WEB自动化测试

遇见 “WebdDriver+Java+Testng”

高江

寄语

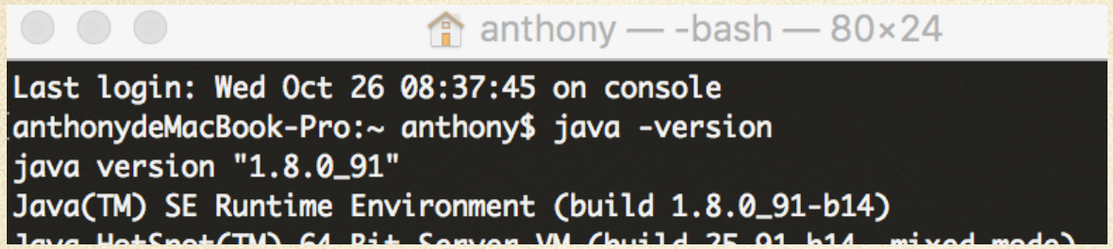
- 自动化测试也只是“一副药”也不是包治百病，是药也三分毒，大多数时候别那么使劲，别那么执着。放自动化一马，也放自己一马。人生是一种折腾，别折腾得自己浑身疼！

必要软件地址

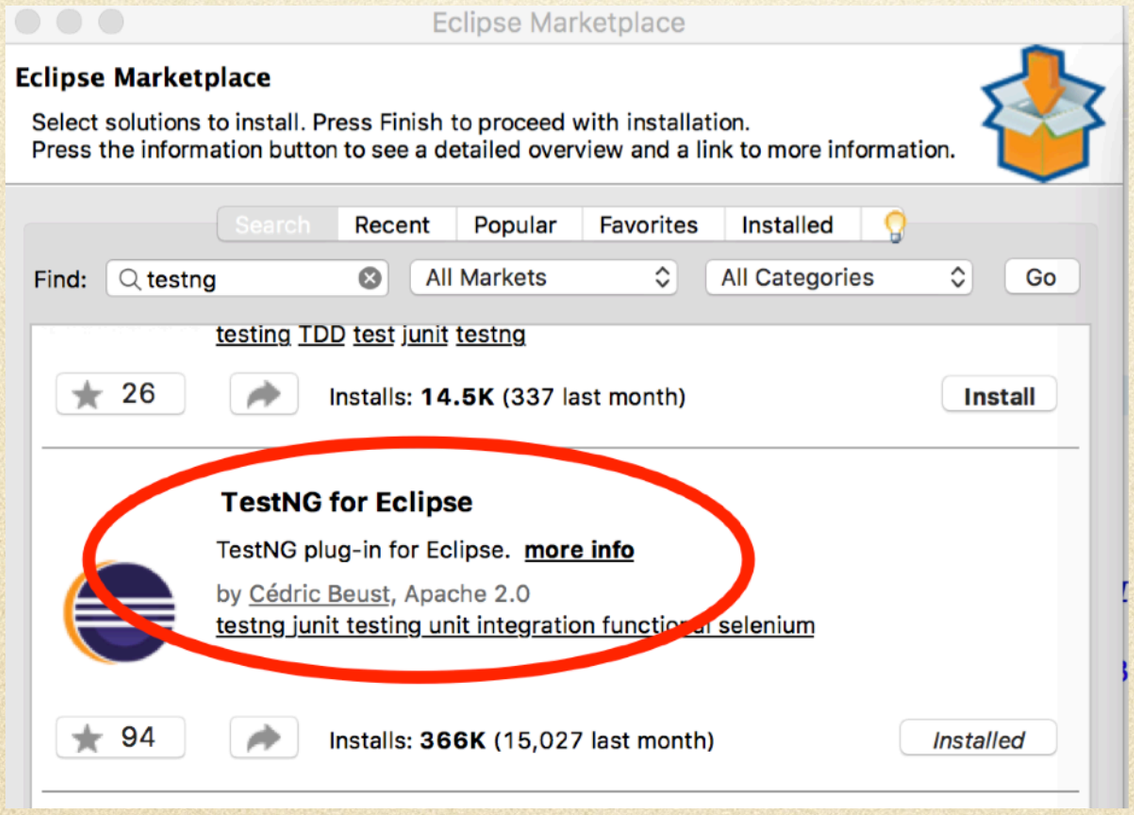
- http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- https://www.eclipse.org/downloads/
- https://git-scm.com/downloads
- http://maven.apache.org/
- http://docs.seleniumhq.org/projects/webdriver/
- http://ant.apache.org/

自动化测试之环境配置

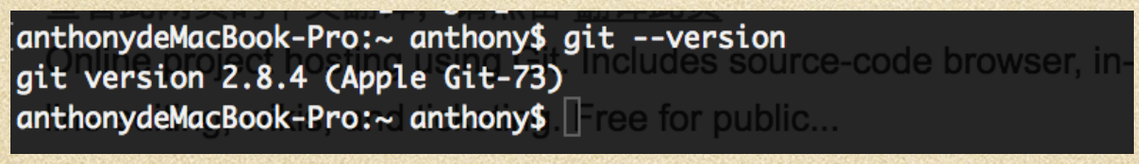
1. 安装JDK，在命令行输入java -version 检查成功与否



2. 安装eclipse并安装testng插件



3. 安装git



自动化测试之必要能力储备

- Java基本编程能力，no算法，no复杂度
- Selenium java版基本语法
- Testng 基本语法
- maven，git 等等
- 。。。。。
- 是不是还没准备好?

