码同学

01

章节

PART

# ui自动化-selenium

**基本介绍：**

Selenium是 ThroughtWorks公司 的，是一个强大的开源的和便携式自动化软件测试工具套件可以用于测试Web应用程序有能力在不同的浏览器和操作系统运行。应用程序有能力在不同的浏览器和操作系统运行。Selenium是一套工具，帮助测试者更有效地基于Web的应用程序自动化。

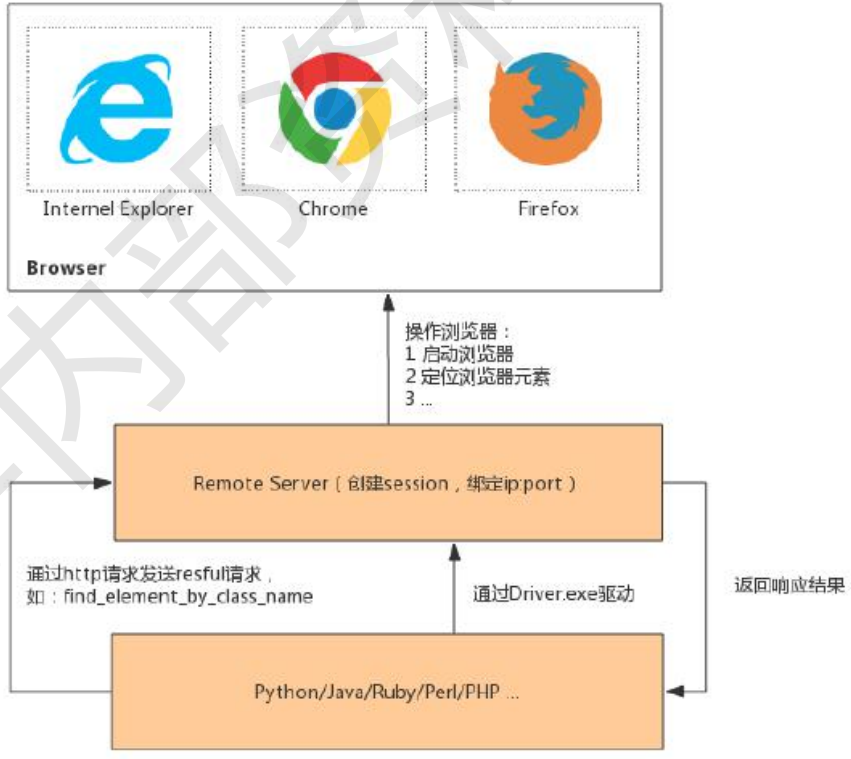
**官网：**

https://docs.seleniumhq.org/

**下载：**

https://docs.seleniumhq.org/download/

对于每一条Selenium脚本，一个http请求会被创建并且发送给浏览器的驱动，
浏览器驱动中包含了一个HTTP Server，用来接收这些http请求，
HTTP Server接收到请求后根据请求来具体操控对应的浏览器，
浏览器执行具体的测试步骤，
浏览器将步骤执行结果返回给HTTP Server，
HTTP Server又将结果返回给Selenium的脚本，如果是错误的http代码我们就会在控制台看到对应的报错信息。



安装对应版本的浏览器

下载浏览器对应版本的driver软件，然后将软件对应的路径添加进系统环境变量path中

Selenium
Pip安装selenium
PyCharm安装selenium

**操作浏览器**

driver =webdriver.Chrome()

**浏览器常用**

driver.close()//关闭当前页面

driver.quit()//关闭由selenium所启动的所有页面

driver.title//返回当前页面的Ttile

driver.current_url//返回当前页面的url

**打开URL**

driver.get(url)

driver.back()

driver.forward()

