目录

[第1章 安装Selenium 1](#_Toc9507214)

[1.1 Selenium的优点 1](#_Toc9507215)

[1.2 Selenium WebDriver 2](#_Toc9507216)

[1.3 在Windows和Linux系统上安装Selenium 4](#_Toc9507217)

[1.3.1 在Windows系统上Python的安装说明 4](#_Toc9507218)

[1.3.2 在Linux系统上Python的安装说明 6](#_Toc9507219)

[第2章 页面导航 7](#_Toc9507220)

[2.1 启动浏览器 7](#_Toc9507221)

[2.2 打开页面 8](#_Toc9507222)

[2.3 浏览器基本操作 9](#_Toc9507223)

[2.4 拖动窗口 10](#_Toc9507224)

[2.5 cookie的调用 12](#_Toc9507225)

[2.6 使用多个窗口 13](#_Toc9507226)

[参考网站 14](#_Toc9507227)

[第3章 元素定位 16](#_Toc9507228)

[3.1 元素定位概述 16](#_Toc9507229)

[3.2 根据 ID 定位 17](#_Toc9507230)

[3.3 根据 name定位 19](#_Toc9507231)

[3.4 根据XPath定位 20](#_Toc9507232)

[3.5 根据标签名定位 21](#_Toc9507233)

[3.6 根据类名定位 22](#_Toc9507234)

[3.7 根据CSS选择器定位 24](#_Toc9507235)

[3.8 根据链接文本定位 25](#_Toc9507236)

[3.9 根据部分链接文本定位 26](#_Toc9507237)

[第4章 利器——PhantomJS 27](#_Toc9507238)

[4.1 PhantomJS 27](#_Toc9507239)

[4.2 下载与安装PhantomJS 27](#_Toc9507240)

[4.2.1 在Windows系统上安装PhantomJS 27](#_Toc9507241)

[4.2.2在Linux系统上安装PhantomJS 28](#_Toc9507242)

[4.2.3 检验安装 28](#_Toc9507243)

[4.3 配置相应的WebDriver 28](#_Toc9507244)

[4.4 第一个PhantomJS小程序 29](#_Toc9507245)

[4.5 通过time.sleep()慢下来 31](#_Toc9507246)

[4.6 PhantomJS的命令行参数 32](#_Toc9507247)

[4.7其他问题 32](#_Toc9507248)

[4.7.1.中文编码问题 32](#_Toc9507249)

[4.7.2不同frame间的转换 33](#_Toc9507250)

[4.7.3 PhantomJS进程不会自动退出 33](#_Toc9507251)

[第5章 基于Python官网的实战 34](#_Toc9507252)

[5.1 通过Selenium访问Python官网 34](#_Toc9507253)

[5.2 通过JavaScript修改Python官网的标题 34](#_Toc9507254)

[5.3 在搜索框搜索 35](#_Toc9507255)

[5.4 获取latest news部分 36](#_Toc9507256)

[5.5 Selenium的等待 38](#_Toc9507257)

[第6章 基于今日头条的实战 42](#_Toc9507258)

[6.1 通过Selenium访问百度热词 42](#_Toc9507259)

[6.2 通过Selenium搜索相关热词 44](#_Toc9507260)

[6.3 获取第一条结果 45](#_Toc9507261)

[6.4 定位元素——抓取内容 46](#_Toc9507262)

[6.5 存储数据 48](#_Toc9507263)

[第7章 数据的编解码和处理 51](#_Toc9507264)

[7.1 读写CSV文件 51](#_Toc9507265)

[7.2 读写JSON文件 53](#_Toc9507266)

[7.3 将字典转化为XML 55](#_Toc9507267)

[7.4 解析XML 56](#_Toc9507268)

[7.5 Base64编码 58](#_Toc9507269)

[7.6词性分析、统计分析NLTK 59](#_Toc9507270)

[7.7 散列表 64](#_Toc9507271)

[7.8 布隆过滤器 65](#_Toc9507272)

[7.9 关系数据库MYSQL 67](#_Toc9507273)

[7.9.1安装MySQL 68](#_Toc9507274)

[7.9.2 MySQL常用命令 71](#_Toc9507275)

[7.10非关系数据库 77](#_Toc9507276)

[7.11总结 77](#_Toc9507277)

[**第8章 基于猫眼电影的实战** 79](#_Toc9507278)

