

春风阁讲堂授课专属

Selenium 与 Web 自动化测试实战

文件建立/修改记录

序号	版本	建立或修改	建立/修改人 日期	审核人 日期	批准人 日期
1	1.0	建立	龙哥 2022-07-05		

目录

27. Selenium 整体介绍	13
27.1. 自动化测试&场景	13
27.1.1. 何谓自动化测试.....	13
27.1.2. 自动化适用场景.....	14
27.2. 自动化测试知识点	16
27.3. WEB 自动化相关技能	17
27.4. SELENIUM 主要介绍	18
27.5. SELENIUM 三剑客	19
27.5.1. WebDriver	19
27.5.2. Selenium-ide.....	19
27.5.3. Selenium-grid.....	19
27.6. 支持语言列表	20
28. Selenium 环境搭建	21
28.1. PYTHON 环境安装	21
28.2. PYTHON 开发环境	22
28.2.1. VScode+Python 插件.....	22
28.2.2. PyCharm 编辑器.....	25
28.2.3. Jupyter-notebook.....	26
28.3. SELENIUM 环境安装	27
28.3.1. selenium 基础安装.....	27
28.3.2. WebDriver 安装.....	27
28.3.3. Selenium-ide 安装.....	28
28.3.4. grid 环境安装.....	29

28.4. 环境测试入门实例30

29. Selenium 元素定位 31

29.1. HTML&元素定位 31

29.2. 元素定位-ID 方式 32

29.3. 元素定位-NAME 方式 33

29.4. 元素定位-CLASS NAME 34

29.5. 元素定位-TAG NAME 35

29.6. 元素定位-CSS SELECTOR 36

29.6.1. *css selector* 分类.....36

29.6.2. *简单选择器&演示*..... 37

29.6.3. *组合器选择器&演示*..... 39

29.6.4. *伪类选择器&演示*..... 41

29.6.5. *属性选择器&演示*..... 44

29.6.6. *伪元素选择器&演示*..... 46

29.7. 元素定位-LINK TEXT 47

29.8. 元素定位-PARTIAL LINK TEXT 48

29.9. 元素定位-XPATH 定位 49

29.9.1. *路径表达式*..... 49

29.9.2. *谓词/条件表达式*..... 50

29.9.3. *通配符匹配*..... 51

29.9.4. *xpath 常用实例*..... 52

29.9.5. *xpath 高级实例*..... 53

29.9.6. *选择器&演示*..... 54

29.10. 传统定位方式总结 55

29.11. CSS&XPATH 定位总结 56

29.12. SELENIUM 相对定位57

29.12.1. 相对定位简介 57

29.12.2. 相对定位样例 1 58

29.12.3. 相对定位样例 2 59

29.13. 万能 JAVASCRIPT 定位60

29.13.1. js 定位分类60

29.13.2. javascript 样例 1 61

29.13.3. javascript 样例 2 62

29.13.4. javascript 样例 3 63

29.14. 万能代码选择器定位64

29.14.1. 代码定位简介 64

29.14.2. 代码定位样例 64

29.15. 多种定位方式演示65

30. Selenium 元素交互 66

30.1. 交互类型汇总66

30.2. 操作 CHEXKBOX67

30.3. 操作 RADIOBOX68

30.4. 操作 FORM 表单 69

30.5. 操作下拉列表70

30.5.1. 下拉列表演示 70

30.5.2. 常用用法 71

30.6. 读取元素信息72

30.6.1. 常用属性&函数 72

30.6.2. property&attribute 73

30.6.3. 演示代码样例 74

31. Selenium 事件交互 75

31.1. 键盘事件75

31.1.1. 常用 Api..... 75

31.1.2. 案例源码..... 76

31.1.3. 常用按键..... 77

31.2. 鼠标事件78

31.2.1. 鼠标按键..... 78

31.2.2. 常用 Api..... 79

31.2.3. 演示代码样例 1..... 81

31.2.4. 演示代码样例 2..... 82

31.2.5. 演示代码样例 3..... 83

31.3. 滚动事件84

31.3.1. 常用 Api..... 84

31.3.2. 演示代码样例 1..... 85

31.3.3. 演示代码样例 2..... 86

31.3.4. 演示代码样例 3..... 87

31.3.5. 演示代码样例 4..... 88

31.3.6. 演示代码样例 5..... 89

32. Selenium 窗口操作 90

32.1. 浏览器操作90

32.1.1. 基础动作..... 90

32.1.2. 演示样例 1..... 91

32.2. IFRAME 操作 92

32.2.1. 常用 Api 操作..... 92

32.2.2. iframe 演示样例..... 93

32.3. 窗口-打开与关闭 94

32.3.1. 常用 API..... 94

32.3.2. 演示样例 1 95

32.3.3. 演示样例 2..... 96

32.3.4. 窗口顺序..... 97

32.4. 窗口-窗口管理 98

32.4.1. 常用 API..... 98

32.4.2. 演示样例 1 99

32.5. 窗口-截图操作 101

32.5.1. 常用 API..... 101

32.5.2. 全屏截图演示..... 102

32.5.3. 元素截图演示..... 103

33. Selenium 其它操作 104

33.1. 对话框-ALERT 104

33.1.1. 常用 Api..... 104

33.1.2. alert 演示样例 1 105

33.2. 对话框-CONFIRM 107

33.2.1. 常用 Api..... 107

33.2.2. 演示样例 1 108

33.3. 对话框-PROMPT 109

33.3.1. 常用 Api..... 109

33.3.2. 演示样例 1 110

33.4. COOKIE 操作 111

33.4.1. 常用 Api 操作..... 111

33.4.2. Cookie 样例 1 112

33.4.3. Cookie 样例 2.....	113
33.5. 文件上传	114
33.5.1. 文件上传演示.....	114
33.6. 异步等待方式	115
33.6.1. 等待方式.....	115
33.6.2. time.sleep.....	116
33.6.3. implicitly_wait()	116
33.6.4. WebDriverWait.....	116
34. Selenium 之 JS 应用	117
34.1. JS 使用场景	117
34.2. JS 同步执行	118
34.3. JS 异步等待	119
34.4. JS 之化繁为简	120
34.5. JS 实现 H5 拖曳	121
34.6. JS 绕过验证码	122
34.6.1. 处理思路.....	122
34.6.2. sessionStorage.....	123
34.6.3. localStorage.....	123
34.6.4. Cookie.....	123
34.7. JS 深入思考	124
35. 与 unittest 集成.....	125
35.1. UT 集成概述	125
35.1.1. unittest 简介.....	125
35.1.2. 什么要集成.....	126
35.2. 用 UT 设计流程.....	127

35.3. 用 UT 分组用例128

35.4. DDT 数据驱动 129

35.5. UT 收集用例结果 130

36. 与 pytest 集成131

36.1. PYT 集成概述 131

36.2. PYT 固件设计流程 132

36.3. FIXTURE 设计流程 133

36.4. PYT 分组用例 134

36.5. PYT 参数化设计 135

36.6. PYT 收集用例结果 136

36.7. 与 JENKINS 集成137

36.7.1. 软件安装..... 137

36.7.2. 初始化..... 142

36.7.3. 中文插件..... 145

36.7.4. allure 插件..... 146

36.7.5. 环境变量..... 147

36.7.6. allure 目录..... 149

36.7.7. 新建任务..... 151

36.7.8. pytest 构建命令..... 153

36.7.9. allure 报告..... 154

36.7.10. 查看报告..... 155

37. pom 设计分层156

37.1. POM 模式简介 156

37.1.1. 什么是 pom 模式..... 156

37.1.2. pom 模式特征..... 157

37.1.3. pom 模式优势.....	158
37.2. POM 模式过程.....	159
37.2.1. 封装基础方法.....	159
37.2.2. 封装页面元素.....	160
37.2.3. 设计测试用例.....	161
37.2.4. 用例场景分组.....	161
37.3. 百度演示案例.....	162
37.4. POM 模式架构.....	163
38. 关键字驱动设计.....	164
38.1. 关键字驱动简介.....	164
38.1.1. 什么是关键字驱动.....	164
38.1.2. 关键字&数据驱动.....	164
38.2. 常用设计方法.....	165
38.2.1. 用例步骤拆分.....	165
38.2.2. 对象提取分离.....	166
38.3. 百度演示案例.....	167
38.4. 深入三种模式.....	168
39. 开发平台综合案例.....	169
39.1. 需求场景说明.....	169
39.2. 测试用例设计.....	170
39.3. 基础分层架构.....	171
39.4. 数据驱动思考.....	171
39.5. 关键字驱动.....	171
39.6. 完整测试项目.....	171

40. 股票项目综合案例 172

40.1. 需求场景说明 172

40.2. 网络爬虫简介 173

40.3. 完整测试项目 174

41. Selenium-ide 175

41.1. 工具简介 175

41.2. 环境安装 175

41.3. 入门样例 176

41.4. 流程样例 177

41.5. 控制台运行 178

42. 分布式 Grid 应用 179

42.1. GRID 简介 179

42.2. 环境搭建 180

42.3. 独立模式 181

42.4. HUB+NODE 182

42.5. DISTRIBUTED 183

42.5.1. hub 内部 183

42.5.2. 环境启动 184

42.6. 远程测试样例 185

42.6.1. 环境安装 185

42.6.2. 案例源码 186

42.7. 个性化样例 187

42.7.1. 节点配置 187

42.7.2. 案例源码 188

43. selenium 最佳实践	189
-------------------------	-----

27. Selenium 整体介绍

27.1. 自动化测试&场景

27.1.1. 何谓自动化测试

普通测试时，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过程一步步执行测试，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，便引入了自动化测试的概念；自动化测试是把以人为驱动测试行为转化为机器执行的一种过程

说直白一点，自动化测试就是以程序测试程序，自动执行指定动作

比如后面我们要学到的接口自动化、UI 自动化、性能自动化，这些都属于典型的自动化测试内容

27.1.2. 自动化适用场景

并非所有的项目均适用自动化，一般采用自动化测试的项目，具有以下几个特征：

1. 需求变动不频繁

测试脚本的稳定性决定了自动化测试的维护成本。脚本的维护本身就是一个代码开发的过程，需要修改、调试；如果软件需求变动过于频繁，导致所花费的成本不低于利用其节省的测试成本，那么自动化测试便是失败的

项目中的某些模块相对稳定，而某些模块需求变动性很大。我们便可对相对稳定的模块进行自动化测试，而变动较大的仍是用手工测试

2. 项目周期足够长

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均需要相当长的时间来完成，这样的过程本身就是一个测试软件的开发过程，需要较长的时间来完成。如果项目的周期比较短，那么没有必要也不值得采用自动化测试

3. 项目初期不适合自动化

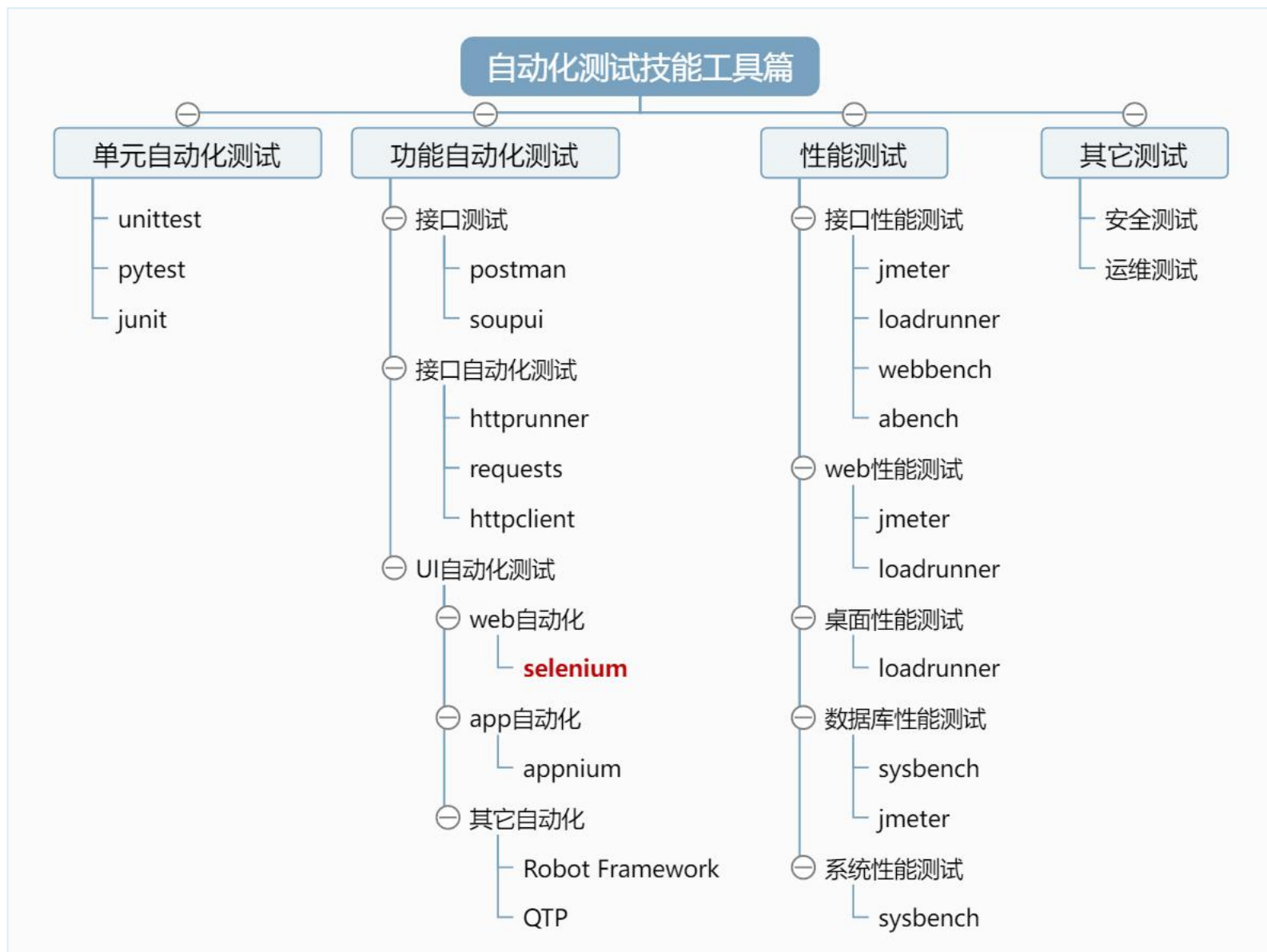
项目建设初期，一般不确定的因素比较多，需求变动也比较大

4. 自动化测试脚本可重复使用

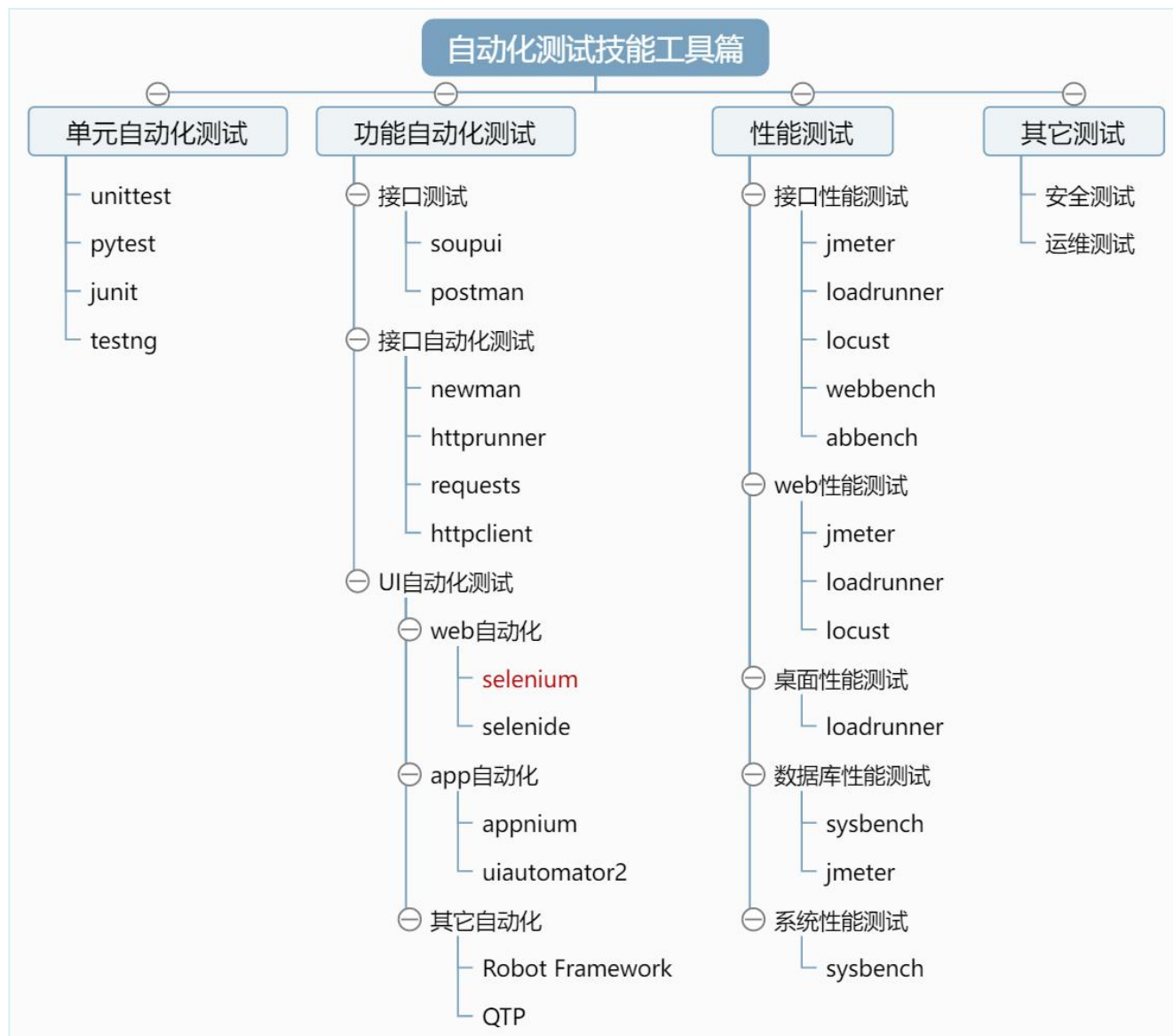
如果费尽心思开发了一套近乎完美的自动化测试脚本，但是脚本的重复使用率很低，致使其间所耗费的成本大于所创造的经济价值，自动化测试价值就无法体现，所以要求测试脚本人员在开发自动化框架和自动化脚本时需要注意到兼容性和可扩展性

自动化测试脚本的重复利用率，是衡量自动化测试成功与否的重要标准

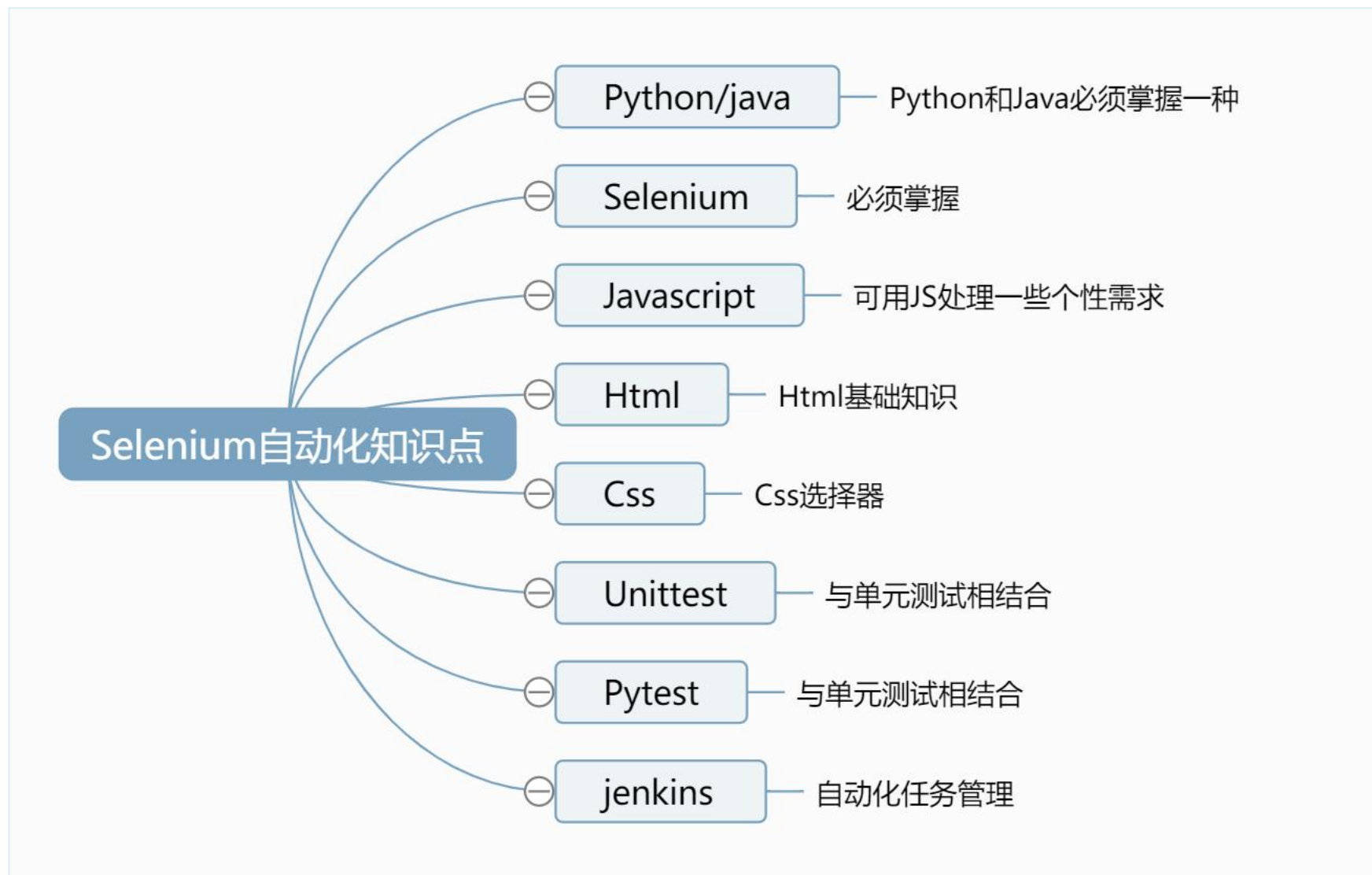
总而言之，我们要看自动化创造的价值是否大于写自动化脚本的工作量



27.2. 自动化测试知识点



27.3. web 自动化相关技能



27.4. Selenium 主要介绍

Selenium 是一个用于 Web 应用程序测试的工具。是一个开源的 Web 的自动化测试工具，最初是为网站自动化测试而开发的，可以按指定的命令自动操作，不同是 Selenium 可以直接运行在浏览器上，它支持所有主流的浏览器，支持的浏览器包括 Edge, Mozilla Firefox, Safari, Google Chrome, Opera 等

