目录

[什么是自动化测试 2](#_Toc20013)

[什么样的项目适合做自动化测试 2](#_Toc586)

[自动化测试的分类 2](#_Toc28548)

[自动化测试的测试流程 3](#_Toc15895)

[什么是Selenium？ 3](#_Toc14029)

[Selenium有什么限制或者缺陷Selenium有什么限制或者缺陷 3](#_Toc27083)

[在selenium中，有哪些不同定位元素方法在selenium中，有哪些不同定位元素方法 3](#_Toc23071)

[什么是imlicitlyWait 3](#_Toc2855)

[selenium中hidden或者是display ＝ none的元素是否可以定位到 4](#_Toc22582)

[selenium中如何保证操作元素的成功率？也就是如何保证点击的元素一定是可以点击的？ 4](#_Toc24074)

[如何提高selenium脚本的执行速度？ 4](#_Toc7478)

[怎么判断元素是否存在？ 5](#_Toc26280)

[自动化测试最大的缺陷是什么？ 5](#_Toc19111)

[如何去提升用例的稳定性？ 5](#_Toc7705)

[什么是分层测试？ 5](#_Toc9252)

[代码管理工具 6](#_Toc21714)

[Selenium：元素等待的4种方法 6](#_Toc28297)

作者：张强

# 什么是自动化测试

自动化测试是把以人为驱动的测试行为转化为机器执行的一种过程。

通常，在设计了测试用例并通过评审之后，由测试人员根据测试用例中描述的规程一步步执行测试，得到实际结果与期望结果的比较。在此过程中，为了节省人力、时间或硬件资源，提高测试效率

# 什么样的项目适合做自动化测试

1.任务测试明确，不会频繁变动

2.每日构建后的验证测试（每日构建 每天发布一个版本 敏捷项目）

3.比较频繁的回归测试

4.软件系统界面稳定，变化少

5.需要在多平台运行相同的测试用例、组合遍历型的测试、大量的重复测 浏览器的兼容、操作系统的兼容

6.软件维护周期长：项目已开发一年以上，版本非常稳定、需要长期维护

7.项目进度压力不太长：项目非常急

8.被测系统软件开发比较规范，能够保证系统的可测试性

9.具备大量的自动化测试平台

10.测试人员具备较强的编程能力

注意：一般情况下，

只要满足如下三个情况就可以开展自动化测试，

软件需求变更不频繁、项目周期较长、自动化测试脚本可重复使用

# 自动化测试的分类

单元测试、接口测试、UI测试

# 自动化测试的测试流程

可行性分析--》测试需求分析--》制定测试计划--》

自动化测试设计--》测试脚本开发--》无人职守测试--》

提交测试报告--》脚本维护阶段

# 什么是Selenium？

Selenium是一个开源的web自动化测试框架，主要是基于web uI的自动化测试。现在的版本，逐步增加了对移动端的自动化测试。Selenium支持多种语言进行开发自动化测试脚本，有Java,python，C#，Javascript等等。Selenium支持跨浏览器平台测试

# Selenium有什么限制或者缺陷Selenium有什么限制或者缺陷

除了基于web的软件和mobile的程序，selenium不支持桌面软件自动化测试。软件测试报告，和用例管理只能依赖第三方插件，例如Junit/TestNG和unittest。由于它是免费的软件，所以没有供应商去提供支持和服务，有问题，只能求助selenium社区。还有一个就是，selenium入门门槛可能有点高，需要具备一定编程语言基础

# 在selenium中，有哪些不同定位元素方法在selenium中，有哪些不同定位元素方法

ID/className/Name/LinkText/PartialLinkText/Xpath/CSS selector

# 什么是imlicitlyWait

imlicitlyWait是隐式等待，一般在查找元素的时候使用。例如，我设置一个查找元素最大时间为10秒，使用了imlicitlyWait后，如果第一次没有找到元素，会在10秒之内不断循环去找元素，知道超过10秒，报超时错误。

# selenium中hidden或者是display ＝ none的元素是否可以定位到

答案-不能

# selenium中如何保证操作元素的成功率？也就是如何保证点击的元素一定是可以点击的？

答案- 添加元素智能等待时间 driver.implicitly\_wait(30)

