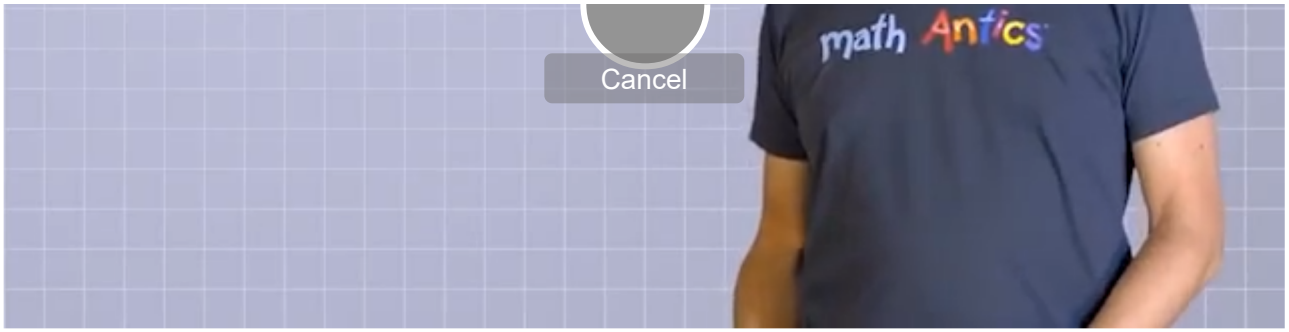
Selenium Automated Testing Python 5: WebDriver Design Pattern



WebDriver design pattern

Welcome to the advanced lectures on WebDriver. This handout will focus on the design of the Selenium WebDriver automation framework, focusing on the Page Object design pattern, and using HTML test reports and integration test reports to automatically send mail.

Page Object design mode

Before discussing the design pattern, let's start with the previous lecture to discuss the automated testing framework.

What is a framework?

A framework is a reusable design of a whole or part of a system, represented by a set of abstract components and methods of interaction between component instances; another definition is that the framework is an application skeleton that can be customized by the application developer. The former is from the application side and the latter is the definition given from the purpose. It can be said that a framework is a reusable design component that specifies the architecture of the application, clarifies the entire design, the dependencies between the collaborative components, the assignment of responsibility and the control flow, represented as a set of abstract classes and their instances. A method of collaboration that provides a contextual relationship for component reuse. Therefore, large-scale reuse of component libraries also requires a framework. In fact, so far, the framework has not been uniformly defined. I prefer the definition given by Ralph Johnson:

TOP

> A framework is a reusable design that is expressed by a set of abstract classes and their collaboration between instances [Johnson 98]. This definition defines the framework from the perspective of the connotation of the framework. Of course, the definition of the framework can be given from the perspective of the use of the framework: a framework is a part of the design and implementation of an application in a given problem domain [Bosch 97].

Why use a framework?

It is a matter of course. Because the development of software systems is very complicated today, especially the server-side software, the knowledge, content, and problems involved are too many. In some ways, using a mature framework is equivalent to letting others do some basic work for you. You only need to concentrate on the business logic design of the system. And the framework is generally mature and robust, he can handle many details of the system, such as transaction processing, security, data flow control and so on. There are also frameworks that are generally used by many people, so the structure is very good, so the scalability is also very good, and it is constantly upgraded, you can directly enjoy the benefits of other people's upgrade code.

Why build an automated test framework?

I used to think that the most important thing in automated testing is to find the object (To Find Test Object). Now I understand the truth. Automated testing without a framework can't find an object, even if it is found, it won't be happy. Just like in reality, people who don't have a car without a house are hard to find an object.

The development of automated testing is usually determined by the needs of automated testing. This demand mainly includes:

- Automated testing is easier to implement. The point is that it is convenient for you to write test scripts.

  A good automated testing framework is for people who don't know the technology and can write

TOP

automated test scripts.