以下是 Selenium 的几个特点：

- 1、免费开源
- 2、跨平台支持：linux 、 windows 、 mac
- 3、支浏览器：Firefox、Safari、Opera、Chrome、Edge 等
- 4、核心功能：就是可以在多个浏览器上进行自动化测试
- 5、可以搭配多种客户端语言使用：Python、Java、C#、JavaScript、Ruby 等
- 6、成熟稳定：目前已经被 google ，百度， 腾讯等许多大公司广泛使用
- 7、支持分布式测试用例的执行，可以把测试用例分布到不同的测试机器的执行，相当于分发机的功能。

Selenium 的主要功能场景包括：

- 1、基于浏览器界面的功能自动化测试
- 2、测试与浏览器的兼容性——测试你的应用程序看是否能够很好得工作在不同浏览器和操作系统之上
- 3、其它作用，诸如爬虫行为

本套课程深入讲解 Selenium 的运行原理和实战技巧，结合 unittest、pytest，熟悉 ddt、pom、关键字等相关测试方法

27.5. Selenium 三剑客

27.5.1. WebDriver

内部实现了对浏览器的各种操作，对外提供了多语言的 API，如果是学习 Selenium，我们 90%的时间应该在和 WebDriver 打交道

27.5.2. Selenium-ide











本质是一个浏览器的插件，可以进行自动化脚本的录制与回放，辅助我们生成一些脚本

27.5.3. Selenium-grid

提供了分布式执行环境，用例同时在多个浏览器同时执行，提搞测试效率

27.6. 支持语言列表

序号	支持语言	说明
1	Python	使用非常多(建议)
2	Java	使用较多(建议)
3	JavaScript	一般
4	Ruby	较少
5	CSharp	较少

Jul 2022	Jul 2021	Change	Programming Language		Ratings	Change
1	3	▲		Python	13.44%	+2.48%
2	1	▼		C	13.13%	+1.50%
3	2	▼		Java	11.59%	+0.40%
4	4			C++	10.00%	+1.98%
5	5			C#	5.65%	+0.82%
6	6			Visual Basic	4.97%	+0.47%
7	7			JavaScript	1.78%	-0.93%
8	9	▲		Assembly language	1.65%	-0.76%
9	10	▲		SQL	1.64%	+0.11%
10	16	▲		Swift	1.27%	+0.20%

28. Selenium 环境搭建

28.1. Python 环境安装

演示版本:

Python==3.10.5

下载地址:

<https://www.python.org/ftp/python/3.10.5/python-3.10.5-amd64.exe>

如果下载不下来, 换一个网络环境, 比如采用手机热点

环境检测:

```
C:\Users\xiang>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

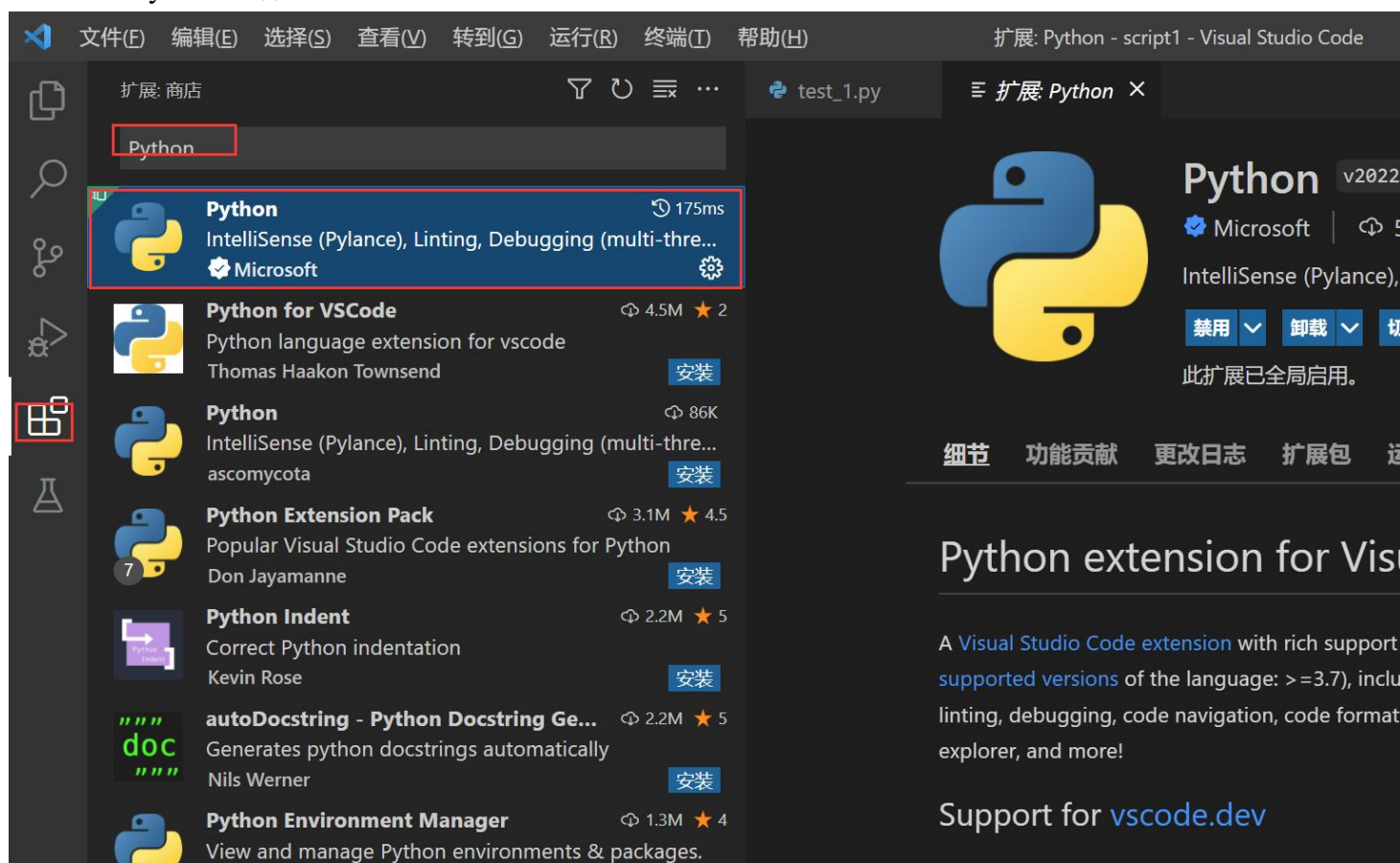
28.2. Python 开发环境

28.2.1. VScode+Python 插件

1. 进官网，下载最新 VScode 版本

<https://code.visualstudio.com/>，目前最新是 1.67.2

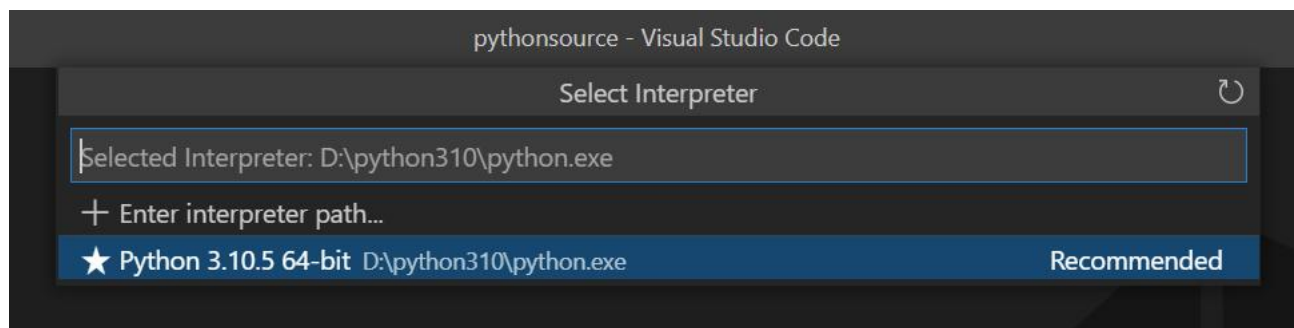
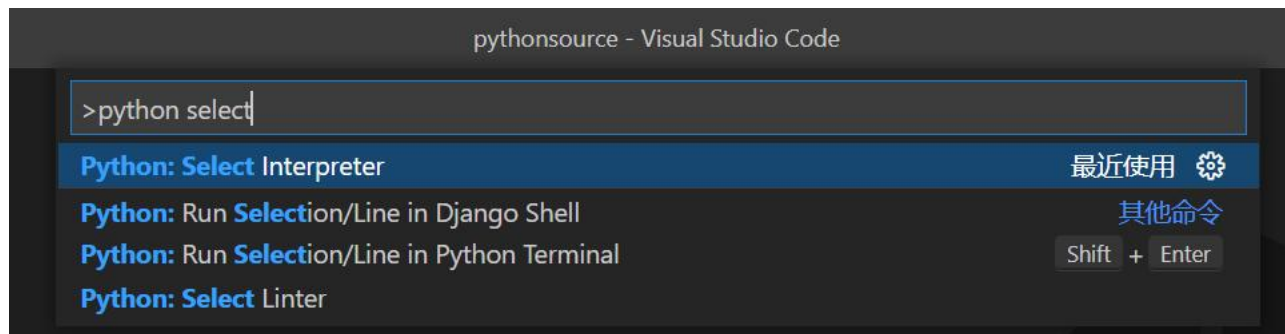
2. 安装 Python 插件



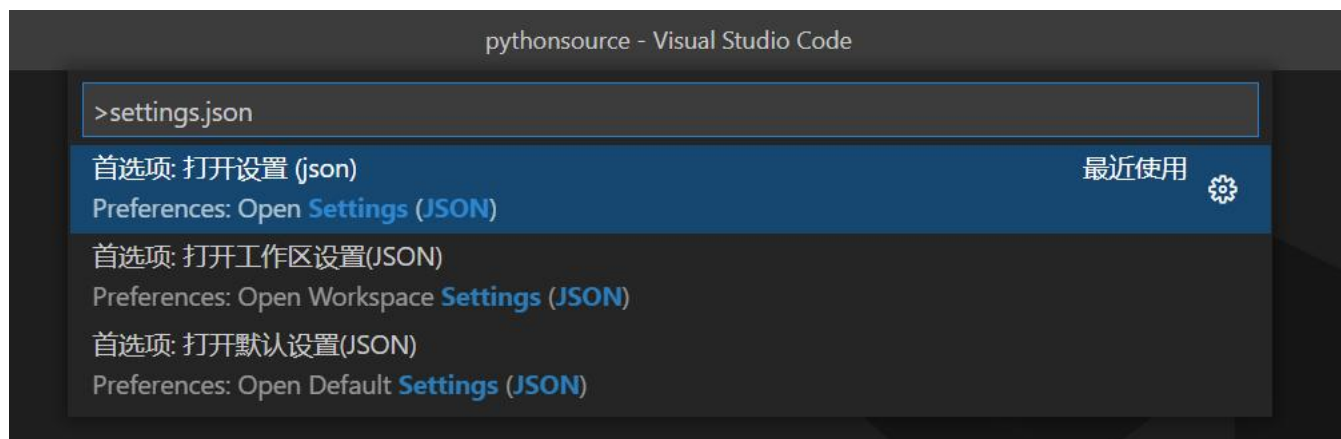
插件配置参考下一页

3. 选择 python 环境

三键组合，ctrl+shift+p，然后输入 python select，选择一个 python 环境



4. 三键组合，ctrl+shift+p，然后输入 settings.json



settings.json 中的值设置如下:

```
{
  "workbench.colorTheme": "Default Dark+",
  "editor.fontSize": 18,
  "editor.formatOnPaste": true,
  "editor.formatOnSave": true,
  "window.zoomLevel": 0,
  "python.analysis.indexing": true,
  "python.languageServer": "Pylance",
  "python.analysis.extraPaths": [
    "C:\\Users\\xiang",
    "D:\\python310\\python310.zip",
    "D:\\python310\\DLLs",
    "D:\\python310\\lib",
    "D:\\python310",
    "D:\\python310\\lib\\site-packages",
  ]
}
```

python.analysis.extraPaths 的值采用如下命令查询:

```
C:\Users\xiang>python -m site

sys.path = [
  'C:\\Users\\xiang',
  'D:\\python310\\python310.zip',
  'D:\\python310\\DLLs',
  'D:\\python310\\lib',
  'D:\\python310',
  'D:\\python310\\lib\\site-packages',
]
```


28.2.2. PyCharm 编辑器

1. 进入官网下载社区版:

<https://www.jetbrains.com/pycharm/download/#section=windows>

28.2.3. Jupyter-notebook

1. `pip install jupyter`

运行时间会比较长，如果失败，将 `pip` 升级到最新版再试试

2. 安装好后，在 `cmd` 运行如下命令即可启动

`jupyter notebook`

28.3. Selenium 环境安装

28.3.1. selenium 基础安装

命令行模式下，执行：

```
pip install selenium==4.4.0
```

28.3.2. WebDriver 安装

https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers，选择相应浏览器，下载最接近的 webDriver

Quick Reference				
Browser	Supported OS	Maintained by	Download	Issue Tracker
Chromium/Chrome	Windows/macOS/Linux	Google	Downloads	Issues
Firefox	Windows/macOS/Linux	Mozilla	Downloads	Issues
Edge	Windows/macOS	Microsoft	Downloads	Issues
Internet Explorer	Windows	Selenium Project	Downloads	Issues
Safari	macOS High Sierra and newer	Apple	Built in	Issues

下载完之后，我们将驱动放在\$PATH 目录下面，或者直接放在 Python 目录下面，如：D:/Program Files/Python310

IE 也是支持的， 不过现在不建议大家学习，毕竟 IE 已经退出历史舞台

28.3.3. Seleniuim-ide 安装

进入下载页 <https://www.selenium.dev/selenium-ide/>下载

目前支持谷歌和火狐浏览器

放在后面章节介绍

28.3.4. grid 环境安装

放在后面章节介绍

28.4. 环境测试入门实例

通过一个最简单的入门实例，完成对环境的测试，以最快的方式了解 Selenium

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
driver.get("https://www.baidu.com")
# 定位元素并发送文本内容
driver.find_element(By.ID, "kw").send_keys("test")
sleep(2)
# 页面按钮点击交互
driver.find_element(By.ID, "su").click()
sleep(2)
driver.quit()
```

Selenium 是模拟人在浏览器操作的过程，我们很容易联想到它的主要功能：

- 1、定位元素(人用大脑和眼睛发现，而 Selenium 则按照一定的定位策略发现)
- 2、元素与事件交互(比如说点击按钮、输入文字、下拉框选择)
- 3、我们还可以通过 Selenium 执行 JS
- 4、浏览器动作(前进、后退……)
- 5、其它功能(在浏览器页面发生的所有，Selenium 均能模拟执行)

29. Selenium 元素定位

29.1. HTML&元素定位

1. HTML 简介

超文本标记语言（英语：HyperText Markup Language，简称：HTML）

html 是一种语言，也是一种通用协议，目前最新版为 html5，而我们的浏览器大多数支持

参考学习网址：<https://www.runoob.com/html/html-tutorial.html>

2. HTML 元素

html 元素以开始标签起始

html 元素以结束标签终止

元素的内容是开始标签与结束标签之间的内容

某些 html 元素具有空内容（empty content）

空元素在开始标签中进行关闭（以开始标签的结束而结束）

3. 大多数 html 元素有属性

4. 大多数 html 元素可以嵌套

5. 浏览器中如查看 HTML 源码

6. 识别常见的 html 元素

Selenium 要求熟悉一般常见的 html 元素

比如像：input(输入)、button(按钮)、select(下拉列表)、a(链接)等等

元素定位就是通过 html 元素不同的特征(比如 id,name,attr 等)去查找元素的过程

29.2. 元素定位-id 方式

1. 参考样例

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.remote.webdriver import WebElement
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 定位元素
# kw = driver.find_element_by_id("kw") #该方式已经废弃
kw: WebElement = driver.find_element(By.ID, 'kw')
sleep(2)
kw.send_keys("演示 id 方式元素定位")
# 打印元素属性
print("标签名称:", kw.tag_name, ",class 类名:", kw.get_dom_attribute('class'))
sleep(5)
driver.quit()
```

2. 适用场景

元素必须有 id 属性，精确定位，效率高

29.3. 元素定位-name 方式

1. 参考样例

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 定位元素
# wd = driver.find_element_by_name("wd") #该方式已经废弃
wd = driver.find_element(By.NAME, 'wd')
sleep(2)
wd.send_keys("演示 name 方式元素定位")
# 打印元素属性
print("标签名称:", wd.tag_name, ",class 类名:", wd.get_dom_attribute('class'))
sleep(5)
driver.quit()
```

2. 适用场景

元素必须有 name 属性，精确定位，效率高

29.4. 元素定位-class name

1. 参考样例

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 定位元素
# s_ipt = driver.find_element_by_class_name("s_ipt") #该方式已经废弃
s_ipt = driver.find_element(By.CLASS_NAME, 's_ipt')
sleep(2)
s_ipt.send_keys("演示 class name 方式元素定位")
# 打印元素属性
print("标签名称:", s_ipt.tag_name, ",class 类名:", s_ipt.get_dom_attribute('class'))
sleep(10)
driver.quit()
```

2. 适用场景

元素必须有 class name 属性，泛型定位，一般要与其它定位策略相结合

29.5. 元素定位-tag name

1. 参考样例

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 定位元素
#inputs = driver.find_elements_by_tag_name("input") #该方式已经废弃
#find_elements: 匹配多个, find_element: 匹配第一个
inputs = driver.find_elements(By.TAG_NAME, "input")
sleep(2)
for input in inputs:
    print(input.get_dom_attribute('id'))
    if(input.get_dom_attribute('id') == "kw"):
        input.send_keys("演示 tag name 方式元素定位")
sleep(10)
driver.quit()
```

2. 适用场景

所有元素均可以定位，泛型定位，一般要与其它定位策略相结合

29.6. 元素定位-css selector

29.6.1. css selector 分类

我们通常把 css 选择器分为五类：

序号	选择器分类	场景说明
1	简单选择器	根据名称、id、类来选取元素
2	组合器选择器	根据它们之间的特定关系来选取元素
3	伪类选择器	根据特定状态选取元素
4	属性选择器	根据属性或属性值来选取元素
5	伪元素选择器	选取元素的一部分并设置其样式(在自动化里面一般用不上)

29.6.2. 简单选择器&演示

选择器	参考样例	样例描述
.class(类选择器)	<code>driver.find_element(By.CSS_SELECTOR, '.class1')</code>	选取所有 <code>class="intro"</code> 的元素。
#id(ID 选择器)	<code>driver.find_element(By.CSS_SELECTOR, '#id1')</code>	选取 <code>id="firstname"</code> 的那个元素。
(通用选择器)	<code>driver.find_element(By.CSS_SELECTOR, '')</code>	选取所有元素。
element(元素选择器)	<code>driver.find_element(By.CSS_SELECTOR, 'p')</code>	选取所有 <code><p></code> 元素。
element,element,..(多元素选择器)	<code>driver.find_element(By.CSS_SELECTOR, 'div,p')</code>	选取所有 <code><div></code> 元素和所有 <code><p></code> 元素。

代码请下翻一页哦^-^

简单选择器代码演示

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# css-id 定位元素
#driver.find_element_by_css_selector("#kw")
kw = driver.find_element(By.CSS_SELECTOR, "#kw")
kw.send_keys("演示 css-id 定位元素")
sleep(5)
# css-id 定位元素
kw = driver.find_element(By.CSS_SELECTOR, ".s_ipt")
kw.clear()
kw.send_keys("演示 css-class 定位元素")
sleep(5)
driver.quit()
```

29.6.3. 组合器选择器&演示

选择器	示例	示例描述
.class1.class2	<code>driver.find_element(By.CSS_SELECTOR, '.class1.class2')</code>	选中同时拥有 class1 和 class2 的元素
.class1 .class2	<code>driver.find_element(By.CSS_SELECTOR, '.class1 .class2')</code>	选中 class1 样式后代元素的所有 class2 样式
element.class	<code>driver.find_element(By.CSS_SELECTOR, 'element.class')</code>	选中 class 样式后代元素的所有 element 标签
element1,element2	<code>driver.find_element(By.CSS_SELECTOR, 'div,p')</code>	同时选中 div 和 p 两个标签元素
element>element	<code>driver.find_element(By.CSS_SELECTOR, 'div>p')</code>	选择其父元素是 <div> 元素的所有 <p> 元素
element+element	<code>driver.find_element(By.CSS_SELECTOR, 'div+p')</code>	选择所有紧随 <div> 元素之后的 <p> 元素
element1~element2	<code>driver.find_element(By.CSS_SELECTOR, 'div~ul')</code>	选择前面有 <p> 元素的每个 元素

代码请下翻一页哦^^

组合选择器代码演示

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(1)
# css-组合定位元素 1
kw = driver.find_element(By.CSS_SELECTOR, "#form span input")
kw.send_keys("css-组合定位元素 1")
sleep(5)
# css-组合定位元素 2
kw.clear()
kw = driver.find_element(By.CSS_SELECTOR, "#form span .s_ipt")
kw.send_keys("css-组合定位元素 2")
sleep(5)
driver.quit()
```


