第 3 章

Selenium 及浏览器驱动的安装配置

Selenium 是一款用于 Web 应用程序 UI 自动化测试的首选工具。本书第 1 章已经详细介绍过 Selenium 在运行自动化测试用例时的优势。与接口自动化相比，它更贴近真实用户使用场景，在 UI 层自动化测试上具有不可替代的作用。

本章主要内容分为两部分，Selenium 的下载与安装及主流浏览器驱动的下载与调试。这是 Web 篇环境准备的最后一部分内容。

3.1 Selenium 的下载及安装

获取 Selenium 的最新特性，首选方式还是在 Selenium 官网下载相关文档进行新特性的了解与使用。Selenium 官网网址是 https://www.seleniumhp.dev，如图 3.1 所示。

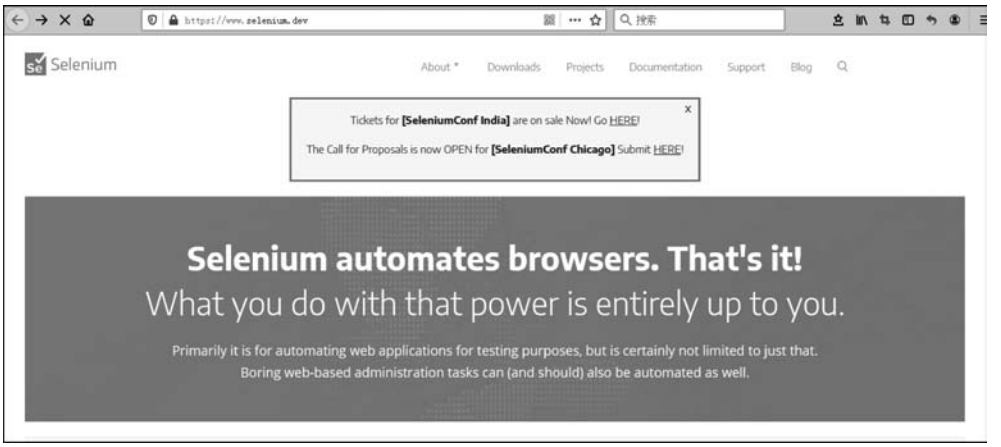

图 3.1　Selenium 官网首页

从 Selenium 官网可以看到目前 Selenium 体系的三大组成部分 Selenium WebDriver、Selenium IDE 和 Selenium Grid，如图 3.2 所示。

本章需要安装配置的是 Selenium WebDriver 组件，基于 Web 自动化测试的后续学习也是围绕着这一部分展开的。Selenium IDE 在早期是以 FireFox 窗口插件形式出现的。其

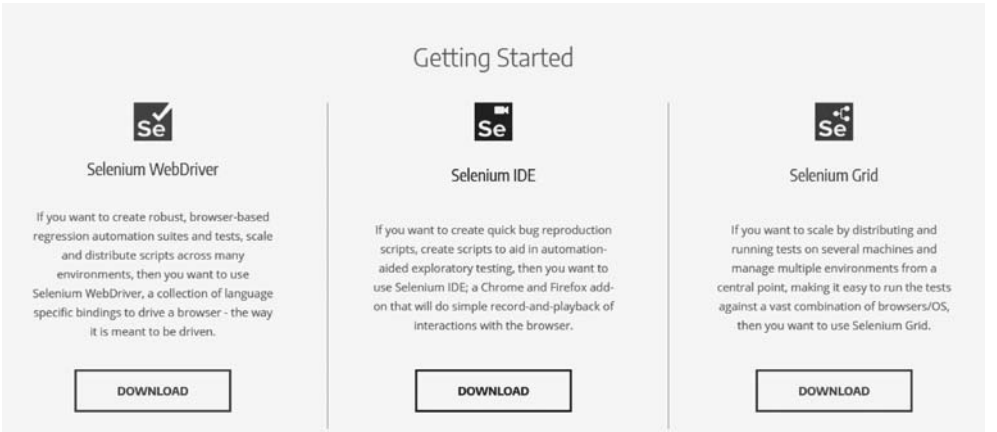

图 3.2　Selenium 体系的组成

主要功能是配合 FireFox 及 Chrome 浏览器完成脚本录制与回放操作，以及简单用例的管理与断言的实现。本书不再进行 Selenium IDE 相关功能的展示与讲解。Selenium Grid 属于 Selenium 脚本运行时的一个多机并发架构实现，属于 Selenium 在使用过程中实现较为复杂的部分，相关安装配置与使用会在本书最后一章进行讲解。

