****

**软件测试技术第二次实验报告**



**学 院 软件学院**

**专 业 软件工程**

**年 级 大三**

**姓 名 张聪**

**2019年 3 月 26 日**

# 软件测试技术第一次实验报告

git网址链接：https://github.com/likeAnEgg/SoftwareTest/tree/master/lab

1. 需求分析

“软件测试名单.xlsx”包含学生信息，http://121.193.130.195:8800可以在登录后查看个人信息（student id为用户名，student id的最后6位为密码）。请检查excel中的每个记录，确保每个学生的信息与网站上的信息一致。

1. 概要设计

使用java编写脚本，通过selenium和火狐浏览器对指定网址进行检测。每个测试样例包括来自本地文件中的对应学生的全部信息（学号，姓名，github网址），通过junit中的Parameterized这一runner可以对每个测试样例进行较为准确的区分并同时检测。

一次启动的大致流程是，先启动火狐浏览器，然后登陆指定网址，输入用户名和密码信息（都来自学号），找到登陆按钮并点击，在跳转后的页面中验证对应信息，最后退出驱动。（其中，真正困扰到我的不是设计这个流程本身，而是本人电脑浏览器启动与退出和网络的不稳定，导致时间的不稳定，最终会导致只使用隐式等待难以处理的套接字超时错误,详细解决办法我会在后面的部分具体说明。）

由于该runner接收的测试样例需要是静态变量，所以不宜在测试类中直接读取excel文件生成测试样例。我这里采用的是新建一个类，设计一个读取函数，在新类中通过main运行一遍这个读取函数，将excel文件中的信息转化成对应的数组元素代码并输出，然后将这段代码直接复制到测试类中的@Parameters函数的返回值中，这样就直接得到了全部的静态测试样例，且节省了在运行测试类时读取excel的时间。

最后，将代码整合在测试类中，进行测试即可。

1. 详细设计

（一）测试样例生成类 ReadExcel.java

**package** Lab2;

**import** java.io.File;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** jxl.Cell;

**import** jxl.Sheet;

**import** jxl.Workbook;

**public** **class** ReadExcel {

**public** **static** **void** main(String[] args) **throws** Exception{

//1:创建workbook

String xlsPath = System.*getProperty*("user.dir") + "/src/Lab2/students.xls";

Workbook workbook=Workbook.*getWorkbook*(**new** File(xlsPath));

//2:获取第一个工作表sheet

Sheet sheet=workbook.getSheet(0);

//3:获取数据

//System.out.println("行："+sheet.getRows());

//System.out.println("列："+sheet.getColumns());

**int** startRow=2;

**int** totalnum=sheet.getRows()-startRow;

totalnum--;//最后有一行空行

String[][] studentArray=**new** String[200][4];

**for**(**int** i=startRow;i<sheet.getRows();i++){

**for**(**int** j=0;j<sheet.getColumns();j++){

Cell cell=sheet.getCell(j,i);

studentArray[i-startRow][j]=cell.getContents();

}

}

//最后一步：关闭资源

workbook.close();

**for**(**int** i=0;i<totalnum;i++){

System.***out***.print("{");

**for**(**int** j=0;j<4;j++){

System.***out***.print("\""+studentArray[i][j]+"\"");

**if**(j!=3)System.***out***.print(",");

}

System.***out***.println("},");