29.6.4. 伪类选择器&演示

选择器	例子	例子描述
:active	a:active	选择活动的链接。
:checked	input:checked	选择每个被选中的 <input>元素。
:disabled	input:disabled	选择每个被禁用的<input>元素。
:empty	p:empty	选择没有子元素的每个<p>元素。
:enabled	input:enabled	选择每个已启用的<input>元素。
:first-child	p:first-child	选择作为其父的首个子元素的每个<p>元素。
:first-of-type	p:first-of-type	选择作为其父的首个<p>元素的每个<p>元素。
:focus	input:focus	选择获得焦点的<input>元素。
:hover	a:hover	选择鼠标悬停其上的链接。
:lang(language)	p:lang(it)	选择每个 lang 属性值以"it"开头的<p>元素。
:last-child	p:last-child	选择作为其父的最后一个子元素的每个<p>元素。
:last-of-type	p:last-of-type	选择作为其父的最后一个<p>元素的每个<p>元素。
:link	a:link	选择所有未被访问的链接。
:not(selector)	:not(p)	选择每个非<p>元素的元素。

:nth-child(n)	p:nth-child(2)	选择作为其父的第二个子元素的每个<p>元素。
:nth-last-child(n)	p:nth-last-child(2)	选择作为父的第二个子元素的每个<p>元素，从最后一个子元素计数。
:nth-last-of-type(n)	p:nth-last-of-type(2)	选择作为父的第二个<p>元素的每个<p>元素，从最后一个子元素计数
:nth-of-type(n)	p:nth-of-type(2)	选择作为其父的第二个<p>元素的每个<p>元素。
:nth-child(odd)	p:nth-child(odd)	选择作为其父的奇数行的每个<p>子元素
:nth-child(even)	p:nth-child(even)	选择作为其父的偶数行的每个<p>子元素
:only-of-type	p:only-of-type	选择作为其父的唯一<p>元素的每个<p>元素。
:only-child	p:only-child	选择作为其父的唯一子元素的<p>元素。
:optional	input:optional	选择不带"required"属性的<input>元素。
:read-only	input:read-only	选择指定了"readonly"属性的<input>元素。
:read-write	input:read-write	选择不带"readonly"属性的<input>元素。
:required	input:required	选择指定了"required"属性的<input>元素。
:root	root	选择元素的根元素。
:target	#news:target	选择当前活动的#news 元素（单击包含该锚名称的 URL）。
:valid	input:valid	选择所有具有有效值的<input>元素。
:visited	a:visited	选择所有已访问的链接。

代码请下翻一页哦^-^

伪类行选择器代码演示

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(1)
vcurrent = driver.current_window_handle
# css-伪类选择器 1
kw = driver.find_element(By.CSS_SELECTOR, "#s-top-left a:first-child")
kw.click()
sleep(5)
# css-伪类选择器 2
driver.switch_to.window(vcurrent)
kw = driver.find_element(By.CSS_SELECTOR, "#s-top-left a:nth-last-of-type(2)")
kw.click()
sleep(5)
# css-伪类选择器 3
driver.switch_to.window(vcurrent)
kws = driver.find_elements(By.CSS_SELECTOR, "#s-top-left a:nth-child(odd)")
for one in kws:
    print(one.text)
sleep(5)
driver.quit()
```

29.6.5. 属性选择器&演示

选择器	例子	例子描述
[attribute]	[target]	选择带有 target 属性的所有元素。
[attribute=value]	[target=_blank]	选择带有 target="_blank" 属性的所有元素。
[attribute~value]	[title~flower]	选择带有包含 "flower" 一词的 title 属性的所有元素。
[attribute =value]	[lang =en]	选择带有以 "en" 开头的 lang 属性的所有元素。
[attribute^=value]	a[href^="https"]	选择其 href 属性值以 "https" 开头的每个 <a> 元素。
[attribute\$=value]	a[href\$=".pdf"]	选择其 href 属性值以 ".pdf" 结尾的每个 <a> 元素。
[attribute*=value]	a[href*="w3school"]	选择其 href 属性值包含子串 "w3school" 的每个 <a> 元素。

代码请下翻一页哦^^

属性选择器代码演示

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(1)
vcurrent = driver.current_window_handle
# css-属性选择器
kw = driver.find_element(By.CSS_SELECTOR, "a[href='http://news.baidu.com']")
kw.click()
sleep(5)
driver.quit()
```

29.6.6. 伪元素选择器&演示

选择器	例子	例子描述
<code>::after</code>	<code>p::after</code>	在每个 <code><p></code> 元素之后插入内容。
<code>::before</code>	<code>p::before</code>	在每个 <code><p></code> 元素之前插入内容。
<code>::first-letter</code>	<code>p::first-letter</code>	选择每个 <code><p></code> 元素的首字母。
<code>::first-line</code>	<code>p::first-line</code>	选择每个 <code><p></code> 元素的首行。
<code>::selection</code>	<code>p::selection</code>	选择用户选择的元素部分。

29.7. 元素定位-link text

1. 参考样例

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(1)
vcurrent = driver.current_window_handle
# link-text 元素定位
#kw = driver.find_element_by_link_text("新闻")
kw = driver.find_element(By.LINK_TEXT, "新闻")
kw.click()
sleep(5)
driver.quit()
```

2. 适用场景

元素必须为超链接，使用的场景有限

29.8. 元素定位-partial link text

1. 参考样例

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(1)
vcurrent = driver.current_window_handle
# partial-link-text 元素定位
kw = driver.find_element_by_partial_link_text("123")
kw = driver.find_element(By.PARTIAL_LINK_TEXT, "123")
kw.click()
sleep(5)
driver.quit()
```

2. 适用场景

元素必须为超链接，使用的场景有限

29.9. 元素定位-xpath 定位

29.9.1. 路径表达式

表达式	描述
nodename	选取此节点的所有子节点。
/	从根节点选取。
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。
[]	指定 xpath 条件查询

29.9.2. 谓语/条件表达式

条件	描述
/bookstore/book[1]	选取属于 bookstore 子元素的第一个 book 元素
/bookstore/book[last()]	选取属于 bookstore 子元素的最后一个 book 元素
/bookstore/book[last()-1]	选取属于 bookstore 子元素的倒数第二个 book 元素
/bookstore/book[position() < 3]	选取最前面的两个属于 bookstore 元素的子元素的 book 元素
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素
//title[@lang= 'eng']	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性
/bookstore/book[price>35.00]	选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00
/bookstore/book[price>35.00]/title	选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00
/bookstore/book[text()=' 百度类']	选取 bookstore 元素的所有 book 元素，且文本内容为:百度类

29.9.3. 通配符匹配

符号	描述
*	匹配任何元素节点
@*	匹配任何属性节点。
node()	匹配任何类型的节点。
以下为样例说明	
/bookstore/*	选取 bookstore 元素的所有子元素。
//*	选取文档中的所有元素。
//title[@*]	选取所有带有属性的 title 元素。

29.9.4. xpath 常用实例

<code>types</code>	选取 <code>types</code> 元素的所有子节点
<code>/types</code>	选取根元素 <code>types</code>
<code>types/book</code>	选取属于 <code>types</code> 的子元素的所有 <code>book</code> 元素
<code>//book</code>	选取所有 <code>book</code> 子元素，而不管它们在文档中的位置
<code>types//book</code>	选择属于 <code>types</code> 元素的后代的所有 <code>book</code> 元素，不管它们位于 <code>types</code> 之下的什么位置
<code>/types/book[1]</code>	选取属于 <code>types</code> 子元素的第一个 <code>book</code> 元素
<code>/types/book[last()]</code>	选取属于 <code>types</code> 子元素的最后一个 <code>book</code> 元素
<code>/types/book[last()-1]</code>	选取属于 <code>types</code> 子元素的倒数第二个 <code>book</code> 元素
<code>/types/book[position()<3]</code>	选取最前面的两个属于 <code>types</code> 元素的子元素的 <code>book</code> 元素
<code>//title[@lang]</code>	选取所有拥有名为 <code>lang</code> 的属性的 <code>title</code> 元素
<code>//title[@lang='eng']</code>	选取所有 <code>title</code> 元素，且这些元素拥有值为 <code>eng</code> 的 <code>lang</code> 属性
<code>/types/book[price>35.00]</code>	选取 <code>types</code> 元素的所有 <code>book</code> 元素，且其中的 <code>price</code> 元素的值须大于 35.00
<code>/types/book[price>35.00]/title</code>	选取 <code>types</code> 元素中的 <code>book</code> 元素的所有 <code>title</code> 元素，且其中的 <code>price</code> 元素的值须大于 35.00
<code>/bookstore/book[text()='百度']</code>	选取 <code>bookstore</code> 元素的所有 <code>book</code> 元素，且文本内容为:百度

29.9.5. xpath 高级实例

`//input[@id='id123']`

查找 input 元素,并且 id='id123'

`//input[@name='name1' and class='class1']`

查找 input 元素,name='name1'

`//a[text()='百度']`

查找 a 元素, 文本='百度'

`//a[contains(text(),"百度")]`

查找 a 元素, 文本包含'百度'

`//a[contains(@class,"class1")]`

查找 a 元素, class 含有 class1

`//div[@id="u1"]//a[@name="name"]`

利用复杂层级定位查询元素

`//a[@name="name3"]/preceding-sibling::a[@name="name1"]`

查找 a 元素(name=name3),再查找它前面的同级的 a 元素(name=name1)

`//a[@name="name3"]/following-sibling::a[@name="name5"]`

查找 a 元素(name=name3),再查找它后面的同级的 a 元素(name=name5)

`//a[@name="name3"]/parent::div/a[@name="name4"]`

查找 a 元素(name=name3),再查找它父元素(div)下的后代 a(name=name4)

`//a[text()='百度']/parent::div/following-sibling::div//a[@name="name1"]`

根据上面, 同学自己行理解一下……

https://www.w3cschool.cn/doc_xslt_xpath/xslt_xpath-xslt-comment.html?lang=en

29.9.6. 选择器&演示

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(1)
vcurrent = driver.current_window_handle
#kw = driver.find_element_by_xpath("//*[@id='s-top-left']/a[1]")
kw = driver.find_element(By.XPATH, "//*[@id='s-top-left']/a[1]")
print("打印 1: ", kw.get_attribute('outerHTML'))
kw = driver.find_element(By.XPATH, "//*[@id='s-top-left']/a[1]/parent::*a[2]")
print("打印 2: ", kw.get_attribute('outerHTML'))
# kw.click()
sleep(1)
driver.quit()
```

29.10.传统定位方式总结

定位方式	场景说明	优先级	
id 方式	元素必须有 id 属性	精确定位，效率高，优先使用	<code>driver.find_element('id', 'kw')</code> <code>driver.find_element_by_id("kw")</code>
name 方式	元素必须有 name 属性	精确定位，效率高，优先使用	<code>driver.find_element('name', 'wd')</code> <code>driver.find_element_by_name("wd")</code>
class name 方式	元素必须有 class name	宽松定位，一般要与其它定位相结合	<code>driver.find_element('class name', 's_ipt')</code> <code>driver.find_element_by_class_name("s_ipt")</code>
tag name 方式	-	宽松定位，一般要与其它定位相结合	<code>driver.find_element("tag name", "input")</code> <code>driver.find_elements_by_tag_name("input")</code>
css selector 方式	-	通用定位，效率与书写有关	<code>driver.find_element("css selector", "#kw")</code> <code>driver.find_element_by_css_selector("#kw")</code>
link text 方式	元素必须为<a>	精确定位，使用场景有限	<code>driver.find_element("link text", "新闻")</code> <code>driver.find_element_by_link_text("新闻")</code>
partial link text 方式	元素必须为<a>	精确定位，使用场景有限	<code>driver.find_element("partial link text", "123")</code> <code>driver.find_element_by_partial_link_text("123")</code>
xpath 方式	-	通用定位，效率与书写有关	<code>driver.find_element("xpath", "//*[@id='s-top-left']/a[1]")</code> <code>driver.find_element_by_xpath("//*[@id='s-top-left']/a[1]")</code>

css selector 方式与 xpath 方式的效率高低处决于表达式本身的效率：

使用优先级：精确定位(id,name 方式)>宽松定位

8 种方法里面均有单个和多个查找方式，如：find_element 和 find_elements

29.11.css&xpath 定位总结

CSS 选择器和 Xpath 有什么区别？我们该如何选择？

相信很同学都有这个疑问，而且这个问题一些面试官会经常问(无聊)

个人理解如下：

- 一，从语法上来说，CSS 表达式偏向简洁，Xpath 相对稍显复杂
- 二，CSS 定位效率相对高一些，这也是所有浏览器都支持的原因
- 三，功能上 Xpath 比 CSS 强大，如 CSS 不支持文本搜索，Xpath 支持文本搜索 `text()`
- 四，Xpath 支持的函数特别多，CSS 选择器支持的函数比较少，所以在复杂元素查找时候，xpath 反而更加简洁
- 五，如果你是前端开发出身，或者有前端开发的基础，相信你喜欢 CSS 选择器

个人建议如下：

- 一，优先使用 CSS 定位器，解决不了再使用 Xpath 定位器(场景非常非常少)
- 二，熟练使用 CSS 定位器，因为 CSS 定位器使用范围比 Xpath 更广，整个前台的样式语法均是采用 CSS 定位
- 三，熟练使用两种，肯定没有任何问题(^-^)

29.12.Selenium 相对定位

29.12.1. 相对定位简介

相对定位器是 selenium4 以后新增加的功能

使用场景：

当目标元素不是很好查找，但周边有元素比较容易查找

使用 API：

1. **above**：在当前元素的上方
2. **below**：在当前元素的下方
3. **toLeftOf**：在当前元素的左边
4. **toRightOf**：在当前元素的右边
5. **near**：在当前元素的附近

在实际使用中，有可能不准确，不推荐过多使用

29.12.2. 相对定位样例 1

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.relative_locator import locate_with
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 定位中间元素
su = driver.find_element(By.CSS_SELECTOR, "#s-top-left > a:nth-child(1)")
# 在中间元素周边查找
location = locate_with(By.CSS_SELECTOR, "a").to_right_of(su)
su = driver.find_element(location)
sleep(2)
su.click()
sleep(5)
driver.quit()
```

29.12.3. 相对定位样例 2

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.relative_locator import locate_with
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 定位中间元素
su = driver.find_element(By.ID, "su")
# 在中间元素周边查找
location = locate_with(By.CSS_SELECTOR, "input").near(su)
su = driver.find_element(location)
su.send_keys("相对定位样例 2")
sleep(5)
driver.quit()
```

29.13.万能 javascript 定位

29.13.1. js 定位分类

JavaScript 作为一种编程语言

它本身拥有着自己一套强大的元素定位方式，

同时可以结合 js 编程的特点，采用诸如条件、循环等复杂处理方式，定位出业务需要的元素

序号	选择器分类	场景说明
1	id 定位	document.getElementById()
2	name 定位	document.getElementsByName()
3	tag 定位	document.getElementsByTagName()
4	class 定位	document.getElementsByClassName()
5	css 定位	document.querySelector()、document.querySelectorAll()

29.13.2. javascript 样例 1

```
from time import sleep
from xml.dom.minidom import Document
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.relative_locator import locate_with
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 定位元素
kw=driver.execute_script("return document.querySelector('#kw')")
print(kw)
kw.send_keys("万能的 js 定位演示 1")
sleep(5)
driver.quit()
```

29.13.3. javascript 样例 2

```
from time import sleep
from xml.dom.minidom import Document
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.relative_locator import locate_with
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 定位元素
kw=driver.execute_script("return document.querySelector('#s-top-left > a:nth-child(1)')")
print(kw)
kw.click()
sleep(5)
driver.quit()
```

29.13.4. javascript 样例 3

```
from time import sleep
from xml.dom.minidom import Document
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.relative_locator import locate_with
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# js 定位元素
script = "let alist=document.querySelectorAll('#wrapper a');"
script = script+"for(let i=0;i<alist.length;i++){\"
script = script+"let a=alist[i];\"
script = script+"if(a&&a.href=='http://news.baidu.com/') return a;\"
script = script+"}\"
kw = driver.execute_script(script)
print(kw)
kw.click()
sleep(5)
driver.quit()
```

29.14.万能代码选择器定位

29.14.1. 代码定位简介

Python/Java 作为一种编程语言

它可以触发所有的基础元素定位方法

同时可以结合编程的特点，采用诸如条件、循环等复杂处理方式，定位出业务需要的元素

29.14.2. 代码定位样例

```
from time import sleep
from xml.dom.minidom import Document
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.relative_locator import locate_with
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 定位元素
wrapper = driver.find_element(By.ID, 'wrapper')
alist = wrapper.find_elements(By.TAG_NAME, 'a')
for a in alist:
    if a.get_dom_attribute('href')== 'http://news.baidu.com':
        a.click()
sleep(5)
driver.quit()
```


29.15.多种定位方式演示



如上图所示，我们需要选择如上元素，有多少种方法可以实现呢？我们一起来想想吧！

id 方式	-不可以
name 方式	-不可以
class name 方式	-与其它相结合
tag name 方式	-与其它相结合
css selector 方式	-可以
link text 方式	-可以
partial link text	-可以
xpath 方式	-可以
相对定位	-可以
javascript 定位	-可以
代码选择定位	-可以

在真正的项目中，我们都是采用效率最高的，而且是自己最为熟悉的定位方式

30. Selenium 元素交互

30.1. 交互类型汇总

基础交互动作类型

序号	元互动作	条件说明
1	click	适用较广，基本上所有可见元素都可以点击
2	send keys	可输入内容的元素，如文本框、文本域
3	clear	可输入内容的元素，如文本框、文本域
4	select	下拉框
.....

基础交互元素类型

序号	元互类型	交互动作
1	text/textarea	send_keys
2	checkbox	click
3	radiobox	click
4	select	select
5	form	send_keys、click
6	a	click
.....

30.2. 操作 ChexkBox

场景说明：模拟 checkBox 的选中与取消

```
from time import sleep
from typing import List
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.remote.webelement import WebElement
driver = webdriver.Chrome()
driver.get("http://www.divcss5.com/yanshi/checkbox.html")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 查找 checkbox 组
eles: List[WebElement] = driver.find_elements(By.NAME, 'Fruit')
for i in range(200):
    for ele in eles:
        # 取水原来选中的
        if (ele.is_selected()):
            ele.click()
    # 选中当前
    index = i % 4
    eles[index].click()
    sleep(0.2)
sleep(5)
driver.quit()
```

30.3. 操作 RadioBox

场景说明：模拟 RadioBox 的选中与取消

```
from distutils.log import debug
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("http://www.divcss5.com/yanshi/radio.html")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 查找 checkbox 组
eles = driver.find_elements(By.NAME, 'Fruit')

for i in range(200):
    index = i % 4
    eles[index].click()
    print(eles[index].is_selected())
    sleep(0.2)
sleep(5)
driver.quit()
```

30.4. 操作 Form 表单

场景说明：模拟表单提交

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com/")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 查找 checkbox 组
sleep(1)
driver.find_element(By.ID, 's-top-loginbtn').click()
sleep(2)
driver.find_element(By.ID, "TANGRAM__PSP_11__userName").send_keys('user001')
sleep(1)
driver.find_element(By.ID, "TANGRAM__PSP_11__password").send_keys('passwd001')
sleep(1)
driver.find_element(By.ID, 'TANGRAM__PSP_11__submit').click()
sleep(5)
driver.quit()
```

30.5. 操作下拉列表

30.5.1. 下拉列表演示

```
from distutils.log import debug
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.select import Select
driver = webdriver.Chrome()
driver.get("http://www.divcss5.com/fanli/form-select.html")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 查找 checkbox 组
sleep(1)
select_element=driver.find_element(By.CSS_SELECTOR,"select:first-of-type")
select_object = Select(select_element)
for i in range(200):
    sleep(0.5)
    #select_object.select_by_index(i%2)
    select_object.select_by_value(str(i%2))
driver.quit()
```

30.5.2. 常用用法

```
# 通过索引选择
select_object.select_by_index(1)
# 通过值选择
select_object.select_by_value('value1')
# 通过可见文本进行选择
select_object.select_by_visible_text('Bread')
```

下面这些可用有 mult-select 上

```
# 通过索引取消选择
select_object.deselect_by_index(1)
# 通过值取消选择
select_object.deselect_by_value('value1')
# 通过可见文本取消选择
select_object.deselect_by_visible_text('Bread')
# 取消所有选择
select_object.deselect_all()
```

```
# 返回选中的选项 list 列表
all_selected_options = select_object.all_selected_options
# 返回选中的第一个选项
first_selected_option = select_object.first_selected_option
# 返回所有选项 list 列表
all_available_options = select_object.options
```

```
# 判断是否可多选
does_this_allow_multiple_selections = select_object.is_multiple
```

30.6. 读取元素信息

30.6.1. 常用属性&函数

序号	属性&函数	适用范围	用法说明
1	ele.id	通用	selenium 内部 ID，并不是我们理解的 html 中的 id
2	ele.text	文本框、文本域	元素的内容
3	ele.tag_name	通用	元素的标签
4	ele.size	通用	元素大小
5	ele.location	通用	元素位置
6	ele.get_attribute('x')	通用	元素的属性，元素 html 特征
7	ele.get_property('x')	通用	元素的属性
8	ele.get_dom_attribute('x')	通用	元素的 html 特征(需在 html 特征中有定义)
9	ele.value_of_css_property('x')	通用	元素的 css 样式值
10	ele.is_displayed()	通用	元素是否可见
11	ele.is_enabled()	通用	元素是否非禁用
12	ele.is_selected()	下拉/选框类	元素是否选中