大家不要慌
我是司马光

Selenium 初探

- 关于selenium的历史这里就不介绍了，听我瞎扯淡还不如Google来得更准确些，工作原理嘛也自己去科普下吧，直接介绍Selenium的几大产品：

- 1. 传说中的录制神器 Selenium IDE(目前Firefox下的一个插件)，可以在website 上录制大多数用户的行为，竟然还可以导出成各种格式的代码，您可以自主选择自己擅长的编程语言风格，然后它就华丽丽的帮你把你的行为自动的转换成代码了，吓尿了有么有？但是还是不推荐用神器，它是个好工具，入门时你可以从中学到很多，但是“前路”真的略窄，你要问我为什么？你猜！

- 2. Selenium Remote Control 也就是RC，从字面意思也能猜出来它能干什么，不仅能让你操作本机的浏览器，也给了你操控其他机器上浏览器的机会。

- 3. Selenium WebDriver，RC 能做的WebDriver也能做，后面要分享的就是基于WebDriver 的自动化测试框架，容后细禀。

- 4. Selenium Grid 这个就NB了，它可以使不同平台上不同浏览器一起执行（at the same time），不需要要你手忙脚乱的像个无头苍蝇一样忙到吐血，它可以让你觉得一切都有组织都有架构，都是可控制的，你可以尽情的往深处想发挥它的作用！

- 初探就这么滴了，总算艰难的迈出了一小步开张了^_^，请轻拍！

WebDriver Element locating
ByID

- 在正式创建测试项目之前有必要详细介绍下怎么样去寻找页面元素，当然就是根据网页元素的属性了，想必大家一定都很熟知常用的浏览器比如Chrome和Firefox了，一般化在网页上选中你想查看的元素然后右击审查元素就可以看到该元素的属性了，知道了这些元素属性又怎么和webdriver结合起来呢？下面就来介绍下webdriver locate elements的方法，webdriver中用findElement(By.**())关键字来locate 元素

- 第一种最快也是最方便的方法：： By ID

- 通常网页上的元素会有一个唯一识别的ID 就像我们的身份证一样，它用来唯一标识某元素，WebDriver通过这种方法去发现元素的语法很简单

- driver.findElement(By.id("******"))；这里只要知晓元素的相对应的id就可以找到该元素了，也许会有人说我怎么知道id？直接检查页面元素就可以在html代码中看到你想要的元素的id了，比如百度页面的搜索框相对应的html代码为：<input type="text" name="wd" id="kw" maxlength="100" class="s_ipt" autocomplete="off">这里可以清楚的看到该文本框的id为kw，相应的发现元素代码就应该为：driver.findElement(By.id("kw"))； 注意：不是所有的元素都有id，这个要根据但是开发的设计。

WebDriver Element locating
ByName

- By Name

- driver.findElement(By.name("*****"));这里只要知道相对应的元素的name就可以成功的找到元素了，还以百度首页的搜索框为例，搜索框的name为wd，相应的代码应为driver.findElement(By.name("wd"))；当然也不是所有的元素都有name。

WebDriver Element locating ByXpath

- By xpath
- driver.findElement(By.xpath("****"));这里主要是通过元素的xpath来定位元素的，xpath的知识有兴趣的可以访问http://www.w3school.com.cn/xpath/去了解下，这里不做介绍，下面介绍几种即使你完全不懂xpath的语法你也照样可以运用xpath来发现元素的方法：
- #1利用firefox的xpath插件在add-on中搜索xpath，然后选择 xpath checker，安装完成后，就可以运用xpath了，比如我想得到百度首页logo的xpath值只要把光标移到图片上右击view xpath 就可以得到xpath的值了：id('lg')/x:img，所以这里相对应的代码就应该为：driver.findElement(By.xpath("id('lg')/x:img"))
- #2利用chrome的xpath插件，这个插件个人感觉比firefox下的插件更好用，出错的几率也相应的小了很多，该插件为PsychoXpath:https://chrome.google.com/webstore/detail/psychoxpath/bpnigkcdmnofjkmojlopmelmhgpbndog 安装成功后，选定相应的元素右击你可以选择绝对路径（Absolute），这里不推荐这种绝对路径的，强烈推荐第二种（short），当然你也可以highlight元素，自己试试绝对感觉很有意思，获取到相应的xpath后就可以把值相应的替换到driver.findElement(By.xpath(“”))中就可以了。

