基于 Selenium 的 OpenStack Horizon 自动化测试的实现

龚智勇
（武汉邮电科学研究院 武汉 430074）

摘　要：基于提高对 OpenStack Horizon 的基本功能进行频繁测试的测试效率，节省测试成本的目的，通过研究 Selenium 测试工具进行 Web 自动化测试的原理和方法，在其基础上对其 WebDriver 组件的部分 API 进行封装，使用 PyUnit 框架编写测试用例并整合成测试用例集，通过单独的配置文件实现代码与数据相分离，并将元素定位全部放在单独文件，最后引入 HTMLTestRunner 模块用于生成 html 格式测试报告，从而形成了完整的自动化测试系统。通过将此系统应用于 Horizon 的自动化测试，能够更快地完成预期的测试任务并得到相应的测试报告，实现了对 OpenStack 的自动化测试，测试效率提高了 50％以上。

关键词：自动化测试；实现；Horizon；测试框架
中图分类号：TN06　　**文献标识码**：A　　**国家标准学科分类代码**：520.40

Implementation of automated testing for OpenStack Horizon based on selenium

Gong Zhiyong
（Wuhan Research Institute of Posts and Telecommunications, Wuhan 430074, China）

Abstract: Based on the purpose of improving the test efficiency of the basic functions of OpenStack Horizon and saving the cost of testing, some of the APIs of WebDriver components are encapsulated on the basis of studying the principle and method of Web automation test by Selenium testing tools. The PyUnit framework prepares the test case and integrates it into test case suites, separating the code from the data through a separate configuration file, placing the element in a separate file, and finally introducing the HTMLTestRunner module to generate the html format test report, resulting in a complete Automated test system. By applying this system to Horizon's automated testing, it is possible to complete the expected test tasks faster and get the corresponding test reports, enabling automated testing of OpenStack and improved test efficiency by more than 50%.

Keywords: automated testing; implementation; Horizon; test framework

1　引　言

OpenStack 作为影响最大的开源云计算平台，受到很多公司的欢迎。在对 OpenStack 云平台开发的过程中，需要公司投入大量人力物力对 Horizon 进行全方位的测试[1]，通常测试采用的是手工进行，不仅耗时耗力，而且由于测试人员个体的差异经常导致测试结果存在很大差异，因此很有必要实现测试的自动化。

测试的自动化离不开测试工具的支持，为此选择了最受欢迎的开源自动化测试工具 Selenium[2]。由于直接使用 Selenium WebDriver 的 API 进行测试可能会因为页面元素定位超时而失败，也会因为被测系统页面元素的变更导致需要修改大量测试用例代码，并且生成的测试结果很不直观，不便于定位测试中发现的问题。

基于上述不足，在 Selenium 的基础上，对 WebDriver 的 API 进行了封装，只有页面找到了目标元素才会进行下一步的操作，否则会进行等待；同时将页面元素的定位信息放在单独的文件中，在用例中直接调用相应的元素定位信息，避免将定位信息放在测试用例代码中，从而当页面布局发送变化时，只需修改这一个文件中的定位信息，

收稿日期：2017-02

提高了系统的适用性；另外还将测试数据放在单独的配置文件中，实现测试数据与测试代码相分离，只需修改配置文件中的数据便可执行不同的测试用例，提高了代码的利用率；最后还引入了 HTMLTestRunner 模块，可以生成美观的 HTML 格式的测试报告，列出了所有测试用例及测试结果，并且给出了出错的信息以便于定位错误。

通过这些改进措施，使得本系统更加适用于对 Horizon 的自动化测试。在实际测试中，不仅提高了测试效率，节省了大量时间，而且由于测试标准统一，测试结果更具说服力。同时还可以经常利用晚上的时间频繁进行测试，验证被测系统的可靠性和稳定性。

2 测试系统设计

2.1 Selenium WebDriver 简介

Selenium 是由 ThoughtWorks 公司开发的主要用于 Web 应用的自动化测试框架[3]，它的测试能够直接运行在浏览器中，就像真正的用户在操作一样。其支持的浏览器包括 IE、Firefox、Chrome、Opera 等，它支持自动录制动作，并且能够自动生成 Net、Java、Perl、Python 等不同语言的测试脚本[4]。目前使用的是 Selenium 2.0 系列（由 Selenium 1.0 和 WebDriver 合并而成），其主要包含 Selenium IDE、Selenium Grid、Selenium RC（Remote Control）、WebDriver 等组件[5]。