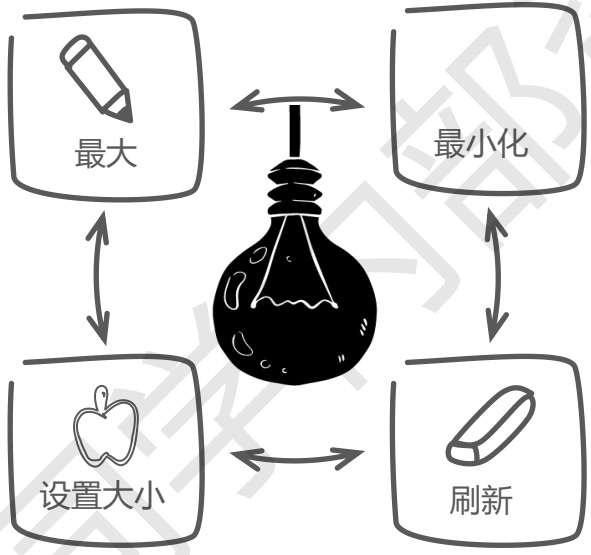
**其他常见**

driver.current_window_handle//返回当前的浏览器的窗口句柄

driver.window_handles//返回当前的浏览器的所有窗口句柄

driver.page_source//返回当前页面的源码

driver.maximize_window()




driver.minimize_window()

driver.set_window_size(480,800)

driver.refresh()


元素定位
内容: 网页元素定位
UI自动化：用程序模拟人和浏览器交互，实现自动运行测试用例，验证系统正确性的一种手段
UI自动化的核心：找到网页元素->操作网页元素
Selenium Webdriver：ThroughtWorks提供的开源工具，是目前最主流的UI自动化工具
定位: 根据HTML标签的类型，属性和文本，找到需要操作的页面元素
元素类型：a, span, input....
元素属性: id, name,class...
元素文本: 开始标签和结束标签中间的文本内容
Selenium八种定位方式


partialLinkText
linkText
链接元素定位
class
tagName
多元素定位
xpath
css
层级定位
元素定位
id
name

# 元素定位

定位方式：

1. id
5.xpath 6.css_selector （定位原则跟css选择器是一样的）
7.link_text 8.partial_link_text

说明：基于元素的id属性定位
前提：元素必须有id属性
方法：driver.find_element_by_id()
特性：一般情况下，在一个页面中，id属性值是唯一；

2. name
说明：基于元素的name属性定位
前提：元素必须有name属性
方法：driver.find_element_by_name()
特性：一般情况下，在一个页面中，name属性值是可以重复的；

3. class
说明：基于元素的class属性定位
前提：元素必须有class属性
方法：driver.find_element_by_class_name()
特性：一般情况下，在一个页面中，class可以有多个值，定位时，使用1个值；

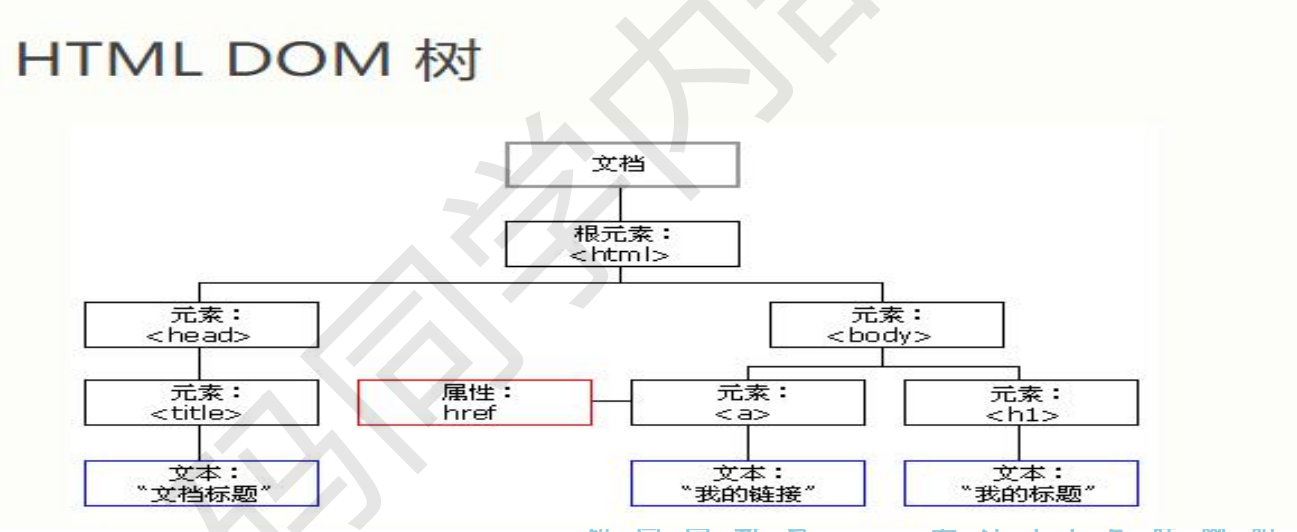
4. tag_name：
说明：一基于元素的**标签名**定位
前提：标签名在页面中是唯一1个，或者有多个但就找第1个；
方法：driver.find_element_by_tag_name()
特性：如果页面存在多个相同元素，默认返回符合条件的第一个元素；

