Automatic testing of complex Web application using Hierarchical Selenium framework

Student: Guangyu Lin (gl8429)
Xiangkun Dai (xd863)

Abstract

In automatic testing of complex Web applications, it will produce a large number of redundant test scripts, and the flexible management and call test is unavoidable demand because of the complex test scenarios. In this project, we stratified automated open source Selenium Web testing framework logically in order to increase the reusability and maintainability of test scripts.

Key Words: Automatic testing, Web applications, Hierarchical, Selenium

1.Introduction

Every software development group tests its products, yet delivered software always has defects. Test engineers strive to catch them before the product is released but they always creep in and they often reappear, even with the best manual testing processes. Automated software testing is the best way to increase the effectiveness, efficiency and coverage of your software testing.

An automated testing tool is able to playback pre-recorded and predefined actions, compare the results to the expected behavior and report the success or failure of these manual tests to a test engineer. Once automated tests are created they can easily be repeated and they can be extended to perform tasks impossible with manual testing. Because of this, savvy managers have found that automated software testing is an essential component of successful development projects. [1]

1.1 Procedure of Automated Testing

The figure below shows the basic flow of Web automatic test.

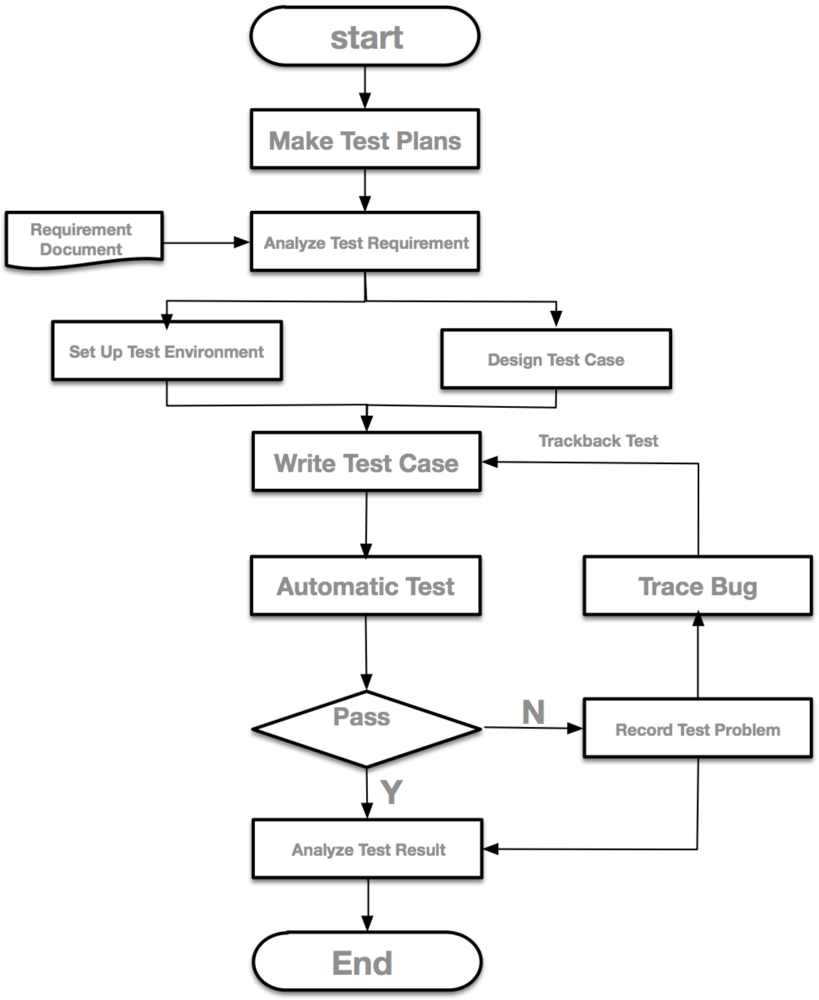


Fig. 1 Flow graph of Web automatic test

1.1.1 Make Test Plan

Clear test object, test purposes, the content of the test items, test methods, and progress of the test requirements and ensure the human, hardware, data and other resources required for testing are prepared. After the development of a good test plan, we deploy them to case design.