30.6.2. property&attribute

1. 基础理解

property 是 dom 属性，是 JavaScript 里面的对象，偏向于后台

attribute 是 html 元素上的特征，偏向于前台

attribute 可以理解为 dom 自带的 property，如 id,name 等

很多情况下 property 和 attribute 一样，所以极难区别

2. 举例说明

```
ele = driver.find_element(  
    By.XPATH, "//*[@id='kw']")  
print("打印结果: ", ele.get_attribute('id'), ele.get_property(  
    'id'), ele.get_dom_attribute('id'))  
print("打印结果: ", ele.get_attribute('outerHTML'), ele.get_property(  
    'outerHTML'), ele.get_dom_attribute('outerHTML'))  
print("打印结果: ", ele.get_attribute('class'), ele.get_property(  
    'class'), ele.get_dom_attribute('class'))  
driver.quit()
```

由于可以看出：

id,name 这样的值的 property 和 attribute 相同

outerHTML 是 property，但非 attribute

class 是 attribute，但非 property

3. 我们可以认为 `ele.get_attribute('x')` 的范围 = `ele.get_property('x') + ele.get_dom_attribute('x')`

所以如果我们为了方便，通常优先使用 `ele.get_attribute('x')`

30.6.3. 演示代码样例

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(5)
ele = driver.find_element(
    By.XPATH, "//node()[@id='kw']")
print("打印结果: ", ele.get_attribute('id'), ele.get_property(
    'id'), ele.get_dom_attribute('id'))
print("打印结果: ", ele.get_attribute('outerHTML'), ele.get_property(
    'outerHTML'), ele.get_dom_attribute('outerHTML'))
print("打印结果: ", ele.get_attribute('class'), ele.get_property(
    'class'), ele.get_dom_attribute('class'))
sleep(5)
driver.quit()
```

31. Selenium 事件交互

31.1. 键盘事件

31.1.1. 常用 Api

1. KeyDown 事件

```
ActionChains(driver).key_down(Keys.SHIFT).send_keys("abc").perform()
```

2. KeyUp 事件

```
ActionChains(driver).key_down(Keys.SHIFT).send_keys("a").key_up(Keys.SHIFT).send_keys("b").perform()
```

3. SendKeys 事件

```
text_input = driver.find_element(By.ID, "textInput")
ActionChains(driver).send_keys_to_element(text_input, "abc").perform()
```

4. 复制粘贴事件

```
# ctrl+a 全选输入框内容
driver.find_element_by_id("kw").send_keys(Keys.CONTROL, 'a')
# ctrl+x 剪切输入框内容
driver.find_element_by_id("kw").send_keys(Keys.CONTROL, 'x')
# ctrl+c 复制输入框内容
driver.find_element_by_id("kw").send_keys(Keys.CONTROL, 'c')
# ctrl+v 粘贴内容到输入框
driver.find_element_by_id("kw").send_keys(Keys.CONTROL, 'v')
```

31.1.2. 案例源码

```
from time import sleep
from selenium import webdriver
from selenium.webdriver import Keys
from selenium.webdriver.chrome.options import Options
driver = webdriver.Chrome()
driver.get("http://www.baidu.com")
driver.find_element_by_id("kw").send_keys("春风阁讲堂 6") # 输入框输入内容
sleep(2)
driver.find_element_by_id("kw").send_keys(Keys.BACK_SPACE) # 删除多输入的一个 6
sleep(2)
driver.find_element_by_id("kw").send_keys(Keys.SPACE) # 输入空格键 + B 站
driver.find_element_by_id("kw").send_keys("B 站")
sleep(2)
driver.find_element_by_id("kw").send_keys(Keys.CONTROL, 'a') # ctrl+a 全选输入框内容
sleep(2)
driver.find_element_by_id("kw").send_keys(Keys.CONTROL, 'x') # ctrl+x 剪切输入框内容
sleep(2)
driver.find_element_by_id("kw").send_keys(Keys.CONTROL, 'v') # ctrl+v 粘贴内容到输入框
sleep(2)
driver.find_element_by_id("su").send_keys(Keys.ENTER) # 通过回车键来代替单击操作
sleep(2)
driver.quit()
```

31.1.3. 常用按键

按键	名称	按键	名称
Keys. BACK_SPACE	回退键 (BackSpace)	Keys. INSERT	插入键 (Insert)
Keys. TAB	制表键 (Tab)	Keys. DELETE	删除键 (Delete)
Keys. ENTER	回车键 (Enter)	Keys. NUMPAD0 ~ NUMPAD9	数字键 1-9
Keys. SHIFT	大小写转换键 (Shift)	Keys. F1 ~ F12	F1 - F12 键
Keys. CONTROL	Control 键 (Ctrl)	(Keys. CONTROL, 'a')	组合键 Ctrf+a, 全选
Keys. ALT	ALT 键 (Alt)	(Keys. CONTROL, 'c')	组合键 Ctrf+c, 复制
Keys. ESCAPE	返回键 (Esc)	(Keys. CONTROL, 'x')	组合键 Ctrf+x, 剪切
Keys. SPACE	空格键 (Space)	(Keys. CONTROL, 'v')	组合键 Ctrf+v, 粘贴
Keys. END	行尾键 (End)	Keys. PAGE_UP	翻页键上 (Page Up)
Keys. HOME	行首键 (Home)	Keys. PAGE_DOWN	翻页键下 (Page Down)
Keys. LEFT	方向键左 (Left)		
Keys. UP	方向键上 (Up)		
Keys. RIGHT	方向键右 (Right)		
Keys. DOWN	方向键下 (Down)		

31.2. 鼠标事件

31.2.1. 鼠标按键

- 0 — 左键(默认)
- 1 — 滚轮(Selenium 目前还不支持)
- 2 — 右键(偶尔用)
- 3 — 上页键(少, 一般设备未有)
- 4 — 下页键(少, 一般设备未有)

通常我们所说的鼠标就是三键鼠标, 五键鼠标比较少见, 另外就是滚轮在 Selenium 当前并不支持

31.2.2. 常用 Api

1. 鼠标点击按住

```
clickable = driver.find_element(By.ID, "clickable")
ActionChains(driver).click_and_hold(clickable).perform()
```

2. 点击释放

```
clickable = driver.find_element(By.ID, "click")
ActionChains(driver).click(clickable).perform()
```

3. 鼠标右击

```
clickable = driver.find_element(By.ID, "clickable")
ActionChains(driver).context_click(clickable).perform()
```

4. 鼠标后翻页(属于触控笔 Api)

```
action = ActionBuilder(driver)
action.pointer_action.pointer_down(MouseButton.BACK)
action.pointer_action.pointer_up(MouseButton.BACK)
action.perform()
```

5. 鼠标前翻页(属于触控笔 Api)

```
action = ActionBuilder(driver)
action.pointer_action.pointer_down(MouseButton.FORWARD)
action.pointer_action.pointer_up(MouseButton.FORWARD)
action.perform()
```

6. 鼠标双击

```
clickable = driver.find_element(By.ID, "clickable")
ActionChains(driver).double_click(clickable).perform()
```

翻页有惊喜哦^-^

7. 移动到元素

```
hoverable = driver.find_element(By.ID, "hover")
ActionChains(driver).move_to_element(hoverable).perform()
```

8. 按窗口偏移(属于触控笔 Api)

```
action = ActionBuilder(driver)
action.pointer_action.move_to_location(8, 12)
action.perform()
```

9. 按当前指针位置偏移

```
ActionChains(driver).move_by_offset(13, 15).perform()
```

10. 元素拖放

```
draggable = driver.find_element(By.ID, "draggable")
droppable = driver.find_element(By.ID, "droppable")
ActionChains(driver).drag_and_drop(draggable, droppable).perform()
```

11. 按偏移量拖放

```
draggable = driver.find_element(By.ID, "draggable")
start = draggable.location
finish = driver.find_element(By.ID, "droppable").location
ActionChains(driver).drag_and_drop_by_offset(draggable, finish['x']-start['x'], finish['y']-start['y']).perform()
```


31.2.3. 演示代码样例 1

```
from time import sleep
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("http://www.baidu.com")
driver.maximize_window()
driver.implicitly_wait(10)
# 形成历史记录
driver.find_element(By.ID, "kw").send_keys("测试")
driver.find_element(By.ID, "su").click()
sleep(2)
driver.back()
sleep(2)
kw = driver.find_element(By.ID, "kw")
head_wrapper = driver.find_element(By.ID, "head_wrapper")
# 鼠标点击, 输入框失去焦点变灰
ActionChains(driver).click_and_hold(head_wrapper).pause(2).perform()
# 鼠标点击, 输入框获得焦点外框变蓝
ActionChains(driver).move_to_element(kw).click_and_hold().pause(2).perform()
# 鼠标释放, 输入框出现历史记录
ActionChains(driver).move_to_element(kw).release().pause(2).perform()
# 移动到历史框记录 并点击第 1 条
ActionChains(driver).move_by_offset(5, 50).click().pause(2).perform()
driver.quit()
```

31.2.4. 演示代码样例 2

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get('https://selenium.dev/selenium/web/mouse_interaction.html')
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 源元素
draggable = driver.find_element(By.ID, "draggable")
# 目标元素
droppable = driver.find_element(By.ID, "droppable")
# 拖动元素
ActionChains(driver).pause(2).drag_and_drop(
    draggable, droppable).pause(5).perform()
driver.quit()
```

<https://sahitest.com/demo/dragDropDataTransfer.htm>

31.2.5. 演示代码样例 3

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get('https://sahitest.com/demo/dragDropDataTransfer.htm')
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
# 源元素
draggable = driver.find_element(By.ID, "drag1")
# 目标元素
droppable = driver.find_element(By.ID, "div3")
# 拖动元素(结果无效)
ActionChains(driver).pause(2).drag_and_drop(
    draggable, droppable).pause(5).perform()
driver.quit()
```

注意: drag_and_drop 函数并不支持 html5 的拖曳操作, 对此, 我们在后面有 JS 的解决方案, 这里不作细节讲述了

31.3. 滚动事件

31.3.1. 常用 Api

1. 滚动到元素

```
ActionChains(driver).scroll_to_element(iframe).perform()
```

2. 给定数量滚动

```
ActionChains(driver).scroll_by_amount(0, step).pause(0.2).perform()
```

3. 元素+滚动数量

```
scroll_origin = ScrollOrigin.from_element(iframe)
ActionChains(driver).scroll_from_origin(scroll_origin, 0, 200).perform()
```

4. 元素(偏移量)+滚动数量

```
scroll_origin = ScrollOrigin.from_element(footer, 0, -50)
ActionChains(driver).scroll_from_origin(scroll_origin, 0, 200).perform()
```

5. 从原点的偏移量滚动

```
scroll_origin = ScrollOrigin.from_viewport(10, 10)
ActionChains(driver).scroll_from_origin(scroll_origin, 0, 200).perform()
```

31.3.2. 演示代码样例 1

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
from selenium.webdriver.common.service import sleep
driver = webdriver.Chrome()
driver.get("https://selenium.dev/selenium/web/scrolling_tests/frame_with_nested_scrolling_frame_out_of_view.html"
)
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(2)
iframe = driver.find_element(By.TAG_NAME, "iframe")
# 滚动到指定的元素
ActionChains(driver).scroll_to_element(iframe).perform()
sleep(5)
driver.quit()
```

31.3.3. 演示代码样例 2

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
from selenium.webdriver.common.service import sleep
driver = webdriver.Chrome()
driver.get("https://selenium.dev/selenium/web/scrolling_tests/frame_with_nested_scrolling_frame_out_of_view.html"
)
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(2)
footer = driver.find_element(By.TAG_NAME, "footer")
delta_y = footer.rect['y']
step = delta_y/100
for tnum in range(100):
    # 滚动给动的数量
    ActionChains(driver).scroll_by_amount(0, int(step)).pause(0.2).perform()
sleep(5)
driver.quit()
```

31.3.4. 演示代码样例 3

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.action_chains import ScrollOrigin
from selenium.webdriver.common.by import By
from selenium.webdriver.common.service import sleep
driver = webdriver.Chrome()
driver.get("https://selenium.dev/selenium/web/scrolling_tests/frame_with_nested_scrolling_frame_out_of_view.html")
)
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(2)
iframe = driver.find_element(By.TAG_NAME, "iframe")
scroll_origin = ScrollOrigin.from_element(iframe)
# 滚动到给定元素+同时滚动数量
ActionChains(driver).scroll_from_origin(scroll_origin, 0, 200).perform()
sleep(5)
driver.quit()
```

31.3.5. 演示代码样例 4

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.action_chains import ScrollOrigin
from selenium.webdriver.common.by import By
from selenium.webdriver.common.service import sleep
driver = webdriver.Chrome()
driver.get("https://selenium.dev/selenium/web/scrolling_tests/frame_with_nested_scrolling_frame_out_of_view.html")
)
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(2)
footer = driver.find_element(By.TAG_NAME, "footer")
scroll_origin = ScrollOrigin.from_element(footer, 0, -50)
# 元素(偏移量)+滚动数量
ActionChains(driver).scroll_from_origin(scroll_origin, 0, 200).perform()
sleep(5)
driver.quit()
```


31.3.6. 演示代码样例 5

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.action_chains import ScrollOrigin
from selenium.webdriver.common.by import By
from selenium.webdriver.common.service import sleep
driver = webdriver.Chrome()
driver.get("https://selenium.dev/selenium/web/scrolling_tests/frame_with_nested_scrolling_frame_out_of_view.html")
)
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(2)
scroll_origin = ScrollOrigin.from_viewport(10, 10)
# 从原点的偏移量滚动
ActionChains(driver).scroll_from_origin(scroll_origin, 0, 200).perform()
sleep(5)
driver.quit()
```

32. Selenium 窗口操作

32.1. 浏览器操作

32.1.1. 基础动作

序号	操作	场景说明
1	<code>driver.get("https://www.baiduc.om")</code>	打开某一个网页
2	<code>driver.back()</code>	浏览器后退
3	<code>driver.forward()</code>	浏览器前进
4	<code>driver.refresh()</code>	浏览器刷新
5	<code>driver.quit()</code>	浏览器退出

32.1.2. 演示样例 1

```
from time import sleep
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("https://www.baidu.com")
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
sleep(5)
driver.back()
sleep(5)
driver.forward()
sleep(5)
driver.refresh()
sleep(5)
driver.quit()
```

32.2. iframe 操作

32.2.1. 常用 Api 操作

1. 按元素切换至 iframe

```
iframe = driver.find_element(By.CSS_SELECTOR, "#modal > iframe")
driver.switch_to.frame(iframe)
```

2. 按 id/name 切换至 iframe

```
driver.switch_to.frame('buttonframe')
driver.find_element(By.TAG_NAME, 'button').click()
```

3. 按索引切换

```
iframe = driver.find_elements_by_tag_name('iframe')[1]
driver.switch_to.frame(iframe)
```

重要：通常我们要定位到 `iframe` 里面的元素，首先要切换至该 `iframe`

32.2.2. iframe 演示样例

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
sleep(2)
driver.get('https://sahitest.com/demo/iframesTest.htm')
# 定位到某个 iframe
iframe = driver.find_elements(By.CSS_SELECTOR, "iframe")
driver.switch_to.frame(iframe[0])
sleep(2)
# 定位 iframe 中的元素
driver.find_element(By.CSS_SELECTOR, 'a:first-of-type').click()
sleep(10)
driver.quit()
```

32.3. 窗口-打开与关闭

32.3.1. 常用 API

1. 获取当前窗口/标签页

```
driver.current_window_handle
```

2. 等待窗口/标签页

```
wait = WebDriverWait(driver, 10)
wait.until(expected_conditions.number_of_windows_to_be(2))
```

3. 切换新窗口

```
driver.switch_to.window(original_window)
```

4. 创建新窗口并切换

```
driver.switch_to.new_window('tab')      # 打开新标签页并切换到新标签页
driver.switch_to.new_window('window')   # 打开一个新窗口并切换到新窗口
```

5. 关闭窗口/标签页

```
driver.close()
```

6. 退出浏览器

```
driver.quit()
```

7. 循环窗口列表

```
for window_handle in driver.window_handles:
    if window_handle != original_window:
        driver.switch_to.window(window_handle)
        break
```

32.3.2. 演示样例 1

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com')
wait = WebDriverWait(driver, 10)
sleep(2)
owindow = driver.current_window_handle
wait = WebDriverWait(driver, 10)
wait.until(expected_conditions.number_of_windows_to_be(2))
sleep(2)
driver.switch_to.new_window('tab')
sleep(2)
driver.get('http://www.baidu.com')
sleep(2)
driver.switch_to.window(owindow)
sleep(10)
driver.quit()
```

32.3.3. 演示样例 2

```
from time import sleep

driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com')
wait = WebDriverWait(driver, 10)
xinwen = driver.find_element(
    By.CSS_SELECTOR, '#s-top-left a:first-of-type')
hao123 = driver.find_element(
    By.CSS_SELECTOR, '#s-top-left a:nth-of-type(2)')
map = driver.find_element(
    By.CSS_SELECTOR, '#s-top-left a:nth-of-type(3)')
sleep(2)
xinwen.click()
sleep(2)
hao123.click()
sleep(2)
map.click()
sleep(2)
for index in range(len(driver.window_handles)):
    driver.switch_to.window(driver.window_handles[index])
    sleep(2)
sleep(10)
driver.quit()
```


32.3.4. 窗口顺序

打开浏览器(百度首页), 然后依次打开百度新闻、hao123、百度地图

在 `driver.window_handles` 存储的顺序分别为:

0: 百度首页

1: 百度地图

2: hao123

3: 百度新闻

32.4. 窗口-窗口管理

32.4.1. 常用 API

1. 取得窗口大小

```
width = driver.get_window_size().get("width")  
height = driver.get_window_size().get("height")
```

2. 设置窗口大小

```
driver.set_window_size(1024, 768)
```

3. 取得窗口位置

```
x = driver.get_window_position().get('x')  
y = driver.get_window_position().get('y')
```

4. 设置窗口位置

```
driver.set_window_position(0, 0)
```

5. 最大化窗口

```
driver.maximize_window()
```

6. 最小化窗口

```
driver.minimize_window()
```

7. 全屏窗口

```
driver.fullscreen_window()
```

32.4.2. 演示样例 1

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com')
wait = WebDriverWait(driver, 10)
sleep(2)
driver.minimize_window()
sleep(2)
driver.maximize_window()
sleep(2)
driver.set_window_size(400, 380)
sleep(2)
driver.set_window_position(200, 200)
sleep(2)
driver.set_window_position(0, 0)
sleep(2)
driver.maximize_window()
sleep(2)
driver.fullscreen_window()
sleep(2)
driver.maximize_window()
sleep(10)
driver.quit()
```


32.5. 窗口-截图操作

32.5.1. 常用 API

1. 屏幕截图

```
driver.save_screenshot('e:/image.png')
```

2. 元素截图

```
ele.screenshot('./image.png')
```

3. 输出 base64 编码

```
base64 = ele.screenshot_as_base64
```

4. 元素截图

```
pngs = ele.screenshot_as_png
```

32.5.2. 全屏截图演示

```
from time import sleep, time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com')
wait = WebDriverWait(driver, 10)
sleep(2)
# driver.save_screenshot('./image.png')
driver.save_screenshot('e:/image.png')
sleep(2)
driver.quit()
```

32.5.3. 元素截图演示

```
from time import sleep, time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com')
wait = WebDriverWait(driver, 10)
sleep(2)
ele = driver.find_element(By.CSS_SELECTOR, '#head_wrapper .s_form_wrapper')
# 将元素保存为图片
ele.screenshot('./image.png')
base64 = ele.screenshot_as_base64
print(base64)
pngs = ele.screenshot_as_png
print(pngs)
sleep(2)
driver.quit()
```

33. Selenium 其它操作

33.1. 对话框-alert

33.1.1. 常用 Api

1. 访问 alert 的话对框文本

```
text = alert.text
```

2. 等待对话框出现

```
wait = WebDriverWait(driver, 10)  
alert = wait.until(expected_conditions.alert_is_present()) # 等待直到对话框出现
```

3. 定位到对话框

```
alert = driver.switch_to.alert
```

4. 模拟点击

```
alert.accept()
```


33.1.2. alert 演示样例 1

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com/')
driver.find_element(By.XPATH, "//*[@id='s-usersetting-top']").click()
# 打开"搜索设置"
driver.find_element(By.XPATH, "//*[@id='s-user-setting-menu']/div/a[1]").click()
driver.find_element(By.XPATH, '//*[@id="se-setting-7"]/a[2]').click() # 点击保存设置
sleep(2)
alter = driver.switch_to.alert # 定位到对话框
sleep(2)
alter.accept() # 模拟确定按钮
sleep(2)
driver.execute_script("alert('对话框演示完结');") # 模拟弹出对话框
wait = WebDriverWait(driver, 10)
alert = wait.until(expected_conditions.alert_is_present()) # 等待直到对话框出现
text = alert.text
print(text)
sleep(2)
alert.accept()
sleep(5)
driver.quit()
```


33.2. 对话框-confirm

33.2.1. 常用 Api

1. 访问 alert 的话对框文本

```
text = alert.text
```

2. 等待对话框出现

```
wait = WebDriverWait(driver, 10)  
alert = wait.until(expected_conditions.alert_is_present()) # 等待直到对话框出现
```

3. 定位到对话框

```
alert = driver.switch_to.alert
```

4. 模拟点击

```
alert.accept()
```

5. 确认框取消

```
alert.dismiss()
```

33.2.2. 演示样例 1

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com/')
sleep(2)
driver.execute_script("window.confirm('是否确定?');") # 模拟弹出确认框
wait = WebDriverWait(driver, 10)
wait.until(expected_conditions.alert_is_present())
alert = driver.switch_to.alert # 定位到确认框
text = alert.text
print("对话框展示:", text)
sleep(5)
# alert.accept() # 确认确认框
alert.dismiss() # 取消确认框
sleep(5)
driver.quit()
```

33.3. 对话框-prompt

33.3.1. 常用 Api

1. 访问 alert 的话对框文本

