# 自动化测试基础

## 软件测试分类

1. 根据项目流程阶段划分软件测试：
   1. 需求分析
   2. 设计
   3. 编码
   4. 单元测试：
      1. 对程序中的单个子程序或具有独立功能的代码段进行测试
   5. 集成测试：
      1. 在单元测试的基础上，通过单元模块组装成系统或子系统，再进行测试
      2. 重点是检查模块之间的接口是否正确
   6. 系统测试：
      1. 针对整个产品系统进行的测试
      2. 验证系统是否满足需求规格的定义
      3. 验证软件系统的正确性和性能等是否满足其规约所指定的要求
   7. 验收测试：
      1. 部署软件之前的最后一个测试操作
      2. 目的是确保软件准备就绪，向软件购买者展示该系统满足用户的需求
2. 根据软件代码的可见程度划分（软件测试领域最基本的概念）：
   1. 黑盒测试：
      1. 把被测的软件看作一个黑盒子，不关心盒子里面的结构是什么，只关心软件的输入数据和输出结果。
      2. 只检查程序呈现给用户的功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接受收入数据而产生正确地输出信息。
      3. 着眼于程序外部结构，不考虑内部逻辑结构，主要针对**软件界面和软件功能**进行测试。
   2. 白盒测试：
      1. 把盒子打开，研究里面的**源代码**和程序**执行结果**
      2. 按照程序内部的结构测试程序，通过测试检测产品内部动作是否按照设计规格说明书的规定正常进行，检验程序中的每条通路是否都能按预定要求正确工作。
   3. 灰盒测试：
      1. 介于黑盒测试和白盒测试之间
      2. 关注输出对于输入的**正确性**，同时也关注**内部表现**。但不如白盒测试详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态
      3. 有时候输出是正确的，但内部已经错误了。如果采用白盒测试效率会比较低。
3. 从对软件的不同测试点进行划分：
   1. 功能测试：
      1. 检查实际的功能是否符合用户的需求。设计软件的目的也就是满足客户对其功能的需求。
      2. 又细分为很多种：逻辑功能测试、界面测试、易用性测试、安装测试、兼容性测试等。
   2. 性能测试：
      1. 性能测试是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统地各项性能指标进行测试。
      2. 时间性能：
         1. 主要指软件的一个具体响应时间。
         2. 比如一个登陆需要的时间，一个交易需要的时间
         3. 需要搭载一个具体且独立的测试环境，否则没什么意义。
      3. 空间性能：
         1. 主要指软件运行时所消耗的系统资源
         2. 比如硬件资源，CPU、内存、网络带宽消耗等
4. 从对软件测试工作的自动化程度划分：
   1. 手工测试：
      1. 由人去一个一个的执行测试用例，通过键盘鼠标等输入一些参数，查看返回结果是否符合预期结果。
      2. 通常是指我们在系统测试阶段进行的功能测试
   2. 自动化测试：
      1. 为了节省人力、时间或硬件资源，提高测试效率，把测试转化为机器执行
      2. 功能自动化测试：
         1. 这是一般说的自动化测试
         2. 通过编码的方式测试一个软件的功能，重复执行程序以重复测试
         3. 如果一个软件一小部分发生改变，只需要修改一部分自动化测试代码即可
      3. 性能自动化测试：
         1. 通过性能测试工具辅助完成
         2. 通过工具模拟成千上万的用户向系统发送请求，来验证系统地处理能力
5. 出现在软件功能测试周期的：
   1. 冒烟测试：
      1. 对一个系统大规模测试之前，先验证一下软件的基本功能是否实现，是否有可测性。也就是先投入较少的人力和时间验证一个软件的主要功能。
      2. 节省大量时间成本和人力成本。
   2. 回归测试：
      1. 修改了旧代码后，重新进行测试以确认修改后没有引入新的错误或者导致其他代码产生错误。
      2. 一般是在软件的第二轮测试开始，验证第一轮中发现的问题是否得到修复
      3. 回归也是个循环的过程，如果回归后通不过，就再次回归知道通过。
   3. 随机测试：
      1. 测试中所有输入数据都是随机生成的
      2. 目的是模拟用户的真是操作，并发现一些边缘性的隐蔽错误
      3. 缺点：
         1. 不系统
         2. 无法统计代码覆盖率和需求覆盖率
         3. 发现的问题难以重现
      4. 一般放在测试的最后执行
6. 其他测试：
   1. 探索性测试：
      1. 是一种测试思维技术，没有很多实际的测试方法、技术、工具
      2. 强调测试人员的主管能动性，强调在碰到问题时及时改变测试策略，抛弃繁杂的测试计划和测试用例设计过程。
   2. 安全测试：
      1. 在产品开发基本完成到发布阶段，对产品进行检验以验证产品符合安全需求定义和产品质量标准
      2. 越来越受到关注和重视，因为由于安全性问题造成的后果是不可估量的。