1.1.2 Analyze Test Requirement

Utility Designers analyze testing requirements and design requirement tree according to the test plan and requirements specification to cover all demand points while designing use case. In general, functional test requirements which are web-based cover the following aspects.

Page link test

1) Page control test
2) Page function test
3) Data processing test
4) Module business logic test

1.1.3 Design Test Case

It can perform automated test case aggregated into automated test cases. Sometimes we usually make username, password, product and client independent. Example will be shown later.

1.1.4 Setup Test Requirement

Automated testers can begin to build a test environment while use case design work carried out at the same time you can begin to build a test environment. Set up a test environment include the following

1) Deployment of test system
2) Test hardware's call
3) Installation and setup the test tool
4) Arranged network environment

1.1.5 Write Test Case

According to the degree of difficulty of test automation problems, we should take the appropriate way to write test scripts script during developing.

1) Get all required the page controls by recording mode
2) Control the execution of script with structured statements
3) Insert checkpoints and abnormal decision feedback statement
4) Make independent public common feature to be share scripts

1.1.6 Track Bug

Bugs in testing record should be written in the tool of failure management, in order that we can trace and deal with them periodically. That is to say, tester should fix bug before regression testing.

1.2 Selenium

Selenium is an automated testing tool for Web applications. It can accurately reproduce the procedure of Test Cases written by software testers through simulating users' various operations on the Web pages. Selenium contains three tools: Selenium-IDE, Selenium-RC and Selenium-Core. Wherein, Selenium-Core is a core part of driving Selenium work, as a test engine written in JavaScript that can operate on various elements of Web pages, such as: click the button, enter text box, and assert the existence of certain Web pages Web text and other elements.

Selenium-IDE is a Firefox plug-in, can record the playback of user behavior in Firefox, and the recorded Selenese (Selenium Commands) into Java/C#/Python/Ruby and other languages, modify multiplexed in the Selenium-RC. For more complex Test Cases, Selenium-IDE limited functionality, often use it to record a rough step, and then converted to testers familiar programming languages, and improve on this basis, to form a more powerful and flexible Selenium-RC Test Cases.

Selenium-RC (Selenium Remote Control) sets up between the Web browser and the Web application you want to test a proxy server (Selenium Server), so that the JavaScript engine and test Web applications homologous to bypass same-origin policy restrictions (Same Origin Policy), and then get to the Web page for various operations privileges.

1.2.1 Common Selenese Commands

Selenium commands, known as selense, are consists of a series of running test cases. These commands are arranged in order to form the test scripts.

● open	● verifyTable
● click/clickAndWait	● waitForPageToLoad
● verifyTitle/assertTitle	● waitForPageToLoad

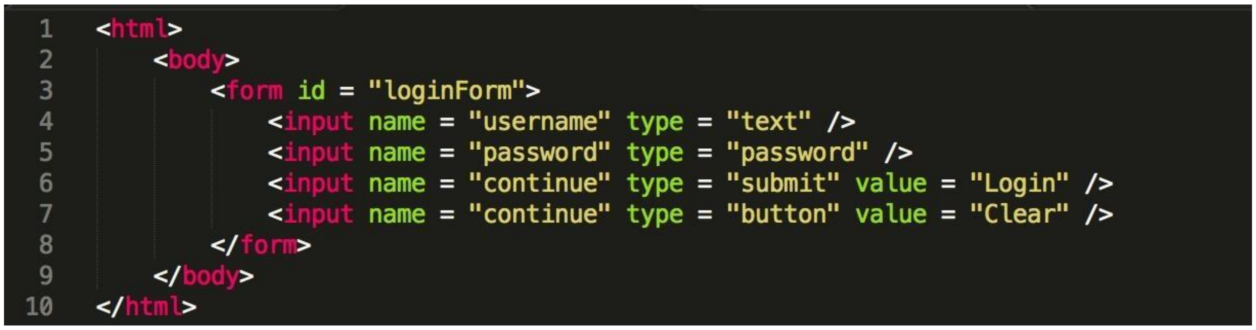
- verifyTextPresent, to verify a specific piece of text in the page
- verifyElementPresent, to verify the existence of a specific UI element
- verifyText, to verify both text and UI element

1.2.2 Element locators

In actual testing work, we found it harder to locate a web element than in a static html file. To better locating them, we use different locators supported by Selenese. For many Selenium commands, target fields are necessary. Target recognizes UI elements in Web pages, by using the form of locatorType = location. There are certain types of locatorType: Default, Identifier, id, DOM, Name, Xpath, CSS Selector, and Link Text.