[8.1 网站分析 79](#_Toc9507279)

[8.2 最初的测试代码 80](#_Toc9507280)

[8.3进一步完善之后的测试代码 83](#_Toc9507281)

[第9章 基于淘宝商品的实战 86](#_Toc9507282)

[9.1 准备工作 86](#_Toc9507283)

[9.2 流程简述 86](#_Toc9507284)

[9.3 代码解读 89](#_Toc9507285)

[9.4 可配置项 93](#_Toc9507286)

[第10章 单元测试 94](#_Toc9507287)

[10.1 为什么要写单元测试 94](#_Toc9507288)

[10.2 怎样写单元测试 94](#_Toc9507289)

[10.3 单元测试的演示 95](#_Toc9507290)

[10.4 单元测试样例 96](#_Toc9507291)

[10.5 Selenium的单元测试 99](#_Toc9507292)

[10.6 美化报告 100](#_Toc9507293)

[第11章 多线程 103](#_Toc9507294)

[11.1 什么是进程线程 103](#_Toc9507295)

[11.2 单线程，多线程对比 103](#_Toc9507296)

[11.3 实际体验 104](#_Toc9507297)

[11.4 threading库中常用的函数 106](#_Toc9507298)

[11.5 锁的概念 107](#_Toc9507299)

[11.6 主程序是线程还是进程 109](#_Toc9507300)

[11.7 更多解决方案 109](#_Toc9507301)

[11.8 基于多线程和Selneium的示例 109](#_Toc9507302)

[11.9 GIL 110](#_Toc9507303)

[**第12章 发送邮件** 112](#_Toc9507304)

[12.1 通信的选择 112](#_Toc9507305)

[12.2 获取授权码 112](#_Toc9507306)

[12.3 发送邮件 114](#_Toc9507307)

[第13章 Selenium IDE 117](#_Toc9507308)

[13.1安装SeleniumIDE 117](#_Toc9507309)

[13.2 Selenium IDE 119](#_Toc9507310)

[13.3 Katalon Recorder 123](#_Toc9507311)

[实例与步骤 125](#_Toc9507312)

[13.4 数据驱动 126](#_Toc9507313)

[13.5 扩展脚本 127](#_Toc9507314)

[**13.5.1 添加扩展脚本** 127](#_Toc9507315)

[**13.5.2 定位器构建器** 128](#_Toc9507316)

[**13.5.3 定位器构建的自定义顺序** 128](#_Toc9507317)

[13.5.4 Prototype附加命令 128](#_Toc9507318)

[**13.6 Katalon Recorder Helper工具** 129](#_Toc9507319)

[**第14章 Python扩展** 131](#_Toc9507320)

[14.1 2to3工具 131](#_Toc9507321)

[14.2 测试类型 131](#_Toc9507322)

[14.3 通配符类型 132](#_Toc9507323)

[14.4 str方法 133](#_Toc9507324)

[**14.5 异常层次结构** 133](#_Toc9507325)

[14.6 兼容Python 2和3 135](#_Toc9507326)

[14.7 兼容性代码 135](#_Toc9507327)

[14.7.1 导入库 136](#_Toc9507328)

[14.7.2 输出函数 136](#_Toc9507329)

[14.7.3 异常捕获 136](#_Toc9507330)

[14.8 类相关 137](#_Toc9507331)

[14.9 推导式 137](#_Toc9507332)

[14.10 经典除法和真除法 138](#_Toc9507333)

[14.11 \_\_future\_\_ 139](#_Toc9507334)

[14.12 类型转换和关键字 139](#_Toc9507335)

[第15章 GUI测试 141](#_Toc9507336)

[15.1 tkinter测试 141](#_Toc9507337)

[15.2 创建界面 141](#_Toc9507338)

[15.3 按钮 142](#_Toc9507339)

[15.4 pack属性 143](#_Toc9507340)

[15.5继承类 143](#_Toc9507341)

[15.6 复选框和单选按钮 145](#_Toc9507342)

[15.7 Entry输入框 147](#_Toc9507343)

[15.8 列表框 151](#_Toc9507344)

[15.9 Text控件 154](#_Toc9507345)

[15.10 Canvas控件 157](#_Toc9507346)

[15.11 Menu控件 159](#_Toc9507347)

[15.12 spinbox控件 161](#_Toc9507348)