WebDriver Element locating cssSelector

- By CssSelector，这种方法据说比用xpath要快，性能比用xpath要好”很多"，在这里我想说，如果真的要钻牛角尖，那真比用xpath要快，但是差距可能也就是0.X秒，但是cssSelector比xpath更容易维护倒是真的，个人也比较喜欢这个方法，也比较容易上手，推荐大家看看css的东西对于加深UI的理解还是很有帮助的（http://www.w3school.com.cn/css/css_syntax_attribute_selector.asp）这里还要提下怎么样找到元素更简单就怎么用，至于脚本开发效率和以后的维护和性能自己权衡利弊，重要的是达成一致的规范，把简单的事情做复杂了那不是牛逼那是傻逼，貌似扯淡了。。。。

- 基本语法为driver.findElement(By.cssSelector("****"));这里介绍一个工具叫firefind，firebug下的插件，这个插件可以帮助你用cssSelector识别元素，举个例子比如说我要找百度首页的那个百度logo，首先用firefox打开百度页面，然后按下F12打开firebug，切换到FireFinder，然后输入CSS属性去匹配，如果属性描述正确，百度logo将被高亮出来如图：

WebDriver Element locating
linkText

- By linkText，这种方法就更直白了，就是根据link的名称来识别元素，比如百度首页有一个地图的link，只要根据link 的名字”地图"就可以识别了,基本语法为：driver.findElement(By.linkText("地图"));

WebDriver Element locating className

- By className，这种方法就是根据元素的class属性来识别元素的，但是可能有好多的元素的class name 是一样的，这样取到的就是元素的集合，基本语法为：driver.findElement(By.className(“**"));

WebDriver Element locating TagName

- 第七种方法为: By TagName，这种方法就是根据元素的TagName来识别元素的，比如有一元素html代码为：<iframe. src="..."></iframe> 就可以用这样的语法来识别该元素：driver.findElement(By.tagName("iframe"));

- 当然还有很多其他的方法，这里就不一一描述了，方法是死的，能用好，能好用才有价值!

TestNG初探

- TestNG是什么?

  - TestNG按照其文档的定义是：TestNG是一个测试框架，其灵感来自JUnit和NUnit的，但引入了一些新的功能，使其功能更强大，使用更方便

  - TestNG是一个开源自动化测试框架;TestNG表示下一代。TestNG是类似于JUnit（特别是JUnit 4），但它不是一个JUnit扩展。它的灵感来源于JUnit。它的目的是优于JUnit的，尤其是当测试集成的类。 TestNG的创造者是Cedric Beust（塞德里克·博伊斯特）

  - TestNG消除了大部分的旧框架的限制，使开发人员能够编写更加灵活和强大的测试。 因为它在很大程度上借鉴了Java注解（JDK5.0引入的）来定义的测试，它也可以告诉你如何使用这个新功能在真实的Java语言生产环境中。

TestNG特点

- TestNG的特点
  - 注解
  - TestNG使用Java和面向对象的功能
  - 支持综合类测试（例如，默认情况下，没有必要创建一个新的测试每个测试方法的类的实例）
  - 独立的编译时间测试代码运行时配置/数据信息
  - 灵活的运行时配置
  - 主要介绍“测试组”。当编译测试，只要问TestNG运行所有的“前端”的测试，或“快”，“慢”，“数据库”等
  - 支持依赖测试方法，并行测试，负载测试，局部故障
  - 灵活的插件API，支持多线程测试

TestNG注解

- 注解 描述
- @BeforeSuite 注解的方法将只运行一次，运行所有测试前此套件中。
- @AfterSuite 注解的方法将只运行一次此套件中的所有测试都运行之后。
- @BeforeClass 注解的方法将只运行一次先行先试在当前类中的方法调用。
- @AfterClass 注解的方法将只运行一次后已经运行在当前类中的所有测试方法。
- @BeforeTest 注解的方法将被运行之前的任何测试方法属于内部类的 <test>标签的运行。
- @AfterTest 注解的方法将被运行后，所有的测试方法，属于内部类的<test>标签的运行。
- @BeforeGroups 组的列表，这种配置方法将之前运行。此方法是保证在运行属于任何这些组第一个测试方法，该方法被调用。
- @AfterGroups 组的名单，这种配置方法后，将运行。此方法是保证运行后不久，最后的测试方法，该方法属于任何这些组被调用。