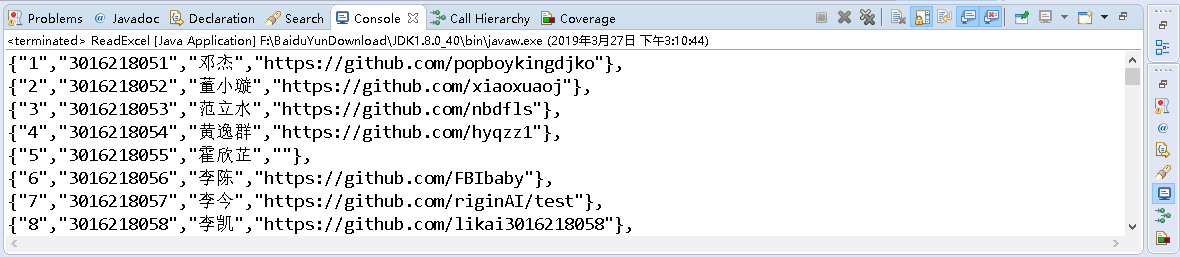
}

}

}

说明：该类使用jxl包对xls格式的excel进行读取操作，由于原excel是xlsx格式，所以需要先打开原excel文件，并将其另存xls格式的新文件。我这里将“软件测试名单.xlsx”另存为了“students.xls”，之后对新文件进行处理。

代码运行后部分结果如图所示：



直接生成了符合runner格式的测试样例中静态数组的代码，将其整合到测试类中即可。

（二）测试类

**package** Lab2;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.junit.Assert.*fail*;

**import** java.io.BufferedInputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.concurrent.TimeUnit;

**import** javax.swing.filechooser.FileSystemView;

**import** org.apache.poi.hssf.usermodel.HSSFRow;

**import** org.apache.poi.hssf.usermodel.HSSFSheet;

**import** org.apache.poi.hssf.usermodel.HSSFWorkbook;

**import** org.apache.poi.poifs.filesystem.POIFSFileSystem;

**import** org.junit.\*;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

**import** org.openqa.selenium.Alert;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.NoAlertPresentException;

**import** org.openqa.selenium.NoSuchElementException;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.firefox.FirefoxDriver;

**import** org.openqa.selenium.support.ui.ExpectedConditions;

**import** org.openqa.selenium.support.ui.WebDriverWait;

**import** jxl.Cell;

**import** jxl.Sheet;

**import** jxl.Workbook;

@RunWith(Parameterized.**class**)

**public** **class** gitTest {

//private boolean started;

**private** WebDriver driver=**null**;

**private** WebDriverWait wait = **null**;

**private** String baseUrl;

**private** **boolean** acceptNextAlert = **true**;

**private** StringBuffer verificationErrors = **new** StringBuffer();

**private** String lsid;

**private** String lsname;

**private** String lsgit;

**public** gitTest(Object num,String x,String y,String z){

**this**.lsid=x;

**this**.lsname=y;

**this**.lsgit=z;

}

@Before

**public** **void** setUp() **throws** Exception {

String driverPath = System.*getProperty*("user.dir") + "/src/Lab2/geckodriver.exe";

System.*setProperty*("webdriver.gecko.driver", driverPath);

driver = **new** FirefoxDriver();

baseUrl = "http://121.193.130.195:8800/login";

//driver.manage().timeouts().pageLoadTimeout(300, TimeUnit.SECONDS);

//driver.manage().timeouts().setScriptTimeout(300, TimeUnit.SECONDS);

//driver.manage().timeouts().implicitlyWait(300, TimeUnit.SECONDS);

wait = **new** WebDriverWait(driver, 59);

//

}

@Test

**public** **void** testIt() **throws** Exception {

Thread.*sleep*(3000);//等待三秒

**try**{//由于浏览器启动时间极其不稳定，且在我的认知里selenium对该问题没有太好的处理，所以我这里对get进行try，后续代码使用finally强制执行，来忽视套接字超时异常

driver.get(baseUrl);

}

**catch**(Exception e){System.***out***.println("SLOW GET");Thread.*sleep*(5000);//超时可能代表电脑繁忙，继续等待}

**finally**{

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*name*("id")));//出现指定元素之前一直等待，以下同理

driver.findElement(By.*name*("id")).clear();

driver.findElement(By.*name*("id")).sendKeys(lsid);

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*name*("password")));

driver.findElement(By.*name*("password")).clear();

driver.findElement(By.*name*("password")).sendKeys(lsid.substring(4));

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*id*("btn\_login")));

WebElement lg=driver.findElement(By.*id*("btn\_login"));

lg.click();

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*id*("student-id")));

String wsid = driver.findElement(By.*id*("student-id")).getText();

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*id*("student-name")));

String wsname = driver.findElement(By.*id*("student-name")).getText();

