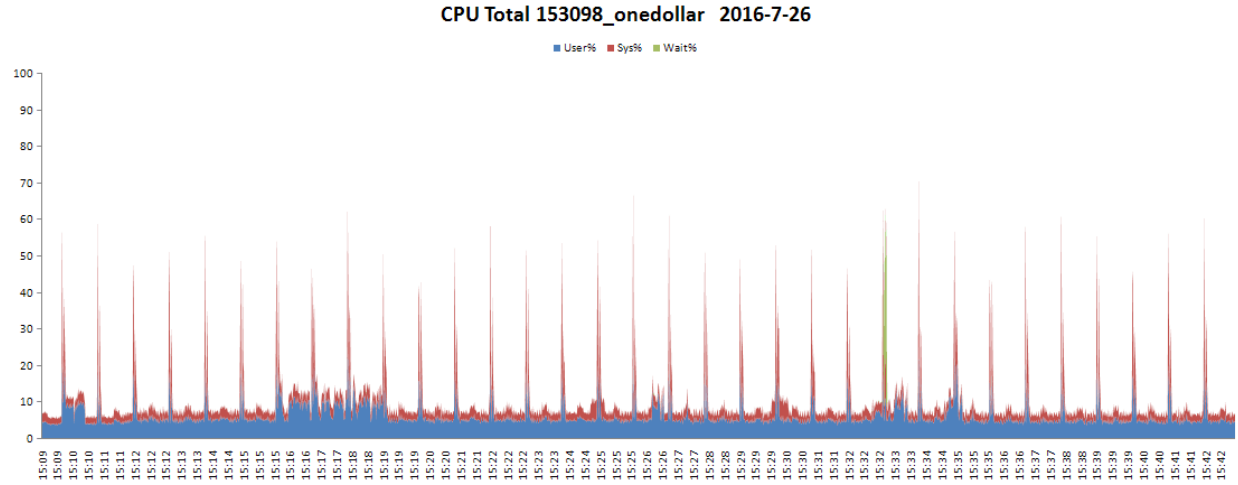
二、具体测试结果:

- HPS:



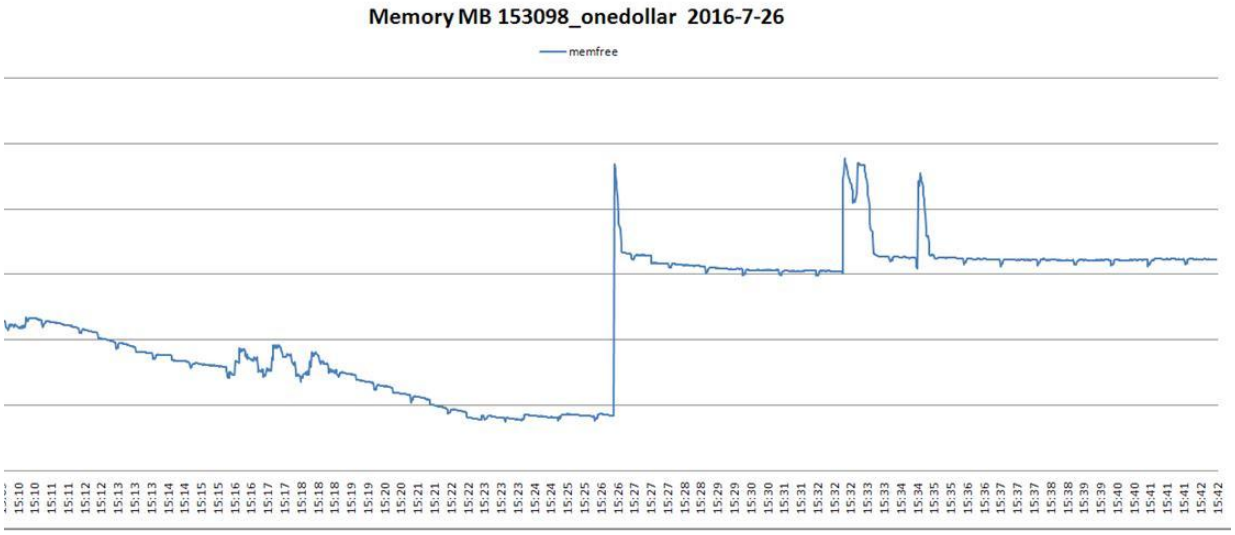
本组结论：整组一元夺宝请求可以达到 HPS300 并保持系统性能稳定。

- CPU 占用:



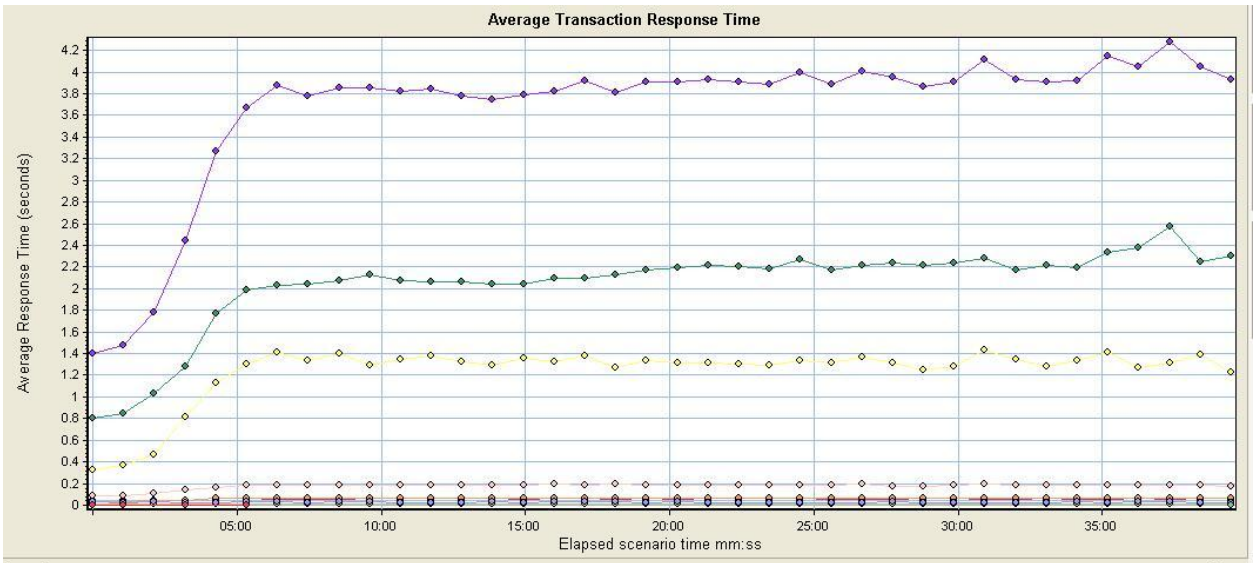
本组结论：整个打压过程中，cpu 占用低于 20%，符合预期

- Mem 占用：



本组结论：整个打压过程中，free 内存一直大于 0，最后会触发 linux 系统的自动清理功能，符合预期。

- Response time:



Legend

Color	Scale	Measurement	Graph's Mini...	Graph's Ave...	Graph's Max...	Graph's Me...	Graph's Std....
	1	Action_Transaction	1.401	3.669	4.278	3.886	0.68
	1	Charge	0.801	2.043	2.578	2.175	0.388
	1	GetclassList	0.007	0.008	0.008	0.008	0
	1	GetdetailInfo	0.042	0.048	0.05	0.049	0.002
	1	GetFirstPage	0.011	0.011	0.011	0.011	0
	1	GetlatestList	0.042	0.043	0.044	0.043	0
	1	GetListCount	0.007	0.008	0.008	0.008	0
	1	GetUserInfo	0.032	0.062	0.071	0.065	0.009
	1	GetVersion	0.007	0.008	0.008	0.008	0
	1	Pay	0.091	0.176	0.194	0.183	0.025
	1	start	0.026	0.027	0.028	0.027	0

本组结论：整个打压过程中，所有接口的 response time 一直小于 1s（每个事务封装有多个请求），charge 组合请求是导致 QPS 上不去的主要原因，后续优化时建议从 charge 请求着手。

三、测试过程中发现的问题:

1. Detail 接口打压一段时间后，返回 404 错误。

解决方案：resin 启动的时候 jvm 参数配置中默认堆控件分配不足，增加 Xms 及 PermSize 配置后，不会再返回 404 错误。

2. 打压 detail→shop 序列请求 QPS 不到 10 时，服务器 shop 接口出现 404 错误，同时 response time 逐渐增加到 5s 以上。

解决方案：添加数据库监控后，发现是数据库查询逐渐增加到 5s 导致的，查看具体 sql 语句发现有一项查询会将所有表都查出来，导致效率很慢，而这项没有用，去掉后，QPS 可以达到 140 左右。

3. 除了 pay 和 charge 的接口，整体打压时，发现 order 接口在高压下，占了 5s 以上。

解决方案：由于数据库没有添加索引，全表查询的，添加索引后，QPS 可以达到 200

以上，问题解决。

四、测试基本信息

测试任务名称：一元夺宝服务器性能测试方案

测试目的：检查在高并发的情况下一元夺宝服务器运行情况，并尝试找出系统瓶颈。

测试人员：曹承臻

测试环境说明

- 硬件/系统配置

CPU：Core 32 核

内存：128G

操作系统：Linux

- 程序配置

无

- 测试数据

无

测试分组

（1）测试分组一

测试目的：并发较多一元夺宝服务器请求时，查看一元夺宝服务器各项性能。

持续时间：1h

测试条件：

一元夺宝服务器服务器可以正常提供服务并保持性能稳定

测试方法：

a) 模拟用户操作过程，组成打压请求脚本。

b) 使用 Loadrunner 向一元夺宝服务器打压

c) 每秒请求数为 20 个，每 2 分钟增加 20 个/秒，一直增加到 500/秒并持续 30 分钟

d) 重点关注同时并发用户数，TPS，响应时间，server 端的 CPU 和内存占用情况

e) 对比得到的性能结果数据，尝试找出性能瓶颈。

测试脚本逻辑:

向一元夺宝服务器发送请求，随机组合发送一元夺宝请求序列，验证返回结果正确性

QTP 使用之
一个灵活的数据驱动的 IF 函数

◆ 译者：杨 俊

在上一篇文章里，我们对于能够覆盖选择案例决策结构的一个数据驱动方法进行了研究，这种灵活的方法让此类结构案例变得更加有效、可管理，并且让代码也变得可读。

而这篇短文描述了以一种类似方式的方法去覆盖“**If-Else If-Else-End”的 If 结构案例。结果是，又一次地，让代码变得更加简洁、高效、灵活、并且方便维护。**

函数签名是:

Function [If](ByVal strExpression, ByVal strDoIfTrue, ByVal dicElseIf, ByVal strDoIfFalse)

这里:

1. strExpression 是一个表达式，返回的是“真”或者“假”
2. strDoIfTrue 是一个有效的存储过程名或者一段代码，当 strExpression 返回 “真”的时候，就被执行
3. dicElseIf 是一个带有键-值对的字典对象，键定义每个条件，值定义当条件为真时的执行命令。相比于多次使用 IF 和 Else 语句，我们可以用先前提到的[Select Case]函数，传入这个字典对象。
4. strDoIfFalse 是一有效的存储过程名称或者一段代码，当 strExpression 返回”假“时被执行。