TestNG注解

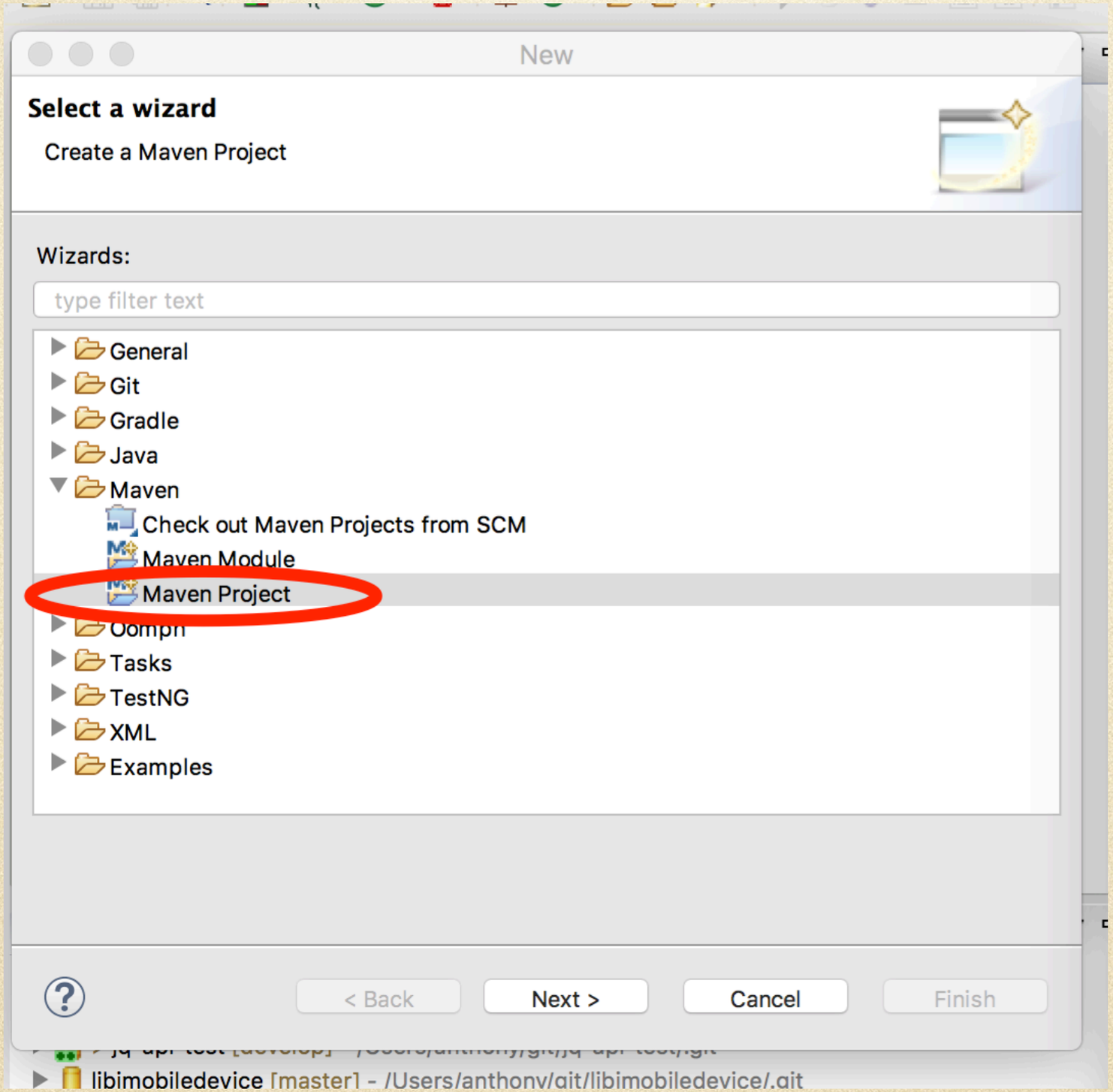
- 注解　　描述
- @BeforeMethod　　注解的方法将每个测试方法之前运行。
- @AfterMethod 被注释的方法将被运行后，每个测试方法。
- @DataProvider
- 标志着一个方法，提供数据的一个测试方法。注解的方法必须返回一个Object[] []，其中每个对象[]的测试方法的参数列表中可以分配。
- 该@Test 方法，希望从这个DataProvider的接收数据，需要使用一个dataProvider名称等于这个注解的名字。
- @Factory　　作为一个工厂，返回TestNG的测试类的对象将被用于标记的方法。该方法必须返回Object[]。
- @Listeners　　定义一个测试类的监听器。
- @Parameters 介绍如何将参数传递给@Test方法。
- @Test　 标记一个类或方法作为测试的一部分。

TestNG进阶参数化

- 简述下testng做参数化的几种方式:
  - @Parameters
  - dataProvider
  - 详情请访问本人博客 http://anthonygao.blog.51cto.com/8653110/1660824

自动化初遇之项目创建

- 打开Eclipse，新建一个maven pro
- 一步步next到设置group id 和 artifact id
- 点击完成结束项目创建


New
Select a wizard
Create a Maven Project
Wizards:
type filter text
General
Git
Gradle
Java
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project
Oomph
Tasks
TestNG
XML
Examples
< Back
Next >
Cancel
Finish
libimobiledevice [master] - /Users/anthonv/git/libimobiledevice/.git

New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.training

Artifact Id: newtrain

Version: 0.0.1-SNAPSHOT

Package: com.training.newtrain

Properties available from archetype:

Name	Value

Add...