```
text = alert.text
```

2. 等待对话框出现

```
wait = WebDriverWait(driver, 10)  
alert = wait.until(expected_conditions.alert_is_present()) # 等待直到对话框出现
```

3. 定位到对话框

```
alert = driver.switch_to.alert
```

4. 模拟输入

```
alert.send_keys("abcdefg")
```

5. 模拟点击

```
alert.accept()
```

6. 确认框取消

```
alert.dismiss()
```

33.3.2. 演示样例 1

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com/')
sleep(2)
driver.execute_script(
    "var ttt=window.prompt('请输入文字','默认值');console.dir('输入值:'+ttt);") # 模拟弹出确认框
wait = WebDriverWait(driver, 10)
wait.until(expected_conditions.alert_is_present()) # 等待输入框的出现
alert = driver.switch_to.alert # 定位到输入框
text = alert.text
print("输入框展示:", text)
alert.send_keys("abcdefg") # 模拟输入文本
sleep(2)
# alert.dismiss()
alert.accept() # 模拟确认
sleep(5)
# driver.quit()
```

33.4. Cookie 操作

33.4.1. 常用 Api 操作

序号	操作	场景说明
1	<code>driver.get_cookies()</code>	取所有 Cookie
2	<code>driver.get_cookie("foo")</code>	取指定名称的 Cookie
3	<code>driver.add_cookie({"name": "key", "value": "value"})</code>	添加指定 Cookie
4	<code>driver.delete_cookie("test1")</code>	删除指定名称 Cookie
5	<code>driver.delete_all_cookies()</code>	删除所有 Cookie

利用 cookie 操作，有时候我们可以跳过复杂的验证码、短信码登录，让自自化测试正常运行，后面会有实例讲解

33.4.2. Cookie 样例 1

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com/')
sleep(2)
cookies = driver.get_cookies()
# 打印当前所有 cookie
for index in range(len(cookies)):
    print(str(index), cookies[index])
# 获取指定名 cookie
bcookie = driver.get_cookie("BIDUPSID")
print('测试值:', bcookie['name'], '--', bcookie['value'])
sleep(5)
driver.quit()
```


33.4.3. Cookie 样例 2

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://www.baidu.com/')
sleep(2)
driver.add_cookie({"name": "longge_1", "value": "longge_value"})
driver.add_cookie({"name": "longge_2", "value": "longge_value2"})
# 打印当前所有 cookie
cookies = driver.get_cookies()
for index in range(len(cookies)):
    print(str(index), cookies[index])
print('华丽的分割线 1-----')
# 删除 cookie
driver.delete_cookie('longge_2')
cookies = driver.get_cookies()
for index in range(len(cookies)):
    print(str(index), cookies[index])
print('华丽的分割线 2-----')
driver.delete_all_cookies()
cookies = driver.get_cookies()
print(cookies)
driver.quit()
```

33.5. 文件上传

33.5.1. 文件上传演示

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait
driver = webdriver.Chrome()
driver.maximize_window()
driver.get('https://sahitest.com/demo/php/fileUpload.htm')
file2 = driver.find_element(By.ID, 'file2')
file3 = driver.find_element(By.ID, 'file3')
# 上传文件
file2.send_keys('C:/Users/xiang/Desktop/files/file2.txt')
sleep(1)
# 上传文件
file3.send_keys('C:/Users/xiang/Desktop/files/file3.txt')
sleep(3)
submit = driver.find_element(
    By.CSS_SELECTOR, 'form:nth-of-type(3) input[type=submit]')
# 提交文件
submit.click()
sleep(10)
driver.quit()
```

33.6. 异步等待方式

33.6.1. 等待方式

在 UI 自动化测试中，必然会遇到环境不稳定，网络慢的情况，这是如果不做任何处理的话，代码会由于没有找到元素而报错。这时我们就要用到 `wait`，而且在 `selenium` 中，我们可以用到一共三种等待。

序号	操作	场景说明
1	<code>time.sleep</code>	强制/固定等待
2	<code>implicitly_wait()</code>	隐式/智能等待
3	<code>WebDriverWait</code>	显示/智能等待

33.6.2. time.sleep

1. 定义：强制等待是最简单粗暴的一种办法
2. 优点：方便快捷
3. 缺点：固定周期，影响效率，一般仅在调试中使用

33.6.3. implicitly_wait()

1. 定义：设置一个最长等待时间，如果在规定时间内加载完，则执行下一步，否则一直等到时间截止
2. 优点：与 sleep 相比，时间上是范围内智能的等待
3. 缺点：程序会一直等待整个页面加载完成，单个无关紧要元素，会卡住整个执行步骤

33.6.4. WebDriverWait

1. 定义：周期检测，如果条件成立，则执行下一步，否则继续等待，直到超过设置的最长时间
2. 优点：可以灵活设置等待条件
3. 缺点：需要自行分析针对性等待条件

WebDriverWait 配合该类的 until()和 until_not()方法，就能够根据判断条件而进行灵活地等待

WebDriverWait 配合元素定位器使用，用来等待元素加载完

```
alert = wait.until(driver.find_element(By.CSS_SELECTOR, '#id'))
```

WebDriverWait 配合 expected_conditions 使用，用来判断一些特定条件的加载

```
wait.until(expected_conditions.alert_is_present())
```

34. Selenium 之 JS 应用

34.1. JS 使用场景

JavaScript（简称“JS”）是一种具有函数优先的轻量级，解释型或即时编译型的编程语言。它最大的使用场景是作为开发 Web 页面的脚本语言，被广泛的应用浏览器

JavaScript 有 ECMAScript5 和 ECMAScript6 两个非常重要的标准，目前大多数浏览器已经逐渐支持 ECMAScript6；目前 Selenium，已经支持 ECMAScript6 标准

JavaScript 学习网址：<https://www.w3school.com.cn/js/index.asp>(语言基础)，https://www.w3school.com.cn/js/js_html5dom.asp(操作 dom 元素)

JS 在 Selenium 自动化中，可以应用于许多场景，可以这么说，你能够用 Selenium 代码的地方，几乎都可以用 JS 实现，但一般我们不这么做，通常我们可以如下场景考虑使用 JavaScript

1. 简化流程：

采用 Selenium 操作时，流程可能非常复杂，比如选择时间，一般用 Selenium 会有年、月、日、甚至时、分、秒好几步操作，但如果用 JS 实现，可能就一行代码，毕竟 Selenium 是基于可视化操作，而 JS 可以直接操作 Dom 元素

2. 补充实现：

Selenium 虽然内置了很多与浏览器交互的方法，但是有一些方面还是不能完全覆盖；有部分场景，可能用 Selenium 实现不了，这时我们可以考虑采用 JS 实现；比如操作隐藏元素，根据业务修改特定的元素值或属性

3. 使用原则：

优先使用 Selenium 原生实现，然后再考虑用 JavaScript 扩展实现

34.2. JS 同步执行

同步执行，是指在执行 Selenium 代码的时候，同时会同步执行 JS 代码，并且返回结果

1. JavaScript 中可以像 web 编程时一样，正常操作 dom
2. JavaScript 中可以返回普通结果，如数字，字符串，字典，列表
3. JavaScript 可以返回 dom 元素，会自动和 WebElement 兼容
4. JavaScript 可以传入外部参数，用 arguments 对象接收

```
# 普通脚本
script = "alert(1);"
driver.execute_script(script)
# 普通脚本
script = "let a=1;a=a+1;console.dir(a);"
driver.execute_script(script)
# 带返回的脚本
script = "let a=1;a=a+1;console.dir(a);return a+2;"
rtn = driver.execute_script(script)
# 脚本可以访问 dom 元素
script = "return document.title;"
rtn = driver.execute_script(script)
# 脚本可以设置 dom 元素
script = "document.title=arguments[0];"
rtn = driver.execute_script(script, '我想修改一下标题')
```

34.3. JS 异步等待

异步等待是指 Selenium 在执行的时候，需要异步显式等待某一个执行结果

关于 `execute_async_script` 函数，同学们在网上或者官网，你是很难找相关说明的

1. `callback` 函数一定要调用，否则我们这段代码会执行超时(特别注意)
2. `callback` 函数中可以传参，并作为 `execute_async_script` 的返回结果
3. 一般我们用于异步 js，我们可以确保后续 selenium 代码是在 ajax 请求之后(比如 ajax)

```
from time import sleep
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("http://www.baidu.com")
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(5)

script = """
    var callback = arguments[arguments.length - 1];
    window.setTimeout(function(){callback('timeout')}, 3000);
"""

driver.execute_async_script(script, '测试一下异步等待,这行 js 会执行 3 秒，大家仔细体会一下')
sleep(5)
driver.quit()
```

34.4. JS 之化繁为简

我们看这么一个场景，携程官网选日期

<https://flights.ctrip.com/online/channel/domestic>

目的地
东京(TYO)

出发日期
2022-08-05 后天

4天

返回日期
2022-08-08 周一

2022年 8月

2022年 9月

日 一 二 三 四 五 六

1 2 今天 3 4 去程 5 6

7 返程 8 9 10 11 12 13

4 5 6 7 8 9 10 教师节

比如说我们测试的场景是选择今天到后天这一范围，如果用正常的 UI 效率去做，大家会发现，非常繁琐
我们发现用 js 去做，非常方便，具体参考源码

34.5. JS 实现 H5 拖曳

Selenium 对原生的 h5 拖曳并不支持，我们可以通过 js 进行变通实现

<https://sahitest.com/demo/dragDropDataTransfer.htm>

```
# coding:utf-8
import os
from selenium import webdriver
from selenium.webdriver import ActionChains
options = webdriver.ChromeOptions()
# 处理 SSL 证书错误问题
options.add_argument('--ignore-certificate-errors')
options.add_argument('--ignore-ssl-errors')
driver = webdriver.Chrome()
driver.get("https://sahitest.com/demo/dragDropDataTransfer.htm")
driver.maximize_window()
driver.implicitly_wait(5)
# 通过文件加载 js
tapth = os.path.dirname(__file__)+'/drag.js'
f = open(tapth, "r")
atext = f.read()
driver.execute_script(atext)
```

34.6. JS 绕过验证码

34.6.1. 处理思路

1. 场景说明:

在有些登陆场景中，存在如验证码、短信码，这时候我们的自动化登录可以受限

2. 处理思路:

前后端的会话保持，在浏览器中，一般采用下面三种技术

`window.sessionStorage`: 将会话信息存在 `sessionStorage` 中

`window.localStorage`: 将会话信息存在 `localStorage` 中

`cookie`: 将会话信息存在 `cookie` 中

如果我们拿到会话信息，直接添加到 `selenium` 中，那么我们可以跳过验证

3. 对于 `window.sessionStorage`, `window.localStorage`, `cookie`, 我们在 `js` 中均可以操作

4. 我们以开发平台为例，进行实例讲解

A: 分析出会话信息存储在浏览器中的哪部分

B: 首先进行人工登录，生成会话信息

C: 将会话信息设置到 `selenium` 中

D: 正常操作自动化测试

34.6.2. sessionStorage

我们可以采用 javascript 操作 sessionStorage 对象

```
scripts = """
    let sessionStorage=window.sessionStorage;
    sessionStorage.setItem('devt-service_userid', 'admin');
    sessionStorage.setItem('devt_type', '0');
    """
sessionStorage = driver.execute_script(scripts)
```

34.6.3. localStorage

我们可以采用 javascript 操作 localStorage 对象

```
scripts = """
    let localStorage=window.localStorage;
    localStorage.setItem('devt-service_userid', 'admin');
    localStorage.setItem('devt_type', '0');
    """
sessionStorage = driver.execute_script(scripts)
```

34.6.4. Cookie

我们可以采用 python 或者 java 操作 cookie 对象

```
driver.add_cookie({"name": "longge_1", "value": "longge_value"})
driver.add_cookie({"name": "longge_2", "value": "longge_value2"})
```

34.7. JS 深入思考

在 Selenium 中，嵌入式 JavaScript 能做哪些事情？

1. 元素定位器的一种
2. 可以用于简化流程，这个要看具体场景而定，一般要求对前台数据流有一定的了解
3. 补充实现 selenium 实现不了的场景(h5 拖曳，会话处理……)
4. 在 dom 这一个范围，没有 js 实现不了需求

JavaScript 在我们前台，属于一门非常重要的语言，如果说你做自动化测试，js 和 css 是必修课
做自动化测试的人员，如果再懂 js，会让你写测试代码的时候轻松不少

35. 与 unittest 集成

35.1. ut 集成概述

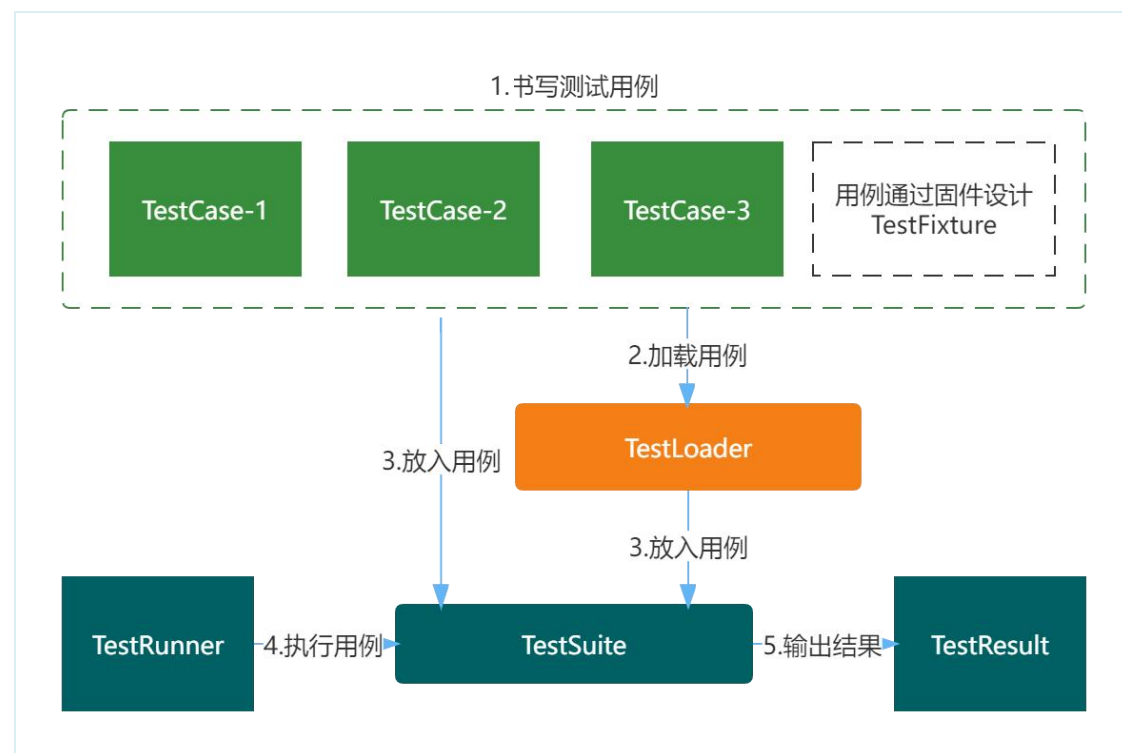
35.1.1. unittest 简介

unittest 是一个 python 内部自带的单元测试框架；

unittest 具备完整的测试结构，支持自动化测试的执行，对测试用例集进行组织，并且提供了丰富的断言方法，最后生成测试报告；

unittest 框架的初衷是用于单元测试，但不限于此，在实际工作中，由于它提供了完整的测试流程，我们也可将它用于 ui 自动化；

关于 unittest 完整教程，大家可以参考龙哥前面介绍的课程



35.1.2. 什么要集成

这时候大家回过头来看一下我们的前面课程的代码，不难发现有如下问题？

- 1、每一个应用例我们写一个模块和入口函数，里面的业务流程，断言逻辑，测试结果需要全部手写，代码复用性非常低
- 2、如何我想批量执行所用例，或者选取一部分场景用例运行，我们该如何操作？
- 3、测试报告的输出，没有统一标准

35.2. 用 ut 设计流程

以下为我们演示场景用例，采用 `unittest` 进行设计用例

序号	测试用例	步骤
1	<hao123>搜索	开打浏览器，搜索<hao123>，断言第一行搜索结果为 hao123 官网，退出浏览器
2	<腾讯视频>搜索	开打浏览器，搜索<腾讯视频>，断言第一行搜索结果为腾讯视频官网，退出浏览器
3	<新浪>搜索	开打浏览器，搜索<新浪>，断言第一行搜索结果为新浪官网，退出浏览器
4	<aqy>搜索	开打浏览器，搜索<aqy>，断言第一行搜索结果为爱奇艺官网，退出浏览器
5	<selenium>搜索	开打浏览器，搜索<selenium>，断言第一行搜索结果为 selenium 官网，退出浏览器
6	<12306>搜索	开打浏览器，搜索<12306>，断言第一行搜索结果为 12306 官网，退出浏览器

从上面的流程，我们不难分析出公用操作：打开浏览器，退出浏览器

因此我们很容易想到如下实现思路：

```
def setUp(self):
    打开浏览器
def tearDown(self):
    退出浏览器
def test_01_hao123:
    业务逻辑 1
def test_02_txsp:
    业务逻辑 2
.....
```

具体实现参考源码工程

35.3. 用 ut 分组用例

为什么分组？

一个功能的验证往往需要多个测试用例，需要把多个测试用例集合在一起执行

不同的用例分组往往代表了不同的功能场景，或者表了用例的有一定的关联性，通过分组，我们可以分功能、分场景针对性进行测试，更快地发现问题，降低测试成本

序号	场景分组	测试用例
1	国内	<hao123>搜索
		<腾讯视频>搜索
		<新浪>搜索
		<aqy>搜索
		<12306>搜索
2	国外	<selenium>搜索
3	中文类	<腾讯视频>搜索
		<新浪>搜索
4	数字类	<12306>搜索
		<hao123>搜索
5	英文类	<aqy>搜索
		<selenium>搜索

按场景组织后，我们整个项目的测试用例代码参考源码工程

35.4. ddt 数据驱动

如果我们需要验证的关键字几百上千的时候，我们用上面的设计方法明显不可取，我这里我们可以参考数据驱动的设计思想
数据驱动使用原则：

1. 数据驱动一般用于同一用例场景的不同数据
2. 将输出数据和输出数据/断言数据进行参数化处理，比如上一堂课中的搜索关键字、预期地址

序号	测试用例	驱动数据(输入)	断言数据(输出)
1	<hao123>搜索	hao123	https://www.hao123.com/
2	<腾讯视频>搜索	腾讯视频	https://v.qq.com/
3	<新浪>搜索	新浪	https://www.sina.com.cn/
4	<aqy>搜索	aqy	http://www.iqiyi.com/
5	<selenium>搜索	Selenium	https://www.selenium.dev/
6	<12306>搜索	12306	https://www.12306.cn/index/
.....

35.5. ut 收集用例结果

unittest+selenium 收集用例结果与单独采用 unittest 无任何区别

```
if __name__ == '__main__':  
    loader = unittest.TestLoader()  
    loader.loadTestsFromTestCase(  
        Test1HTMLTestRunner)  
  
    f = open(os.path.dirname(__file__)+'/test.html', 'wb')  
    runner = HTMLTestRunner.HTMLTestRunner(  
        f, title='演示 HTML', description='描述', verbosity=1)  
    unittest.main(testRunner=runner, testLoader=loader)
```

36. 与 pytest 集成

36.1. pyt 集成概述

Pytest 是 Python 语言下的一种单元测试框架，与 Python 自带的 unittest 测试框架类似，但是比 unittest 框架使用起来更简洁，效率更高。从目前使用率上来讲，pytest 比 unittest 份额更高。

如果没有说明的话，我们后面均采用 pytest+selenium 进行讲解。

36.2. pyt 固件设计流程

以下为我们演示场景用例，采用 `pytest` 进行设计用例

序号	测试用例	步骤
1	<hao123>搜索	开打浏览器，搜索<hao123>，断言第一行搜索结果为 hao123 官网，退出浏览器
2	<腾讯视频>搜索	开打浏览器，搜索<腾讯视频>，断言第一行搜索结果为腾讯视频官网，退出浏览器
3	<新浪>搜索	开打浏览器，搜索<新浪>，断言第一行搜索结果为新浪官网，退出浏览器
4	<aqy>搜索	开打浏览器，搜索<aqy>，断言第一行搜索结果为爱奇艺官网，退出浏览器
5	<selenium>搜索	开打浏览器，搜索<selenium>，断言第一行搜索结果为 selenium 官网，退出浏览器
6	<12306>搜索	开打浏览器，搜索<12306>，断言第一行搜索结果为 12306 官网，退出浏览器

从上面的流程，我们不难分析出公用操作：打开浏览器，退出浏览器

因此我们很容易想到如下实现思路：

```
def setup_method(slef):
    打开浏览器
def teardown_method(slef):
    退出浏览器
def test_01_hao123:
    业务逻辑 1
def test_02_txsp:
    业务逻辑 2
.....
```

具体实现参考源码工程

36.3. fixture 设计流程

以下为我们演示场景用例，采用 unittest 进行设计用例

序号	测试用例	步骤
1	<hao123>搜索	开打浏览器，搜索<hao123>，断言第一行搜索结果为 hao123 官网，退出浏览器
2	<腾讯视频>搜索	开打浏览器，搜索<腾讯视频>，断言第一行搜索结果为腾讯视频官网，退出浏览器
3	<新浪>搜索	开打浏览器，搜索<新浪>，断言第一行搜索结果为新浪官网，退出浏览器
4	<aqy>搜索	开打浏览器，搜索<aqy>，断言第一行搜索结果为爱奇艺官网，退出浏览器
5	<selenium>搜索	开打浏览器，搜索<selenium>，断言第一行搜索结果为 selenium 官网，退出浏览器
6	<12306>搜索	开打浏览器，搜索<12306>，断言第一行搜索结果为 12306 官网，退出浏览器

从上面的流程，我们不难分析出公用操作：打开浏览器，退出浏览器

因此我们很容易想到如下实现思路：