# 层级定位

对不好定位的元素采用层级定位，层级定位是结合网页结构找到元素的一种方式，Selenium支持两种层级定位：

CSS Selector定位器

XPath定位器



# 元素定位总结

当页面元素有id或者name属性时，用id或者name来定位。

当有链接需要定位时，建议用linkText或partialLinkText方式。

当要定位多个相同类型的元素，建议用tagName

当要定位多个样式相同的元素，建议用css

普通定位无法找到元素，建议使用xpath, css 实现层级定位

# 页面输入框

```
//找到输入框元素：
element = driver.find_element_by_id("kw")
//将输入框清空：
element.clear();
//在输入框中输入内容：
element.send_keys("test")
//获取输入框的文本内容：value不要改，取得就是value属性的值 element.get_attribute("maxlength")
```

# 元素等待

隐士等待：
概念：定位元素时，如果能定位到元素则直接返回该元素，不触发等待； 如果不能定位到该元素，则间隔一段时间后再去定位元素； 如果在达到最大时长时还没有找到指定元素，则抛出元素不存在的异常 NoSuchElementException
方法：driver.implicitly_wait(timeout)
(timeout：为等待最大时长，单位：秒)
说明：隐式等待为全局设置（只需要设置一次，就会作用于所有元素
显士等待：
概念：定位指定元素时，如果能定位到元素则直接返回该元素，不触发等待； 如果不能定位到该元素，则间隔一段时间后再去定位元素； 如果在达到最大时长时还没有找到指定元素，则抛出超时异常 TimeoutException

# 元素等待

1. 导包 等待类 --> from selenium.webdriver.support.wait import WebDriverWait
2. WebDriverWait(driver, timeout, poll_frequency=0.5)
1). driver：浏览器驱动对象
2). timeout：超时的时长，单位：秒
3). poll_frequency：检测间隔时间，默认为0.5秒
3. 调用方法 until(method)：直到...时x 是driver
1). method：函数名称，该函数用来实现对元素的定位
2). 一般使用匿名函数来实现：lambda x: x.find_element_by_id("userA")
4. **element = WebDriverWait(driver, 10, 1).until(lambda x: x.find_element_by_id("userA"))**

区别：
1. 作用域：隐式为全局元素，显式等待为单个元素有效
2. 使用方法：隐式等待直接通过驱动对象调用，而显式等待方法封装在WebDriverWait类中
3. 达到最大超时时长后抛出的异常不同：隐式为NoSuchElementException，显式等待为TimeoutException

# 窗口截图

说明：在Selenium中，提供了截图方法，我们只需要调用即可

方法：

driver.get_screenshot_as_file(imgpath)

imgpath：图片保存路径

# 对select操作

说明：Select类是Selenium为操作select标签特殊封装的。
实例化对象：
select = Select(element)
element: <select>标签对应的元素，通过元素定位方式获取，
例如：driver.find_element_by_id("selectA")
操作方法：
1. select_by_index(index) --> 根据option索引来定位，从0开始
2. select_by_value(value) --> 根据option属性 value值来定位
3. select_by_visible_text(text) --> 根据option显示文本来定位
操作方法：
1. select_by_index(index) --> 根据option索引来定位，从0开始
2. select_by_value(value) --> 根据option属性 value值来定位
3. select_by_visible_text(text) --> 根据option显示文本来定位(显示到页面的文本