## 适合自动化的项目

1. 不是所有项目都适合实施自动化测试的。
2. 适合自动化测试的项目的特点：
   1. 任务明确，不会频繁变动
   2. 每日构建后的测试验证
   3. 比较频繁的回归测试
   4. 软件系统界面稳定，变动少
   5. 需要在多平台运行的相同测试案例、组合遍历型测试，大量的重复任务
   6. 软件维护周期长
   7. 项目进度压力不太大
   8. 被测试的软件系统开发比较规范，能够保证可测试性
   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
3. 在普遍经验中，基本满足以下三点即可展开自动化测试：
   1. 软件需求编程不频繁：
      1. 如果软件需求变动过于频繁，则对测试脚本的维护成本将比较高
      2. 如果项目某些模块相对稳定，某些模块需求变动大，则可分别用自动化和手工
   2. 项目周期较长：
      1. 自动化测试需求的确定、自动化测试框架的设计、测试脚本的编写与调试都需要比较长的时间
      2. 因此如果项目的周期短的连测试软件都来不及开发，那自然不需自动化测试
   3. 自动化测试脚本可重复使用：
      1. 衡量所测试的项目之间是否有很大的差异
      2. 衡量所选择的测试工具是否适应项目间的差异
      3. 衡量测试人员是否有能力开发出适应这种差异的自动化测试框架

## 自动化测试及工具

1. 自动化测试的狭义和广义概念：
   1. 广义上，所有借助工具进行的软件测试
   2. 狭义上，主要指基于UI层的自动化测试（一般都是指这个）
   3. 其他：基于代码编写阶段的单元自动化测试，基本集成测试阶段的接口自动化测试
2. QTP自动化测试工具（目前主流）：
   1. HP Quick Test Professional Software，是一种企业级自动化测试工具
   2. 提供了强大易用的录制回放功能
   3. 至此B/S与C/S两种架构
3. Robot Framework自动化测试工具：
   1. 用python编写的功能自动化测试框架
   2. 具备良好的可扩展性，支持关键字驱动
   3. 可以同时测试多种类型的客户端或者接口
   4. 可以进行分布式测试执行
4. Watir自动化测试工具：
   1. Web Application Testing in Ruby，是一个Ruby语言库，使用Ruby进行脚本开发
   2. 基于Web模式
5. Selenium自动化测试工具（在web自动化领域应用越来越广泛）：
   1. 用于Web应用程序测试
   2. 支持多平台、多浏览、多语言