Xpath locator is a kind of language that locating elements in XML documents, which is widely used in our project. Xpath extends locating methods by id and name, and provides more possibilities.

Here is an example of Xpath locator.



```
 1  <html>
 2      <body>
 3          <form id = "loginForm">
 4              <input name = "username" type = "text" />
 5              <input name = "password" type = "password" />
 6              <input name = "continue" type = "submit" value = "Login" />
 7              <input name = "continue" type = "button" value = "Clear" />
 8          </form>
 9      </body>
10  </html>
```

Xpath locating begins at “//”, therefore “xpath=” tag is not necessary.

1) //form[1], locating the form element in line 3
2) //form[@id = ‘loginForm’], locating the element with id ‘loginForm’
3) //form[input/\@name = ‘username’], locating the form with input sub-element of which name = ‘username’
4) //form[@id=‘loginForm’]/input[1], locating the first input element under the form with id ‘loginForm’.
5) //input[@name=‘continue’][@type=’button’], locating the input with name ‘continue’ and with type ‘button’.

The table below shows the example of some commands.

Command	Target	Value
verifyText	//table/tr/td/div/p	This is my text and it occurs right after the div inside the table
verifyElementPresent	//div/p/img	
verifyTextPresent	Marketing Analysis	

1.2.3 Selenium Working Process

Selenium Server is started by the HTTP GET request from Junit Test/Ruby Test/uUnit Test/perl Test. Then Selenium launches browsers. GET request goes through proxy, and, finally, request is proxied through the Internet.