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*id*("student-git")));

String wsgit = driver.findElement(By.*id*("student-git")).getText();

*assertEquals*(lsid,wsid);

*assertEquals*(lsname,wsname);

*assertEquals*(lsgit, wsgit);

**try**{//关闭浏览器，具体情况与启动类似

driver.quit();

}**catch**(Exception e){System.***out***.println("SLOW QUIT");Thread.*sleep*(3000);

//如果超时，可能代表电脑繁忙，继续等待}

**finally**{Thread.*sleep*(2000);}

}

/\*

String verificationErrorString = verificationErrors.toString();

if (!"".equals(verificationErrorString)) {

fail(verificationErrorString);

}\*/

}

@Parameters

**public** **static** Collection<Object[]> getDate(){

**return** Arrays.*asList*(**new** Object[][]{

{"1","3016218051","邓杰","https://github.com/popboykingdjko"},

{"2","3016218052","董小璇","https://github.com/xiaoxuaoj"},

//代码中包含143个测试样例，此处文档中省略大部分测试样例，在git链接上的代码中可见全部样例

{"142","3016218191","祝开元","https://github.com/ZHUKAIYUAN0/test"},

{"143","3016218192","邹璐文","https://github.com/jianxin-YL"},

});

}

}

1. 调试分析

（一）问题

本人电脑浏览器启动与退出和网络的不稳定，导致时间的不稳定，最终会导致只使用隐式等待driver.manage().timeouts()难以处理的套接字超时错误。

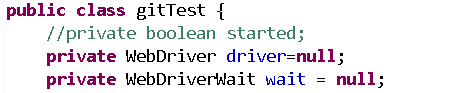
（二）解决

这个问题其实可以分成两个问题解决，一个是浏览器启动与退出的时间不稳定，一个是网络不稳定。

1.网络不稳定

让我开心的是，对于这个问题，selenium中有较为成熟的解决方案。selenium显式等待中的WebDriverWait和ExpectedConditions可以完美解决网络不稳定带来的问题，且把时间损耗降到很低。

先声明WebDriverWait



在@Before中定义好WebDriverWait



之后在测试中，每次需要寻找某个元素，都可以写一句

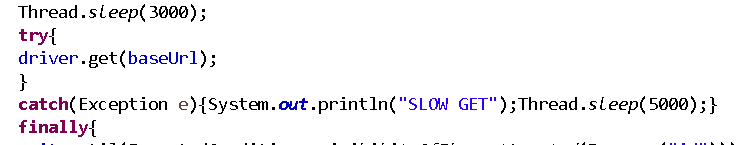
wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("id")));

其中的By.name("id")可以是任意的元素by，如By.id("btn\_login")等。这样，在59秒之内selenium会等到这个元素被检测到才会继续执行后面的代码。不仅如此，ExpectedConditions还支持其他各种各样的情况，我认为使用它们对代码的健壮性有很大提高。

2.浏览器启动退出不稳定

但浏览器启动与退出的时间这个东西selenium没有太好的检测办法，所以我通过java的异常处理特性进行了强制处理，并使用Thread.sleep合理分配时间以求对测试时间的影响降到最低（如果电脑有更好的gpu的话可能不会遇到这个问题）。

如图

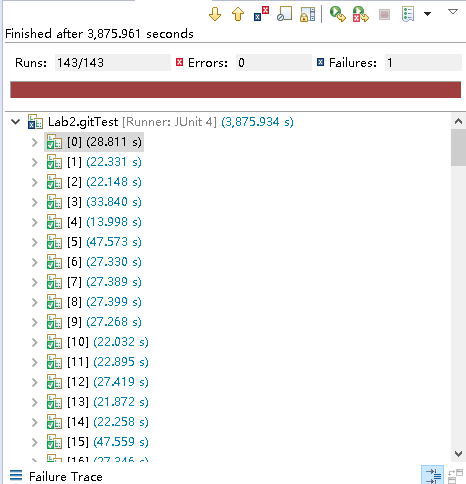


超时代表系统繁忙，我选择继续等待一段时间后，强制执行后面的代码，这么解决如果在性能更好的电脑上会造成时间的浪费，但如果仅对于本机完成测试而言，问题能得到较好的解决。