[15.13 messagebox 162](#_Toc9507349)

[15.14 filedialog 163](#_Toc9507350)

[15.15 Message 163](#_Toc9507351)

[15.16 tkinter界面布局 164](#_Toc9507352)

[15.17 tkinter支持的模块 165](#_Toc9507353)

[15.18 \_tkinter接口 166](#_Toc9507354)

[15.19 tkinter.ttk 166](#_Toc9507355)

[15.20 底层实现流程 166](#_Toc9507356)

[第16章 基于知乎的实战 168](#_Toc9507357)

[16.1 知乎分析 168](#_Toc9507358)

[16.2 文字部分 170](#_Toc9507359)

[16.3 图片部分 173](#_Toc9507360)

[16.4 建议 177](#_Toc9507361)

[16.4.1 从干净的状态开始 178](#_Toc9507362)

[16.4.2 测试的隔离 178](#_Toc9507363)

[16.4.3 Anaconda 178](#_Toc9507364)

[16.4.4 报告 179](#_Toc9507365)

[16.4.5 IP代理 179](#_Toc9507366)

[16.4.6 过滤请求 180](#_Toc9507367)

[16.5 一些注意事项 180](#_Toc9507368)

[16.5.1 Captchas 180](#_Toc9507369)

[16.5.2 性能测试 181](#_Toc9507370)

[16.5.3 六度分割理论及其实际使用 181](#_Toc9507371)

[16.5.4 HTTP状态码 181](#_Toc9507372)

[16.6 常见问题 183](#_Toc9507373)

[16.6.1 元素定位失败 183](#_Toc9507374)

[16.6.2 Webdriver调用失败 185](#_Toc9507375)

[第17章 基于新浪微博的实战 186](#_Toc9507376)

[17.1 新浪微博分析 186](#_Toc9507377)

[17.2 实现思路 188](#_Toc9507378)

[17.3 代码改进 193](#_Toc9507379)

[17.4 结语 196](#_Toc9507380)

[附录A 如何阅读源代码 197](#_Toc9507381)

[附录B Git分布式计算 200](#_Toc9507382)

[B.1 Git 的背景介绍 200](#_Toc9507383)

[B.2 Git 与其他版本控制系统的区别 200](#_Toc9507384)

[B.3 Git中的复制、修改和推送 202](#_Toc9507385)

[B.4 Git 分支 203](#_Toc9507386)

[B.5 Git 自定义命令 205](#_Toc9507387)

[B.6 Git stash命令 205](#_Toc9507388)

[B.7 Git 文件的对比 206](#_Toc9507389)

[B.8 Git 版本日志、回退 207](#_Toc9507390)

# 

# 第1章 安装Selenium

人与动物最大的区别在于人善于使用工具，只要用对了工具就能事半功倍，而Python正是当前最完美的工具之一。Python如此强大，并且方方面面都做得很好，以至于没人敢说“精通Python”，本书只讲了其中关于数据挖掘的很小一部分，详细讲述了Selenium这个库的使用，帮助读者在技术深造的过程中更上一层楼。

## 1.1 Selenium的优点

本书侧重于讲解基于Selenium开源程序的网络爬虫方面的自动化技术。很多人听过这个名词，却并没有真正去了解过它，经常有人问：“如何理解和看待Selenium项目？”对于该问题，Selenium官方有一个很经典的回答：“自动化操作的浏览器，就是这样……做你想要做的事情，一切取决于你。”

Selenium项目最早是为了测试浏览器、网页而诞生的，而在大数据时代，Selenium则广泛应用于网络爬虫。需要注意的是，Selenium项目并不是单个特定的软件，而是由多个工具组成的，每个工具具有特定的功能。

Selenium WebDriver是Selenium项目基于浏览器的一部分，也是我们主要使用的一部分，它的灵活性很强，几乎支持所有主流的浏览器（甚至包括一些很小众的浏览器），并且可以完美地在支持这些浏览器的操作系统运行，换句话说，当使用Selenium的时候，你完全不必担心兼容性问题，可以放心地进行代码移植。

但不要轻易更换你的WebDriver调用的浏览器。Selenium WebDriver对浏览器的操作通过浏览器原生的API来实现，这样一来，虽然速度和稳定性大大提高，也交给了浏览器的开发商，但也带来了一些副作用，因不同浏览器Web元素的实现和呈现方法不同，Selenium WebDriver也要分不同的浏览器而提供不同的实现方法。