备注：方括号是 VBScript 一不太常用的特性，里面可以使用特殊字符或者保留关键字来标识。

- **覆盖 IF 语句的[IF]方法如下:**

```
-------------------------------------------------------------------------------
Function [If](ByVal strExpression, ByVal strDoIfTrue, ByVal dicElseIf, ByVal strDoIfFalse)
  [If] = Eval(strExpression)
  If([If])Then
        PrintLog("If", "Executing " & strDoIfTrue)
        Execute(strDoIfTrue)
  ElseIf ([Select Case](dicElseIf)) Then
        PrintLog("If", "Executed ElseIf")
  Else
        PrintLog("If", "Executing " & strDoIfFalse)
        Execute(strDoIfFalse)
  End If
End Function
-------------------------------------------------------------------------------
```

在上面我们可以看到，[IF]函数里面用了**[Select Case]**的函数(该函数里面加了日志事件，如下所示)

```
-------------------------------------------------------------------------------
Function [Select Case](ByVal dicCases)
      Dim [Case], [Cases]
      [Select Case] = False
      If (Not TypeName(dicCases) = "Dictionary") Then
            PrintLog("Select Case", "Nothing to do")
            Exit Function
      End If
      If (dicCases.Count = 0) Then
            PrintLog("Select Case", "Nothing to do")
            Exit Function
      End If
      [Cases] = dicCases.keys
      For each [Case] in [Cases]
            [Select Case] = Eval([Case])
            If([Select Case])Then
```

```
                PrintLog("Select Case", "Executing " & dicCases([Case]))
                Execute(dicCases([Case]))
                Exit For
            End If
        Next
    End Function
```

我们可以看到以上两个函数都使用了自定义函数 **PrintLog**，如下所示。该函数将步骤细节都录入到 **UFT** 报告中，并且在输出面板中打印出事件日志来。

```
Function PrintLog(ByVal strStepName, ByVal strDetails)
Print(strStepName & " - " & strDetails)
Reporter.ReportEvent micDone, strStepName, strDetails
End Function
```

- 使用

```
Dim arrTemp, i
Dim dic
Dim Condition, IfIsTrue, IfIsElseIf, IFIsFalse
Set dic = CreateObject("Scripting.Dictionary")
Condition = "arrTemp(i)>70"
IfIsTrue = "PrintLog(Now()&"" Wear a T-Shirt!"")"
Set dic = CreateObject("Scripting.Dictionary")
dic.Add "arrTemp(i)>60", "PrintLog(arrTemp(i)&"" Wear a hat!"")"
dic.Add "arrTemp(i)>50", "PrintLog(""Testing [If]"", arrTemp(i)&"" Wear a long-sleeved shirt!"")"
Set IfIsElseIf = dic
IfIsFalse = "Wscript.Echo(Now()&"" Wear a coat!"")"
arrTemp = Array(50, 55, 60, 65, 70, 80)
For i = LBound(arrTemp) To UBound(arrTemp)
PrintLog("Testing [If]", "Temperature is " & arrTemp(i) & ": " & [If](Condition, IfIsTrue, IfIsElseIf, IfIsFalse))
Next
```

- 输出 (UFT 日志)

Testing [If] – Temperature is 50: Wear a coat!

Testing [If] –Temperature is 55: Wear a long-sleeved shirt!

Testing [If] –Temperature is 60: Wear a long-sleeved shirt!

Testing [If] –Temperature is 65: Wear a hat!

Testing [If] –Temperature is 70: Wear a hat!

Testing [If] –Temperature is 80: Wear a T-Shirt!

鱼儿妈的 4 年测试生活

◆作者：小鱼儿

提笔写点东西已经是自毕业以后不需要靠作文博得分数之后的第一次，这第一次给了基本已融入我每日生活的 51Testing，我觉得测试这个工作与我的生活已经紧密联系在一起了，可能我的以下行文没有什么逻辑调理，但是基本是我这个测试人这 4 年大致接触测试的主脉络，希望通过这种方式与同样在测试这条阵列的朋友们一起交流，分享。

2012 年-2015 年

2012 年我的大三生活即将结束，很幸运我在上海这个我父母工作和生活的大都市找到了一份月薪 2000 的实习工作，我的爸妈在上海呆了近 10 年，老一辈人希望子女离家近一点，虽然上海不是我们的家，可是家人在哪里，家就在哪里，我也是一个恋家的乖乖女，很幸运自己可以在父母居住的地方找到一份工作，其实当时的我并不知道自己到公司之后要干嘛，具体负责什么，只知道我的英语六级和数据库 2-4 级让面试官勉为其难的录用了我，之所以是勉为其难实则面试的时候表现真真是不尽如人意，后来进公司了面试官也是我的 leader，他开玩笑说我当时答的真是让他哭笑不得，说错也不能算错，反正是答非所问，没办法，谁叫我的英语还是不错的，实习生嘛也不能要求太高，反正是需要调教的。

进入这家台湾企业做服务器测试之后才发现真的不能用一个累字来形容，现在想想当时自己也坚持下来了，虽不知这种坚持对于我的择业人生怎样，可能要是当初不去做测试我就靠着那张会计证当会计去了，现在想想其实也蛮佩服自己的，毕竟服务器那大家伙来来回回拆卸对于一个小女子来说真真的一丁点不怜香惜玉啊，在台企的 3 年吃了无数的凤梨酥，也拆拆装装了无数的服务器，主板，有聚餐时的欢声笑语，也有测试环境搭不出来，然后被 leader 说的偷偷哭哭啼啼，但也都过去了。

这 3 年前一年基本是熟悉部门的机器和做常规的测试，即是二次换料测试 ，就是一个服务器中间替换了一个原材料，产品做完了基本的测试之后就拿到系统来做整合测试，基本不会有什么问题，但是量多，重复一样的动作，拆，装，刷分位，装系统，常规的

功能和压力测试。有时候一个人要测试 20 多片板子，要把每一片板子放进机壳里面，拧螺丝，插电源，开机，开测，要是操作有问题或者因为其他人为因素导致测试跑不过，这样的动作又要重来。测试真的是一个需要耐心和细心的活，所以很多人就说测试适合女孩子来干，可是有时候小女子也会.....