本测试系统主要使用了 WebDriver 组件，它是基于客户端/服务器模式设计的。客户端也就是进行测试的代码，测试代码中的对页面的各种操作，比如打开浏览器，转跳到特定的 URL，点击某个按钮，填写文本框，勾选选择框等都是通过 HTTP 请求的方式发送给浏览器[6]。

服务器端也就是 Remote Server，任何浏览器均可作为服务器端。当脚本通过 WebDriver 启动浏览器后，这个浏览器就是本测试的 Remote Server，它会一直等待接收客户端发送的请求并执行相应操作，在 Response 中返回执行状态和返回值等信息[7]。

WebDriver 测试流程如图 1 所示，测试过程会经历以下 4 个步骤。

图 1 WebDriver 测试流程

1）当执行测试代码时，客户端（测试用例）通过 WebDriver 的驱动会在一个新的线程中启动代码中指定的浏览器；

2）浏览器启动成功后，WebDriver 会将浏览器绑定到特定的端口上，该浏览器实例便可以作为 WebDriver 的 Remote Server，也就是作为服务器端，这时客户端和服务器端之间便建立了连接；

3）客户端创建 1 个 Session 会话，在该 Session 会话中向服务器发送 HTTP 请求，服务器收到请求后会对其进行解析，完成浏览器页面的相应操作并向客户端返回响应；

4）客户端接收到响应之后，对其进行分析，接着进行下一步的操作，当执行至断言判断语句时，则会将分析结果与预期结果相比较，判断测试是否成功[8]。

2.2 系统框架设计

由于 Selenium 是开源软件，使用较为灵活，但是若直接使用 Selenium WebDriver 的 API 进行测试也会有一些不足。测试时可能会因为页面元素定位超时而失败，也会因为被测系统页面元素布局的变更导致需要修改大量测试用例代码，并且生成的测试结果很不直观，不便于定位测试中发现的问题。

因此系统在 Selenium 的基础上，同时结合被测系统 Horizon 的特点，进行了有针对性的改进。首先根据实际情况对 WebDriver 的部分 API 进行了封装，便于使用；之后基于 PyUnit 框架编写测试用例，在测试用例中调用封装后的方法，用例中的数据则通过读取配置文件获得；而通过使用 PyUnit 的 TestSuite 测试套件类可将大量分散的测试用例根据功能模块予以整合，便于测试和维护；最后则是为了直观查看测试报告而引入了 HTMLTestRunner 模块，生成 html 格式测试报告，系统框架如图 2 所示。

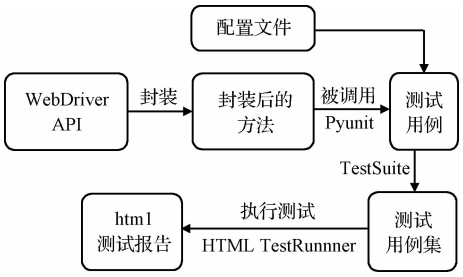

图 2 测试系统框架

3 测试系统核心模块实现

本测试系统在 Selenium 的基础上，实现了测试数据与代码相分离和页面元素集中定位，避免了页面元素定位超时，并能够生成 HTML 格式的测试报告。

3.1 测试数据与代码相分离的实现

在本测试系统采用了数据驱动模式，将测试数据与代码相分离，也就是将登录的用户名、密码、被测页面 url 以及页面各种文本框待输入的内容等各种基本数据全部保存在 config.cfg 配置文件中，而不是直接把数据写在代码中，在代码中通过读取配置文件来获取数据。使得当需要对其它环境进行测试或测试数据发生变化时，只需修改配置文件，而不需修改代码，方便了代码的后期维护。config.cfg 中的部分配置信息如下：

```
[server]
url=http://10.127.2.31/auth/login/
waittimes=5
picdir=pictures
```

```
cases = Snapshots, Volumes, Images, Networks, Instances
[user]
user=admin
passwd=Abc12345
```

在 base.py 中实现了对配置文件的读取操作：

```
conf = ConfigParser.ConfigParser()
conf.read('config.cfg')
```

在配置文件中，每个配置项由一个键值对构成，各个配置项又根据类型或功能进行分组，一个组即为一个 section，用中括号([])予以区分。上述配置文件中，在 server 部分配置了系统登录页面的 url、执行命令过程中等待时间、截图放置文件夹及需要进行测试的用例等；在 admin 部分定义了系统管理员的登录用户名和密码；user 部分定义了一个普通用户的用户名和密码。