Remove

Advanced

< Back | Next > | Cancel | Finish

自动化初遇之配置项目依赖

- 打开新建的maven项目的pom.xml文件
- 添加项目依赖如下图并保存

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.anthony.test</groupId>
  <artifactId>training</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>training</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>3.0.0-beta1</version>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.9.10</version>
        </dependency>
        <dependency>
            <groupId>net.sourceforge.jexcelapi</groupId>
            <artifactId>jxl</artifactId>
            <version>2.6.10</version>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.5</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.0</version>
        </dependency>
  </dependencies>
</project>
```

项目组织

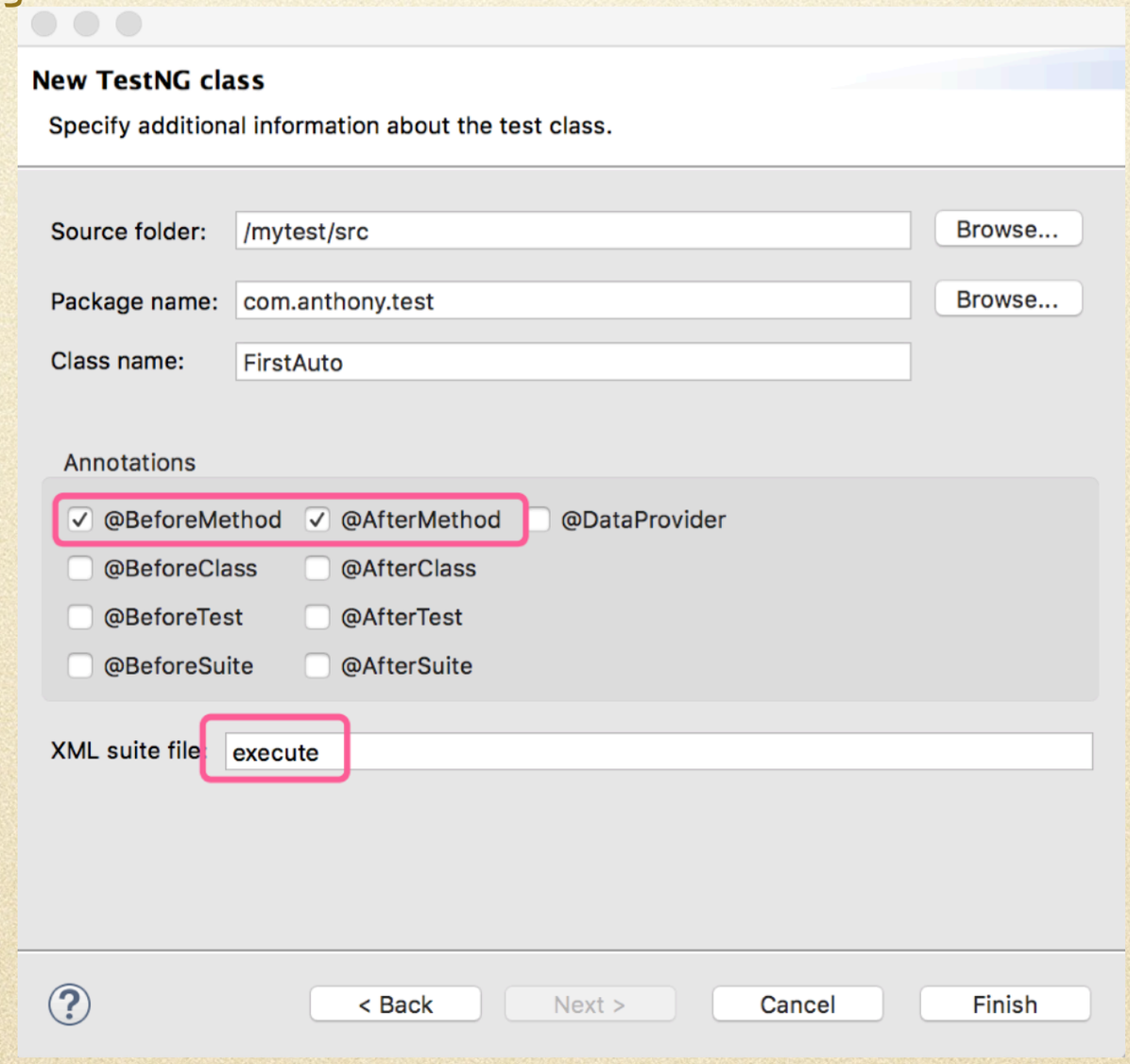
项目依赖类库，会进行自动下载
不需要手工添加

自动化测试初例之
测试分析

- 测试场景：访问携程官网，完成登陆
- 进行粗略分析，完成这个场景需要以下几步
    - 打开携程官网首页
    - 点击登录链接 / 按钮
    - 输入用户名&密码
    - 点击登录
    - 验证登录结果

自动化测试之初次打码

- 在刚刚创建的项目中新建一个testng测试类如右图所示，会自动生成初始化的测试代码



自动化测试之初次打码

在@Test的f(){}中添加以下代码就可以完成我们想要的自动化测试了