说明：在Selenium中封装了如何切换frame框架的方法
方法：
1). driver.switch_to.frame(frame_reference) --> 切换到指定frame的方法
frame_reference：可以为frame框架的name、id，index或者**定位到的frame元素**
2). driver.switch_to.default_content() --> 恢复默认页面方法
driver.switch_to.parent_frame()--->恢复到默认页面方法
在frame中操作其他页面，必须先回到默认页面，才能进一步操作

定义一个加法函数

1. 文件名字命名：文件名不能写中文，不能写unittest，不能直接写testcase

2.步骤

1.导包import unittest

2.继承unittest.TestCase

```
class Add(unittest.TestCase)
        def test_1(self):
            pass
```

3.方法名称必须以test字母开头

3.需求：定义一个加法函数,并在unittest框架中测试此函数

UnitTest核心要素

1.TestCase　　测试用例(一条测试用例)

2.TestSuite　　测试套件（把多条用例组合到一起）

　　1.TestSuite().addTest()

　　2.TestLoader()

3.TestRunner　测试套件执行

　　1.TextTestRunner()

　　2.HTMLTestRunner()

4.Fixture(装置函数)

　　首先被执行，结束的时候会被执行（具有特殊意义的函数）

```
def setUp(self):
        pass
def tearDown(self):
    pass
```

级别不一样：类级别，函数级别（类级别要高于函数级别的装置函数）

跳过
对于一些未完成的或者不满足测试条件的测试函数和测试类，可以跳过执行。

使用方式

```
# 直接将测试函数标记成跳过
@unittest.skip('代码未完成')
# 根据条件判断测试函数是否跳过
@unittest.skipIf(condition, reason)
```

备注：上面的跳过装饰器可以装饰函数的测试用例，也可以装饰类

TestSuite (测试套件)

说明：
多条测试用例集合在一起，就是一个TestSuite (测试套件)

使用：
1. 实例化： suite = unittest.TestSuite() (suite：为TestSuite实例化的名称)
2. 添加用例：suite.addTest(ClassName("MethodName"))
(ClassName：为类名；MethodName：为方法名)

说明：
TextTestRunner是用来执行测试用例和测试套件的

使用：
1. 实例化： runner = unittest.TextTestRunner()
2. 执行： runner.run(suite) # suite：为测试套件名称

TestLoader
说明：
用来加载TestCase到TestSuite中，即加载满足条件的测试用例，并把测试用例封装成测试套件。
使用：
unittest.TestLoader，通过该类下面的discover()方法自动搜索指定目录下指定开头的.py文件， 并将查找到的测试用例组装到测试套件；
用法：
suite = unittest.TestLoader().discover(test_dir, pattern='test*.py') 自动搜索指定目录下指定开头的.py文件，并将查找到的测试用例组装到测试套件
test_dir: 为指定的测试用例的目录
pattern：为查找的.py文件的格式，默认为'test*.py'
也可以使用unittest.defaultTestLoader 代替 unittest.TestLoader() ->suite
运行：
runner = unittest.TextTestRunner()
runner.run(suite)

从类种加载测试用例

suite = unittest.TestLoader().loadTestsFromTestCase(TestRunScript)
备注：TestRunScript为测试用例的类

Fixture

说明：Fixture是一个概述，对一个测试用例环境的初始化和销毁就是一个Fixture

Fixture控制级别：
1. 方法级别 2. 类级别

方法级别
使用：
1. 初始化(前置处理): def setUp(self) --> 首先自动执行
2. 销毁(后置处理): def tearDown(self) --> 最后自动执行
3. 运行于测试方法的始末，即：运行一次测试方法就会运行一次setUp和tearDown