从配置文件读取需要测试的用例集(这些用例集在配置文件中列出，以“,”分隔)：

```
test_cases = config.server.cases.split(',')
```

由于用例模块的名称在配置文件中是通过逗号分隔的，在读取时使用了 split()函数并以逗号为分割符来分割用例模块，返回一个列表，下一步则可遍历这个列表来对所有用例执行测试。

虽然上述将测试数据放在单独的配置文件中，在测试用例中读取需要的测试数据比直接将测试数据放在测试用例代码中看起来要复杂一些，但是这种处理方式却有很大的优点，主要如下：

1)在实际测试中，会针对多个项目进行测试，不同项目的环境信息不同，这时不必在代码中一个个地去寻找和修改相关信息，而只需简单的修改配置文件中的 url 等信息即可，方便快捷且不会因为部分数据遗漏未改而导致测试出错；

2)由于 Horizon 中对用户有着不同的角色权限划分，最常使用的是系统管理员、租户管理员和普通用户这 3 个角色，不同角色对应的功能也有一些区别，在测试时为了分别测试不同角色的功能，因此需要切换不同的用户，这时便可通过简单地修改配置文件中的用户名密码等信息而快速切换不同的用户进行测试；

3)如果某次开发人员对某个模块的代码进行了修改，仅需要对这部分的功能进行验证测试时，则可修改配置文件 server 部分的 cases 的值，只填写需要测试的功能模块即可完成对指定模块的测试，快速得到测试结果，而不必等待较长时间进行全部功能的测试。

3.2 页面元素集中定位的实现

Selenium 能够模仿手工对页面的各种元素的基本操作，前提是必须能够使用某种方式对页面的各种元素进行定位，这样才能让程序对相应元素进行操作。目前，常见的定位方式主要有 id、name、tag、link、XPath、CSS 等[9]，可根据实际情况选择不同的定位方式，主要选择了 id 定位和 XPath 定位两种方式。

在开发过程中，Web 页面可能会发生局部调整变化，导致部分页面元素也可能发生变化，之前对该元素的定位方式也可能失效了[10]。为了应对 Web 页面发生变化时导致的元素定位变化问题，将所有页面元素的定位全部集中在 locators.py 中，在测试用例中通用调用来完成元素定位，而不是分别在所有测试用例中对元素进行定位，避免了页面变化时需要修改所有涉及的测试用例，而只需对 locators.py 中的相应元素的定位进行更改，极大提高了自动化测试系统的可维护性[11]。在 locators.py 中对于各种元素的定位是：

```
username = (By.ID, 'id_username')
passwd = (By.ID, 'id_password')
button_xpath = '//button[text() = "%s"]'
label_xpath = '//label[text() = "%s"]'
option_xpath = '//option[text() = "%s"]'
```

其中登录的用户名和密码均直接采用了 id 定位，方便快捷；而页面其他的按钮、标签(文本框)、选择框等元素均采用了 XPath 定位，通过 text()函数得到元素的文本信息予以定位，若页面中有多个相同文本时，会在调用时加上相应的数字表示第几个元素[12-13]。

对于定位元素的调用，此处以登录功能为例予以说明。在 basecontrol.py 文件中有如下代码：

```
from locators import LoginPageLocators as LPL
def login(self, username, passwd):
    self.driver.find_element(LPL.username).
        send_keys(username)
```

代码中 LPL.username 即为 locators 中的 username，通过 id 定位找到登录页面用户名文本框，send_keys()的参数为 login 方法传入的用户名参数，作为文本框的输入值。在测试用例中通过读取配置文件的用户名密码来调用此 login 方法完成 Horizon 页面的登录，即可接着进行下一步的测试。

3.3 页面元素定位超时的避免

本系统对页面元素的基本操作均通过 WebDriver 提供的 API 进行[14]。为了使用方便，同时也为了避免定位页面元素超时导致的测试失败，选择了对 WebDriver 的部分常用的 API 进行封装，得到了页面元素基本操作常用的方法，其中一个方法示例如下：

```
def click_button_by_text(self, type_button, index = 0):
        button_element = WebDriverWait(
            self.driver, int(self.config.server.
waittimes)).until(lambda driver:
            driver.find_elements_by_xpath(
CL.button_xpath %type_button))
        button_element[index].click()
```