3.1.1　Selenium 在线安装

Selenium 最常用的安装方式为在线安装。第 2 章 Python 配置过程中已将 Scripts 目录配置进系统环境变量，在此基础上就可以进行在线安装了。

1. 安装

打开命令提示行工具，输入 pip install selenium 命令，按下回车键进行安装，如图 3.3 所示。

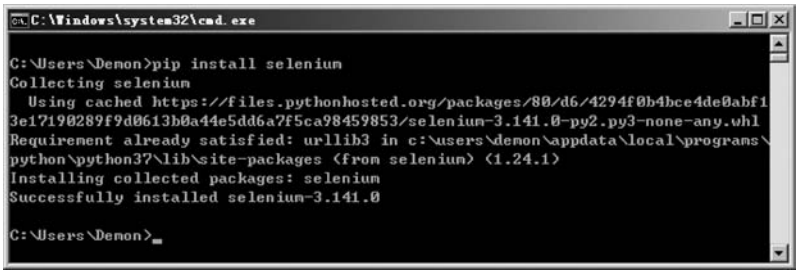

图 3.3　Selenium 在线安装

少数情况下会出现安装失败提示。通常操作系统是 Windows 10 的时候有可能出现这种情况。这是因为 Windows 10 对 pip 命令的支持出现异常，此时用 pip3 命令进行安装就可以了。方法和上面所示 pip 的使用方法一样。

2. 验证

安装成功后输入 python 命令，按下回车键进入 Python 命令模式，输入 import selenium，按下回车键进行验证，如图 3.4 所示。

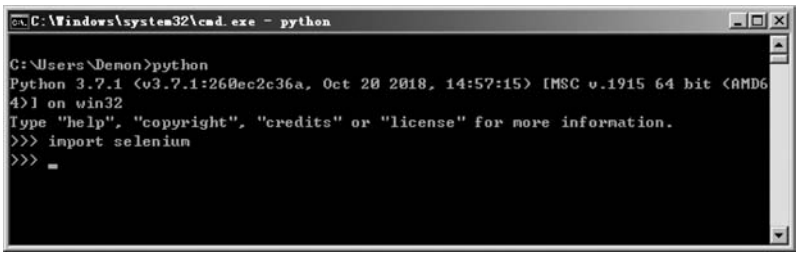

图 3.4 Selenium 验证

如果没有任何提示，就说明 import 导入 Selenium 包成功了，在线安装完成。

3. 指定版本安装

这种操作在实际工作环境中比较少见。常见的情况是在已经配置好的脚本框架中用到了某个版本的某些属性方法，而这些属性方法在新版本中被移除了，脚本修改起来又比较有难度，这时会考虑将 Selenium 进行降级安装。这种情况只涉及降级，因为在写一个脚本时不太可能用到未来 Selenium 版本的功能。在命令提示符窗口中输入 pip install selenium==<指定版本号>后回车进行降级安装，如图 3.5 所示。