一年之后我从北方一所普通大学毕业了，大学最要好的马儿在那个美丽的滨海城市送别了我，她也踏上了更北的继续求学生活，毕业之后我留在了实习单位，拿到毕业证的同时我也转正了，工资涨到了 3500，还有值得高兴的就是我不用每日重复一样的测试工作，我被调到别的 team，可以接触新项目，新的项目每日学的东西很多，做的事情比实习的那一年有意思多了，这其实也就意味着我犯错误的机会更多了，可是在一位出色女监理的指导下我也进步很快，只能说我对那位女项目经理是又爱又恨，我渴望成为她那样出色的人，也在她的压榨下觉得她有点无情，生活就是这样，工作也是这样，你需要配合团体的需要。

第一个项目结束，又接着做了 2 个项目，每个项目差不多都持续了一年，测试的日子继续着，我也找到了人生的另一半，只能说近水楼台先得月，很快我也有了我可爱的小鱼儿，生活的重心不在只是测试，因为鱼儿的到来生活有了另外的色彩，我也为了家庭回到了鱼儿爸爸的故乡，开启了另外一个南方城市的测试生活。

2015 年

在鱼儿爸爸的老家我很辛苦的找到了一份电信业务的软件测试工作，毕竟到另外一个陌生的城市找工作并不容易。业务名为：固网手机推送，我负责模拟线上环境搭建测试环境，可以理解为 web 与手机测试一体，因为电信手机流量业务是用户手机端操作，数据反馈到后台服务器，可以理解为 B/S 架构，只是这个 B 不单单只是 PC 端的，也是手机端的 B，是手机流量推送业务。最忙的是节假日，因为会有促销活动，环境在节日之前提前完成测试，部署，等待用户的反馈。这段工作经历很短，一则是由于照顾还在哺乳期的鱼儿，照顾鱼儿花了我太多体力和精力加之工作地点太远，交通太堵，只能很抱歉的辞职了。

2016 年

因为自己的不成熟和种种原因我离开了我的鱼儿来到了西子湖畔，钱塘江边，应聘到了一家做笔记本的公司做组件和 web 测试。我负责公司内部网站的测试和线上环境部

署，同时也做组件测试。我更加倾向于把我负责的组件测试叫 APP 测试。对于我负责的 Web 测试因为环境已经相当成熟，做起来也不是很累，只是前期搭建测试环境的时候稍微麻烦点，因为这个内网用 Windows 系统搭建测试环境，所以对于我来说还是应付得来的，之前那家公司用 Linux 搭建测试环境，着实对于我这个没有 Linux 基础的测试人员来说有点手忙脚乱。虽然很多人包括我的老板在内都说还有谁用 windows 搭环境，感觉太 low，但是也没有谁去改变，也就是抱怨抱怨。测试环境搭建好之后基本就不会大动了，每次根据业务需求，开会，然后讨论测试逻辑，写测试 case，case review，code review，测试，测试完内部发报告，找老板 review，老板看过后发正式的报告，QA，最后部署线上环境，线上测试一下，发一份线上测试报告，没什么问题，over。组件测试的业务流程与 web 测试大同小异，只是组件测试相对于 web 测试稍有不同，因为针对的只是单个组件，所以测试的就很细，很深入，会从安装，到功能，到兼容性等等方面进行测试。

现在我还在钱塘江边做着测试的工作，冬天快到了，天气冷了，人也懒了了，可是每日不变的还是打开电脑，打开网页浏览 51Testing 的习惯没有变，在这里我找到了很多同行，也接触了测试这个领域很多自己没有涉及的专业和业务知识，每个行业都有同行，同行之间的对话以及业务交流会让我们提升的更快，也让我们的视野更广，51Testing 已经成为我工作的精神家园，鱼儿不在身边的日子里她陪伴着我成为更好的我，以后我还会一如既往关注 51Testing，关注测试人的生活。

再论纯软件测试方法

◆ 作者：顾 翔

2009 年，我在网上发布了一篇文章《浅谈纯软件测试方法》，这些日子我对这个想法有了更深入的理解。

大家都知道李小龙吧，他汇集了各路武林高手，并向他们学习，最后结合武术、柔道、摔跤、拳击、泰拳...的精华，并且结合武术的本质“攻”和“防”创建了自己的拳术截拳道， 达到了武术的最高境界，成为一代宗师。

在软件测试领域，软件测试一直是被开发牵着鼻子走的，虽然产生了各种软件测试模型（V 模型、X 模型、H 模型）和方法（基于传统、基于质量、基于风险和基于经验）也解决不了提高质量的本质。另外，最近 Scrum 敏捷方法被各大公司使用。首先我的确认为 Scrum 是个好东西，尤其是 ATDD, BDD, TDD 摆脱了测试被开发牵着鼻子走的形式，然而 Scrum 给测试带来了很大的工作量，尤其是在 Sprint 后期，随着老功能越来越多，回归测试的工作量越来越多。有人提出了想法，让开发人员帮助测试，可是大家都知道，测试需要有一定发现 Bug 天赋的，有些人善于做测试，而有些人不善于。