该方法通过设置等待时间避免了定位页面元素超时导致的测试失败。首先会实例化 WebDriver 中的 Web-

DriverWait 类，该类的第一个参数为 driver，第二个参数为 timeout，此处会从配置文件读取 waittimes 的值（在配置文件中据实际测试环境的网络状况更改，例如可设置为 10 s）作为参数，然后调用该类的 until()方法，以一个 lambda 匿名函数作为参数，以 XPath 定位方式在 10 s 内，默认每隔 0.5 s 进行一次检查，不断查找目标元素，找到了就立即进行下一步操作而不管等待时间是否达到了 10 s，若 10 s 后还没有找到目标元素，则会抛出 TimeoutException 异常。

该方法实现了根据按钮的名称来定位并点击该按钮这些操作，方法传入的第一个参数为按钮名称，用于定位按钮。与此同时，根据被测系统的实际情况，例如在虚拟机列表中有多台虚拟机时，列表每行右侧均有一个名称相同的按钮由于对虚拟机进行操作，则需要一个参数进行选择，故引入了第二个参数，用于当页面中有多个按钮具有相同名称时选择点击哪一个按钮。

之后便可基于 Python 单元测试框架 PyUnit 来编写测试用例，调用这些封装后的方法来对页面元素进行基本的操作时，会等待页面加载完全，找到目标元素才会进行下一步的操作，避免了因为未找到目标元素而导致的测试失败，降低了非被测系统自身 Bug 而导致的失败的概率，使得测试结果更可信。

3.4 HTML 格式测试报告的生成

对于测试结果，Selenium 默认只在命令行显示，对于后期查看及分析测试结果很不方便，因此在本自动化测试系统中引入了 HTMLTestRunner 模块，用于生成 html 格式的测试报告[15]。将下载的 HTMLTestRunner.py 文件放在 python 的 Lib 目录下[16]，同时在 testRunner.py 中进行了如下定义：

```
suite = unittest.TestSuite()
test_cases = config.server.cases.split(',')
for test in tests:
    suite.addTest(test)
now_time = time.strftime("
%Y-%m-%d-%H_%M_%S",time.localtime(time.
time()))
filename = 'E:\python\selenium\webControl\
results\'+now_time+'_report.html'
  fp = file(filename, 'wb')
        runner = HTMLTestRunner.
HTMLTestRunner(
      stream=fp,
      title=u'OpenStack Horizon 测试报告' )
runner.run(suite)
```

首先会从配置文件中读取所有需要测试的用例，通过使用 PyUnit 框架的 TestSuite 测试套件类将大量零散的测试用例组织起来，统一进行管理和测试，最后便可一次性执行用例集中的所有用例。

对于测试结果的收集和测试报告的生成，首先会调用 time 模块的 time.localtime(time.time())函数来获取并返回当前进行测试的时间，并通过 strftime()方法进行自定义格式化，得到想要的时间显示格式；然后指定测试报告的存放路径和报告名称，此测试最终会生成名称形如 2016-11-01-15_37_59_report.html 所示的测试报告；之后则是实例化 file 类，向测试报告文件中写入测试数据及定义测试报告标题等信息；最后调用该模块的 run()方法，将所需测试的测试用例组成的测试用例集 TestSuite 对象作为方法的参数，执行测试，并且返回 TestResult 结果对象[17]，得到测试报告，对测试用例执行情况进行了较为直观的统计，并给出了消息的出错信息便于定位问题。

4 结果分析

通过运行 testRunner.py 执行测试，可以得到 html 格式的测试报告，在测试报告中，对测试用例执行情况进行了统计，并且对未通过的测试用例还给出了报错信息，便于分析定位失败原因，如图 3 所示。

OpenStack Horizon测试报告

Start Time: 2016-11-01 15:37:59
Duration: 0:01:52.143000
Status: Pass 1 Error 1

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
allcases.Volumes.CreateVolume.CreateVolume	1	1	0	0	Detail
test_create_volume			pass		
allcases.Snapshots.CreateSnapshot.CreateSnapshot	1	0	0	1	Detail
test_create_snapshot	error [x] ft1.1: Traceback (most recent call last): File "E:\python\selenium\allcases\Snapshots\CreateSnapshot.py", line 18, in test_create_snapshot self.control.click_link_by_text('云硬盘') File "E:\python\selenium\basecontrol.py", line 89, in click_link_by_text self.driver, int(self.config.server.waittimes)).until(lambda driver: File "E:\python\selenium\selenium\webdriver\support\wait.py", line 80, in until raise TimeoutException(message, screen, stacktrace) TimeoutException: Message:				
Total	**2**	**1**	**0**	**1**	