## Selenium介绍

1. selemium自动化测试浏览器，主要用于Web应用程序的自动化测试，但也不局限于此，同时支持所有基于web的管理任务自动化
2. Selemium的特点：
   1. 开源，免费
   2. 多浏览器支持：FileFox，Chrome，IE，Opera
   3. 多平台支持：Linux，Windows，Mac
   4. 多语言支持：java，python，C#，Javascript等
   5. 对web页面有良好的支持
   6. API简单，驱动灵活
   7. 支持**分布式**测试用例执行
3. Selenium1.0的三个工具：
   1. SeleniumIDE：
      1. 这是嵌入到FireFox浏览器的一个插件，实现简单的浏览器操作的录制和回放
      2. 测试过程中，发现bug后可以通过录制重现的步骤，以便更容易重现bug
      3. IDE录制的脚本可以转换成多种语言，从而帮助我们快速开发脚本
   2. Selenium Grid：
      1. 这是一种自动化测试的辅助工具
      2. 通过利用现有的计算机基础设施，加快web应用的功能测试
      3. 可以很方便的**同时**在多台机器上和**异构环境**中**并行**运行多个测试用例
      4. 特点有：
         1. 并行执行
         2. 通过一个主机统一控制用例在不同环境、不同浏览器下执行
         3. 灵活地添加、变动测试机
   3. Selenium RC：
      1. 这是selenium家族的核心工具，支持多种不同语言编写自动化测试脚本
      2. 通过selenium RC的服务器作为代理服务器访问应用从而达到测试的目的
      3. Selenium RC分为Client Libraries和selenium Server：
         1. Client Libraries主要用于编写测试脚本，用来控制selenium Server的库
         2. selenium Server负责控制浏览器的行为，分为三个部分：
            1. Selenium Core：本身就是一堆JS函数的集合，被嵌入到浏览器页面中，实现用程序对浏览器进行操作
            2. Launcher：用于启动浏览器，把Selenium Core加载到浏览器页面，同时把浏览器的代理设置为Selenium Server的Http Proxy
            3. Http Proxy：作为selenium Server在浏览器的代理
4. Selenium2.0的更改：
   1. Selenium2.0主要加了WebDriver，也是主要推荐的
   2. WebDriver是seleniumRC的替代品，为了向下兼容才没有抛弃seleniumRC
   3. WebDriver和SeleniumRC的区别：
      1. seleniumRC：
         1. 在浏览器中运行JavaScript应用，使用浏览器内置的JavaScript翻译器来翻译和执行selense命令（selenium命令集合）
      2. WebDriver：
         1. 针对各浏览器开发，通过原生浏览器支持或者浏览器扩展直接控制浏览器。
         2. 取代了嵌入javascript代码的方式，避免JavaScript安全模式导致的限制。
         3. 除了来自浏览器厂商的支持，还利用了操作系统级的调用模拟用户输入

## 前端工具介绍

1. FireBug：
   1. 这是FireFox浏览器下的一套开发类插件
   2. 集成了HTML查看和编辑、JavaScript控制台、网络状况监视器、Cookie
   3. 用它开发Javascript、CSS、HTML比较方便，可以查看页面上的元素，从而根据属性进行定位
2. FirePath：
   1. 这是FireBug插件扩展的一个开发工具
   2. 用来编辑、检查和生成Xpath1.0表达式、CSS3选择器和JQuery选择器
   3. 可以快速通过XPATH和CSS定位页面上的元素
   4. 当通过FireBug的鼠标箭头选择一个页面元素后，FirePath输入框将给出XPath表达式，快速定位
   5. 可以点击”XPath:”按钮切换到CSS定位方式
3. Chrome自带的开发人员工具：
   1. 通过选项-》工具-》开发者工具，或者F12，或者Ctrl+Shift+I打开
4. IE浏览器也加入了开发人员工具：
   1. 通过选项-》工具-》开发人员工具，或者F12打开
   2. 提供了浏览器的兼容模式，向下兼容，便于测试浏览器的兼容性

# WebDriver环境搭建

1. Pycharm装selenium包：
   1. File-》settings-》Project:workplace-》Project Interpreter-》点+号-》搜索selenium-》install package
2. 下载geckodriver.exe：
   1. 下载地址：<https://github.com/mozilla/geckodriver/releases>
   2. 下载解压后将getckodriver.exe复制到Firefox的安装目录下，如（C:\Program Files\Mozilla Firefox），并在环境变量Path中添加路径：C:\Program Files\Mozilla Firefox；
3. 第一个webdriver脚本：