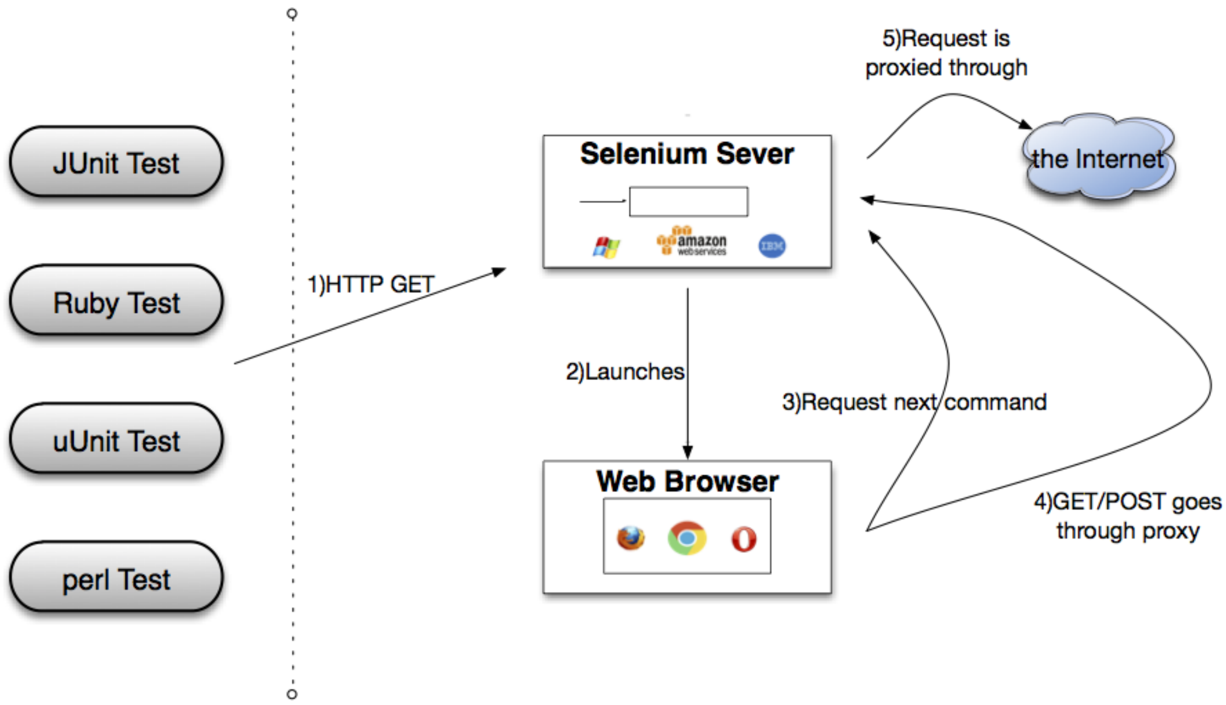


Fig 2. Selenium Working Process

1.3 TestNG

While using Selenium to test Web, there are various behaviors depending on input parameters of testers, e.g., choosing from drop-down menu and type characters in the text box. It is easy to pass test parameters if we use TestNG, which will improve reusability and maintainability of test cases.

2.Hierarchical Testing Framework Based on Selenium

2.1 Test objective

We are trying to include most common Web operations in our project. Specifically, we tested on two types of web applications: Search Engine and Gmail functions.

For search engine test, we verified the searching results by Google, Bing, and Baidu. For Gmail test, we simulated and verified behaviors like login, sending mail, deleting mail, and searching mail.

2.2 Description of the framework

In order to test several Web elements, we construct a hierarchical framework to take advantage of reusability. There are three levels in this framework: appObjects, tasks, and test cases.

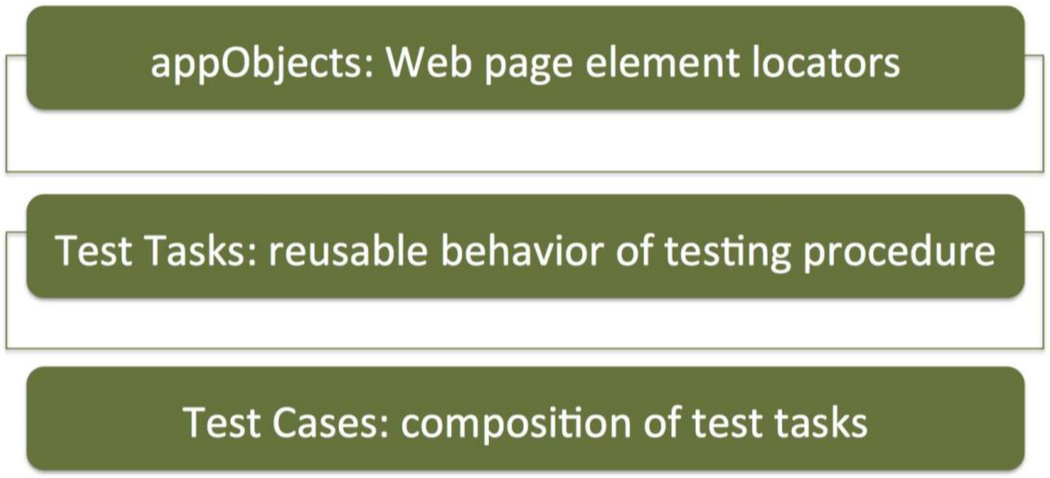

appObjects: Web page element locators
Test Tasks: reusable behavior of testing procedure
Test Cases: composition of test tasks

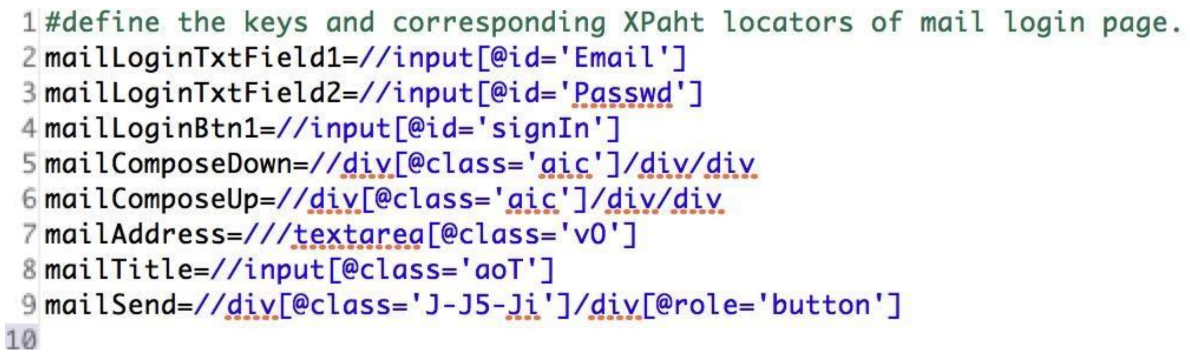
Fig 3. Hierarchical Testing Framework Based on Selenium

The Test Suite consists of two different types of Test Cases (search engine test and Gmail test), each Test Cases is consisted of several reusable test tasks. Test tasks can complete homogenous but different behaviors by passing distinct parameters. AppObjects level, located under Test Tasks, includes locators information in Web pages to be tested.