SeleniumWebDriver支持的浏览器主要包括：

* Firefox浏览器，
* 谷歌浏览器，
* 欧朋浏览器，
* Edge浏览器，
* IE浏览器。

SeleniumWebDriver支持的支持的操作系统主要包括：

* Windows，
* Linux，
* Mac OS（含iOS），
* Android。

不仅包括以上这些，而且Selenium还支持多种编程语言（Java、R、Python、PHP 、Ruby、Perl、Haskell、Objective-C、C#等语言）。在这个信息泛滥的互联网时代，如果要选择Web方面的自动化工具，我会将Selenium作为首选（一个支持多种浏览器，多种操作系统，多种编程语言的工具，恐怕没有什么比它更合适了）。

使用Selenium项目的自动化技术具有以下好处。

* 可靠性

企业的日常工作常常包含一些完全必要的重复性工作，传统的企业大多采用人力来解决，这就容易导致一些严重的错误。比如，误删数据库，系统宕机，无论在经济上，还是在声誉上，对于企业来说，这样的错误都是一种毁灭性的打击。而当把人为因素从这里排除时，绝大多数人为错误导致的重大损失就可以避免。

* 性能优化

自动化技术人员在日常工作中经常需要提高工作效率，以传统方式解决这种问题是很难的。然而，当应用了自动化技术之后，我们就可以通过更加高效的系统能源利用方式来更迅速地执行任务，加快工作流程，处理更加庞大的工作负载。

* 解放生产力

没有一家企业不想降低成本，要提高企业的利润率，就需要招聘更多的人，招收更多的人就意味着更高的人力成本，这个问题却可以通过自动化技术迎刃而解。自动化的工作没有人力成本，这就意味着通过自动实现代码的编写可以将工作人员从大部分繁琐的运维工作中解放出来，就像人力资源中的概念——“为了创造物质财富而投入于生产活动中的一切要素通称为资源，包括人力资源、物力资源、财力资源、信息资源、时间资源等，其中人力资源是一切资源中最宝贵的资源，是第一资源”。

作为一个跨语言的项目，很多编程语言都可以使用Selenium，如Java，但在本书中我们采用Python来介绍Selenium的用法。为什么采用Python呢？Python以其快捷著称，比如，要解决数独问题，C语言要写300~500行代码，Java只需要写100~200行代码，而Python只需要100行以内的代码。再从长远的角度来说，作为目前蓬勃兴起的一门语言，Python具有美好的前景，在科学与数字计算、网络编程和系统网络运维方面的应用非常广泛。而Selenium自动化测试与这些方面具有密不可分的关系，基于Python进行Selenium的学习，更加有利于读者未来的发展。

## 1.2 Selenium WebDriver

大名鼎鼎的Selenium在2004年由一个叫作Jason Huggins的年轻人开发。Jason Huggins在芝加哥的ThoughtWorks公司工作，他的主要职责是为内部的时间和费用（Time and Expenses）系统开发一个基于Java脚本的运行测试系统（Java Script Test Runner），作为一个推崇自动化的技术人员，他意识到相对于手工测试每次改动，他应该用一个程序来代替自己，来有效利用自己的时间。

于是他开发了一个可以驱动页面进行交互的 JavaScript 库，用于让浏览器测试后自动返回测试结果。这个库最初只是一些能够与Web页面交互的JavaScript的函数库，这个库为Selenium RC 和Selenium IDE的核心部分奠定了基础。后来随着库的不断丰富，Selenium也就逐渐发展成了一个实现模拟人类操纵浏览器的行为的项目。

讲到这里，我们要再提起Selenium的名字，如果你翻查字典，你就会惊讶发现Selenium这个单词不仅和自动化测试一点关系也没有，相反，它是一种化学元素，并且对于化学元素汞（ Mercury，元素符号Hg）有拮抗作用。然而，其中确实存在着一定的历史渊源，在早期，ThoughtWorks公司确实想过通过购买商业测试工具 Mercury系列（QTP、LR和TD），但它失败了，这家起步不久的公司难以负担起这一系列商业工具高昂的价格，因此在成功开发后，他们就开源了自己开发的项目，并且起名为Selenium（化学元素硒），来表达自己的情绪。