#coding = utf-8

#为防止乱码问题，以及方便添加中文注解，通过以上语句把编码统一成UTF-8

from selenium import webdriver#导入webdriver模块

#获取浏览器对象，启动浏览器

browser = webdriver.Firefox()#Firefox已经在webdriver包里，可以直接用，其他浏览器要先装驱动

#向浏览器发送网址

browser.get("http://www.baidu.com")

#定位到百度的输入框，并通过send\_keys向输入框输入selenium

browser.find\_element\_by\_id("kw").send\_keys("selenium")

#定位到搜索按钮，并引发点击事件

browser.find\_element\_by\_id("su").click()

#退出并关闭窗口的每一个相关的驱动程序

browser.quit()

## 安装浏览器驱动

1. WebDriver支持Firefox、IE、Opera和Chrome，还支持Android和iPhone移动测试
2. 它还包含一个基于HtmlUnit的无界面测试，称为HtmlUnitDriver
3. 各个浏览器驱动下载地址：<http://npm.taobao.org/mirrors/chromedriver/>
4. 安装XXX浏览器驱动，下载XXXDriver\_win32.zip(根据自己系统下载不同的版本驱动)，解压得到chromedriver.exe 放入某个文件夹，并设置PATH环境变量
   1. 为了方便就直接放在FireFox根目录下，反正已经加入环境变量了
5. 安装完成后可以用IE 和chrome 来替换firefox 运行上面的例子。
   1. browser = webdriver.Firefox()替换为：browser = webdriver.Ie()
   2. 或browser = webdriver.Chrome()
   3. 如果程序能调用相应的浏览器运行，说明我们的浏览器驱动安装成功。

# Python WebDriver API

## 浏览器的操作

1. 浏览器最大化：
   1. Driver.maximize\_window()：将浏览器最大化显示
   2. 如果selenium与浏览器版本不兼容，会报错
      1. WebDriverException: Message: Failed to find width field
2. 设置浏览器宽、高：
   1. Driver.set\_window\_size(480,800)
3. 浏览器前进、后退：
   1. 这两个功能很少用到
   2. Driver.forward()
   3. Driver.back()

## 简单对象的定位

1. 对象（元素）的定位和操作时自动化测试的核心部分。
   1. 除了操作元素外，获取元素属性、文本、数量都是需要定位元素的
   2. 可以通过唯一区别于其他元素的属性来定位一个元素。
   3. Webdriver提供了一系列定位元素的方法：
      1. 通过id定位：find\_element\_by\_id()
      2. 通过name定位：find\_element\_by\_name()
      3. 通过classname定位：find\_element\_by\_class\_name()
      4. 通过tagname定位：find\_element\_by\_tag\_name()
      5. 通过link text定位：find\_element\_by\_link\_text()
      6. 通过partial link text定位：find\_element\_by\_partial\_link\_text()
      7. 通过xpath定位：find\_element\_by\_xpath()
      8. 通过css selector定位：find\_element\_by\_css\_selector()
2. Id和name定位：
   1. Id和name是最常用的定位方式，因为大多数元素都有这两个属性，而且对控件的id和name命名时一般也有不同的名字。通过它们可以很容易地找到某个元素
3. Tag name定位：
   1. 最不靠谱的一种，因为每种元素都有标签名
   2. 但如果该标签只有一个，tag name也不失为一种办法
4. Link text和partial link text定位：
   1. 可以取标签中的文字进行定位<a>abc123</a>
   2. Link text必须匹配全部文字abc123，partial link text可以取文字的一部分abc
5. Xpath定位：
   1. Xpath是一种在XML文档中定位元素的语言，而HTML可以看做XML的一种实现
   2. Xpath扩展了id和name定位方式，更灵活，定位能力更强大
   3. 比如有一段html文本：