2.2.1 Definition and Collection of Web element locators

Selenium locates Web element by XPath. We create documents in appObjects layer, the Web page elements fall locators, which is easy to maintain and use.

An Example of Locator Document



```
#define the keys and corresponding XPaht locators of mail login page.
mailLoginTxtField1=//input[@id='Email']
mailLoginTxtField2=//input[@id='Passwd']
mailLoginBtn1=//input[@id='signIn']
mailComposeDown=//div[@class='aic']/div/div
mailComposeUp=//div[@class='aic']/div/div
mailAddress=///textarea[@class='v0']
mailTitle=//input[@class='aoT']
mailSend=//div[@class='J-J5-Ji']/div[@role='button']

```

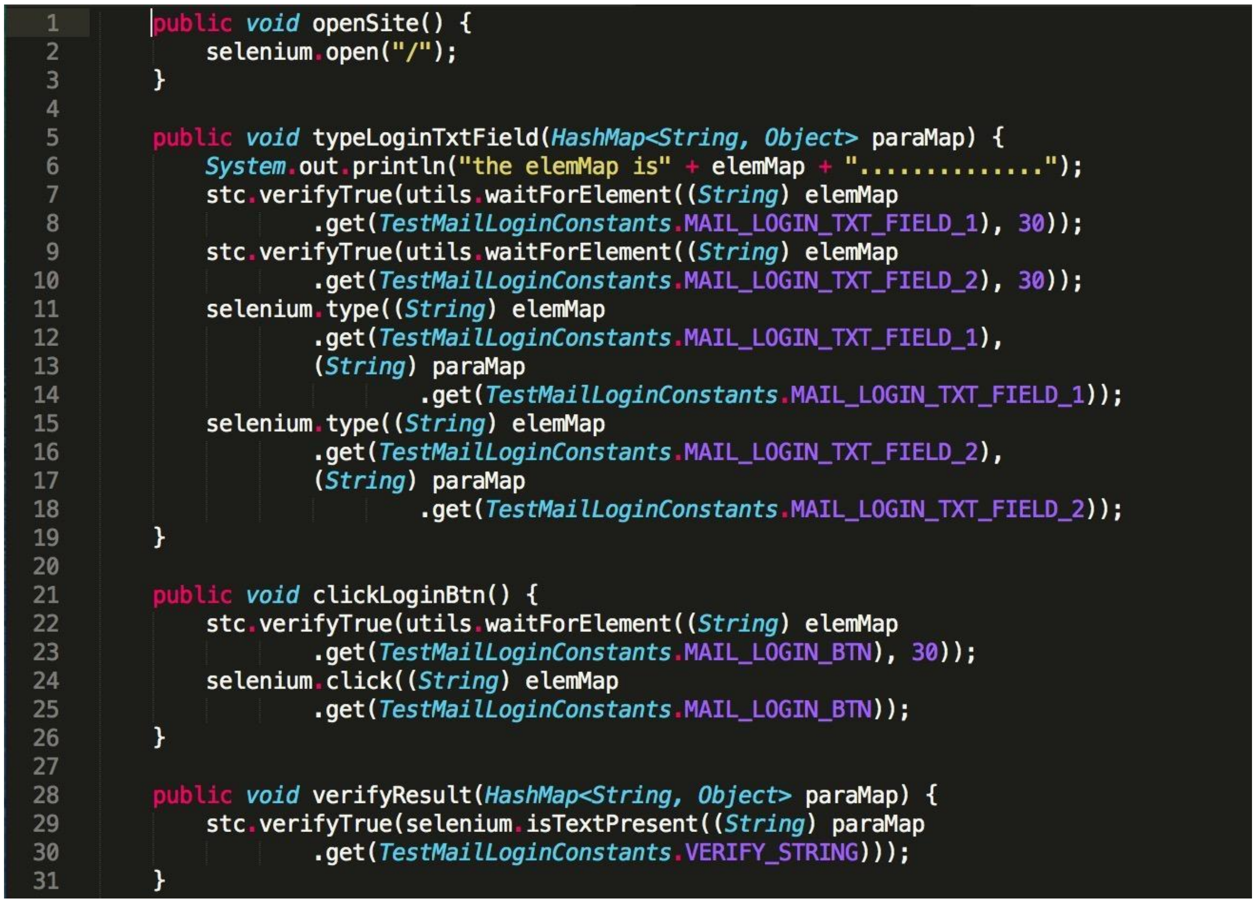
2.2.2 Implementation of test tasks

We divided test tasks into several process, which is shown in Table 1.

Search Process	Email Login	Email Sending	Email Deleting	Email Searching
1. search keywords 2. search button 3. verify the results page	1. input email address 2. input email password 3. login button 4. verify the results	1.email login process 2.compose button 3.input sending information(receiver address, subject, body) 4.send button 5.verify the result page	1.email login process 2.checkout deleted item 3.delete button 4.verify the total mail	1.email login process 2.search keywords 3.search button 4.verify the results page

Table 1. Decomposition of tasks

An Example of Test Task



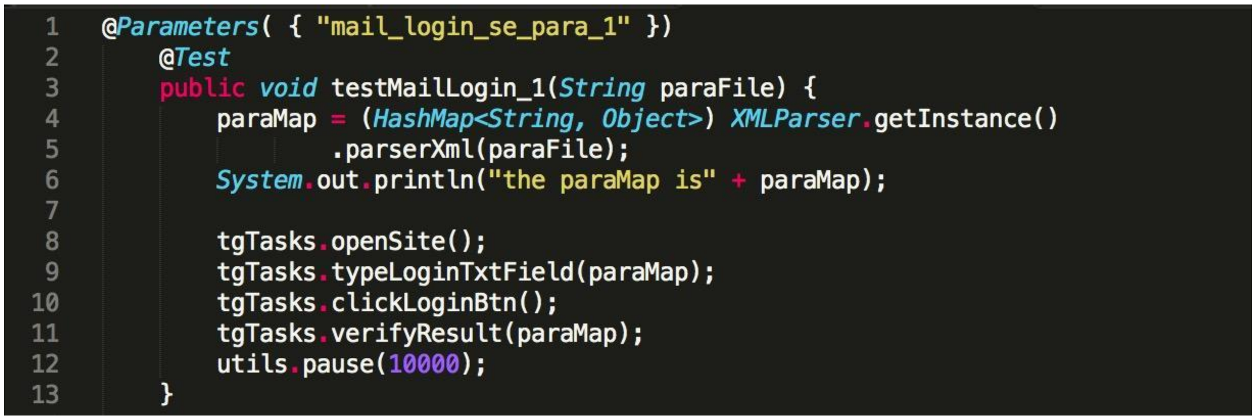
```
public void openSite() {
    selenium.open("/");
}