于是我们又回到了软件测试的本质，“测”与“试”。所谓“测”就是检测软件系统是否按照用户显性需求的运转，比如功能；“试”可以理解为试错，尝试。比如找到系统最大负载点，系统对错误输入，异常环境是否可以适应，一旦程序发生错误多久可以修复。我们也可以把“测”理解为证“真”，“试”理解为证“伪”。

下面我们来讨论在一个 Sprint 中如何运用软件测试的本质“测”与“试”来进行测试工作。我们假设一个 Sprint 为 1 个月，即 22 个工作日，我们把这 22 个工作日分成前、中、后三部分。前（第 1 个工作日到第 7 个工作日），中（第 7 个工作日到第 14 个工作日），后（第 15 个工作日到第 22 个工作日）。

在 Sprint 前期测试人员的主要工作为书写这个 Sprint 新功能的测试用例，这些测试

用例可以是自动化，也可以是手工的，并且在这些测试用例中，以证“真”的“测”的方法为主，证“伪”的“试”的方法为辅。我把这种测试方法叫做纸上谈兵式测试，在这里测试用例没有具体的格式，目的只要可以给相应的开发人员看懂就行，关键需要关注的是对新特性的覆盖度。从第二个工作日开始，在每日站会后，测试人员把他们前一天写的测试用例交给相应的开发人员，与他们进行一对一的评审，由于测试用例是每日提交的，所以评审的时间不会很长。一旦通过评审后，这个测试用例就交给开发人员了。开发人员在开发期间需要 100%达到这些测试用例所希望的结果，并且根据开发人员的自身需求，在适当的时候执行测试用例。

在 Sprint 中期，测试人员主要工作专向基于经验的探索式测试和非功能性测试，在这个阶段，主要运用证“伪”的试的方法。而开发人员的主要责任在重新运行一遍交给自己的测试用例，以及对缺陷的修改。

在 Sprint 后期，开发人员协助测试人员一起完成回归测试，开发人员按照以前的测试用例执行测试，测试人员仍旧以探索式方式来测试，以保证整个 Sprint 结束是一个高质量可交付的产品。一旦发现缺陷，开发人员优先回去修改缺陷。

我们在日常测试工作中仍旧需要抓住软件测试的本质，“测”与“试”，发挥测试人员的经验优势，同时邀请开发人员来协助产品测试，从而提高产品的质量。

利用代码覆盖率进行精准测试和回归

◆极测：linxuan

一、前言

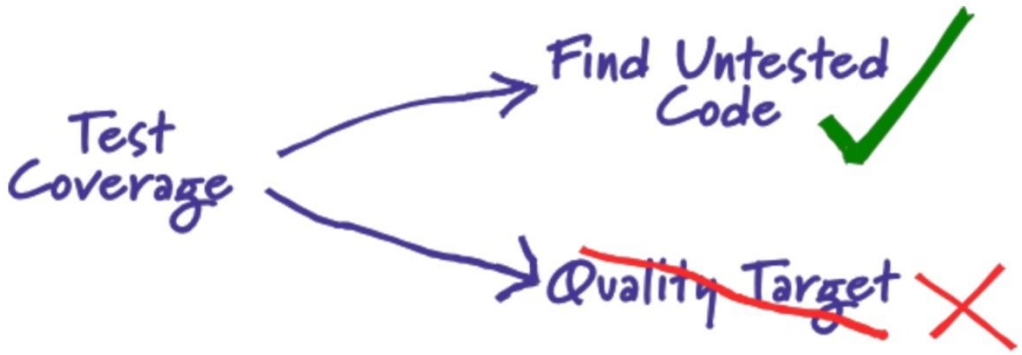
如何评估测试是否充分以及产品质量好坏一直是一个很难的事情，传统上:

从测试工程师角度，我们通常用自动(手动)测试用例数量，这些用例对需求覆盖率，HSF/Http 等接口覆盖率，bug 数等几个维度去评估。

从开发工程师角度，我们通常使用单元测试对产品代码的行/路径覆盖率去评估。

但是这些方式都有很多缺点，比如很多数据依赖于人去手动去统计，或者拍脑袋去评估，以及产品开发的习惯，单元测试缺失等，导致我们评估产品质量好坏和测试是否充分只能通过上线后发现的故障去评估，都是事后去补救，下面我谈谈如何在产品上线前通过所有测试活动对产品代码的覆盖率去评估测试是否充分，实现精准测试和回归

二、统计代码覆盖率的价值



1. 分析未覆盖部分的代码，从而反推在前期测试设计是否充分，没有覆盖到的代码是否是测试设计的盲点，为什么没有考虑到？需求/设计不够清晰，测试设计的理解有误，工程方法应用后的造成的策略性放弃等等，之后进行补充测试用例设计。

2. 检测出程序中的废代码，可以逆向反推在代码设计中思维混乱点，提醒设计/开发人员理清代码逻辑关系，提升代码质量。

3. 代码覆盖率高不能说明代码质量高，但是反过来看，代码覆盖率低，代码质量不会高到哪里去，可以作为测试自我审视的重要工具之一

所以，对于我们来说，重点关注的是核心类的还未覆盖的代码，很可能这些代码就会产生故障

三、获取代码覆盖率

市面上统计代码覆盖率的工具有很多，比如 Cobertura，Emma，Jacoco，Jmockit 等，其中 Jacoco 基于 java 字节码的原理，能够统计单元测试和集成测试时候的覆盖率，所以我以 Jacoco 为例，介绍如何在实际项目测试活动使用代码覆盖率进行统计，达到精准测试的目的从而降低每次回归的成本。