- try 方式进行 id,name,clas,x path, css selector 不同方式进行定位，如果第一种失败可以自动尝试第二种-Selenium保证元素成功率是通过元素的定位，当然它的定位方法很多，一定能有合适的。但是在自动化工程的实施过程中，高质量的自动化测试不是只有测试人员保证的。需要开发人员规范开发习惯，如给页面元素加上唯一的name,id等，这样就能大大地提高元素定位的准确性。当然如果开发人员开发不规范，我们在定位元素的时候尽量使用相对地址定位，这样能减少元素定位受页面变化的影响。只要我们元素定位准确，就能保证我的每一个操作符合我的预期

# 如何提高selenium脚本的执行速度？

答案-- Selenium脚本的执行速度受多方面因素的影响，如网速，操作步骤的繁琐程度，页面加载的速度，以及我们在脚本中设置的等待时间，运行脚本的线程数等。所以不能单方面追求运行速度的，要确保稳定性，能稳定地实现回归测试才是关键。

我们可以从以下几个方面来提高速度：

一，减少操作步骤，如经过三四步才能打开我们要测试的页面的话，我们就可以直接通过网址来打开，减少不必要的操作。

二，中断页面加载，如果页面加载的内容过多，我们可以查看一下加载慢的原因，如果加载的内容不影响我们测试，就设置超时时间，中断页面加载。

三，在设置等待时间的时候，可以sleep固定的时间，也可以检测某个元素出现后中断等待也可以提高速度。

四，配置testNG实现多线程。在编写测试用例的时候，一定要实现松耦合，然后在服务器允许的情况下，尽量设置多线程运行，提高执行速度。

# 怎么判断元素是否存在？

答案--判断元素是否出现，存在两种情况，一种是该元素压根就没有，自然不会出现；另外一种是有这样的元素，但是是hidden状态

可以通过先判断是否存在，如果不存在返回false；如果存在再去判断是否displayed

# 自动化测试最大的缺陷是什么？

不稳定

-可靠性

-不易维护

-成本与收益

# 如何去提升用例的稳定性？

time.sleep( )

-driver.implicitly\_wait(30)

-多用 try 捕捉，处理异常

# 什么是分层测试？

行业里面提的一般是金字塔的分层模型

UI测试

集成/接口测试

单元测试

# 代码管理工具

Git svn 区别

# Selenium：元素等待的4种方法

1. 使用Thread.sleep()，这是最笨的方法，但有时候也能用到而且很实用。

2. 隐示等待，隐性等待是指当要查找元素，而这个元素没有马上出现时，告诉WebDriver查询Dom一定时间。默认值是0,但是设置之后，这个时间将在WebDriver对象实例整个生命周期都起作用。

WebDriver dr = new FirefoxDriver();

dr.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

3. 使用javascript

WebElement element = driver.findElement(By.xpath(test));

((JavascriptExecutor)driver).executeScript("arguments[0].style.border="5px solid yellow"",element);

4. 显示等待,推荐使用显示等待

WebDriverWait wait = new WebDriverWait(dr, 10);

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

显式等待 使用ExpectedConditions类中自带方法， 可以进行显试等待的判断。

显式等待可以自定义等待的条件，用于更加复杂的页面等待条件

（1）页面元素是否在页面上可用和可被单击：elementToBeClickable(By locator)

（2）页面元素处于被选中状态：elementToBeSelected(WebElement element)

（3）页面元素在页面中存在：presenceOfElementLocated(By locator)

（4）在页面元素中是否包含特定的文本：textToBePresentInElement(By locator)

（5）页面元素值：textToBePresentInElementValue(By locator, java.lang.String text)

（6）标题 (title)：titleContains(java.lang.String title)

只有满足显式等待的条件满足，测试代码才会继续向后执行后续的测试逻辑

如果超过设定的最大显式等待时间阈值， 这测试程序会抛出异常。

WebDriverWait wait = new WebDriverWait(driver,5);

wait.until(ExpectedConditions.presenceOfElementLocated(By.cssSelector("")));