- Solve problems with automated test scripts such as exception handling and scene recovery.
- Testing is easy to maintain. Automated testing projects, basically without good management and maintenance, must be a big pit. I can say with great responsibility that automated testing is not available for a year and a half, and you can't see the output. Therefore, management and maintenance have become the most important thing. A good framework can reduce the manpower and resources you put into management and maintenance.
- Reusability. One of the meanings of the framework is that it can be reused. So in the framework, you can implement some common functions to simplify the script development process.
- Beautiful and easy to read test report. For Selenium, the test report it produces is based on test scripts, and there is no such report based on the test set, so if you want, the test framework can be implemented.

There are still a lot of testing requirements, I can't list them one by one, and most of the requirements can be customized in the testing framework. Now you can answer the above question, record & playback is not happy, you need an automated test framework.

Please carefully consider whether you need automated testing

Features of automated testing (high cost and high risk)

TOP

Automated testing is a very proud thing, it is a very pick project. First, the project cycle is long, but the requirements are not changed frequently; secondly, most objects in the system can be identified, and there are no large number of third-party plugins. And you have to be clear, you can't expect automated testing to help you discover new bugs, and automated testing itself is not imaginative (as opposed to manual testing). Its strength lies in iterative iteration, and its value is based on long-term regression testing to ensure long-term stable version updates of the tested products.

With regard to the entry point for automated testing, it is usually necessary to have the basic conditions for introducing automated testing after a complete system test.

At present, the automated test success stories I have done have good management and excellent testing framework. The lack of both, automated testing will inevitably become a big pit. The cost of filling the pit is very high.

Page Object Design Principle

The Page Object design pattern is one of the best design patterns for the Selenium automated test project, emphasizing the separation of test, logic, data, and drivers.

Page Object mode is a test design pattern in Selenium. It mainly designs each page as a Class, which contains the elements (buttons, input boxes, titles, etc.) that need to be tested in the page, so that it can be passed in the Selenium test page. Calling the page class to get the page element, this cleverly avoids the need to change the test page code when the page element id or position changes. When the page element id changes, you only need to change the properties of the page in the test page Class.

Its benefits are as follows:

- Centrally manage element objects
- Centrally manage public methods within a page
- Easy maintenance in the later stage

TOP

Object of Page Object

1. WebDriver package
    1. Here is the package code for Selenium, the basic package code after the package is completed.
2. Page base class
    1. A basic Page class is designed so that all pages inherit, which identifies the basic and public functions of a sub page class.
3. Sub Pages(s) subclass
    1. The specific page class defines the functionality of a particular page.
4. Tests class
    1. This section describes the specific test cases.
5. Define Test Suite
    1. Multiple test cases are added to a Test Suite and executed together.
6. Define Test Runner
    1. Design the test Runner, open the entire test, and generate an HTML test report for the test results, and send it to the specified mailbox by mail.

TOP

7. Define the main entry for the test

    1. Define the main entry class for the test, the entry to the code

HTML test report

HTML test report needs to introduce HTMLTestRunner

```
from ranzhiWeekend import HTMLTestRunner
```

HTMLTestRunner is based on Python 2.7. Our course notes are based on Python 3.x, so you need to make some changes to this file.

The sample code for the test is as follows

```
                #declare a test suite
        suite = unittest.TestSuite()
                 # Add test cases to test suites
        suite.addTest(RanzhiTests("test_ranzhi_login"))

                 # Create a new test result file
        buf = open("./result.html", "wb")

                 # declare the object that the test runs
        runner = HTMLTestRunner.HTMLTestRunner(stream=buf,
                                               title="Ranzhi Test Result",
                                               description="Test Case Run Result")
                 # Run the test and generate the result as HTML
        runner.run(suite)

```

TOP

```
                # Close file output
        buf.close()
```

Integration test report

The script for sending HTML test reports using email is as follows

```
                #Open test report results
        f = open("./result.html", "rb")

                # Put the test results in the body of the message
        mailBody = f.read()
                # Close the test result file
        f.close()

                # Declare a mail object with the body of the mail you just got
        msg = MIMEText(mailBody, "html", "utf-8")
                # Set the subject of the message
        msg["subject"] = Header("Automation Test Result", "utf-8")

                # Create an SMTP service object
        # simple message transfer protocol
                #Simple message transfer protocol
        smtpMail = smtplib.SMTP()

                #Connect SMTP server
        smtpMail.connect("mail.51testing.com")

        # Login SMTP server
        smtpMail.login("liutingli@51testing.com", "123456789")

                # Send mail using SMTP server
```