```
System.setProperty("webdriver.chrome.driver", "webdriverServer/chromedriver");
WebDriver driver = new ChromeDriver();
driver.get("http://www.ctrip.com");

WebElement loginLink = driver.findElement(By.linkText("登录"));
loginLink.click();
WebElement userName = driver.findElement(By.id("txtUserName"));
userName.sendKeys(“******”);
WebElement password = driver.findElement(By.name("txtPwd"));
password.sendKeys(“******”);
WebElement loginButton = driver.findElement(By.cssSelector("input[type='submit']"));
loginButton.click();

Assert.assertTrue(driver.findElement(By.cssSelector("p.user_name")).getText().contains("尊敬的会员"), “会员信息出现了耶～");
```

然后在FirstAuto类中直接右击，选择run as TestNG test就可以观察到测试代码的执行了，是不是很简单？很容易实践？但是，是不是还觉得没底？自动化测试就这么做？老司机下次再讲

开启简易测试框架之路
MyBy篇

之前有记录了关于怎样搭建webdriver的测试环境，也记录了一些小技巧，也给出实例怎样真正开始动手写一个自动化测试的case.但是真正的自动化测试项目中，只会那些还是远远不够的，细想一下，如果每一个步骤要有driver.findElement只论可读性的话就该够你喝一壶的了，更别谈维护了。这时脑子上灯突然闪了下，貌似你好像有了些灵感，那些重复的语句，可以封装吗？那么我们就先从findElement的方法开始我们的框架设计之旅。

我们首先来看看原生的webdriver找元素的方法,以id为例： driver.findElement(By.id(" ")),暂时抛开driver, findElement方法中包含了2个部分By 和id（元素的属性）By是原生的关键字，这里我们也先不动它，我们先从id入手，其实不光id，有很多通过元素属性找元素的方法，比如: id,name,cssSelector,xpath,linkText,tagName等等等。。这里我们就可以做文章了，比如我们实例化一个By locator 的方法，把有可能用到的元素属性都列在里面，根据用户提供的元素属性的不同调用不同的方法，元素属性包含2个部分：属性名称，属性value中间可用’：’隔开。

1. 如果用户的输入没有’:’，那么默认调用By.id()方法
2. 如果用户的输入包含’:’,那么根据实际情况来判断，通过’:’把元素用户输入分开，通过’:’左边分离出来的字符串来判断具体的调用方法。通过’:’右边分离出来的字符串来传入实际的属性value

具体实现如下：

```java
/**
 * @author Anthony
 *
 * 1. 如果用户的输入没有‘:’，那么默认调用By.id()方法
 * 2. 如果用户的输入包含':',那么根据实际情况来判断
 * 通过‘:’把元素用户输入分开，通过':'左边分离出来的字符串来判断具体的调用方法。
 * 通过‘:’右边分离出来的字符串来传入实际的属性value
 *
 */
public abstract class MyBy extends By {

  public static By locator(String locator) {
    if (!locator.contains(":")) {
      return By.id(locator);
    } else {
      String[] lArr = locator.split(":");
      String by = lArr[0];
      String using = locator.substring(by.length() + 1);
      if (by.equalsIgnoreCase("id")) {
        return By.id(using);
      } else if (by.equalsIgnoreCase("name")) {
        return By.name(using);
      } else if (by.equalsIgnoreCase("xpath")) {
        return By.xpath(using);
      } else if (by.equalsIgnoreCase("cssSelector")) {
        return By.cssSelector(using);
      } else if (by.equalsIgnoreCase("linkText")) {
        return By.linkText(using);
      } else if (by.equalsIgnoreCase("partialLinkText")) {
        return By.partialLinkText(using);
      } else if (by.equalsIgnoreCase("tagName")) {
        return By.tagName(using);
      } else if (by.equalsIgnoreCase("className")) {
        return By.className(using);
      } else {
        print(" Element " + locator + "cannot be found.. ");
        throw new IllegalArgumentException("Can't find element");
      }
    }
  }
```

开启简易测试框架之路
MyWebElement篇

定义完我们自己的MyBy后，那我们怎么用呢？如何才能把找元素的方法封装完毕呢？接下来我们需要
定义自己的MyWebElement并且实现原生WebElement,添加属于自己的getElement方法来调用MyBy的找元素方法