public void typeLoginTxtField(HashMap<String, Object> paraMap) {
    System.out.println("the elemMap is" + elemMap + ".............");
    stc.verifyTrue(utils.waitForElement((String) elemMap
            .get(TestMailLoginConstants.MAIL_LOGIN_TXT_FIELD_1), 30));
    stc.verifyTrue(utils.waitForElement((String) elemMap
            .get(TestMailLoginConstants.MAIL_LOGIN_TXT_FIELD_2), 30));
    selenium.type((String) elemMap
            .get(TestMailLoginConstants.MAIL_LOGIN_TXT_FIELD_1),
            (String) paraMap
                    .get(TestMailLoginConstants.MAIL_LOGIN_TXT_FIELD_1));
    selenium.type((String) elemMap
            .get(TestMailLoginConstants.MAIL_LOGIN_TXT_FIELD_2),
            (String) paraMap
                    .get(TestMailLoginConstants.MAIL_LOGIN_TXT_FIELD_2));
}

public void clickLoginBtn() {
    stc.verifyTrue(utils.waitForElement((String) elemMap
            .get(TestMailLoginConstants.MAIL_LOGIN_BTN), 30));
    selenium.click((String) elemMap
            .get(TestMailLoginConstants.MAIL_LOGIN_BTN));
}

public void verifyResult(HashMap<String, Object> paraMap) {
    stc.verifyTrue(selenium.isTextPresent((String) paraMap
            .get(TestMailLoginConstants.VERIFY_STRING)));
}
```

The actual code is shown above. The parameter determines the behavior of the test task, which have received a paraMap data structures and their content appropriate behavior in the method. In this way, test cases can be used to drive test parameter configuration file task to implement its desired behavior.