TOP

```
        smtpMail.sendmail("liutingli@51testing.com", targetEmail, msg.as_string())

                # Exit SMTP object
        smtpMail.quit()
```

Automated test framework example

WebDriver package

```
# coding=utf-8
from selenium import webdriver
from selenium.webdriver.support.select import Select


class AutomateDriver(object):
    """
    a simple demo of selenium framework tool
    """

    def __init__(self):

        driver = webdriver.Firefox()
        try:
            self.driver = driver
        except Exception:
            raise NameError("Firefox Not Found!")


    def clearCookies(self):
        """
        clear all cookies after driver init
        """
```

TOP

```
        self.driver.delete_all_cookies()

    def refreshBrowser(self):
        self.driver.refresh()

    def maximizeWindow(self):
        self.driver.maximize_window()

    def navigate(self, url):
        self.driver.get(url)

    def quitBrowser(self):
        self.driver.quit()

    def closeBrowser(self):
        self.driver.close()

    def getElement(self, selector):
        """
        to locate element by selector
        :arg
        selector should be passed by an example with "i,xxx"
        "x,//*[@id='langs']/button"
        :returns
        DOM element
        """
        if ',' not in selector:
            return self.driver.find_element_by_id(selector)
        selector_by = selector.split(',')[0]
        selector_value = selector.split(',')[1]

        if selector_by == "i" or selector_by == 'id':
            element = self.driver.find_element_by_id(selector_value)
        elif selector_by == "n" or selector_by == 'name':
```

TOP

```
            element = self.driver.find_element_by_name(selector_value)
        elif selector_by == "c" or selector_by == 'class_name':
            element = self.driver.find_element_by_class_name(selector_value)
        elif selector_by == "l" or selector_by == 'link_text':
            element = self.driver.find_element_by_link_text(selector_value)
        elif selector_by == "p" or selector_by == 'partial_link_text':
            element = self.driver.find_element_by_partial_link_text(selector_value)
        elif selector_by == "t" or selector_by == 'tag_name':
            element = self.driver.find_element_by_tag_name(selector_value)
        elif selector_by == "x" or selector_by == 'xpath':
            element = self.driver.find_element_by_xpath(selector_value)
        elif selector_by == "s" or selector_by == 'selector_selector':
            element = self.driver.find_element_by_css_selector(selector_value)
        else:
            raise NameError("Please enter a valid type of targeting elements.")

        return element

    def type(self, selector, text):
        """
        Operation input box.

        Usage:
        driver.type("i,el","selenium")
        """
        el = self.getElement(selector)
        el.clear()
        el.send_keys(text)

    def click(self, selector):
        """
        It can click any text / image can be clicked
        Connection, check box, radio buttons, and even drop-down box etc..
```

TOP

```
        Usage:
        driver.click("i,el")
        """
        el = self.getElement(selector)
        el.click()

    def selectByIndex(self, selector, index):
        """
        It can click any text / image can be clicked
        Connection, check box, radio buttons, and even drop-down box etc..

        Usage:
        driver.select_by_index("i,el")
        """
        el = self.getElement(selector)
        Select(el).select_by_index(index)

    def clickByText(self, text):
        """
        Click the element by the link text

        Usage:
                Driver.click_text("News")
        """
        self.getElement('p,' + text).click()

    def submit(self, selector):
        """
        Submit the specified form.

        Usage:
        driver.submit("i,el")
        """
```

TOP

```
        el = self.getElement(selector)
        el.submit()

    def executeJs(self, script):
        """
        Execute JavaScript scripts.

        Usage:
        driver.js("window.scrollTo(200,1000);")
        """
        self.driver.execute_script(script)

    def getAttribute(self, selector, attribute):
        """
        Gets the value of an element attribute.

        Usage:
        driver.get_attribute("i,el","type")
        """
        el = self.getElement(selector)
        return el.getAttribute(attribute)

    def getText(self, selector):
        """
        Get element text information.

        Usage:
        driver.get_text("i,el")
        """
        el = self.getElement(selector)
        return el.text

    def getDisplay(self, selector):
```