在2006年，Google已是Selenium的重要用户，但是Google的测试工程师们不得不绕过它的限制进行测试，因为浏览器对JavaScript资源产生的同源问题，以及浏览器对安全性方面的提高，导致Selenium Core有很大局限性。为了解决在长期以来使用Selenium遇到的问题，Google的工程师Simon Stewart 开始基于Selenium项目进行二次开发，这个项目命名为 WebDriver 。他们需要一款更先进的测试工具，这个工具能通过浏览器和操作系统的本地方法来直接和浏览器进行交互，WebDriver 项目的目标就是解决 Selenium 存在的问题。

Selenium的项目历史上存在着3个重大版本，分别是Selenium 1.0（如图1-1所示）、Selenium 2.0和Selenium 3.0。Selenium1.0版本的主要内容是Selenium RC（Remote Control），现在已经被官方正式弃用，不再推荐Selenium的用户使用Selenium RC。

可以说Selenium 1.0和Selenium 2.0最大的区别就在于WebDriver，在Selenium官方网页上有这样一个等式：Selenium 1.0 + WebDriver = Selenium 2.0

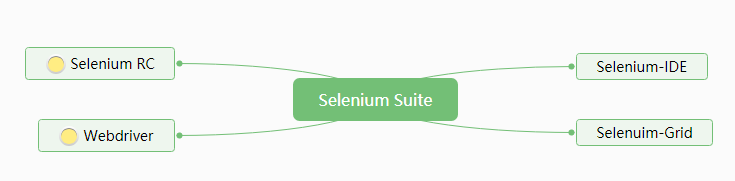
现在比较先进也广泛使用的版本是Selenium 3.0（如图1-2所示）。Selenium 3.0也包含了Selenium WebDriver，但不同的是移除了对Selenium RC核心代码的支持。

Selenium Grid和Selenium IDE（如图1-3所示）则一直独立在Selenuim的主要项目之外，作为一个分支而存在。

图1-1 Selenuim RC 图1-2 Selenium Grid 图1-3 Selenium IDE

细心的读者不难发现，三者的区别在于右上角的小标志。

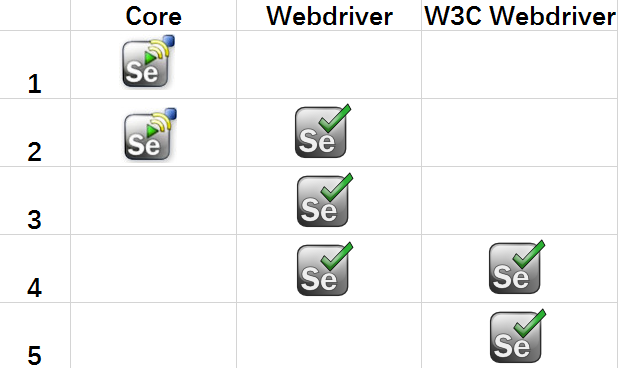
Selenium Grid 的主要目的在于允许你在多台机器上并行进行测试。换句话说，Selenium Grid 支持分布测试执行，可以让你的自动化测试在一个分布式环境中运行。

而现行的Selnium主要分为两个部分——Selenium WebDriver和Selenium IDE。前者主要用来创建基于浏览器的回归自动化套件和测试，后者主要用来创建快速的Bug重现脚本，创建脚本以帮助进行自动化辅助的探索性测试。具体的Selenium项目分支如图1.4所示。

图字修改：图1.4中Webdriver请改为WebDriver，Selenium-Grid 请改为Selenium Grid，Selenium-IDE 请改为Selenium IDE

图1-4 Selenium项目的工具分支

Selenium项目未来将更加标准化，加大对W3C标准的践行，取消对部分不符合W3C标准的WebDriver内容的支持。Selenium项目未来版本的区别及发展趋势如图1-5所示。

图1-5 Selenium版本区别及未来方向

（编辑注：图1-5中WebDriver请改为WebDriver）

## 1.3 在Windows和Linux系统上安装Selenium

工欲善其事，必先利其器。本节讨论Selenium的安装及使用。

### 1.3.1 在Windows系统上Python的安装说明

本书基于Python来讲解Selenium。在这里，假定读者使用过Python，并了解Python的基本语法、配置方法。

当然，要下载Python的Selenium库，可以在从PyPI官方网站上搜索Selenium，在弹出的页面中单击相应的链接。然而，更好的方法是使用pip来下载。可以使用以下命令来安装Selenium。