2.2.3 Implementation of Test Case and Call of Test Tasks

Test Cases, which is a collection of test procedures, can be achieved by calling the number of Test Tasks. In a hierarchical Selenium test framework, Test Cases is in accordance with the requirements of calling the existing Test Tasks.

An Example of Test Case

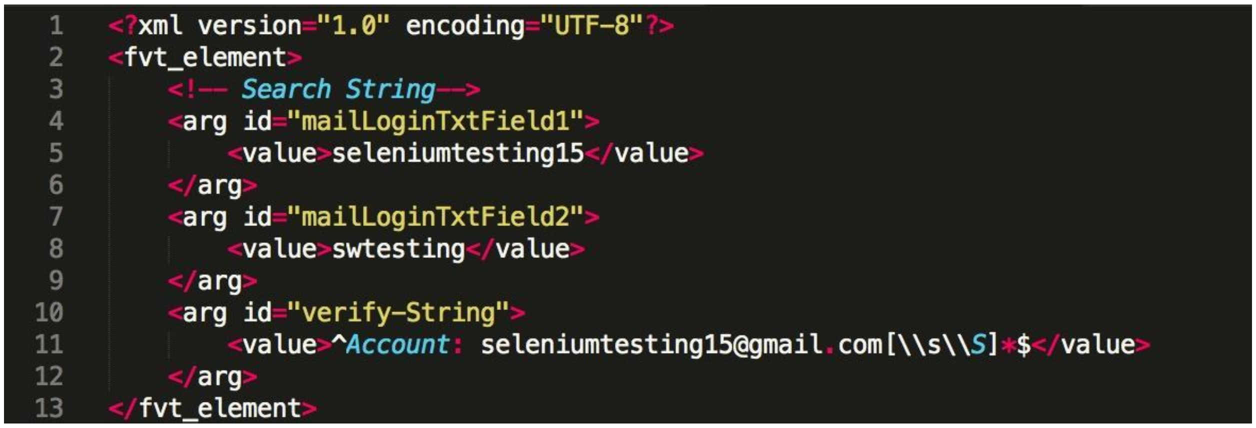


```java
@Parameters( { "mail_login_se_para_1" })
    @Test
    public void testMailLogin_1(String paraFile) {
        paraMap = (HashMap<String, Object>) XMLParser.getInstance()
                .parserXml(paraFile);
        System.out.println("the paraMap is" + paraMap);

        tgTasks.openSite();
        tgTasks.typeLoginTxtField(paraMap);
        tgTasks.clickLoginBtn();
        tgTasks.verifyResult(paraMap);
        utils.pause(10000);
    }
```

2.2.4 Parameter parser and parameter file

It is worth noting the paraMap parameters in two piece of codes above. The hash table is given by the parameter file parser which we defined. The parameter mechanism of TestNG makes it flexible for Test Cases to specify the parameter file to drive different Test Cases.