<html class="a">

<body>

<div class="b-wrap">

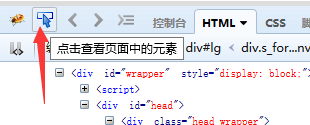
<div id="c" name="d">

<form target="e" action="http://www.f.com">

<span id="g">

<input id="h" type="text">

* 1. 可以用绝对路径方式定位：
     1. find\_element\_by\_xpath(“/html/body/div[2]/form/span/input”)
     2. 缺点是路径可能很长，难以阅读和维护
  2. 可以用相对路径定位：
     1. 举例：
        1. find\_element\_by\_xpath("//input[@id=’h’]") #通过自身的id 属性定位
        2. find\_element\_by\_xpath("//span[@id=’e’]/input") #通过上一级目录的id 属性定位
        3. find\_element\_by\_xpath("//div[@id=’c’]/form/span/input") #通过上三级目录的id 属性定位
        4. find\_element\_by\_xpath("//div[@name=’d’]/form/span/input")#通过上三级目录的name 属性定位
     2. 还可以做布尔逻辑运算，如//div[@id=’c’ or @name=’d’]
     3. 缺点是：
        1. 性能比其他大多数方式差
        2. 不健壮，Xpath会随着页面元素布局的改变而改变
        3. 兼容性不好，不同浏览器Xpath的实现不同
  3. 通过firefox的firebug查看xpath：
     1. 点击左上角鼠标箭头：



* + 1. 点击页面任何一个地方，将进入到所在html的位置
    2. 多想要的元素右键，获取Xpath：



* 1. 通过firepath插件查看xpath：
     1. 选中元素后，将在FirePath的XPath输入框显示xpath：



1. CSS定位：
   1. CSS定位语法比Xpath更加简洁，可以比较灵活选择控件的任意属性，一般情况下定位速度要比XPath快。一般更推荐CSS
   2. CSS常见语法：

|  |  |  |
| --- | --- | --- |
| \* | 通用元素选择器，匹配任何元素 |  |
| E | 标签选择器，匹配所有使用E 标签的元素 | css=div匹配<div> |
| .info | class 选择器，匹配所有class 属性中包含info 的元素 | css=div.formd匹配<div class=”formd”> |
| #footer | id 选择器，匹配所有id 属性等于footer 的元素 | css=ul#record匹配<ul id=”record”> |
| E,F | 多元素选择器，同时匹配所有E 元素或F 元素，E 和F 之间用逗号分隔 |  |
| E F | 后代元素选择器，匹配所有属于E 元素后代的F 元素，E 和F 之间用空格分隔 |  |
| E > F | 子元素选择器，匹配所有E元素的子元素F |  |
| E + F | 毗邻元素选择器，匹配紧随E元素之后的同级元素F（只匹配第一个） |  |
| E ~ F | 同级元素选择器，匹配所有在E元素之后的同级F元素 |  |
| E[att='val'] | 属性att 的值为val 的E 元素（区分大小写） | css=input[name=’ab’]匹配<input name=’ab’ |
| E[att^='val'] | 属性att 的值以val 开头的E 元素（区分大小写） | css=input[name^=’ab’]匹配<input name=’abcd’ |
| E[att$='val'] | 属性att 的值以val 结尾的E 元素（区分大小写） | css=input[name$=’ab’]匹配<input name=’xxab’ |
| E[att\*='val'] | 属性att 的值包含val 的E 元素（区分大小写） | css=input[name$=’ab’]匹配<input name=’ccabc’ |
| E[att1='v1'][att2\*='v2'] | 属性att1 的值为v1，att2 的值包含v2 （区分大小写） | Css=input[name=’ab’][id=’cd]匹配<input id=’cd’ name=’ab’> |
| E:contains('xxxx') | 内容中包含xxxx 的E 元素 | css=li:contains(“gg”)匹配<li>gg</li> |
| E:not(s) | 匹配不符合当前选择器的任何元素 | Css=li:not(contains(“gg”)匹配<li>ab</li> |