TOP

```
        """159        Gets the element to display,The return result is true or false.

        Usage:
        driver.get_display("i,el")
        """
        el = self.getElement(selector)
        return el.is_displayed()

    def getTitle(self):
        '''
        Get window title.

        Usage:
        driver.get_title()
        '''
        return self.driver.title

    def getUrl(self):
        """
        Get the URL address of the current page.

        Usage:
        driver.get_url()
        """
        return self.driver.current_url

    def acceptAlert(self):
        '''
            Accept warning box.

            Usage:
            driver.accept_alert()
            '''
```

TOP

```
        self.driver.switch_to.alert.accept()

    def dismissAlert(self):
        '''
        Dismisses the alert available.

        Usage:
        driver.dismissAlert()
        '''
        self.driver.switch_to.alert.dismiss()

    def implicitlyWait(self, secs):
        """
        Implicitly wait. All elements on the page.

        Usage:
        driver.implicitly_wait(10)
        """
        self.driver.implicitly_wait(secs)

    def switchFrame(self, selector):
        """
        Switch to the specified frame.

        Usage:
        driver.switch_to_frame("i,el")
        """
        el = self.getElement(selector)
        self.driver.switch_to.frame(el)

    def switchDefaultFrame(self):
        """
        Returns the current form machine form at the next higher level.
        Corresponding relationship with switch_to_frame () method.
```

TOP

```

        Usage:
        driver.switch_to_frame_out()
        """
        self.driver.switch_to.default_content()

    def openNewWindow(self, selector):
        '''
        Open the new window and switch the handle to the newly opened window.

        Usage:
        driver.open_new_window()
        '''
        original_windows = self.driver.current_window_handle
        el = self.getElement(selector)
        el.click()
        all_handles = self.driver.window_handles
        for handle in all_handles:
            if handle != original_windows:
                self.driver._switch_to.window(handle)
```

Base Page class

```
class RanzhiBasePage():
    def __init__(self, driver, baseUrl):
        """
                Construction method
                :param driver: packaged webdriver
                :param baseUrl: The basic url of the system http://[localhost:808]/ranzhi/www
        """

        self.baseUrl = baseUrl
```

TOP

```
        self.driver = driver

    def openPage(self, url):
        """
                Open the page of the system, by splicing the URL
        :param url: /sys/index.html
        :return:
        """
        self.driver.navigate(self.baseUrl + url)
```

Sub Page class

```
from ranzhiWeekend.ranzhi_base_page import RanzhiBasePage


class RanzhiSubLoginPage(RanzhiBasePage):
    def __init__(self, driver, baseUrl):
        """

        :param driver:
        :param baseUrl:
        """
                # Call its base class RanzhiBasePage constructor
                # Implement the function of the constructor of the base class
        super().__init__(driver, baseUrl)
        self.loginPageUrl = "/sys/user-login.html"
        self.mainPageUrl = "/sys/index.html"
        self.driver.clearCookies()

    def login(self, userName, password):
        self.openPage(self.loginPageUrl)
        # self.driver.clearCookies()
```

TOP

```
        self.driver.implicitlyWait(5)
        self.driver.type("account", userName)
        self.driver.type("password", password)
        self.driver.click("submit")

    def getMainPage(self):
        return self.baseUrl + self.mainPageUrl
```

Tests Case class

```
import unittest
from time import sleep

from ranzhiWeekend.automate_driver import AutomateDriver
from ranzhiWeekend.ranzhi_sub_login_page import RanzhiSubLoginPage

"""
1. Import unittest
 2. Inheritance unittest.TestCase
 3. Write the use case The method starts with test
 4. Consider using setUp() and tearDown()
"""


class RanzhiTests(unittest.TestCase):
    def setUp(self):
        """
                Start preparations before each test
        :return:
        """
        self.autoDriver = AutomateDriver()
```