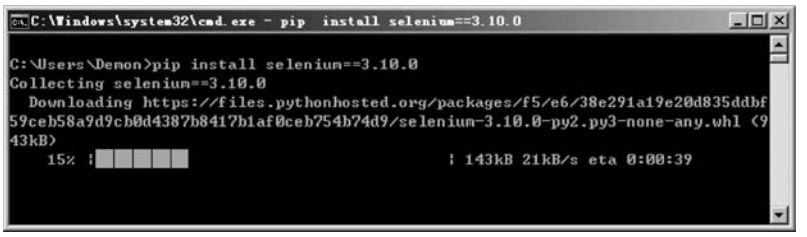

图 3.5 Selenium 降级安装

3.1.2 Selenium 离线安装

还有一种更不常见的情况，就是所有在线安装方式都无法完成 Selenium 的安装。这种情况通常也出自 Windows 10 操作系统下，这时就需要使用离线安装模式了。如果你的 Selenium 已经安装成功，可以跳过此节。

首先需要在 Selenium 官网找到离线文件的下载位置。如果查找不方便，则可以输入 https://pypi.org/project/selenium/#files 直接定位到离线文件的位置，如图 3.6 所示。

在页面中选择 selenium-3.141.0.tar.gz 文件进行保存，然后解压文件。打开命令提示符窗口，进入解压后的目录。输入 pip setup.py install 命令按下回车键进行安装，如图 3.7 所示。

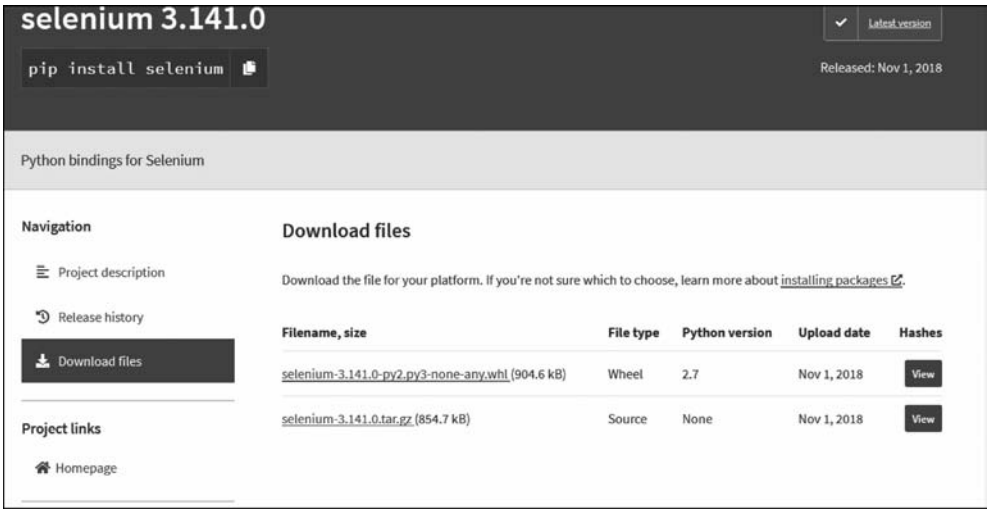

图 3.6　Selenium 离线文件下载

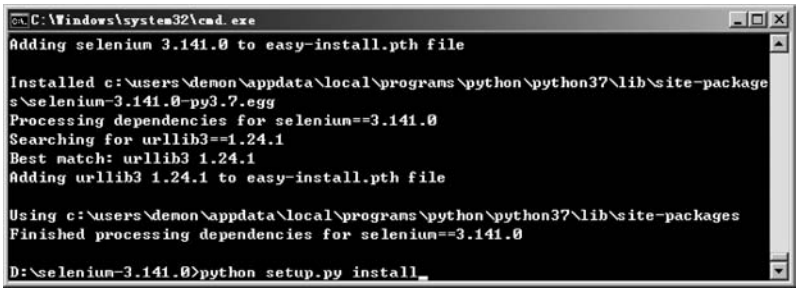

图 3.7　Selenium 离线安装

图 3.7 所示结果为命令方式离线安装后的结果。安装完成后，使用 3.1.1 节的方法验证安装的正确性。

3.2　基于 FireFox 浏览器的驱动配置

Selenium 最早是作为 FireFox 浏览器的一个插件出现的，早期版本中的很多功能只能在 FireFox 浏览器中实现。例如早期的脚本录制插件 Selenium IDE，最初只支持在 FireFox 下安装和使用。随着 UI 自动化测试在 Web 端的普及，Selenium 对其他浏览器的功能支持也日趋完善。