c) 结构性定位：根据元素父子、同级位置来定位：

i.注意，文本节点也被看作子元素

|  |  |
| --- | --- |
| E:nth(n)  E:eq(n) | 在其父元素中的E 子元素集合中排在第n+1 个的E 元素(第一个n=0) |
| E:first | 在其父元素中的E 子元素集合中排在第1个的E 元素 |
| E:last | 在其父元素中的E 子元素集合中排在最后1个的E 元素 |
| E:even | 在其父元素中的E 子元素集合中排在偶数位的E 元素(0,2,4…) |
| E:odd | 在其父元素中的E 子元素集合中排在奇数的E 元素(1,3,5…) |
| E:lt(n) | 在其父元素中的E 子元素集合中排在n 位之前的E 元素(n=2,则匹配0,1) |
| E:gt(n) | 在其父元素中的E 子元素集合中排在n 位之后的E 元素(n=2，在匹配3,4) |
| E:only-child | 父元素的唯一一个子元素且标签为E |
| E:empty | 不包含任何子元素的E 元素 |

## 操作测试对象

1. 通过元素定位后，可以通过WebElement接口:
   1. 对返回的元素对象进行操作
   2. 获取返回的元素对象的数据属性
2. 操作：
   1. clear()：
      1. 清除元素的内容（如果允许的话）
   2. send\_keys(“str”)：
      1. 在元素上模拟键盘输入
   3. click()：
      1. 鼠标单击元素
   4. submit()：
      1. 提交表单
      2. 有时候可以和click()相互替代使用，但：
         1. submit()强调对信息的提交
         2. click()强调事件的独立性
   5. 比如driver.find\_element\_by\_id("kw").send\_keys("selenium")
3. WebElement接口常用方法：
   1. size：返回元素的尺寸，为高和宽的字典
      1. 比如size=driver.find\_element\_by\_id("kw").size
   2. text：返回元素的文本内容（即一对标签之间的文字）
      1. 顺序递归地通过空格拼接自身和所有子元素的文本内容
   3. get\_attribute(attStr)：返回元素的属性的值
      1. attStr为属性对应的字符串，比如’id’，’name’，’class’等属性
   4. is\_displayed()：布尔判断该元素是否可见

## 鼠标事件

1. 有关鼠标的操作，都包含在ActionChains类中
2. 先导入ActionChains模块：
   1. from selenium.webdriver.common.action\_chains import ActionChains
3. 示范执行右击鼠标操作：
   1. ActionChains(driver).context\_click(right).perform()：
      1. ActionChains用于生成用户的行为
      2. driver是webdriver实例
      3. context\_click(driver.find\_element\_by\_id("kw"))，参数为元素对象
      4. 所有的行为都将存储在actionchains对象中
      5. 最后通过perfrom()执行所有存储对象
4. 鼠标双击：
   1. double\_click(element):鼠标左键双击
   2. 对web应用来说，双击用得比较少
5. 鼠标拖放操作：
   1. drag\_and\_drop(source,target)：在源元素上按下左键，然后移动到目标元素释放
      1. Source：鼠标按下的源元素
      2. Target：鼠标释放的目标元素
6. 鼠标移动到某个元素上：
   1. move\_to\_element(element)：把鼠标移到某个元素上，但不点击
7. 鼠标按住不放：
   1. click\_and\_hold(element)：鼠标左键按住某个元素但不放