```
def fixture1():
    打开浏览器
    yield
    退出浏览器

@pytest.mark.useFixtures('fixture1')
def test_01_hao123:
    业务逻辑 1
def test_02_txsp:
    业务逻辑 2
.....
```

具体实现参考源码工程

36.4. pyt 分组用例

为什么分组？

一个功能的验证往往需要多个测试用例，需要把多个测试用例集合在一起执行

不同的用例分组往往代表了不同的功能场景，或者表了用例的有一定的关联性，通过分组，我们可以分功能、分场景针对性进行测试，更快地发现问题，降低测试成本

序号	场景分组	测试用例
1	国内	<hao123>搜索
		<腾讯视频>搜索
		<新浪>搜索
		<aqy>搜索
		<12306>搜索
2	国外	<selenium>搜索
3	中文类	<腾讯视频>搜索
		<新浪>搜索
4	数字类	<12306>搜索
		<hao123>搜索
5	英文类	<aqy>搜索
		<selenium>搜索

按场景组织后，我们整个项目的测试用例代码参考源码工程

36.5. pyt 参数化设计

如果我们需要验证的关键字几百上千的时候，我们用上面的设计方法明显不可取，我这里我们可以参考数据驱动的设计思想
数据驱动使用原则：

1. 数据驱动一般用于同一用例场景的不同数据
2. 将输出数据和输出数据/断言数据进行参数化处理，比如上一堂课中的搜索关键字、预期地址

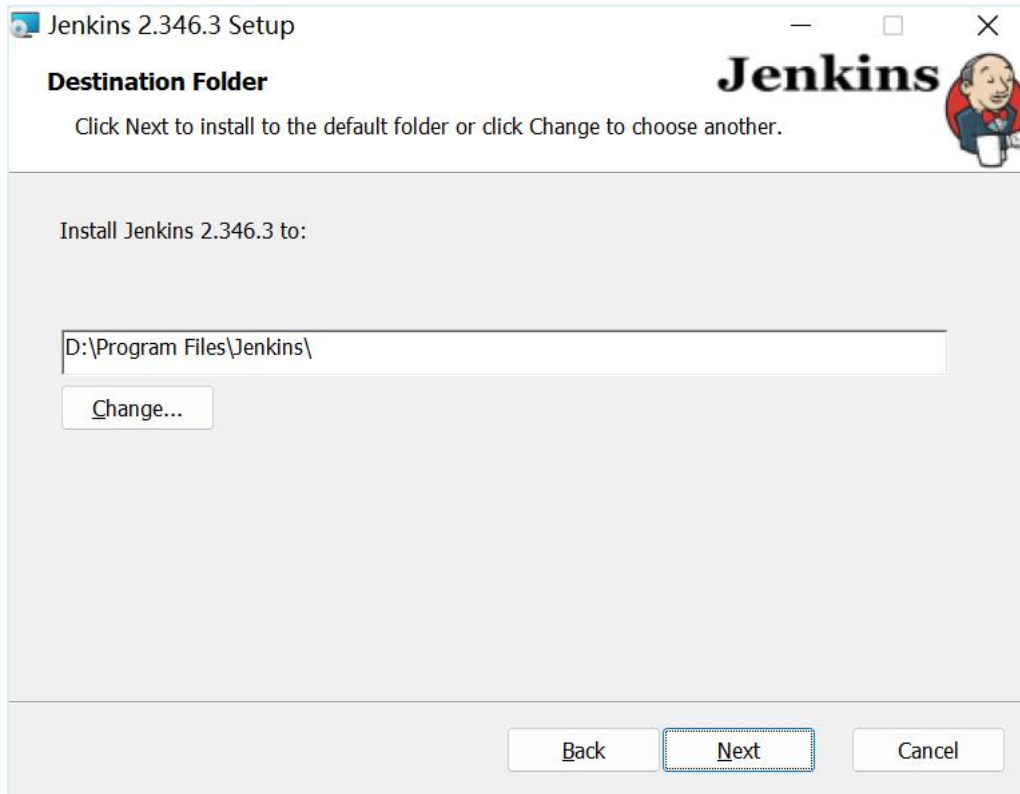
序号	测试用例	驱动数据(输入)	断言数据(输出)
1	<hao123>搜索	hao123	https://www.hao123.com/
2	<腾讯视频>搜索	腾讯视频	https://v.qq.com/
3	<新浪>搜索	新浪	https://www.sina.com.cn/
4	<aqy>搜索	aqy	http://www.iqiyi.com/
5	<selenium>搜索	Selenium	https://www.selenium.dev/
6	<12306>搜索	12306	https://www.12306.cn/index/
.....

36.6. pyt 收集用例结果

采用 allure-pytest 收集报告

36.7. 与jenkins 集成


36.7.1. 软件安装



Jenkins 2.346.3 Setup

Service Logon Credentials

Enter service credentials for the service.

Jenkins 

Jenkins 2.346.3 installs and runs as an independent Windows service. To operate in this manner, you must supply the user account credentials for Jenkins 2.346.3 to run successfully.

Logon Type:

☒ Run service as LocalSystem (not recommended)

☐ Run service as local or domain user:

Account:

Password:

Test Credentials

Back Next Cancel

Jenkins 2.346.3 Setup


Port Selection

Choose a port for the service.

Please choose a port.

Port Number (1-65535):

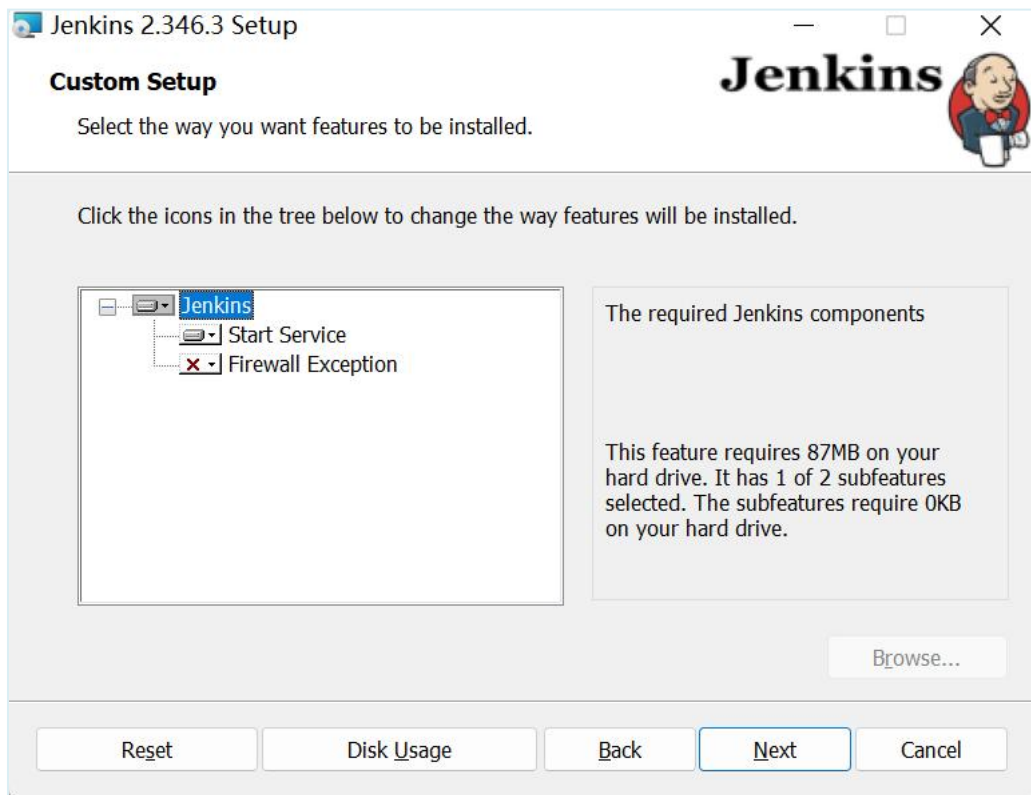
8080

Test Port 

It is recommended that you accept the selected default port.

Back Next Cancel





36.7.2. 初始化

1. 键入 `http://127.0.0.1:8080`，从下面文件中复制密码登陆

入门

解锁 Jenkins

为了确保管理员安全地安装 Jenkins，密码已写入到日志中（[不知道在哪里?](#)）该文件在服务器上：

```
C:\ProgramData\Jenkins\.jenkins\secrets\initialAdminPassword
```

请从本地复制密码并粘贴到下面。

管理员密码

2. 创建管理员密码

新手入门

创建第一个管理员用户

Username:

Password:

Confirm password:

Full name:

实例配置

Jenkins URL:

http://127.0.0.1:8080/

Jenkins URL 用于给各种Jenkins资源提供绝对路径链接的根地址。这意味着对于很多Jenkins特色是需要正确设置的，例如：邮件通知、PR状态更新以及提供给构建步骤的BUILD_URL环境变量。

推荐的默认值显示在**尚未保存**，如果可能的话这是根据当前请求生成的。最佳实践是要设置这个值，用户可能会需要用到。这将会避免在分享或者查看链接时的困惑。

36.7.3. 中文插件

+ New Item

👤 People

📅 Build History

⚙️ Manage Jenkins

1

👤 My Views

📁 New View

Build Queue

▼

No builds in the queue.

Build Executor Status

▼

1 Idle

Manage Jenkins

Building on the built-in node can be a security issue. You should set up distributed builds. See [the documentation](#).

Set up agentSet up cloudDismiss

You are running Jenkins on Java 1.8, support for which will end on or after September 1, 2022. Please refer to [the documentation](#) for details on upgrading to Java 11.

More InfoDismiss

System Configuration

⚙️ Configure System

Configure global settings and paths.

🔧 Global Tool Configuration

Configure tools, their locations and automatic installers.

⚙️ Manage Plugins

2

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

☁️ Manage Nodes and Clouds

Plugin Manager

Updates

Available

Installed

Advanced

🔍 中文

✕

Install

Name ↓

Localization: Chinese (Simplified)

1.0.24

☐

localization

Jenkins Core 及其插件的简体中文语言包, 由 [Jenkins 中文社区](#) 维护。

Install without restart

Download now and install after restart

Update information obtained: 1 hr 15 min ago

Check now

36.7.4. allure 插件

Plugin Manager

可更新

可选插件

已安装

高级

Q

allure

Install

Name ↓

Allure 2.30.2

☒

Build Reports

This plugin integrates Allure reporting tool into Jenkins.

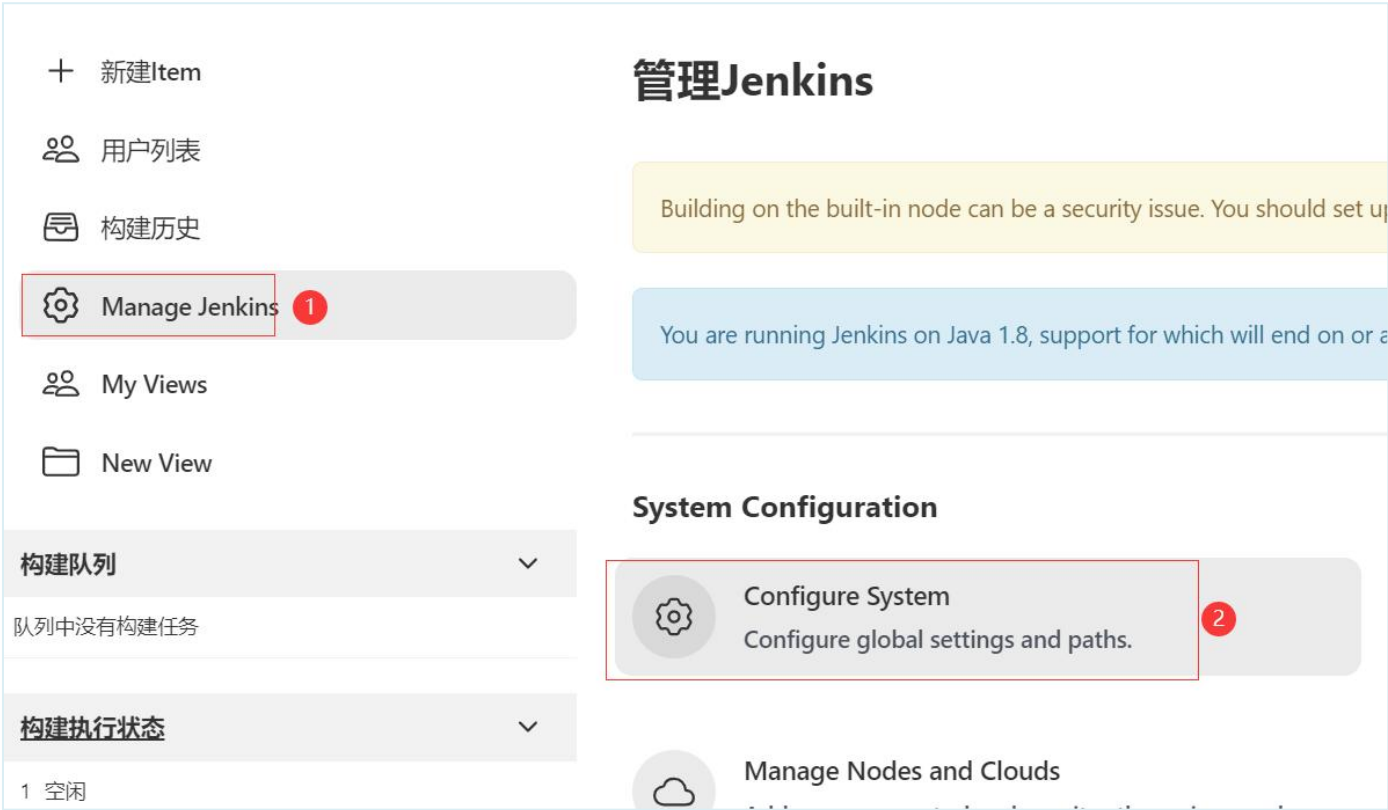
Install without restart

Download now and install after restart

1 小时 25 分 之前获取了更新信息

立即获取

36.7.5. 环境变量



全局属性

☒ Environment variables

键值对列表 ?

键

path

值

C:\Users\xiang\AppData\Local\Google\Chrome\Application;D:\python310\;D:\python310\Scripts\

浏览器路径和python环境变量

新增

☐ Tool Locations

36.7.6. allure 目录

+ 新建Item

👤 用户列表

📅 构建历史

⚙️ Manage Jenkins

1

👤 My Views

📁 New View

构建队列

▼

队列中没有构建任务

构建执行状态

▼

1 空闲

2 空闲

管理Jenkins

Building on the built-in node can be a security issue. You should set up distributed builds. See [the documentation](#).

You are running Jenkins on Java 1.8, support for which will end on or after September 1, 2022. Please refer to [the documentation](#) for details on

System Configuration

⚙️

Configure System

Configure global settings and paths.

🔧

Global Tool Configuration

Configure tools, their locations and automatic installers.

2

☁️

Manage Nodes and Clouds

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Allure Commandline

Allure Commandline 安装

系统下Allure Commandline 安装列表

新增 Allure Commandline

≡ Allure Commandline

×

别名

allure-home

安装目录

D:\utils\allure-2.18.1

allure安装目录

☐ Install automatically ?

新增 Allure Commandline


保存

应用

36.7.7. 新建任务

输入一个任务名称

» 必填项



Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

确定

- ☐ Discard old builds ?
- ☐ This project is parameterized ?
- ☐ 关闭构建 ?
- ☐ 在必要的时候并发构建 ?
- ☐ 静默期 ?
- ☐ 重试次数 ?
- ☐ 该项目的上游项目正在构建时阻止该项目构建 ?
- ☐ 该项目的下游项目正在构建时阻止该项目构建 ?
- ☒ 使用自定义的工作空间 ?

目录

D:\utils\allure-2.18.1\space

该目录用于保存allure报告

36.7.8. pytest 构建命令

pytest --clean-alluredir --alluredir=allure-result E:/pythonsource/selenium/part36/test6

构建

≡ Execute Windows batch command ?

命令

参阅 [可用环境变量列表](#)