3.2.1　GeckoDriver 驱动配置的下载与配置

基于 FireFox 浏览器的 GeckoDriver 驱动与浏览器的版本向上兼容性较差。随着浏览器版本的升级，常常会出现驱动失效的现象，因此需要有一个能够及时查找最新驱动的

地方。

推荐一个好用的资源类网站，阿里云开发者社区中的镜像站。访问网址 https://npm.taobao.org/mirrors，如图 3.8 所示。

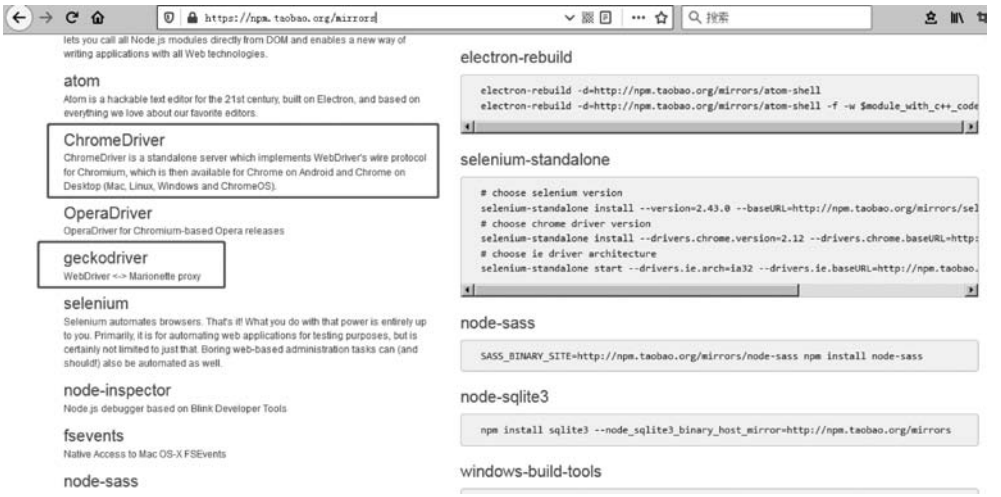

图 3.8　NPM 镜像站

在此处可以找到最新的 FireFox 浏览器和 Chrome 浏览器驱动文件，也可以到官网进行下载。单击图 3.8 中的 geckodriver，进入驱动下载页，如图 3.9 所示。

图 3.9　GeckoDriver 下载页面

找到最新的驱动版本，进入版本子目录，选择适合自己操作系统的版本进行下载。这里笔者选择了 geckodriver-v0.26.0-win64.zip 进行下载。下载完成后将文件解压并放入 Python 安装主目录的 Scripts 文件夹下即可。

3.2.2 调用 FireFox 驱动测试

配置完成后，编写一段代码对配置驱动进行验证。此处验证有两个目的：一是验证浏览器与驱动版本是否匹配；二是验证 Selenium 安装是否正确。如果没有特殊情况，则后面几节的驱动验证不再进行说明，代码如下：

```
#第 3 章/FirefoxDriverTest.py
#引入 selenium 包
from selenium import webdriver

#声明驱动对象并打开 FireFox 浏览器
driver = webdriver.Firefox()

#打开百度首页
driver.get('http://www.baidu.com/')
sleep(2) #暂停两秒

#关闭驱动及 FireFox 浏览器
driver.quit()
```

3.3 基于 Chrome 浏览器的驱动配置

Chrome 是一款由谷歌公司开发的网页浏览器。虽然现在国内不支持 Google 搜索引擎和它配套的应用商店功能，但作为浏览器本身的优点，其运行速度快、加载页面内容迅速，至今仍是 Web 架构软件运行浏览器的首选。