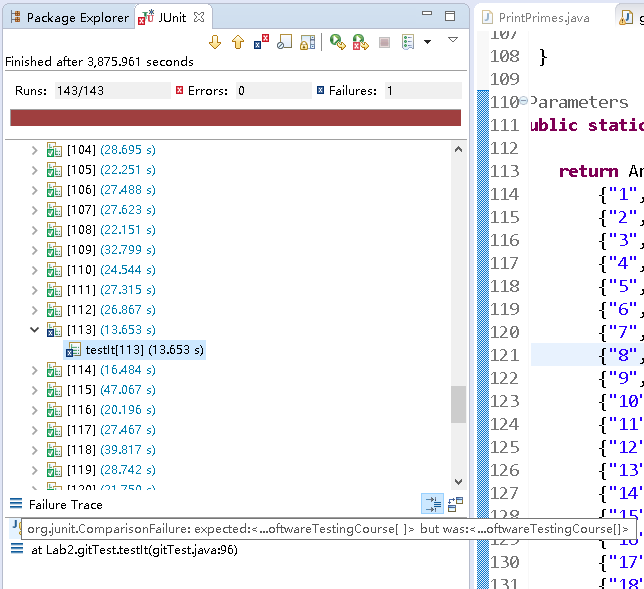
1. 测试结果

输入既是ReadExcel通过excel文件中的信息生成的测试样例，共143个。输出可以说就是网页上登录后检测到的信息。两相匹配，全部相同则为正确。

测试结果如图所示：



测试时长约为一小时，143个测试样例中，有一个样例出现了问题



比对发现，113号测试样例（0开始，所以是第114个同学祝晶晶）的github网址出现了问题，excel中她的网址结尾有一个空格，而实际通过浏览器访问对应网站的网页上的github网址结尾没有空格。

在我看来，这是由于空格在HTML中本身是没有意义的，浏览器只解析代码，空格的代码是&nbsp，所以编写网页的时候即使打了空格，浏览器解析后也很可能会自动忽略，这就导致我检测到的字符串和原字符串不匹配。事实上，人们在输入网址的时候末尾的空格对浏览器也照样没有影响，所以这个Failure也可以忽视，实际的检测最终结果，为全部通过。

1. 总结

经过本次实验,我初步了解了如何使用selenium对网页进行测试,并认识到这些工具对软件开发中的测试活动带来的便利.对软件测试流程有了更深入的了解和掌握,且对课堂上老师讲的黑盒测试等知识点有了更深刻的了解,受益匪浅。

1. 附录

（一）安装Selenium, Firefox and SeleniumIDE的简要描述

在搜索引擎中搜索"火狐浏览器"，下载对应安装包，并进行安装。

然后打开Firefox，在网址栏输入about:addons,回车，在出现的页面中搜索Selenium插件并进行安装。

下载老师提供的安装包selenium-java-3.9.0.zip，解压。在eclipse中右键项目，选择properties->java build path->add ecternal JARS 将之前解压得到的所有jar包添加进路径。