```
public WebElement getWebElement() {
 driver.manage().timeouts().implicitlyWait(IMPLICITLY_WAIT_TIME, TimeUnit.SECONDS);
 return driver.findElement(MyBy.locator(locator));
}

public MyWebElement(String locator) {
 driver.manage().timeouts().implicitlyWait(IMPLICITLY_WAIT_TIME, TimeUnit.SECONDS);
 this.locator = locator;
}
```

同时重新实现WebElement自身的必要函数，比如click，submit……等，详情见下文：

```java
/**
 * @author anthony
 * @date 2018年5月29日
 * @updateTime 上午11:45:59
 */
public class MyWebElement implements WebElement {

  private String locator;
  private final int IMPLICITLY_WAIT_TIME = 30;

  public <X> X getScreenshotAs(OutputType<X> arg0) throws WebDriverException {
    return getWebElement().getScreenshotAs(arg0);
  }

  public void click() {
    getWebElement().click();
  }

  public void submit() {
    getWebElement().submit();

  }

  public void sendKeys(CharSequence... keysToSend) {
    getWebElement().sendKeys(keysToSend);
  }

  public void clear() {
    getWebElement().clear();

  }
```

```
public String getTagName() {
  return getWebElement().getTagName();
}

public String getAttribute(String name) {
  return getWebElement().getAttribute(name);
}

public boolean isSelected() {
  return getWebElement().isSelected();
}

public boolean isEnabled() {
  return getWebElement().isEnabled();
}

public String getText() {
  return getWebElement().getText();
}
```

开启简易测试框架
BasePage篇

定义好了我们自己的MyWebElement后，我们是不是已经大体猜到了此次框架设计的小心思，是不是自动化测试中一个经常用的词要脱口而出了？对，就是 **对象库设计模式**

定义自己的BasePage，封装一些最常用的公用方法如，打开浏览器，访问页面url，同时以对象库的方式来存储页面元素从而实现页面的元素的高效管理

开启简易测试框架
BasePage篇

```
/**
 * @author anthony
 * @date 2018年5月29日
 * @updateTime 下午1:25:25
 */
public class BasePage {

  public static WebDriver driver;


  public static void openBrowser(String browser) {
    try {
        if (browser.equalsIgnoreCase("firefox")) {
            driver = new FirefoxDriver();
        } else if (browser.equalsIgnoreCase("winchrome")) {
            System.setProperty("webdriver.chrome.driver", "webdriverServer/chromedriver.exe");
            driver = new ChromeDriver();
        } else if (browser.equalsIgnoreCase("macchrome")) {
            System.setProperty("webdriver.chrome.driver", "webdriverServer/chromedriver");
            driver = new ChromeDriver();
        } else if (browser.equalsIgnoreCase("safari")) {
            driver = new SafariDriver();
        } else {
            System.setProperty("webdriver.ie.driver", "webdriverServer/IEDriverServer.exe");
            driver = new InternetExplorerDriver();
        }
        driver.manage().window().maximize();
    } catch (Exception e) {
        e.printStackTrace();
    }
  }

}
```

开启简易测试框架
BasePage篇

```
/**
 * 打开页面
 *
 * @param url
 */
public static void navigateTo(String url) {
    driver.get(url);
}

public static MyWebElement myWebelement(String locator) {
  return new MyWebElement(locator);
}

}
```

简易测试框架里程碑

至此，简易测试框架就基本搭建完成了，核心三要素是什么?

MyBy
MyWebElement
BasePage

简单的测试框架已经完成，那如何才能使用起来呢？写用例的方式也还和以前一样么？亮点又在哪里呢？接下来我们进入自动化测试进阶章节……

UI自动化测试进阶

首先我们从自动化测试分析的设计开始，比如请看如下自动化测试用例：

1. 访问 http://www.ctrip.com
2. 点击登录链接
3. 在登录页面输入用户名和密码完成登录

检查点：

登录成功后，页面跳转回首页，并出现欢迎词

分析后我们可以得出如下结果：

UI自动化测试进阶

本用例共经历2个页面，5个元素

测试页面	页面元素	测试用例	检查点
携程首页	登录链接 欢迎词	打开携程首页 点击登录按钮	首页出现欢迎词
携程登录页	用户名输入框 密码输入框 登录按钮	输入用户名 输入密码 点击登录	

UI自动化测试进阶
测试脚本开发篇

定义对象库CtripHomePage
定义对象searchBox，searchButton，flightLink，loginLink，userInfo

```
/**
 * @author anthony
 */

public class CtripHomePage extends BasePage {

        public static MyWebElement searchBox = myWebelement("_allSearchKeyword");

        public static MyWebElement searchButton = myWebelement("search_button_global");

        public static MyWebElement flightLink = myWebelement("linkText:机票");

        public static MyWebElement loginLink = myWebelement("c_ph_login");

        public static MyWebElement userInfo = myWebelement(“cssSelector:p.user_name");