类级别
使用：
1. 初始化(前置处理): @classmethod def setUpClass(cls): --> 首先自动执行
2. 销毁(后置处理): @classmethod def tearDownClass(cls): --> 最后自动执行
3. 运行于测试类的始末，即：每个测试类只会运行一次setUpClass和tearDownClass

断言
概念：让程序代替人为判断测试程序执行结果是否符合预期结果的过程

【掌 握】
assertEqual(expected, actual, msg=None) 验证expected==actual，不等则fail
assertIn(member, container, msg=None) 验证是否member in container

参数化：unittest本身没有参数化这个功能，需要第三方辅助，可以用ddt或者通过插件parameterized来实现
第一种parameterized
下载
pip install parameterized

使用方式 导包：
from parameterized import parameterized
使用@parameterized.expand装饰器可以为测试函数的参数进行参数化

```
# 方式一
@parameterized.expand([(1, 1, 2), (1, 0, 1), (0, 0, 0)])
def test_add(self, x, y, expect):
        pass
#方式二
data = [(1, 1, 2), (1, 0, 1), (0, 0, 0)]
@parameterized.expand(data)
def test_add(self, x, y, expect):
        pass
# 方式三
def build_data():
    return [(1, 1, 2), (1, 0, 1), (0, 0, 0)]
@parameterized.expand(build_data)
def test_add(self, x, y, expect):
        pass
```

下载
pip install ddt
导包 import ddt

```
import ddt
import unittest
def add(x,y):
    return x+y
# data = [(2,3),(5,6)]
data = [{'x':2,'y':3},{'x':5,'y':6}]
# data = ((1,2),(2,3))
@ddt.ddt
class TestStudy(unittest.TestCase):
    @ddt.data(*data)
    def test_add(self,data):
        print(data)
        # result = add(data[0], data[1])
        result = add(data['x'],data['y'])

        print(result)
```

# PageObject

PageObject是指UI界面上用于与用户进行交互的对象。它可以指整个页面，也可以指Page上的某个区域
PageObject实现了测试代码的分层：页面元素、元素操作 和 页面业务的分离

**说白了就是把元素与业务相分离**

下载
pip install openpyxl
# 导包
import openpyxl
wb = openpyxl.load_work(文件的路径)
ws = wb['sheet的名字']

for rows in **ws.iter_rows(min_row=,max_row=,min_col=,max_col=)**->生成器
    row_list = [row.value for row in rows]

ws.iter_cols()

ws.max_row最大行
ws.max_column最大列
ws.cell(row，col).value 具体的某一个cell的值

# 格式化信息

**format参数中可能用到的格式化信息：**

| 占位符 | 描述 |
| --- | --- |
| **%(name)s** | Logger的名字 |
| **%(levelname)s** | 文本形式的日志级别 |
| **%(filename)s** | 调用日志输出函数的模块的文件名 |
| **%(lineno)d** | 调用日志输出函数的语句所在的代码行 |
| **%(asctime)s** | 字符串形式的当前时间。默认格式是“2003-07-08 16:49:45,896” |
| **%(message)s** | 用户输出的消息 |

# 定义格式器

定义日志格式

```
fmt = '%(asctime)s %(levelname)s [%(name)s]
[%(filename)s(%(funcName)s:%(lineno)d)] -
%(message)s'
formatter = logging.Formatter(fmt)
```