3.3.1 ChromeDriver 驱动配置的下载与配置

Chrome 浏览器和 FireFox 浏览器相仿，版本更新比较频繁。ChromeDriver 的优点是对浏览器版本兼容性比较好，一般不会存在版本兼容问题，从而导致无法调用浏览器的情况。ChromeDriver 驱动下载的网址见 3.2.2 节中的阿里云开发者社区镜像站，如图 3.10 所示。

页面中的版本在后期是与浏览器版本一一对应的，驱动更新速度快，通常会与浏览器同步升级。读者可以在这里找到与你所用 Chrome 浏览器版本相近的驱动进行下载。下载完成后解压并放至 Python 主目录下的 Scripts 目录下即可。

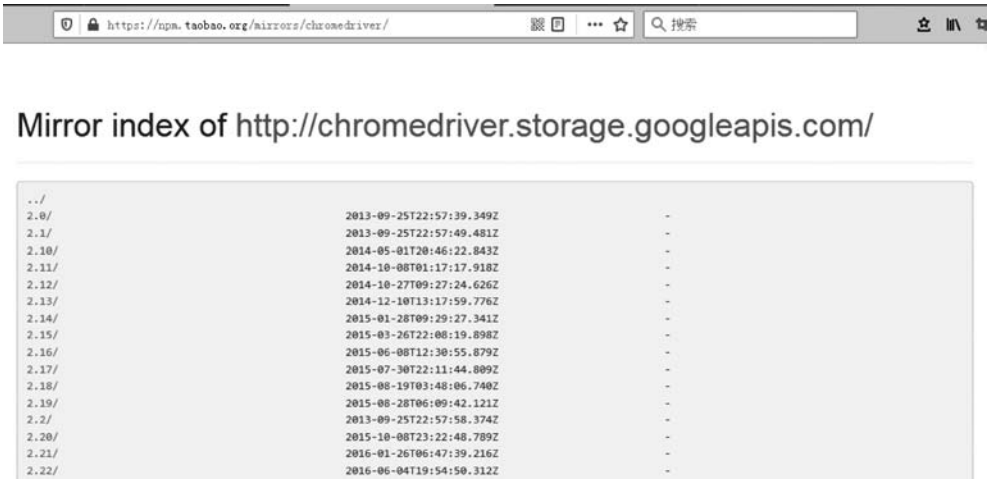

图 3.10 ChromeDriver 下载页面

3.3.2 调用 Chrome 驱动测试

调用 Chrome 驱动浏览器,代码如下:

```
#第 3 章/ChromeDriverTest.py
#引入 selenium 包
from selenium import webdriver

#声明驱动对象并打开 Chrome 浏览器
driver = webdriver.Chrome()

#打开百度首页
driver.get('http://www.baidu.com/')
sleep(2) #暂停两秒

#关闭驱动及 Chrome 浏览器
driver.quit()
```

3.4 基于 IE 浏览器的驱动配置

IE 是一款在 Web 软件领域使用最久的浏览器。微软将在 2020 年 1 月结束对 IE 10 的支持,IE 11 将会成为 IE 浏览器的最后一次更新。大多数网站在开发过程中会对 IE 浏览器的兼容性进行测试。IE 浏览器的 Trident 内核至今仍被很多国内主流浏览器所支持,因此对 IE 浏览器的 UI 层测试,在最近几年内仍将持续进行。

3.4.1　IEDriverServer 驱动配置的下载与配置

目前微软官网已不提供 IEDriverServer 的下载，前面推荐的镜像站里也没有相关驱动。这里分享一个资源链接 http://selenium-release.storage.googleapis.com/index.html，如图 3.11 所示。在这个资源网站中可以找到支持 Selenium 的各种 IEDriverServer 版本，可根据实际情况选择相关驱动版本。

图 3.11　IEDriverServer 下载页面

3.4.2　调用 IE 驱动测试

调用 IE 驱动浏览器，代码如下：

```
#第 3 章/IEDriverTest.py
#引入 selenium 包
from selenium import webdriver

#声明驱动对象并打开 IE 浏览器
driver = webdriver.Ie()

#打开百度首页
driver.get('http://www.baidu.com/')
```

```
sleep(2) #暂停两秒

#关闭驱动及 IE 浏览器
driver.quit()
```