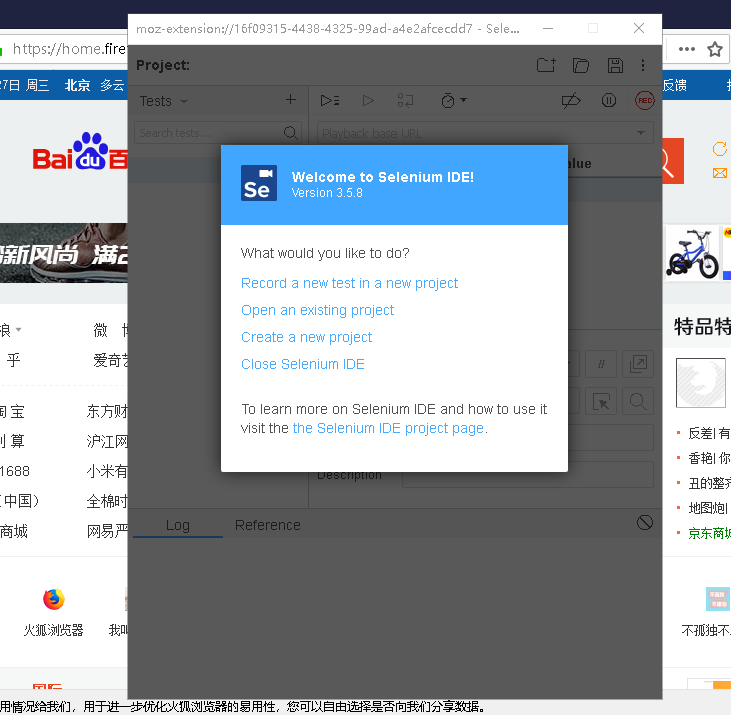
注：java版本应在1.8以上

（二）脚本录制和导出

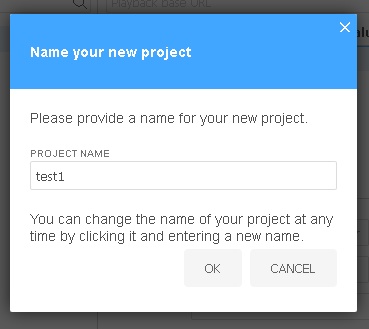
通过学习老师发的Selenium的ppt教程和网上自学，实验脚本由我直接编写代码并调试完成。

下面对通过火狐浏览器和Selenium的脚本录制和导出的流程进行简要描述。

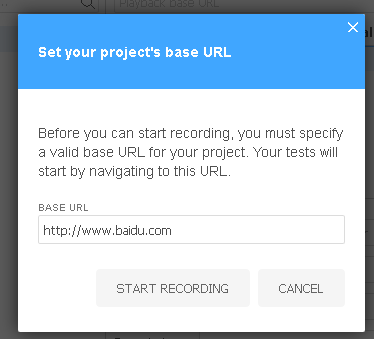
打开火狐浏览器，点击右上角的SeleniumIDE，看到：



选择第一个选项，Record a new test in a new project

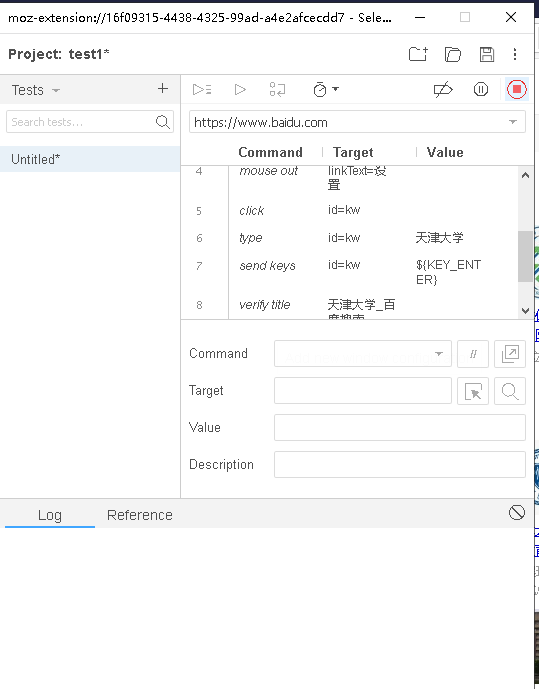


起一个名字，点击ok

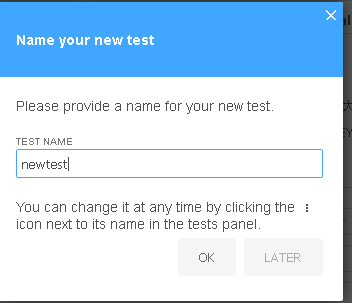


输入base url，点击开始记录

然后可以随意操作，想结束操作时按右上角的SeleniumIDE



点右上角红点停止录制。



起名字

点击save，选择路径保存，得到导出的脚本



（三） url

https://github.com/likeAnEgg/SoftwareTest/tree/master/lab