TOP

```
        self.baseUrl = "http://localhost:808/ranzhi/www"

    def tearDown(self):
        """
                End the cleanup after each test
        :return:
        """
        self.autoDriver.quitBrowser()

    def test_ranzhi_login(self):
        """
                Test case: test and log in
        :return:
        """
                #
        loginPage = RanzhiSubLoginPage(self.autoDriver, self.baseUrl)

                # Use the page object to log in
        loginPage.login("admin", "admin")
        sleep(2)
                #assertion Whether the login is successful
                self.assertEqual(loginPage.getMainPage(), self.autoDriver.getUrl(), u "login failed")
```

Tests Runner class

```
import smtplib
import unittest
from email.header import Header
from email.mime.text import MIMEText

from ranzhiWeekend import HTMLTestRunner
from ranzhiWeekend.ranzhi_tests_0605 import RanzhiTests
```

TOP

```python

class RanzhiTestRunner():

    def runTest(self):
        """
                Running test cases
        :return:
        """

                #declare a test suite
        suite = unittest.TestSuite()
                # Add test cases to test suites
        suite.addTest(RanzhiTests("test_ranzhi_login"))

                # Create a new test result file
        buf = open("./result.html", "wb")

                # declare the object that the test runs
        runner = HTMLTestRunner.HTMLTestRunner(stream=buf,
                                               title="Ranzhi Test Result",
                                               description="Test Case Run Result")
                # Run the test and generate the result as HTML
        runner.run(suite)

                # Close file output
        buf.close()

    def sendEmail(self, targetEmail):
        """
                send email
        :param targetEmail:
        :return:
        """
```

TOP

```

                #Open test report results
        f = open("./result.html", "rb")

                # Put the test results in the body of the message
        mailBody = f.read()
                # Close the test result file
        f.close()

                # Declare a mail object with the body of the mail you just got
        msg = MIMEText(mailBody, "html", "utf-8")
                # Set the subject of the message
        msg["subject"] = Header("Automation Test Result", "utf-8")

                # Create an SMTP service object
        # simple message transfer protocol
                #Simple message transfer protocol
        smtpMail = smtplib.SMTP()

                #Connect SMTP server
        smtpMail.connect("***.******.com")

                # Login SMTP server
        smtpMail.login("*******@*****.com", "*********")

                # Send mail using SMTP server
        smtpMail.sendmail("*******@********.com", targetEmail, msg.as_string())

                # Exit SMTP object
        smtpMail.quit()
```

Main function entry

TOP

```python
if __name__ == "__main__":
        # instantiate a runner
    runner = RanzhiTestRunner()

        #
    runner.runTest()

        #
    runner.sendEmail("********@******.com")
```

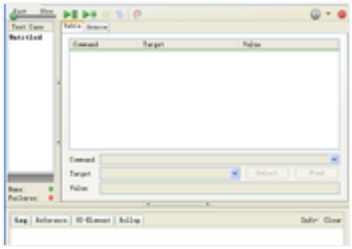
Reprinted source: http://www.jianshu.com/p/b5957c487350

TOP

Intelligent Recommendation

Automated testing using selenium module webdriver

A, webdriver using command Second, the tab navigation General positioning the label to find the label # Consider the case of using xpath xpath label positioning tab navigation complex XPath is the XML...



Automated testing tools Webdriver (1) Selenium IDE

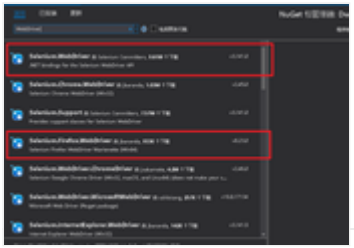
1, ready to work Firefox browser selenium-ide-2.9.0.xpi 2, installation As the Firefox browser support for selenium better than Google browser, so install the Firefox browser; The selenium-ide-2.9.0.x...

Selenium webdriver implements Canvas canvas automated testing

Canvas is a canvas, which can only be positioned on the canvas when positioning elements. As shown in the following page, there is an eChart report picture similar to the following figure on the web p...