```
pytest --clean-alluredir --alluredir=allure-result E:/pythonsource/selenium/part36/test6
```

高级...

增加构建步骤 ▼

36.7.9. allure 报告

该目录与上一节课中的的目录必须相同

构建后操作

≡ Allure Report

☐ Disabled

Results:
Paths to Allure results directories relative from workspace.
E.g. **target/allure-results**.

Path

新增

36.7.10. 查看报告

37. pom 设计分层

37.1. pom 模式简介

37.1.1. 什么是 pom 模式

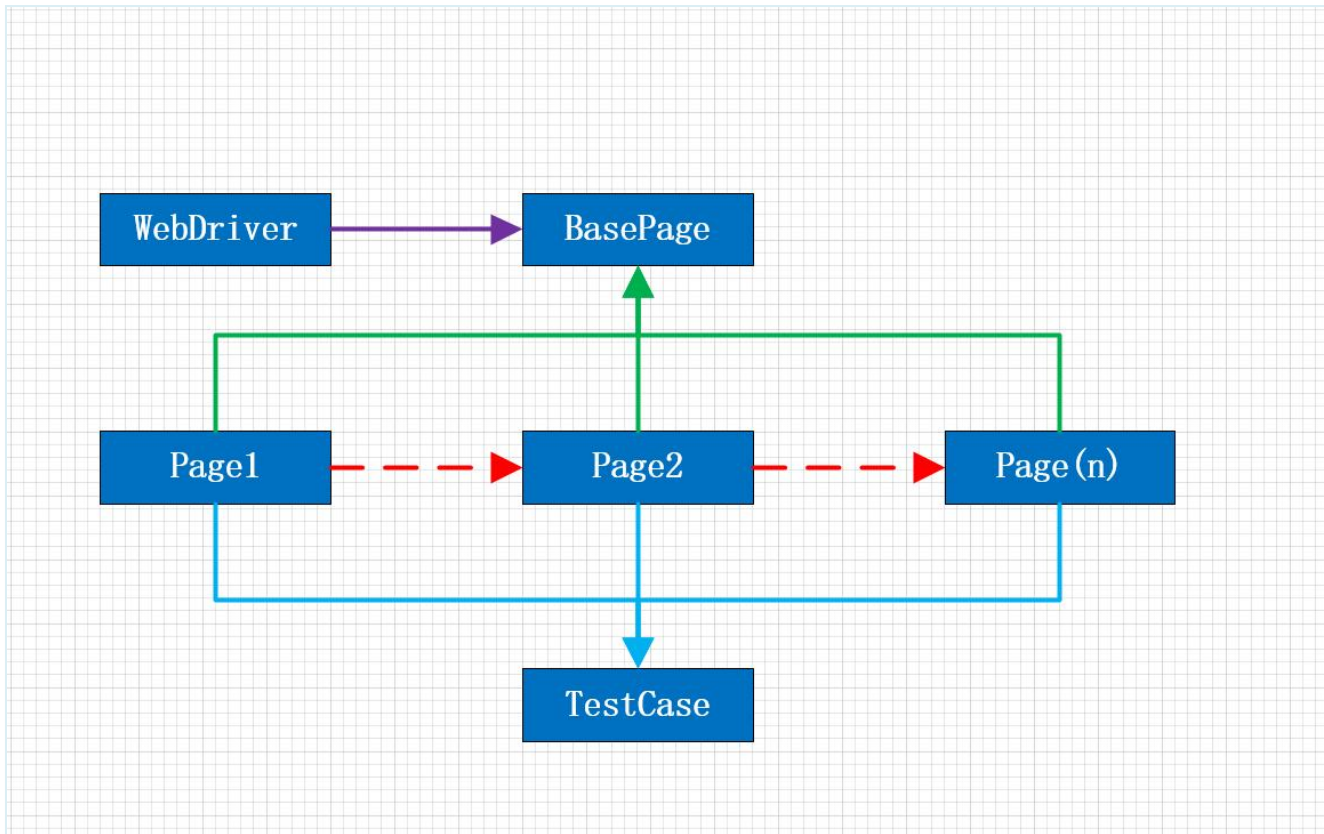
pom 全称：Page Object Model，可理解为每一个页面都应有自己的对象模型，通常也称为：页面驱动测试

这也是是目前在自动化测试中，非常经典的一种设计思想，简单来说：将页面 UI 元素对象、逻辑、业务、数据等分离开来，使得代码逻辑更加清晰，复用性，可维护性更高的一种方法

37.1.2. pom 模式特征

具有明显的分层特征

1. 基础层：又叫 base 层，主要封装 selenium 基础的原生方法、框架方法
2. 页面对象层：又叫 page 层，主要是用于存放页面的元素和动作
3. 测试用例层：又叫 testcase 层，主要试用例场景



37.1.3. pom 模式优势

传统的线性脚本，一般会碰到如下问题：

1. 易用性差：杂乱无章的定位元素方法，例如：find_element
2. 扩展性不好：用例孤立，无法扩展
3. 复用性差：无公共方法，很难复用
4. 可维护性差：一旦元素变化或测试步骤变化，需要维护大量代码和用例

pom 模式的优点

1. 页面元素定位和业务操作方法分离，使得代码更加清晰，减少冗余代码
2. 测试方法单独剥离，这样提高用例的可读性
3. 针对 ui 变化频繁的项目和测试步骤的变化，提高了测试用例的维护性

主要解决解决线性脚本问题，如代码很能重复利用、后期的维护困难问题

37.2. pom 模式过程

37.2.1. 封装基础方法

包括但不限于：

1. 封装 webdriver 基本方法

webdriver 对象的创建

页面打开方法

超链接点击方法

元素定位方式

文本输入方法

对象关闭方法

窗口切换方法

键盘输入方法

.....

2. 封装其它框架方法

文件数据的加载

日志统一记录

参数化方法

.....

37.2.2. 封装页面元素

内容主要包括：

1. 初始化页面元素相关属性值、业务元素定位方式
用户名输入元素/定位
提交按钮/定位
.....
2. 封装针对该元素的一些基础动作、处理逻辑、业务逻辑
输入元素
选择下拉库
点击提交按钮
.....
3. 封装需要断言判断需要的值，或者是其它业务值
.....

37.2.3. 设计测试用例

这一层主要是输入业务数据，完成场景下的测试用例

1、通常我们会将数据和业务分离，比如采用 unittest 中的 ddt 或者是 pytest 中的 parametrize

简单内部数据输入

读取外部文件数据

.....

2、调用封装页面元素层的方法，完成测试用例的设计

步骤拆分

调用业务元素的基础动作、处理逻辑、业务逻辑

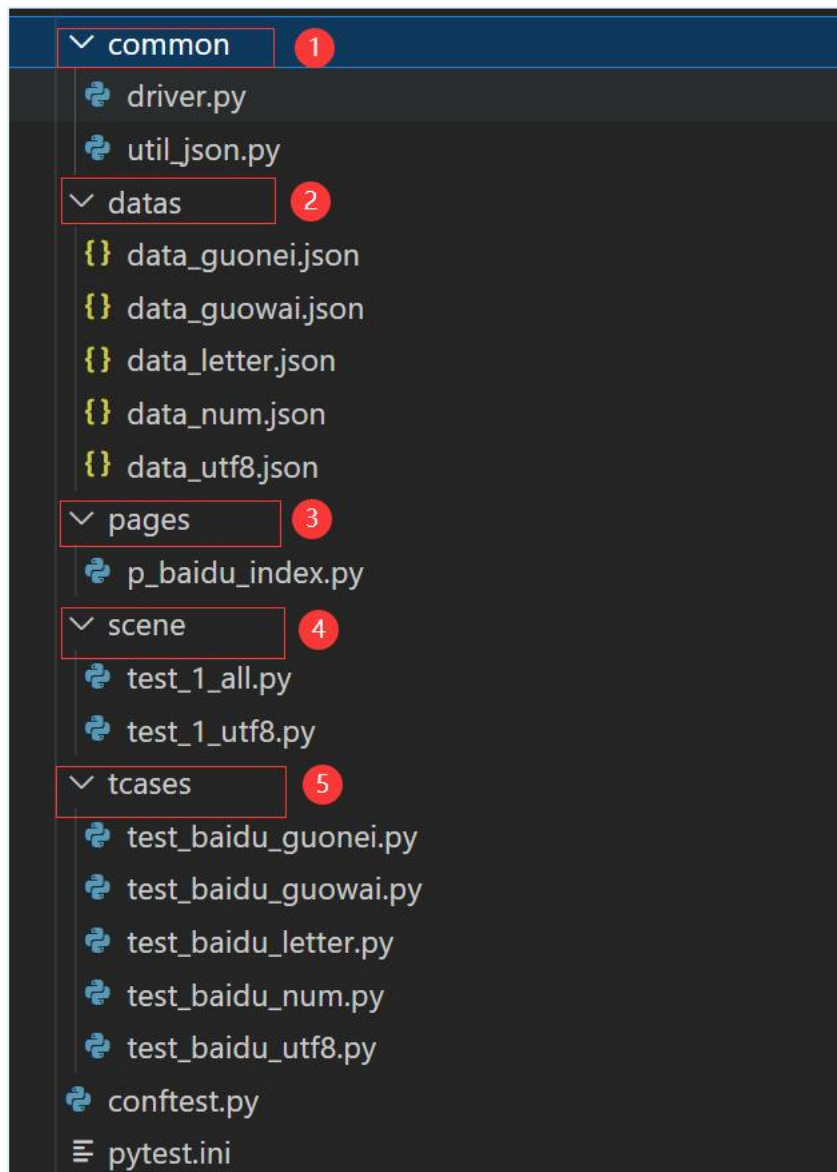
断言结果

.....

37.2.4. 用例场景分组

对不同的测试用例按场景进行分组

37.3. 百度演示案例

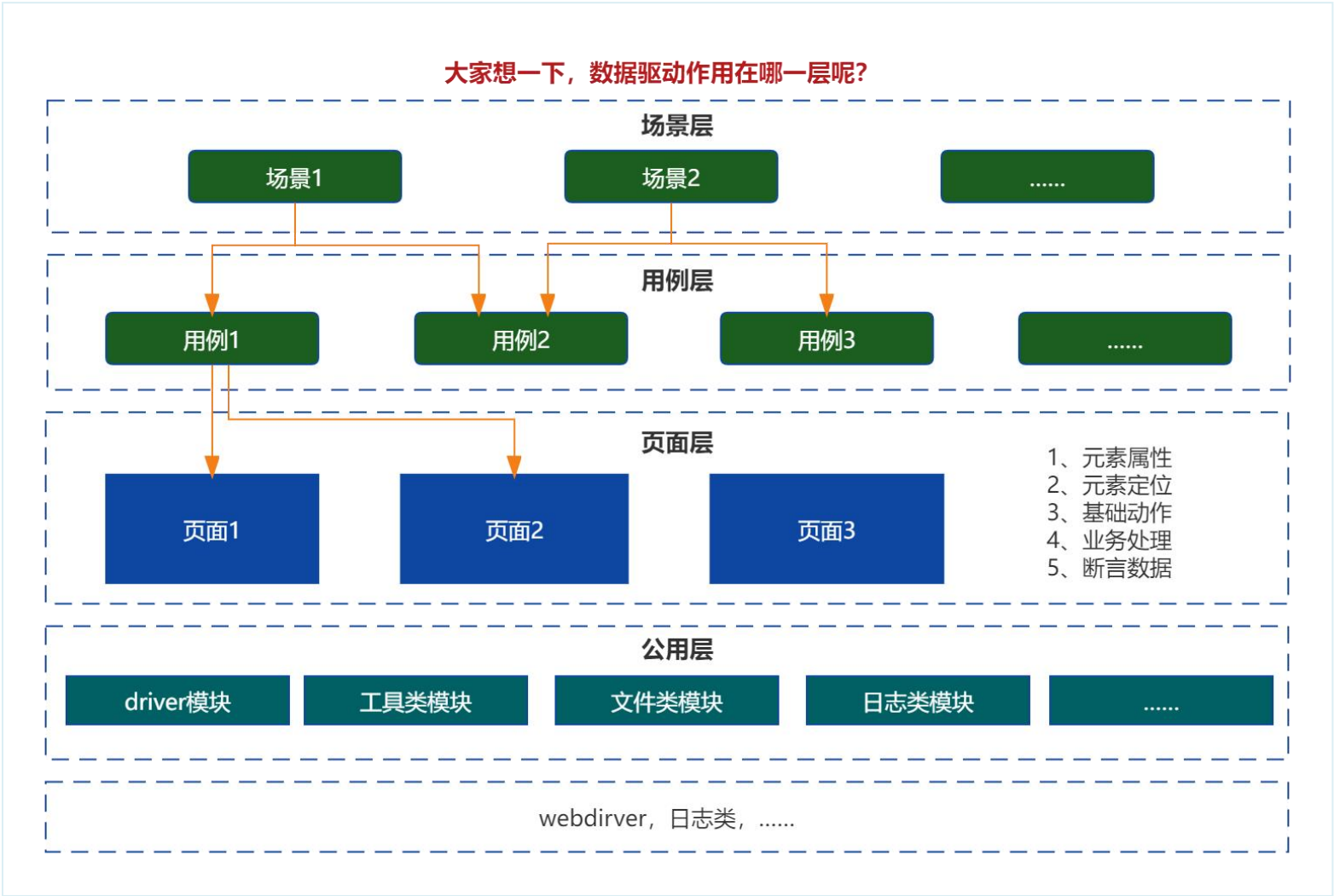


参考百度演示网址，进行 pom 分层设计，如下：

- 1、通用层：webdriver 封装，工具类……
- 2、数据层：json, yaml, csv……
- 3、页面层：元素属性，元素定位……
- 4、场景层：不同用例组合场景
- 5、用例层：测试用例

具体参考源码演示说明

37.4. pom 模式架构



38. 关键字驱动设计

38.1. 关键字驱动简介

38.1.1. 什么是关键字驱动

关键字驱动表示把项目中的一些逻辑/步骤封装成关键字(理解成一个函数或者方法)。

这样达到一个测试用例和测试步骤分离的效果

不同的测试用例则通过不同的关键字步骤进行组合

38.1.2. 关键字&数据驱动

	关键字驱动	数据驱动
本质目标	测试用例和测试步骤分离(过程抽象)	测试用例与测试数据分离
作用对象	测试用例整个过程	测试用例之上
设计过程	逻辑抽象为一个关键字	测试数据提炼

38.2. 常用设计方法

38.2.1. 用例步骤拆分

	序号	测试步骤	其它
百度搜索案例	1	打开浏览器
	2	输入百度地址
	3	搜索框输入文本
	4	点击搜索按钮
	5	点击链接
	6	检查结果

38.2.2. 对象提取分离

	序号	测试步骤	关键字/函数	操作元素/对象	操作值/数据/动作	
百度搜索案例	1	打开浏览器	----		谷歌浏览器
	2	输入百度地址	open_url		www.baidu.com
	3	搜索框输入文本	input_kw	id=kw	腾讯视频
	4	点击搜索按钮	click_su	id=su	click
	5	点击链接	click_link_		
	6	检查结果	assert_result		

这一步需要将对象进行提取分离，主要包括：

- 1、测试元素，如文本框、按钮
- 2、测试数据，如上面的百度地址，业务关键字等
- 3、测试动作，如点击按钮，输入文本框
- 4、测试逻辑，断言逻辑，业务逻辑

38.3. 百度演示案例

```
class TestBaiduIndex():
    const_url = 'http://www.baidu.com'
    const_input = '测试输入'
    const_json = 'datas/data_guowai.json'

    def setup(self):
        self.baidu_index = BaiduIndex()

    @pytest.mark.guowai
    @pytest.mark.parametrize('param', readJson(const_json))
    def test_02_guowai(self, param):
        """
            国外网址测试
        """
        self.baidu_index.open_url(TestBaiduIndex.const_url)
        self.baidu_index.input_kw(param['kw'])
        self.baidu_index.click_su()
        self.baidu_index.click_link()
        assert self.baidu_index.get_current_url() == param['check']
        sleep(5)

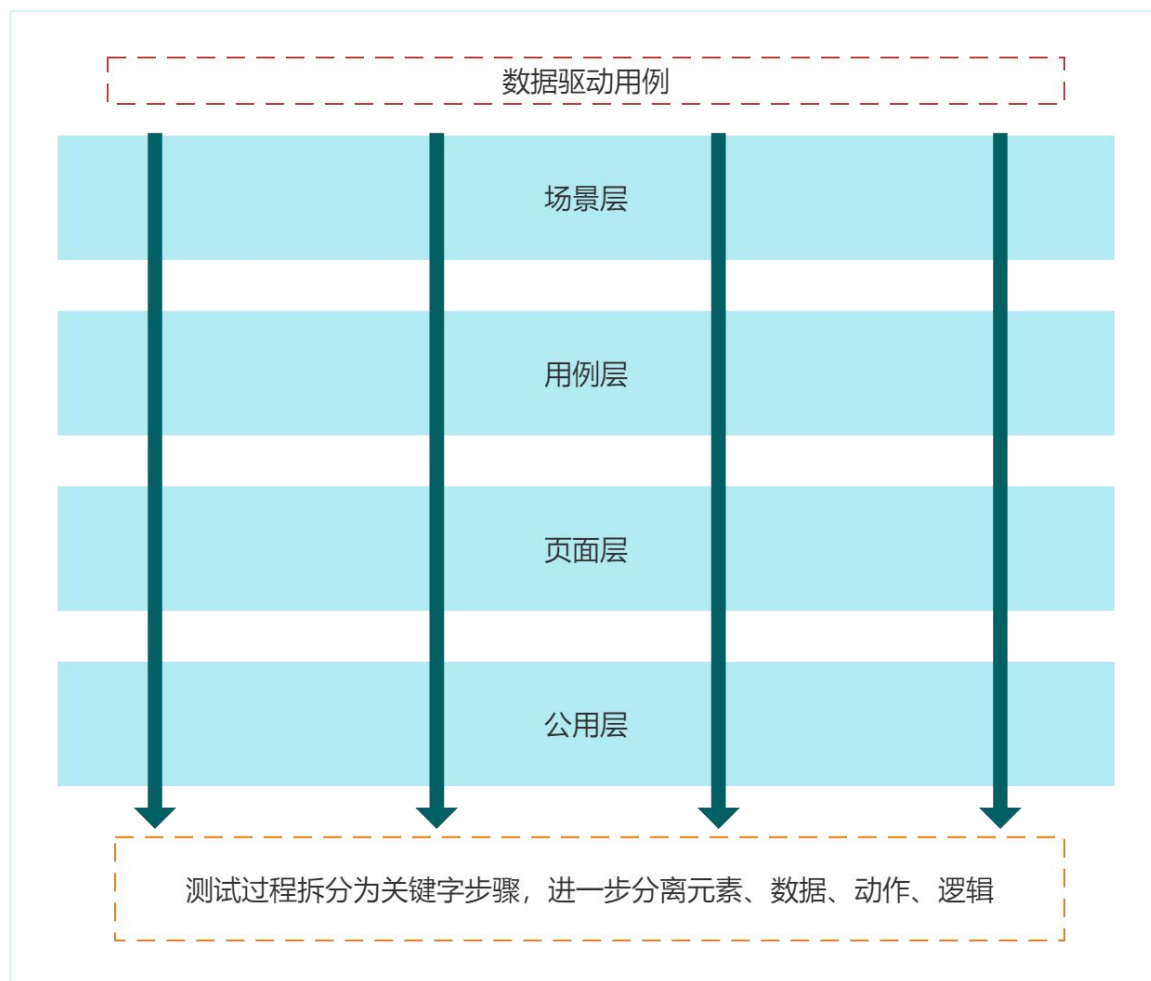
    def teardown(self):
        self.baidu_index.driver.quit()
```

#这里我们可以理解为测试步骤 2
#这里我们可以理解为测试步骤 3
#这里我们可以理解为测试步骤 4
#这里我们可以理解为测试步骤 5

38.4. 深入三种模式

pom 分层架构、数据驱动、关键字驱动，它们都是一种设计思想

同设计开发领域周期中的设计理念相比(项目分包，工程分层，配置文件，面向过程，面向对象，面向切面)，两者有异曲同工之妙



1. pom 设计思想是一种最顶层设计，决定整个工程代码架构
2. 数据驱动是一种面向用例的设计，将数据与用例进行剥离
3. 关键字驱动一种面向过程设计，将业务逻辑进行解耦

我们在书写测试代码的时候，往往需要考虑 pom 分层架构、数据驱动、关键字驱动多种设计理念

39. 开发平台综合案例

39.1. 需求场景说明

序号	测试用例	用例说明	其它
1	登录用例	多账号模拟登录
2	组件列表	查询组件列表
3	新增组件	新增组件
5	组件删除	删除组件

39.2. 测试用例设计

序号	测试用例	步骤拆分	其它说明
1	登录用例	打开界面	无依赖
		输入用户名、密码	
		登录系统	
		断言结果	
2	组件列表	输入查询条件	依赖登录用例
		查询列表	
		断言结果	
3	新增组件	输入数据	依赖登录用例、组件列表、组件删除
		新增结果	
		结果断言	
5	组件删除	选择组件	依赖登录用例、组件列表、组件新增
		删除组件	
		断言结果	

39.3. 基础分层架构

采用页面驱动测试，简化代码，提高可读性，可维护性

一般采用 Base 层、Page 层、TestCase 层、场景层、数据层等分层架构，

39.4. 数据驱动思考

采数据驱动测试，分离数据与测试用例

39.5. 关键字驱动

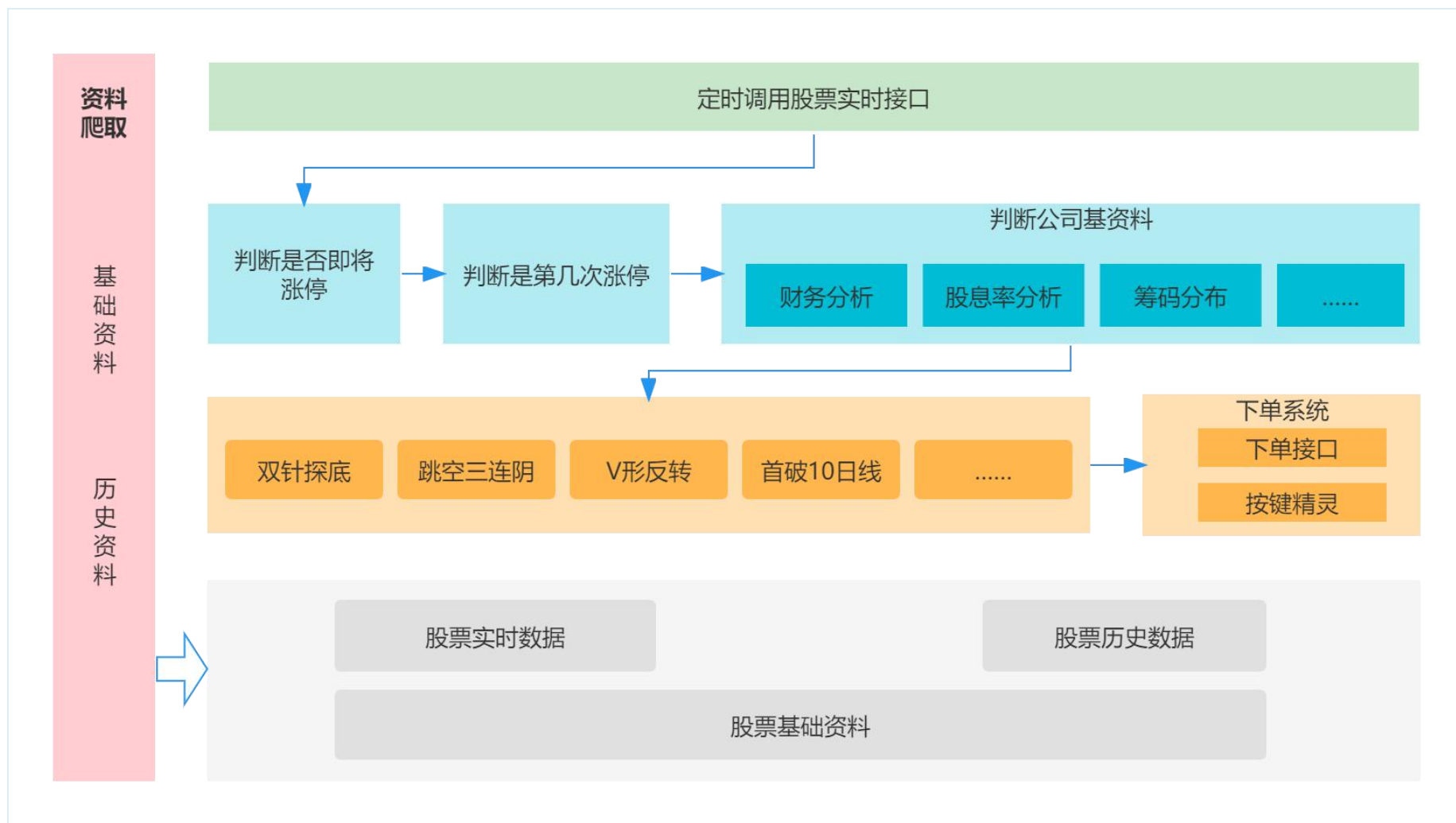
面向过程采用关键字驱动思维，分离测试用例和步骤

39.6. 完整测试项目

参考源码

40. 股票项目综合案例

40.1. 需求场景说明



40.2. 网络爬虫简介

1. 网络爬虫原理

网络爬虫指按照一定的规则（模拟人工登录网页的方式），自动抓取网络上的数据。通俗地说，就是将你上网所看到页面上的有价值的内容获取下来，并进行存储

2. 网络爬虫有何作用？

A、有时候我们需要大量的数据，比如作数据分析和挖掘，如果手工造数据，缺乏真实性，而且难度也很大，比如想获取 1000 万张图片

B、互联网中的数据量大，如果我们人工的去收集数据，这样会很浪费时间与金钱

C、而爬虫有个特点就是能批量、自动化的获取和处理数据

3. 网络爬虫的流程

选取 url->模拟浏览器请求->解析网页、数据->存储数据

40.3. 完整测试项目

参考网地址：http://quote.eastmoney.com/center/gridlist.html#hs_a_board，具体参考源码工程

热门实时资金流向 | 主力排名 | 板块资金 | 行业资金流向 | 概念资金流向 | 地域资金流向 | 资金流监测

自选指标: 市净率

序号	代码	名称	相关链接	最新价	涨跌幅	涨跌额	成交量(手)	成交额	振幅	最高	最低	今开	昨收	量比	换手率	市盈率(动态)	市净率	加自选
1	301270	N汉仪	股吧 资金流 数据	39.51	53.86%	13.83	12.21万	4.93亿	33.26%	45.05	36.51	38.01	25.68	-	64.09%	75.65	3.72	+
2	001259	N利仁	股吧 资金流 数据	28.44	44.00%	8.69	6654	1883.32万	24.00%	28.44	23.70	23.70	19.75	-	3.60%	28.87	2.89	+
3	603237	N五芳斋	股吧 资金流 数据	49.42	44.00%	15.10	1.32万	6485.94万	24.01%	49.42	41.18	41.18	34.32	-	5.23%	10.27	2.69	+
4	300336	*ST新文	股吧 资金流 数据	2.15	20.11%	0.36	94.70万	1.92亿	22.35%	2.15	1.75	1.76	1.79	2.93	12.84%	-17.00	-3.89	+
5	300149	睿智医药	股吧 资金流 数据	12.90	20.00%	2.15	33.23万	4.03亿	20.47%	12.90	10.70	10.75	10.75	5.07	6.66%	3.37	2.20	+
6	301025	读客文化	股吧 资金流 数据	13.99	19.98%	2.33	27.88万	3.81亿	22.04%	13.99	11.42	11.50	11.66	5.31	36.54%	89.38	8.96	+
7	300950	德固特	股吧 资金流 数据	25.62	14.89%	3.32	27.64万	6.68亿	18.61%	26.27	22.12	22.12	22.30	1.55	46.06%	80.44	6.53	+
8	300857	协创数据	股吧 资金流 数据	21.40	10.59%	2.05	9.79万	2.08亿	17.00%	22.39	19.10	19.46	19.35	6.05	6.91%	31.10	3.25	+
9	000416	民生控股	股吧 资金流 数据	4.47	10.10%	0.41	76.36万	3.24亿	12.81%	4.47	3.95	3.96	4.06	0.93	14.36%	-243.48	2.71	+
10	002175	东方网络	股吧 资金流 数据	4.37	10.08%	0.40	18.05万	4.41亿	14.11%	4.37	3.81	3.90	3.97	1.02	12.97%	101.37	15.44	+
11	002343	慈文传媒	股吧 资金流 数据	6.35	10.05%	0.58	32.48万	2.02亿	10.40%	6.35	5.75	5.75	5.77	2.01	6.86%	34.67	3.22	+
12	000909	数源科技	股吧 资金流 数据	11.61	10.05%	1.06	97.80万	10.89亿	9.48%	11.61	10.61	10.61	10.55	1.09	25.24%	115.92	2.83	+
13	600683	京投发展	股吧 资金流 数据	4.82	10.05%	0.44	37.51万	1.79亿	8.68%	4.82	4.44	4.44	4.38	9.20	5.06%	10.14	1.20	+
14	002205	国统股份	股吧 资金流 数据	12.16	10.05%	1.11	46.71万	5.49亿	11.86%	12.16	10.85	11.05	11.05	1.63	25.14%	-24.02	2.73	+
15	002238	天威视讯	股吧 资金流 数据	8.11	10.04%	0.74	17.25万	1.38亿	4.61%	8.11	7.77	7.80	7.37	0.61	2.15%	59.14	2.89	+
16	002115	三维通信	股吧 资金流 数据	5.59	10.04%	0.51	21.41万	1.20亿	0.00%	5.59	5.59	5.59	5.08	0.49	3.09%	45.72	1.96	+
17	603598	引力传媒	股吧 资金流 数据	9.87	10.03%	0.90	36.28万	3.41亿	11.48%	9.87	8.84	8.94	8.97	4.43	13.55%	58.85	9.73	+
18	002112	三变科技	股吧 资金流 数据	12.18	10.03%	1.11	47.07万	5.56亿	12.01%	12.18	10.85	10.85	11.07	1.42	23.68%	101.50	5.77	+
19	002962	五方光电	股吧 资金流 数据	19.32	10.02%	1.76	84.27万	15.48亿	16.34%	19.32	16.45	17.28	17.56	1.40	49.77%	68.91	3.21	+
20	000716	黑芝麻	股吧 资金流 数据	4.94	10.02%	0.45	230.21万	10.35亿	20.04%	4.94	4.04	4.06	4.49	1.52	33.08%	271.21	1.40	+

1256

1

2

3

...

256

下一页

转到

1

GO

意见反馈

41. Selenium-ide

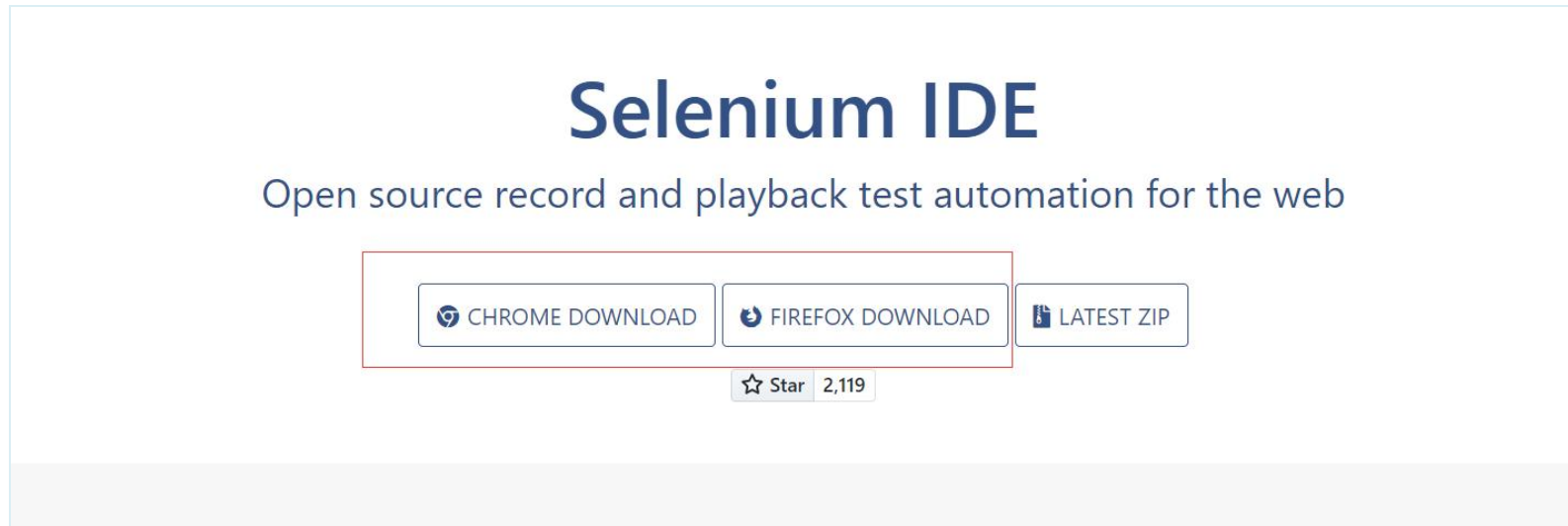
41.1. 工具简介

Selenium-IDE(集成开发环境)是一个浏览器插件,用于开发 Selenium 测试案例，它提供了图形界面，在初期学习和录制脚本方面，非常便利

41.2. 环境安装

<https://www.selenium.dev/selenium-ide/>

进入后，选择不同的浏览器进行下载(有可能网速比较慢)



41.3. 入门样例

Project: test

Test suites +

Search tests...

▼ ✓ Default Suite

✓ test_1

▶ ▶ ⌂ ⌚ ▼

https://www.baidu.com

	Command	Target	Value
1	✓ open	www.baidu.com	
2	✓ set window size	918x831	
3	✓ click	id=kw	
4	✓ type	id=kw	测试1
5	✓ click	id=su	
6	✓ click	id=kw	
7	✓ type	id=kw	春风阁讲堂
8	✓ send keys	id=kw	\${KEY_ENTER}

Command

open

//

Target

www.baidu.com

Value

Description

41.4. 流程样例

1. 条件类流程

序号	条件类	场景说明
1	if else if else end	通常结合 execute script、\${变量名}使用

2. 循环类流程

序号	times 循环	do 循环	while 循环	forEach 循环	
1	times end	do repeat if	while end	forEach end	通常结合 execute script、\${变量名}使用

具体参考源码演示

41.5. 控制台运行

参考文档: <https://www.selenium.dev/selenium-ide/docs/en/introduction/command-line-runner>

命令行运行程序需要以下依赖项才能工作:

1. 安装 node 环境
2. 安装 npm, 通常与 node 一起
3. 安装 selenium-side-runner
`npm install -g selenium-side-runner`
4. 浏览器驱动
`npm install -g chromedriver`
`npm install -g edgedriver`
`npm install -g geckodriver`

控制台运行

1. 启动控制台测试
`selenium-side-runner /path/to/your-project.side`

2. 在不同浏览器测试