## 键盘事件

1. 使用键盘事件需要先导入keys类包：

from selenium.webdriver.common.keys import Keys

1. 经常用到的键盘操作：
   1. send\_keys(Keys.CONTROL,'x') 剪切（Ctrl+X）
   2. send\_keys(Keys.CONTROL,'v') 粘贴（Ctrl+V）
   3. send\_keys(Keys.BACK\_SPACE) 删除键（BackSpace）
   4. send\_keys(Keys.SPACE) 空格键(Space)
   5. send\_keys(Keys.TAB) 制表键(Tab)
   6. send\_keys(Keys.ESCAPE) 回退键（Esc）
   7. send\_keys(Keys.ENTER) 回车键（Enter）
   8. send\_keys(Keys.CONTROL,'a') 全选（Ctrl+A）
   9. send\_keys(Keys.CONTROL,'c') 复制（Ctrl+C）

## 打印信息

1. 在设计功能测试用例时，有些情况需要获得更多信息来证明用例执行结果是成功的
2. 通常通过页面的title、URL地址、标志性信息（比如登录成功的“欢迎”）来判断用例执行成功
3. 通过driver.title获取title信息，类型为字符串
4. 通过driver.current\_url获取url信息，类型为字符串

## 设置等待时间

1. 为了保证脚本运行的稳定，或者能够看到结果，需要在脚本中添加等待时间
2. time.sleep(second):
   1. 设置固定的休眠时间
   2. 需要导入time包，import time
   3. 或者直接导入sleep方法，from time import sleep
   4. 参数为秒，也可以为小数，比如time.sleep(0.1)
3. implicitly\_wait(second)：
   1. 超时等待。等待命令被完成或元素被发现，如果超时则抛出异常
   2. 需要导入webdriver，from selenium import webdriver
4. WebDriverWait(driver,timeout,poll\_frequency=0.5,ignored\_exceptions=None)：
   1. 在设置时间内，默认每隔一段时间检测一次当前页面元素是否存在，如果超时检测不到则抛出异常。如果检测到存在则返回该页面元素
   2. 需要导入包：
      1. from selenium.webdriver.support.ui import WebDriverWait
   3. driver为WebDriver的驱动程序
   4. timeout为最长超时时间，单位为秒
   5. poll\_frequency为休眠时间的间隔（步长），默认为0.5秒
   6. ignored\_exceptions为超时后的异常信息，默认抛出NoSuchElementException
   7. 一般与until(method,message=””)和util\_not(method,message=””)配合使用：
      1. WebDriverWait().until()
      2. until(method,message=””)：设置时间内一直调用method，直到为True。
      3. until\_not(method,message=””)：设置时间内一直调用method，直到为False。

## 定位一组对象

1. 通常用于以下场景：
   1. 批量操作对象，比如将checkbox全部打上勾
   2. 先获取一组对象，然后再在这组对象中过滤出想要的
2. 区别于获取单个对象find\_element\_by\_xx(‘xx’)，获取一组对象为：
   1. find\_elements\_by\_xx(‘xx’)

## 层级定位

1. 先定位父元素，再定位子孙元素
2. 方法就是连用find:
   1. driver.find\_element\_by\_xx('xx').find\_element\_by\_xx('xx')

## 定位frame中的对象

1. 在web应用中经常会出现frame嵌套的应用，比如Aframe里面还有一个Bframe
2. 而webdriver每次只能在一个页面识别元素，因此需要跳到frame中
3. driver.switch\_to\_frame(id/name)：
   1. 将当前页面切换到frame

## 对话框处理

1. 现在很多前端框架的对话框是div形式的
2. 通过二次的定位：
   1. 第一次定位找到弹出的对话框
   2. 第二次定位对话框中的元素

## 浏览器多窗口处理

1. 如果要在多个窗口间切换，需要知道每个窗口的唯一标识符（句柄）
2. 方法：
   1. driver.current\_window\_handle：获取当前窗口句柄
   2. driver.window\_handle：获取所有窗口句柄
   3. driver.switch\_to\_window(handle)：切换到对应窗口
   4. driver.close()：关闭当前窗口