四、获取单元测试的代码覆盖率

单元测试框架我们使用的是 Junit + Jmockit + Jacoco 的经典组合，Jmockit 是非常优秀的 mock 工具，它也自带代码覆盖率的统计，但是相比 Jacoco 来说，还是太弱了。

获取单元测试覆盖率很简单，只需要在 POM 中加上 jacoco 的 maven 插件就行

在 Maven 中配置 Jacoco 统计单元测试代码覆盖率:

```
<plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.7.8</version>
            <executions>
                <execution>
                    <id>default-prepare-agent</id>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                    <configuration>
                        <destFile>
                            ${project.build.directory}/coverage-reports/jacoco-ut.exec
                        </destFile>
                        <propertyName>surefireArgLine</propertyName>
                    </configuration>
                </execution>
                <execution>
                    <id>default-report</id>
                    <phase>test</phase>
                    <goals>
                        <goal>report</goal>
                    </goals>
                    <configuration>
                        <dataFile>${project.build.directory}/coverage-reports/jacoco-ut.exec</dataFile>
                        <outputDirectory>${project.build.directory}/jacoco-ut</outputDirectory>
                    </configuration>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.18.1</version>
            <configuration>
                <useFile>false</useFile>
                <includes>
                    <include><strong>/</strong>Test.java</include>
                </includes>
                <argLine>${surefireArgLine}</argLine>
            </configuration>
        </plugin>
```

配置好以后，我们就可以通过 mvn clean test 命令执行写好的单元测试用例并自动生成覆盖率报告，覆盖率报告的二进制文件以及 html 格式文件分别在:

```
target/coverage-reports/jacoco-ut.exec
target/jacoco-ut/index.html
```

Jacoco 的二进制文件很重要，我们可以利用它提供的 API 来合并单元测试和集成测试的覆盖率二进制文件。

五、获取集成测试的代码覆盖率

这里说的集成测试包括：接口/UI 的自动化测试，手动的探索性测试

我们利用 Jacoco 的 Java agent 在产品运行时中统计在集成测试进行中的代码覆盖率，在产品代码中加 agent，我们需要修改产品的基线文件 placeholder.sh，增加一行:

```
CATALINA_OPTS="${CATALINA_OPTS} -javaagent:/home/admin/jacocoagent.jar=port=6300,output=tcpserver,address='*',includes
='com.[redacted]*'"
```

其中 jacocoagent.jar 和需要统计的 package，依据你产品实际情况修改，更多细节参考 Jacoco 的官方文档：Jacoco Agent，注意，千万不要在生产环境上加 agent，否则会引起严重的安全问题

利用 placeholder.sh 重启应用服务器后生效

六、通过 Jacoco API 实时获取集成测试后的代码覆盖率

引入 Jacoco SDK:

```
<dependency>
          <groupId>org.jacoco</groupId>
          <artifactId>org.jacoco.core</artifactId>
 </dependency>
 <dependency>
          <groupId>org.jacoco</groupId>
          <artifactId>org.jacoco.report</artifactId>
 </dependency>
```

通过 TCP Socket 从应用服务器获取覆盖率二进制文件:

```
public void dumpData() throws Exception {
    final FileOutputStream localFile = new FileOutputStream("/Users/gengcheng/coverage/jacoco-it.exec"); //二进制保存地址
    final ExecutionDataWriter localWriter = new ExecutionDataWriter(localFile);
    // Open a socket to the coverage agent:
    final Socket socket = new Socket(InetAddress.getByName("[redacted]"), [redacted]); //应用服务器地址和端口
    final RemoteControlWriter writer = new RemoteControlWriter(socket.getOutputStream());
    final RemoteControlReader reader = new RemoteControlReader(socket.getInputStream());
    reader.setSessionInfoVisitor(localWriter);
    reader.setExecutionDataVisitor(localWriter);

    // Send a dump command and read the response:
    writer.visitDumpCommand(true, false);
    reader.read();

    socket.close();
    localFile.close();
  }
```

将单元测试结果和集成测试结果合并统计:

```
public void mergeData() throws Exception {
    ExecFileLoader loader = new ExecFileLoader();
    loader.load(new File("/Users/gengcheng/coverage/jacoco-it.exec")); //集成测试二进制文件
    loader.load(new File("/Users/gengcheng/coverage/jacoco-ut.exec")); //单元测试二进制文件
    loader.save(new File("/Users/gengcheng/coverage/jacoco-merged.exec"),true); //合并后的二进制文件
  }
```