在运行这段代码的时候，有时会报 Unexpected error launching Internet Explorer. Protected Mode settings are not the same for all zones 错误，翻译成中文就是“启动 Internet Explorer 时出现意外错误。所有区域的保护模式设置不同”。这个错误和 3.4.1 节配置的驱动没有什么关联，主要是 IE 浏览器设置的问题。打开 IE 浏览器→工具菜单→Internet 选项→安全选项卡，如图 3.12 所示。

图 3.12 Internet 选项

将安全选项卡中所标注的四项的“启动保护模式”全部设置为失选或全选状态。这里要求 4 个选项必须统一。选择完成后，单击“确定”按钮保存更改。再次运行测试代码，就可以正常调用及运行 IE 浏览器并打开百度首页了。

3.5 第一个 Web 自动化测试脚本

至此，Web 篇所用到的环境配置就全部完成了。接下来需要设计一个操作较为复杂的脚本实例来验证环境的正确性，在正式开始对 Selenium 进行讲解之前来感受一下用例自动化的过程。前面调试了三款主流浏览器驱动，在后续章节里，将以 Chrome 浏览器为主进行相关知识的讲解。有特殊情况需要换驱动操作的会进行特别说明。

下面根据自动化测试脚本的流程设计一段代码。代码中所涉及细节在后面章节会进行讲解，此处只针对流程本身进行讲解。在正式开始学习自动化测试知识前，先进行一次感性接触。

1. 手工用例准备

所有的自动化测试用例最早都来源于手工用例。多数情况下，UI 自动化测试用来做回归，因此可以直接把手工用例中的正向用例部分直接转换成自动化测试脚本。本次被测网站使用百度新闻网页进行验证。

【例 3.1】 百度新闻搜索测试。

步骤 1：

(1) 打开百度新闻首页。

(2) 在打开的页面上单击热点新闻。

(3) 关闭热点新闻页面。

(4) 在百度新闻搜索框中输入新闻内容,单击“搜索”按钮。

步骤 2：热点新闻在新页面打开,内容与标题一致。

在这个最基本的手工测试用例中只对步骤 2 做预期判断。

2. 自动化脚本

自动化测试脚本实现过程如下：

(1) 调用驱动,打开 Chrome 浏览器。

(2) 输入网址,打开百度新闻网站。

(3) 单击热点新闻第一条,对弹出热点新闻做结果判断。

(4) 关闭热点新闻页面。

(5) 在百度新闻搜索框中输入关键字进行搜索。

(6) 对最终搜索内容做结果判断。

将上述内容用 Python 脚本实现,代码如下：

```
#第 3 章/TestBaiDuNews.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://news.baidu.com/')

#单击当天热点新闻
sleep(1)
driver.find_element_by_xpath('//*[@id="pane-news"]/div/ul/li[1]/strong/a[1]').click()

#切换句柄
search_Windows = driver.current_window_handle
all_handles = driver.window_handles
for handle in all_handles:
    if handle != search_Windows:
        driver.switch_to.window(handle)

#结果判断,''内根据当时热点新闻关键字进行修改
assert '' in driver.title

#关闭当前页
sleep(1)
driver.close()

#切换句柄
for handle in all_handles:
```

```
        if handle == search_Windows:
            driver.switch_to.window(handle)

    #关键字搜索
    sleep(1)
    driver.find_element_by_xpath('//*[@id="ww"]').send_keys('吴哥窟')
    sleep(1)
    driver.find_element_by_xpath('//*[@id="s_btn_wr"]').click()
    sleep(2)

    #搜索结果判断
    assert '吴哥窟' in driver.page_source
    driver.quit()
```

3. 运行结果

代码 TestBaiDuNews.py 中加入了大量等待时间，方便读者查看每一步操作的结果。此处就不放置运行结果图了，热点新闻每天都会有所变化。