## alert/confirm/prompt处理

1. 对javascript产生的alert/confirm/prompt弹窗：
   1. 通过alert = driver.switch\_to\_alert()可以定位到弹窗
   2. text()：返回弹窗文字信息
   3. accept()：点击确认按钮
   4. dismiss()：点击取消按钮（如果有的话）
   5. send\_keys(smessage)：输入值（如果没有却用了这个函数，会报错）

## 下拉框处理

1. 下拉框由<select>构成，因此第一次先定位到select
2. 下拉框的元素有<option>构成，因此从select再定位到option，进行click()操作

## 上传文件处理

1. 可以通过send\_keys()上传本地文件的路径，从而达到上传文件的目的

## 下载文件

1. 可以通过requests模块查找内容类型。requests模块是一个Python的HTTP客户端，不再python环境包里，需要自己下载
2. 通过requests.head(‘http://www.xxx.xxx’).headers[‘content-type’]获取文件类型
3. 设置Firefox的默认配置文件：
   1. 先获取配置文件对象：fp = webdriver.FirefoxProfile()
   2. 设置文件对象属性：fp.setpreference(“key”,value)
   3. 在创建驱动的时候传入配置文件对象：driver = webdriver.Firefox(firefox\_profile = fp)
4. 下载文件本质上也是对页面对象的click()

## 调用JavaScript

1. 如果有些操作没法用webDriver完成，可以借用JavaScript完成
2. 可以以两种场景执行JS：
   1. 在页面上直接执行JS
   2. 在某个已经定位的元素上执行JS
3. execute\_script(script, \*args=None)：
   1. 在当前window/frame同步执行javaScript
   2. 其中script为JS代码
   3. args为js操作的对象，可以在script中的arguments[x]对应参数的索引，从0开始

## 控制浏览器滚动条

1. 滚动条并非页面上的元素，可以借助JavaScript完成
2. 操作滚动条的两个场景：
   1. 需要滚动条拉到最下方以判断用户完成协议的阅读
   2. 要操作的页面元素不再视觉范围内，无法进行操作，需要拉动滚动条
3. 通过控制scrollTop完成操作：
   1. driver.execute\_script(‘var q=document.documentElement.scrollTop=10000’)
   2. 0为顶部，越大则越底部，通常10000为底部

## Cookie处理

1. 基于真实cookie的测试是无法通过白盒和集成测试完成的
2. WebDriver可以读取、添加、删除cookie信息
3. WebDriver操作cookie的方法：
   1. get\_cookies()：获得所有cookie信息
   2. get\_cookie(name)：获得特定name的cookie信息
   3. add\_cookie(cookie\_dict)：添加cookie键值对
      1. driver.add\_cookie({'name':'xxx', 'value':'yyy'})
   4. delete\_cookie(name)：删除特定name的cookie信息

## 获取对象的属性

1. get\_attribute(att)：
   1. 对一个element对象使用，即driver.find\_element\_by\_xx(“xx”).get\_attribute(att)

## 验证码问题

1. 验证码现在有数字的，有汉字的，有题目的，还有拉动拼图的，还有图片选择的
2. 处理验证码的几种方法：
   1. 去掉验证码：
      1. 如果还处在开发阶段，先把验证码代码注释掉即可
   2. 设置万能码：
      1. 开后门，只要输入这个验证码就视为通过
   3. 验证识别技术：
      1. 比如Python-tesseract，能识别常规图片
      2. 但识别率基本不能到100%，准确率不能保证
   4. 记录cookie：
      1. 向浏览器添加cookie可以绕过登录的验证码
      2. 在用户登录前，通过add\_cookie()方法将用户名和密码写入浏览器cookie，再次登录时将自动登录
      3. 通过get\_cookies()找到登录的所有cookie信息，从而找到用户名和密码的name对象名字，也就是key

## Webdriver原理

1. WebDriver启动目标浏览器，并绑定到指定端口。该浏览器实例作为webdriver的remote server。
2. Client端通过CommandExcuter发送HTTPRequest给remote server监听端口
3. Remote Server需要依赖原生的浏览器组件，来转化浏览器的native调用