An Example of Parameter Definition File

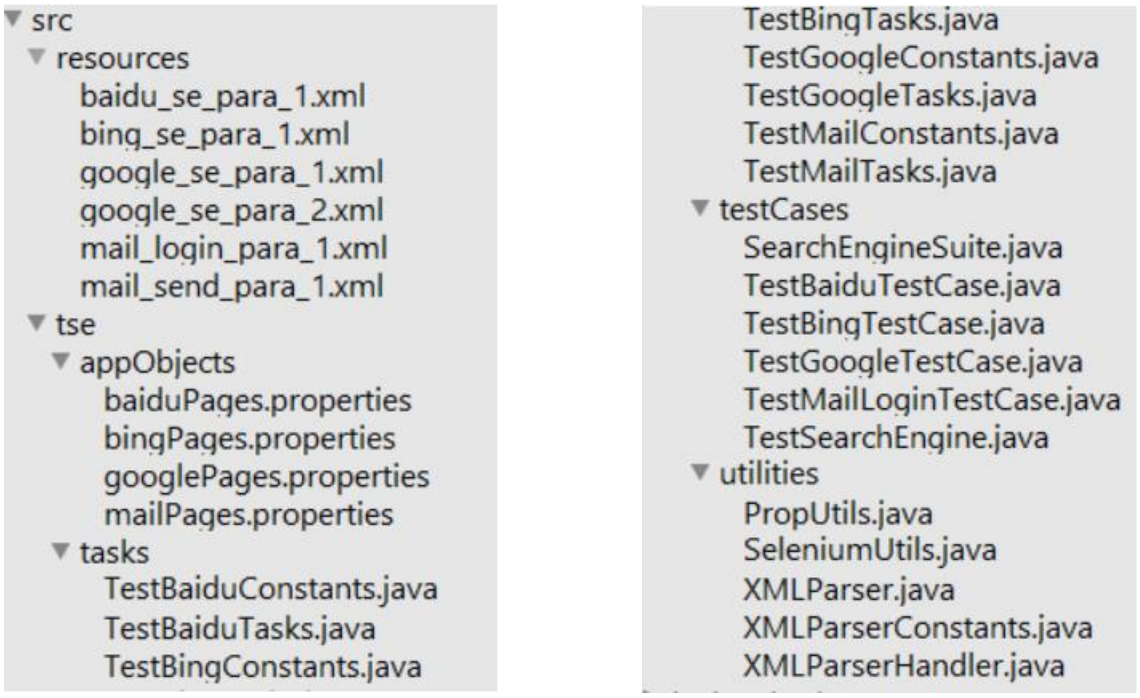


```xml
<?xml version="1.0" encoding="UTF-8"?>
<fvt_element>
    <!-- Search String-->
    <arg id="mailLoginTxtField1">
        <value>seleniumtesting15</value>
    </arg>
    <arg id="mailLoginTxtField2">
        <value>swtesting</value>
    </arg>
    <arg id="verify-String">
        <value>^Account: seleniumtesting15@gmail.com[\\s\\S]*$</value>
    </arg>
</fvt_element>
```

"<fvt_element>" is in the outermost layer, and sub-elements within it are as specific parameter values. "<arg>" represents a page element and its corresponding input. We provided a parser to parse these parameter files.

2.2.5 Source code structure of the project

Until here, we organize our work logically into a three-level structure. Parser and XML files are implemented outside the test suite.



```
▼ src
  ▼ resources
      baidu_se_para_1.xml
      bing_se_para_1.xml
      google_se_para_1.xml
      google_se_para_2.xml
      mail_login_para_1.xml
      mail_send_para_1.xml
  ▼ tse
    ▼ appObjects
        baiduPages.properties
        bingPages.properties
        googlePages.properties
        mailPages.properties
    ▼ tasks
        TestBaiduConstants.java
        TestBaiduTasks.java
        TestBingConstants.java
        TestBingTasks.java
        TestGoogleConstants.java
        TestGoogleTasks.java
        TestMailConstants.java
        TestMailTasks.java
    ▼ testCases
        SearchEngineSuite.java
        TestBaiduTestCase.java
        TestBingTestCase.java
        TestGoogleTestCase.java
        TestMailLoginTestCase.java
        TestSearchEngine.java
    ▼ utilities
        PropUtils.java
        SeleniumUtils.java
        XMLParser.java
        XMLParserConstants.java
        XMLParserHandler.java
```

Fig 4. Structure of source code

3.Conclusion

Selenium provides a method for Web testing. In supporting of hierarchical framework, it becomes more powerful, and has strong reusability. By specifying different parameter files, examples in this article can be tested for search engines to verify different keywords, the same method can be applied to Gmail testing.

Reference

[1] http://support.smartbear.com/articles/testcomplete/manager-overview/

[2] http://www.ibm.com/developerworks/cn/java/j-lo-selenium/index.html

[3] http://www.ibm.com/developerworks/cn/web/wa-testweb/

[4] https://github.com/fool2fish/selenium-doc/blob/master/official-site/selenium-web-driver.md