}
```

UI自动化测试进阶
测试脚本开发篇

定义测试用例库 CtripLoginTest
定义测试用例searchTest

```
/**
 * @author anthony
 *
 */

public class CtripLoginTest {
	@BeforeMethod
	public void beforeMethod() {
		openBrowser("macchrome");
		navigateTo("http://www.ctrip.com/");

	}

	@Test(dataProviderClass = TestDataProvider.class, dataProvider = "userInfo")
	public void searchTest(String username, String password) {
		CtripHomePage.loginLink.click();
		CtripLoginPage.userName.input(username);
		CtripLoginPage.password.input(password);
		CtripLoginPage.loginButon.click();
		Assert.assertTrue(CtripHomePage.userInfo.getText().contains("尊敬的会员"), "会员信息出现了耶～");
	}

	@AfterMethod
	public void afterMethod() {
		closeBrowser();
	}

}
```

UI自动化测试进阶
测试脚本开发篇

自动化测试过程最长用的场景就是批量数据的测试，我们怎么做呢？这里推荐使用TestNG的DataProvider来进行批量数据的执行，这里我自己封装了一个getData的工具用来读取Excel中的测试数
据,DataProvider的定义方式只要按照下面的形式定义即可接下来就是准备好测试数据就好了。

```
/**
 *
 * @author anthony
 *
 */

public class TestDataProvider extends BaseTestSuite {

	/*
	 * Read test data from test data file
	 */
	public static Object[][] getData(String xlFilePath, String sheetName, String tableName) {

		String[][] data = (String[][]) DataStream.getDataByTableName(xlFilePath, sheetName, tableName, 0);
		return data;
	}

	@DataProvider(name = "userInfo")
	public static Object[][] userInfo() throws Exception {
		Object[][] accountInfo = getDataTable("testData.xls", "用户信息", "userInfo");
		return accountInfo;
	}

}
```

UI自动化测试进阶
测试执行控制

花了很长的篇幅讲解了自动化测试的初打码，测试简易框架开发，基于框架的测试用例开发，接下来我们再来聊聊测试执行，本人推荐使用TestNG作为测试用例执行的控制器

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Training">
      <test name="Training" preserve_order="true">
            <classes>
                  <class name="com.anthony.test.training.CtripLoginTest" />
            </classes>
      </test>
</suite>
```

如果你有更多的测试用例库的话，只要继续添加 <class name="*******" />,这样做还有一个好处就是还可以控制测试用例库的执行顺序，做过自动化测试的小伙伴们可能会有一个痛点：测试用例执行顺序不好控制，用例执行顺序影响测试结果，这里我们先不讨论用例设计的合理性，单从执行顺序上下文章的话，我们应该怎么做？testNG可以完美的解决你的痛点

UI自动化测试进阶
测试用例控制篇

在TestNG xml配置文件中，关于<test>的配置里面，有一个属性叫preserve-order，是用来控制<test>里面所有<classes>的执行顺序的。<test>默认下的preserve-order为true，表示<test>下所有<classes>顺序执行，例如下面这段testng的配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="aa">
<test name="aa" preserve_order="true">
<classes>
<class name="com.anthony.BTest" />
<class name="com.anthony.ABTest" />
<class name="com.anthony.ATest" />
</classes>
</test>
</suite>
```

会依次执行BTest,ABTest,ATest中的测试方法。

UI自动化测试进阶
测试用例控制篇

通过测试用例命名a-z的顺序

```
@Test
public void a(){
System.out.println("我是Atest 的a 方法");
}

@Test
public void aa(){
System.out.println("我是Atest 的aa 方法");
}

@Test
public void ab(){
System.out.println("我是Atest 的ab 方法");
}
```

上述代码执行后的输出就是：

我是Atest 的a 方法
我是Atest 的aa 方法
我是Atest 的ab 方法

UI自动化测试进阶
测试用例控制篇

- 使用priority指定执行顺序(默认值为0)，数值越小，越靠前执行

```
@Test(priority=1)
public void a(){
System.out.println("我是Atest 的a 方法");
}

@Test(priority=2)
public void ab(){
System.out.println("我是Atest 的ab 方法");
}

@Test(priority=3)
public void aa(){
System.out.println("我是Atest 的aa 方法");
}
```

这样指定后，ATest执行完成的输出应该就是：

```
我是Atest 的a 方法
我是Atest 的ab 方法
我是Atest 的aa 方法
```

UI自动化测试进阶
测试用例控制篇

- 在xml里面使用<include>指定需要执行的方法和顺序

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="aa">
   <test name="aa" preserve_order="true">
<classes>
   <class name="com.anthony.ATest" />
<methods>
<include name = "ab"/>
<include name = "aa"/>
<include name = "a"/>
</methods>
</classes>
   </test>
</suite>
```

这样指定后，ATest执行完后的输出就应该是：

我是Atest 的ab 方法
我是Atest 的aa 方法
我是Atest 的a 方法

到此，Web自动化测试就介绍完毕了

谢谢大家～