# 以时间分隔记录日志文件

```
fh = logging.handlers.TimedRotatingFileHandler(filename,
when='h', interval=1, backupCount=0)
```

将日志信息记录到文件中，以特定的时间间隔切换日志文件。

filename: 日志文件名

when: 时间单位，可选参数

S - Seconds

M - Minutes

H - Hours

D - Days

midnight - roll over at midnight

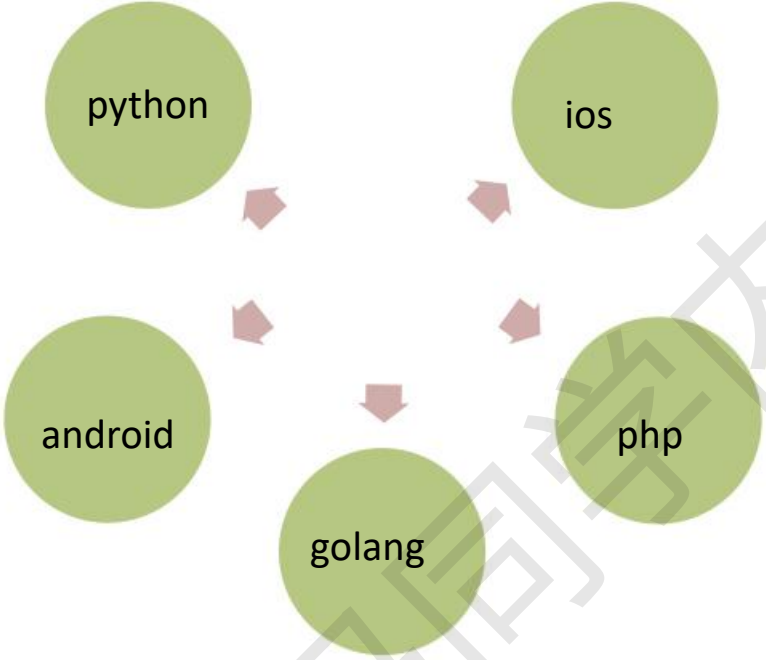
W{0-6} - roll over on a certain day; 0 - Monday interval: 时间间隔(秒)

backupCount: 日志文件备份数量。如果backupCount大于0，那么当生成新的日志文件时，将只保留backupCount个文件，删除最老的文件。

什么是代码覆盖率
描述源代码被测的比例和程度，测试有效性的一个度量
意义

帮助找出未测试到的代码
增加测试用例

# 各语言覆盖率




| | |
|---|---|
| C/CPP | ccover, gcov |
| JAVA | jacoco, cobertura |
| PHP | phpunit+xdebug, php- |
| GOLANG | gotest |
| PYTHON | coverage.py |
| ANROID | jacoco |
| IOS | xcodebuild, gcov |

# coverage cmd

- **run** – 运行python程序并收集覆盖率数据.
- **report** – 产生覆盖率结果报告.
- **html** – 产生HTML格式的覆盖率结果报告.

cmd的操作

coverage help

coverage help run

coverage run run.py

coverage report

coverage html -d coverage_html

# coverage api

```
import coverage
cov = coverage.Coverage()
cov.start()
# .. call your code ..
cov.stop()
cov.save()
cov.html_report()
```

PEP8

使用4个空格进行缩进
制表符与空格不能混用
行的最大长度为79个字符
顶层函数和类之间使用两个空行
类的方法之间使用一个空行
首选编码为utf-8编码
在文件顶部导入库，在所有变量与常量之前
采用标准库，外部库，自定义库顺序导入
紧贴着圆括号、方括号和花括号时不加空格
紧贴在逗号，分号或冒号之前不加空格

二元运算符前后各加一个空格
函数参数的赋值符号前后不加空格
块注释在代码之前，和代码缩进一致
块注释每行以#开头，然后紧跟一个空格
文档字符串紧跟模块，类，函数定义
类名要首字母大写，内部类加上前导下划线
函数名用小写，用下划线分隔
用self作为实例方法的第一个参数
用cls作为类方法的第一个参数
常量在模块中定义，用全大写和下划线分隔

pylint
- 安装
- pip install pylint
- 指令
- pylint [options] modules_or_packages
- 使用

| 类型 | 解释 |
| --- | --- |
| C | 违反编码风格标准 |
| R | 需要重构的代码 |
| W | 轻微的程序问题 |
| E | 代码错误很可能是bug |
| F | 严重的错误导致pylint不能 |

工具

| 语言 | 工具 |
| --- | --- |
| C/C++ | CppCheck/Clang |
| Java | PMD/Findbugs/Fox |
| Android | Godeyes/Infer/Findbugs/Fox/AndroidLint |
| PHP | PHPMD/Rips/PHPCheckstyle |
| Object-C | Godeyes/Infer |
| Python | Pylint |
| Go | Govet/Golint |
| JavaScript | ESLint |