```
selenium-side-runner -c "browserName=chrome"
```

```
selenium-side-runner -c "browserName='internet explorer'"
```

```
selenium-side-runner -c "browserName=edge"
```

```
selenium-side-runner -c "browserName=firefox"
```

```
selenium-side-runner -c "browserName=safari"
```

42. 分布式 Grid 应用

42.1. Grid 简介

1. 使用场景:

我们的系统需要在不同的系统里+不同的浏览器下去运行，比如测试兼容性

对于大型的测试套件，我们需要并行执行

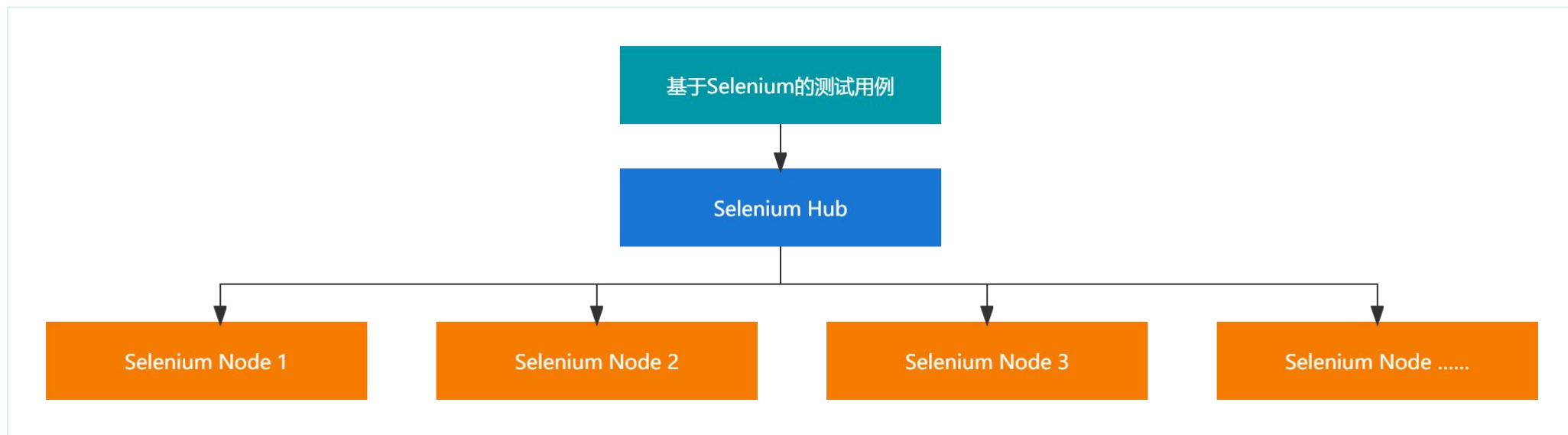
2. Grid 用处:

Selenium Grid 不是用来写脚本代码的，它只负责运行

Selenium Grid 给我们提供了两种角色：一个叫 hub，一个叫 node

hub 被称为总控节点，它负载加载所有的测试机器，一个 grid 里面只有一个 hub，hub 用于管理和分发代码

node 称之为运行/代理节点；node 用于接收代码，且在不同的浏览器中运行代码



42.2. 环境搭建

参考文档:

https://www.selenium.dev/zh-cn/documentation/grid/getting_started/

依赖要求:

1. jdk 11+

进官网下载, <https://jdk.java.net/java-se-ri/11>

配置好环境变量

2. 安装好浏览器

3. 对应浏览器的 webdriver 驱动

驱动要放在 path 环境变量下面

4. 对应的 selenium-server 版本

42.3. 独立模式

顾名思义”独立的”，就是将主控和代理节点都在同一台机器上

standalone 模式能够在一台机器上面执行完整的分布式功能，是 Selenium Grid 的最简单的模式，默认情况下，服务会在 `http://localhost:4444` 地址监听，我们需要通过 `RemoteWebDriver` 指向这个地址，

启动命令参考：

```
java -jar selenium-server-4.4.0.jar standalone
```

`--host` : 指定 IP 地址，多网卡时有用，`0.0.0.0` 代表所有

`--port` : 此参数后跟端口号；设置启动 hub 或 node 服务的端口号；默认端口是 4444，也可以自己设置

`--max-sessions` : 设置最大会话请求数，默认最大会话请求数是 20

`-I` : 指定浏览器 `chrome,firefox,edge`

`--log` : 把日志写到指定路径指定文件

42.4. Hub+node

可以将服务节点（hub）和多个代理节点（node）部署在不同机器上

1. 启动 hub 角色

```
java -jar selenium-server-4.4.0.jar hub --host 0.0.0.0 --port 4444
```

2. 启动 node 角色

节点 1:

```
java -jar selenium-server-4.4.0.jar node --host 0.0.0.0 --port 5555 -I chrome
```

节点 2:

```
java -jar selenium-server-4.4.0.jar node --host 0.0.0.0 --port 6666 -I firefox
```

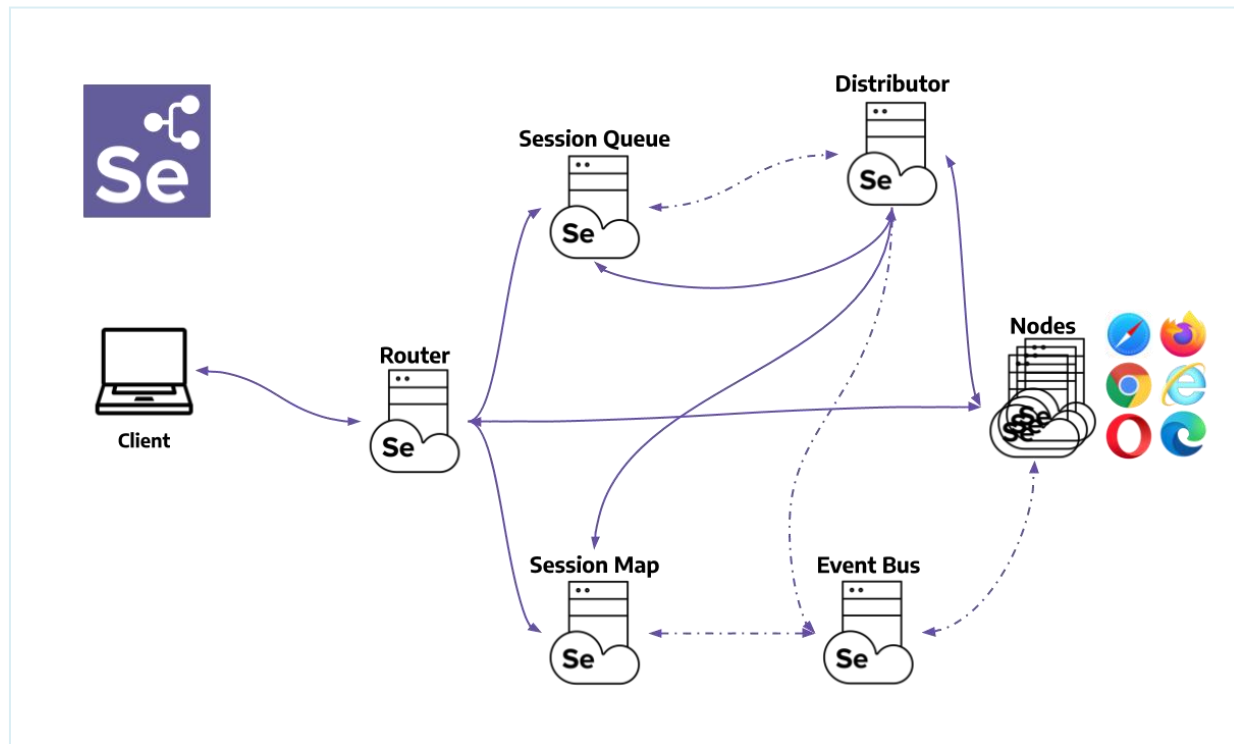
节点 2:

```
java -jar selenium-server-4.4.0.jar node --host 0.0.0.0 --port 7777 -I edge
```

这是我们最常用的一种模式

42.5. Distributed

42.5.1. hub 内部



Event Bus: 其他组件之间的通信路径

Session Map: 负责将 session ID 映射到运行会话的节点 (默认端口 5556)

New Session Queue: 将新会话请求添加到队列中, 然后让分发服务器处理它 (默认端口 5559)

Distributor: 分发服务器 (默认端口 5553)

Router: Grid 的入口, 负责将请求定向到正确的组件 (默认端口 4444)

Hub+node 模式我们看成是 Distributed 的简化版

42.5.2. 环境启动

1. 启动 event bus:

```
java -jar selenium-server-4.4.0.jar event-bus --port 5557
```

2. 启动 sessionqueue:

```
java -jar selenium-server-4.4.0.jar sessionqueue --port 5559
```

3. 启动 sessions:

```
java -jar selenium-server-4.4.0.jar sessions --port 5556
```

4. 启动 distributor:

```
java -jar selenium-server-4.4.0.jar distributor --sessions http://localhost:5556 --sessionqueue http://localhost:5559 --port 5553  
--bind-bus false
```

5. 启动 router:

```
java -jar selenium-server-4.4.0.jar router --port 4444 --sessions http://localhost:5556 --sessionqueue http://localhost:5559  
--distributor http://localhost:5553
```

6. 启动 node

```
java -jar selenium-server-4.4.0.jar node --publish-events tcp://localhost:4442 --subscribe-events tcp://localhost:4443
```


42.6. 远程测试样例

42.6.1. 环境安装

1. 安装 chrome、firefox、edge 驱动

\$PATH 目录下面，含有如下三个驱动：chromedriver.exe，geckodriver.exe，msedgedriver.exe

2. 分别启动三个节点

启动 hub 角色

```
java -jar selenium-server-4.4.0.jar hub --host 0.0.0.0 --port 4444
```

启动 node 角色

节点 1:

```
java -jar selenium-server-4.4.0.jar node --host 0.0.0.0 --port 5555 -I chrome
```

节点 2:

```
java -jar selenium-server-4.4.0.jar node --host 0.0.0.0 --port 6666 -I firefox
```

节点 2:

```
java -jar selenium-server-4.4.0.jar node --host 0.0.0.0 --port 7777 -I edge
```

3. 火狐在测试过程中有可能出现如下问题:

```
console.warn: SearchSettings: "get: No settings file exists, new profile?" (new NotFoundError("Could not open the file at  
C:\\Users\\xiang\\AppData\\Local\\Temp\\rust_mozprofileAAcBVH\\search.json.mozlz4"
```

解决方案：将 geckodriver.exe 在 \$PATH 和 \$TMP 环境变量下均复制一份

42.6.2. 案例源码

```
options = webdriver.ChromeOptions()
#options.set_capability('browserName', 'chrome')
#options.set_capability('platformName', 'Windows 10')
#options.set_capability('browserName', 'firefox')
#options.set_capability('platformName', 'Windows 10')
#options.set_capability('browserName', 'MicrosoftEdge')
#options.set_capability('platformName', 'Windows 10')
driver = webdriver.Remote(
    command_executor='http://127.0.0.1:4444',
    options=options
)
# 浏览器最大化
driver.maximize_window()
driver.implicitly_wait(10)
driver.get("https://www.baidu.com")
# 定位元素并发送文本内容
driver.find_element(By.ID, "kw").send_keys("test")
sleep(2)
driver.find_element(By.ID, "su").click()
sleep(2)
driver.quit()
```

42.7. 个性化样例

42.7.1. 节点配置

我们可以对浏览器进行个性化配置，参考：https://www.selenium.dev/documentation/grid/configuration/cli_options/#node

1. 节点 1 [java -jar selenium-server-4.4.0.jar node --port 5555 --config node1.toml]

```
[node]
```

```
detect-drivers = false
```

```
[[node.driver-configuration]]
```

```
display-name = "chrome-node-1"
```

```
stereotype = '{"browserName": "chrome", "networkname:nodename": "node-1" }'
```

```
max-sessions = 20
```

```
webdriver-path="D:/python310/chromedriver1.exe"
```

2. 节点 2 [java -jar selenium-server-4.4.0.jar node --port 6666 --config node1.toml]

```
[node]
```

```
detect-drivers = false
```

```
[[node.driver-configuration]]
```

```
display-name = "chrome-node-2"
```

```
stereotype = '{"browserName": "chrome", "networkname:nodename": "node-2" }'
```

```
max-sessions = 20
```

```
webdriver-path="D:/python310/chromedriver2.exe"
```

个性化参数需含有冒号，切记

42.7.2. 案例源码

```
from time import sleep
from selenium import webdriver
from selenium.webdriver.common.by import By

options = webdriver.ChromeOptions()
options.set_capability('browserName', 'chrome')
options.set_capability('networkname:nodename', 'node-1')
#options.set_capability('networkname:nodename', 'node-2')
#options.set_capability('networkname:nodename', 'node-3')
driver = webdriver.Remote(
    command_executor='http://127.0.0.1:4444',
    options=options
)
# 浏览器最大化
driver.maximize_window()
# 设置隐式等待
driver.implicitly_wait(10)
driver.get("https://www.baidu.com")
# 定位元素并发送文本内容
driver.find_element(By.ID, "kw").send_keys("test")
sleep(2)
# 页面按钮点击交互
driver.find_element(By.ID, "su").click()
sleep(20)
driver.quit()
```

43. selenium 最佳实践

1. Selenium 的出发点是用来做自动化测试，也就是功能测试，不迁用于做性能测试
2. 测试用例设计过程中，要考虑 pom 分层架构，数据驱动，关键字驱动三大思想的应用
3. 每个测试用例建议单开浏览器，或者是刷新浏览器
4. 测试用例运行时尽量具有独立性，幂等性
5. 含有验证码类似的场景，最好从程序上去掉
6. 测试用例之间在运行时尽量不要有依赖

龙哥在这里祝大家生活愉快，工作顺利，早日实现自己的学习目标