Automated testing using Selenium WebDriver dotnet (1)

TOP



Automated testing using Selenium WebDriver dotnet (1) Quote via nuget Need to quote: using OpenQA.Selenium; Need to quote: using OpenQA.Selenium.Firefox; Language: C# Environment: .net framework 4.5 (...

Preliminary application of Selenium WebDriver for automated testing

table of Contents 1. Import the Selenium WebDriver module 2. Create a new object and start the browser 3. Call the maximize_window() method to maximize the browser window 4. Call the get() method to o...

TOP

More Recommendation

Automated testing Selenium basics (1)-WebDriver

One,Seleniuminstallation Prerequisites: 1. Python is installed and environment variables are configured 2. Pip is installed and environment variables are configured. (python -m pip install -U pip) 1) ...

Automated testing of a white learning record --Python + Selenium + pip + webdriver download and install automated test environment configuration python

White self-test software for a few days of automated testing briefly summarize recent knowledge of science ((● '◡' ●)) 1. First install python on the computer I installed the 3.5.4 version -64-bit con...

TOP

```
coding:utf-8
setup(self):
self.driver = webdriver.Remote(
command_executor='http://localhost:4444/wd/hub',
desired_capabilities={'platform': 'WINDOWS',
'browserName': 'firefox', 'version': '',
'javascriptEnabled': True})
self.driver.implicitly_wait(30)
self.base_url = "https://thewebsite.org/"
self.verificationErrors = []
self.accept_next_alert = True
```

python+selenium automated software testing (Chapter 5): Selenium Gird

5.1 Distributed (Grid) Selenium grid is a tool for distributed execution of test case scripts. For example, testers often need to test the compatibility of multiple browsers, then grid can be used. He...

Python and selenium automated testing

https://www.cnblogs.com/glumer/p/6088258.html My gut...

Selenium- automated testing -python

Selenium testing tools What is Selenium Selenium is a web automated testing tools, including IDE, Grid, RC (selenium 1.0) WebDriver (selenium 2.0) and the like. Selenium IDE firefox is a browser plug-...

TOP

Related Posts

- Selenium WebDriver automated testing
- Selenium Automated Testing Python II: WebDriver Foundation
- Selenium and webdriver automated testing-helloworld
- Selenium WebDriver of automated testing technology
- Python crawler tool Selenium webdriver-automated testing tool
- The best design pattern in selenium automated testing (PO design pattern)
- Python+selenium automated testing framework design
- Selenide: Selenium WebDriver-driven automated testing framework
- Automated testing The difference between WebDriver and WebElement in selenium
- Selenium Automated Testing - Learning Summary - WebDriver (1)

TOP

TOP

Popular Posts

- The important role of CRM data analysis
- LeetCode 30th Shuangli Race

TOP

- Simple understanding of the usage of String.format()
- 3. Pseudo-distributed running Hadoop case
- Array pointer
- "Big Data: Spark Client Operation API"
- 96 different binary search trees
- BoneWeb environment build application
- Centos6 offline installation CDH5.14.2 most complete detailed tutorial-7 browser installation ClouderaManager
- Swagger2 to build powerful RESTful API documentation in Spring Boot

TOP

Recommended Posts

TOP

- "Sword refers to offer" brush question - [backtracking method] interview question 13: robot's range of motion (java implementation)
- vue-mou class-life cycle
- Java basic notes (11)
- Object properties
- How to solve the blockchain bifurcation_hejiu must divide? The inevitability of blockchain forks
- 2021-03-03 bit count
- Comprehensive case of VRRP
- LG T43830 Chino's score
- TP-LINK router manually set the gateway, DNS, turn on DHCP and specify the IP range of the address pool
- Springboot framework learning accumulation---SpringBoot custom starter

Related Tags

Selenium

TOP

automatic testing

08-DEVOPS-TEST

software test

python

automated test

reptile

Python

test

Selenide

TOP

TOP

- Hard work, too tired, take a break
- Contact us to advertise

TOP