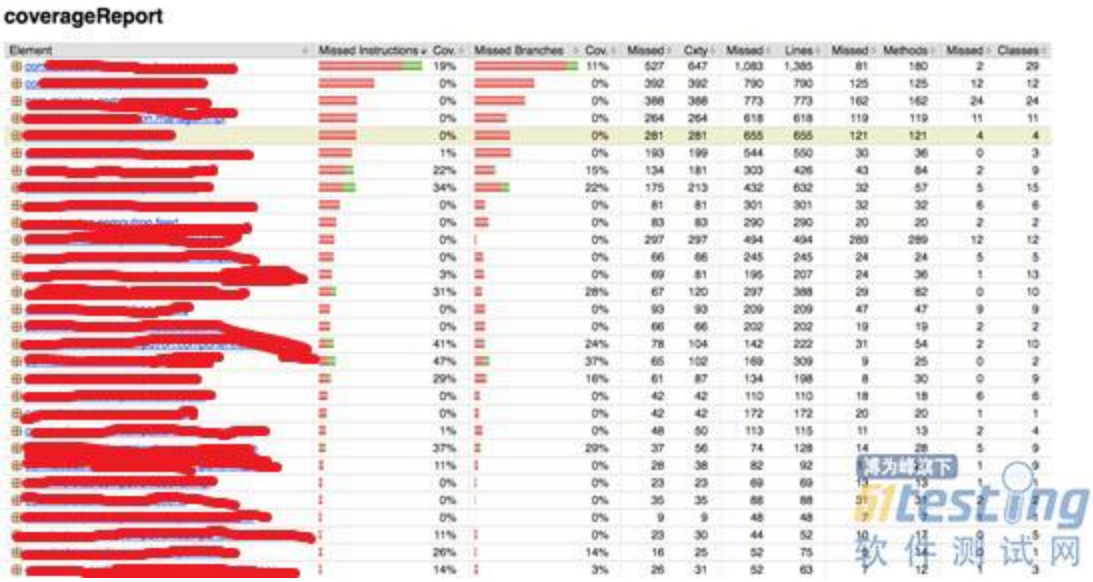
生成合并后的代码覆盖率报告

```
public void getCoverage() throws Exception{
    ExecFileLoader execFileLoader = new ExecFileLoader();
    execFileLoader.load(new File("/Users/gengcheng/coverage/jacoco-merged.exec")); //合并后的二进制文件
    final CoverageBuilder coverageBuilder = new CoverageBuilder();
    final Analyzer analyzer = new Analyzer(execFileLoader.getExecutionDataStore(), coverageBuilder);
    analyzer.analyzeAll(new File("/Users/gengcheng/core-scene/target/classes")); //被测产品编译后的classes目录
    IBundleCoverage analysisResult = coverageBuilder.getBundle("coverageReport"); //报告标题

    final HTMLFormatter htmlFormatter = new HTMLFormatter();
    final IReportVisitor visitor =
        htmlFormatter.createVisitor(new FileMultiReportOutput(new File("/Users/gengcheng/CoverageReport/"))); //HTML
格式的覆盖率报告地址
    visitor.visitInfo(execFileLoader.getSessionInfoStore().getInfos(),
        execFileLoader.getExecutionDataStore().getContents());

    visitor.visitBundle(analysisResult,
        new DirectorySourceFileLocator(new File("/Users/gengcheng/core-scene/target/classes/"), "utf-8", 4));
    visitor.visitEnd();
  }
```

注意，生成 HTML 代码覆盖率及产品源代码信息，需要产品编译后的 classes 文件，这个文件可以通过下面两种方式获取:

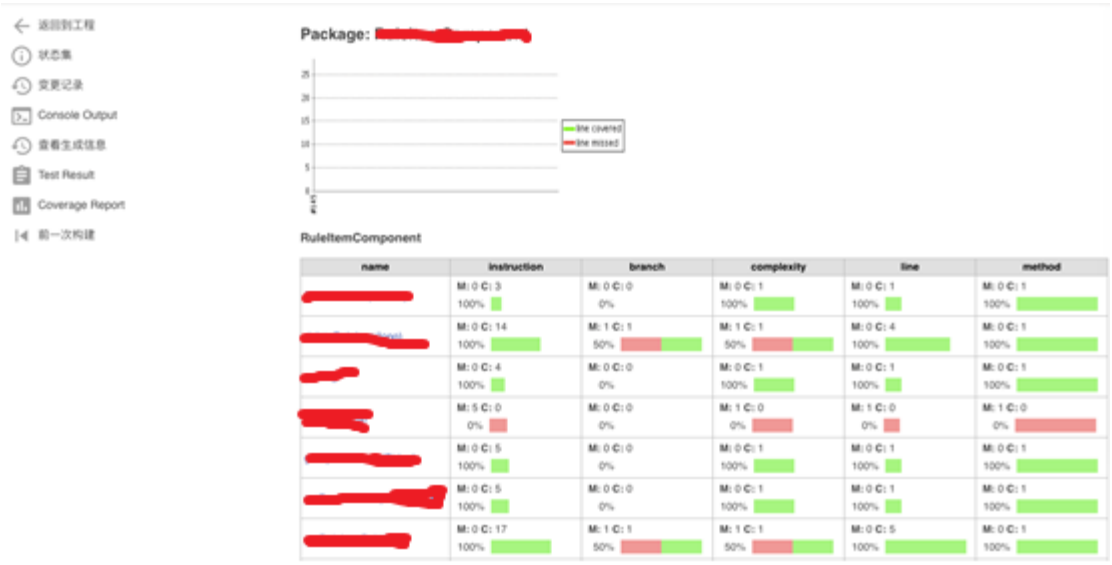
1. clone 产品代码自己编译
2. 应用服务器上下载 war 并解压，找到里面的 clasess 目录

生成后的代码覆盖率报告样式:

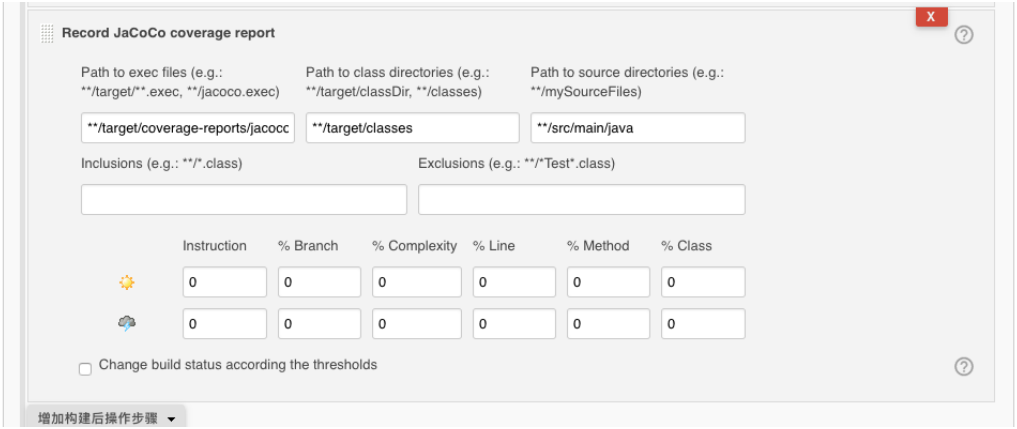


七、和 Jenkins 结合

Jenkins 上有 Jacoco 的的代码覆盖率插件，简单的配置以后就可以展现非常丰富的覆盖率信息和趋势，效果如下:



配置截图:



八、总结

通过以上的配置并结合 jenkins，我们就可以每天生成所有测试活动的代码覆盖率报告，通过这个报告，我们可以发现还未测试到的代码，通过补充单测用例和集成测试用例，尽可能全的覆盖产品代码，通过这个指标来统计测试的效率和产品质量，不再需要盲人摸象了，拍脑袋想覆盖是否全面。当然代码测试也有很多局限性和难题，在《代码级测试理念》中有也有所拓展，感兴趣可以深入了解。

现实公司环境中的实际测试过程是什么样的？

◆ 译　者：Lily

今天我从读者那里得到了一个非常有趣的问题，在公司里，也就是现实环境中，测试工作是如何开展的呢？那些刚从大学出来开始找工作的同学经常有这样的好奇心，好奇在公司这样实际的环境中测试工作是怎样进行的呢？

在本文中，我的重点将放在公司里面的软件测试过程上面。由于从事软件测试职业，日复一日的进行着测试活动，到目前为止我拥有丰富的软件测试的经验。在这里我会试着分享一些更为实际的信息而不是理论的知识。

当我们开始任何新的项目时，都会有一个初始的项目熟悉会议。在这个会议上，我们一般都会讨论一下客户是谁，项目的持续周期以及何时交付，谁是项目的总负责人，也就是项目经理，谁担任技术主管、质量保证主管、开发、测试等等。

项目的计划是根据 SRS（项目需求规范）建立起来。测试人员的主要职责是根据 SRS 和项目计划设计软件测试用例。开发人员从设计开始编码。项目工作被分割为不同的模块，这些模块被分发到开发人员之间。同时，测试人员要负责配置测试环境并依据分配的模块编写测试用例。我们试图利用 SRS 来设计覆盖几乎所有的功能点的测试用例。在这个过程中可以使用一些 excel 测试用例模板或者缺陷追踪工具来手动的维护数据。

当开发人员完成了单个模块的设计后，这些模块就会被分配给测试人员。测试人员将会在这些模块上面进行冒烟测试，如果模块未通过测试，则会将它们分配给相应的开发人员进行修复。对于那些通过冒烟测试的模块，将会依照书面的测试用例对它们展开手动测试。在这个过程中如果发现了任何缺陷，这个模块将会被分配给模块开发人员进行修复，同时测试人员登录缺陷跟踪工具对发现的缺陷进行记录。测试人员在缺陷修复

测试仪上进行缺陷的确认及所有相关模块的回归测试。如果缺陷通过了验证后，会标记为已验证和已关闭，如果没有通过验证，那么上述提到的缺陷周期将会重复进行（缺陷的生命周期将在其他的文章中进行说明）。首先在单个的模块上执行不同的测试，然后在集成的模块上进行集成测试。这些测试包含了兼容性测试，比如说使用不同的硬件，不同版本的操作系统，软件平台，不同的浏览器等等。同时还要通过 SRS 进行压力测试和负载测试。最后，将会搭建一个虚拟的客户环境进行系统测试。当软件通过了所有的这些测试用例，测试报告就形成了，也是时候下决定发布产品了。

以下是一个项目生命周期过程的简单描述。

这是根据 IEEE 和 ISO 标准写出的在每一个软件质量和测试生命周期中进行的测试活动的详细步骤:

1) 回顾软件的需求文档

2) 设定主要版本的目标

- **计划软件发布的目标日期**

制定详细的项目计划。这个详细的项目计划包含了设计规范

- **根据设计规范制定测试计划**

测试计划：测试计划包含了测试的目标和测试使用的方法，功能测试点和非功能测试点，测试风险标准，测试日程安排，多平台支持和测试的资源配置。

- **测试规范**

测试规范文档包含的技术细节（软件需求）需要在测试前给出

- **编写测试用例**

1) 冒烟（BVT）测试用例

2) 理智的测试用例

3) 回归测试用例

4) 负面的测试用例

5) 扩展的测试用例

6) 开发-分模块的开发方式

7) 安装程序绑定：安装程序是依据每个独立的产品构建的

- **构建过程:**

1) 一个构建包含安装多平台的可用产品
2) 测试
3) 冒烟测试（BVT）是一种基本的应用程序测试，它决定了进一步的测试方向
4) 新功能的测试
5) 跨平台的测试
6) 压力测试以和内存泄露测试

- **缺陷报告**

1) 创建缺陷报告
2) 开发-代码冻结
3) 在某个节点上不增加新的功能
4) 测试
5) 构建和回归测试
6) 决定发布版本
7) 长期对象发布后的场景

《51 测试天地》（四十四）下篇 精彩预览

马上阅读