图 3 测试报告示例

在测试报告示例中，共进行了2个用例的测试，创建卷和快照。其中，创建卷成功了，创建快照失败了，给出了异常信息，方便定位错误原因。

通过测试过程的顺利进行和测试报告的正确呈现，说明实现了针对OpenStack的自动化测试功能，设计的自动化测试系统基本达到了预期目的，能够用于被测系统的自动化测试。之后还将此自动化测试系统应用到了公司的实际测试工作中，能够正常执行基本测试，并能及时发现被测系统存在的问题，并且可以利用晚上的空闲时间执行测试，节省了大量时间，体现了本自动化测试系统的实用性。自动化测试与手工测试的对比如表1所示。

表1 Selenium自动化测试和手工测试对比

测试方法	手工测试	Selenium自动化测试
学习成本	低	高
单次测试花费时间（50个用例）	3～4 h	2 h内
是否可重复使用	否	是
测试覆盖范围	无限制	有限制，部分用例无法实现自动化

虽然自动化测试需要的学习成本比手工测试高，对测试人员的水平有一定的要求，并且不是所有的测试用例都可实现自动化。但是自动化测试存在着很大的优点，经过试验，进行50个测试用例的测试时，手工测试视测试人员熟练程度通常需耗费3～4 h，而自动化测试耗费时间在2 h以内，而且这2 h中，只需在开始测试前进行配置文件修改，测试过程中基本不需人工参与，因此测试人员耗费的时间只有几分钟，极大节省了人力。

此外，对于手工测试，每进行一次都要花费大量的时间，而自动化测试只在前期需花费一定的时间用于设计测试系统框架和编写测试用例，后期只需花费少量的时间进行维护，代码可重复使用进行测试，从而提高了后期的测试效率，节省了时间。

5 结 论

实现了OpenStack Horizon基本功能的自动化测试，通过将测试数据与代码向分离，当测试页面发生变化时只需进行配置文件和元素定位文件的修改，而不必对所有涉及的用例代码进行修改，提高了测试代码的可维护性。同时引入了HTMLTestRunner模块，用于生成html格式的测试报告，便于直观统计和分析测试结果，从而使得本自动化测试系统更具实用性。

参 考 文 献

[1] 黄凯，毛伟杰，顾骏杰. OpenStack实战指南[M]. 北京：机械工业出版社，2014.
[2] 戢友. OpenStack开源云王者归来[M]. 北京：清华大学出版社，2014.
[3] 张子凡. OpenStack部署实践[M]. 北京：人民邮电出版社，2014.
[4] 丁小盼，周浩，贺珊，等. 基于OpenStack的云测试平台及其性能分析研究[J]. 软件，2015，36(1)：6-10.
[5] 秦海光. 基于Selenium自动化测试框架的改进与应用[D]. 北京：中国科学院大学，2014.
[6] 温素剑. 零成本实现Web自动化测试—基于Selenium和Bromine[M]. 北京：电子工业出版社，2011.
[7] 关春银，王林，周晖，等. Selenium测试实践—基于电子商务平台[M]. 北京：电子工业出版社，2011.
[8] 吴海峰，詹文法，程一飞. 独立于测试数据的字典编码方法[J]. 电子测量与仪器学报，2016，30(4)：638-644.
[9] 陈能技. 软件自动化测试成功之道—典型工具、脚本开发、测试框架和项目实战[M]. 北京：人民邮电出版社，2010.
[10] 齐永龙，宋斌，刘道煦. 国外自动测试系统发展综述[J]. 国外电子测量技术，2015，34(12)：1-4.
[11] 黄华林. 使用Selenium进行Web应用自动化测试的研究[J]. 电脑开发与应用，2012，25(4)：54-56.
[12] 张添. 基于Selenium的Web自动化测试[D]. 北京：北京交通大学，2014.
[13] 张玲. Web应用自动化测试框架的研究和应用[D]. 上海：华东理工大学，2014.
[14] 陈琪. 自动化测试平台的设计与实现[D]. 西安：西安电子科技大学，2014.
[15] 石敏. 面向Web网页的自动化测试技术研究[D]. 上海：东华大学，2014.
[16] 樊付星，黄大庆，周未，等. 基于Web的自动化测试框架的研究与实现[J]. 电子设计工程，2012，20(1)：126-128.
[17] 吴伶琳. 基于Selenium的软件自动化测试的研究与应用[J]. 计算机与现代化，2013(2)：113-116.

作 者 简 介

龚智勇，1991年出生，硕士研究生，主要研究领域为云计算。
E-mail：gzy833231819@163.com