#通过pip来安装

pip install selenium

如果不巧的是，你使用的是较为老旧的Python 2.*x*版本，那么你可以通过通过手动安装pip或者easy\_install工具来简化安装。使用easy\_install安装Selenium的命令如下。

#通过easy\_install工具安装

easy\_install selenium

当你安装完成后，如果读者还不放心是否安装成功，那么可以通过以下方式来检验是否安装成功。

C:\Users\xuyichenmo>python

Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>>

没有消息就是好消息。如果你输入import selenium而没有任何提示，那么就证明你已经成功安装了Selenium。

接着，我们要下载Selenium服务器（这一项是可选的，你不一定要安装Selenium服务器，它主要用于运行Selenium IDE录制的脚本）。

因为Selenium基于Java编写，所以我们还要安装Java Runtime Environment（JRE） 1.6或者更高版本的JRE（下载地址参见Oracle官网）。

同样，作者可以在Selenium项目官方网站的一个子页面中下载Selenium Standalone Server，截至2018年4月份，Selenium Standalone Server的最新版本为3.13.0（下载页面参见seleniumhq网站）。

安装完JRE后可以通过Windows+R快捷键打开“运行”窗口，然后在“打开”选项中输入cmd，来打开命令行窗口，接着输入以下这条命令来启动Selenium服务器。

java -jar F:\Python\_ADDED\selenium-server-standalone-3.13.0.jar

这里，你需要把-jar参数后面的selenium-server-standalone文件所在的目录替换成你自己的文件所在的目录。

如果你刚刚安装完JRE后就直接运行这条命令，那么你可能会因为端口占用问题而失败。如果出现这种情况，那么你可以通过以下这几条命令来解决问题。

netstat -aon | findstr "[端口号]"

tasklist | findstr "[PID号]"

taskkill /pid [PID号] /

我们可以看到，在第一次使用调用服务器命令后Java后提示我们失败了，Selenium默认调用4444端口，而这个端口已经被占用了，在我们解除了对端口的占用后，就可以成功调用了。

下面介绍上述命令中几个选项的含义。

* --debug和-debug：布尔型，用于启用LogLevel.FINE日志文件，记录调试信息。

默认值是false。

* --version和-version：用于显示版本和退出。默认值是false
* -browserTimeout：整型的秒数，用于指定允许浏览器会话的秒数。在WebDriver命令运行时（如driver.get（url））允许会话暂停的最大秒数，当达到超时值后，会话会退出。最小值为60s，如果未指定，零或负值意味着无限期等待。
* -config：字符串形式的文件名，使用独立服务器的JSON配置文件，覆盖默认值。
* -host：字符串形式的IP地址或主机名，通常自动确定，并且在跨国网络连接的配置中有用（例如，使用VPN的网络）。
* -jettyThreads和-jettyMaxThreads：Jetty（Jetty是一个基于Java编写的网页服务器和Java Servlet容器）中允许的最大线程数。如果未指定，设置为零或负整值，就表示将使用Jetty默认值200。
* -log：字符串形式的文件名，用于记录的文件名。如果省略，将输出到STDOUT。
* -port：整数，表示服务器将使用的端口号。
* -role：字符串形式的可选项，可以设置为[hub]、[node]和[standalone]。
* -timeout和-sessionTimeout：以整数表示的秒数，在服务器自动关闭上一个没有任何活动的会话前，用于指定服务器超时的时间。然后将释放测试槽以供另一个测试使用。这通常用于处理客户端冲突。对于[hub]和[node]格式，还必须设置cleanUpCycle。

### 1.3.2 在Linux系统上Python的安装说明

当你使用Linux系统的时候，通常Linux都自带Python。当然，如果你的Linux系统安装的是2.*x* 版本的Python，这里推荐你再安装一个3.*x*版本的Python。

使用以下这条命令来下载安装Python管理工具包。

sudo apt-get install python-pip

这条命令用于安装setuptools，这条命令和下面一条命令二选一即可。

sudo apt-get install python-setuptools

这条命令用于安装pip。

sudo pip install selenium

# 第2章 页面导航

1.2节提起过Selenium项目可以使我们实现一些模拟人类操纵浏览器的行为。这些基本的行为也是我们实现更加复杂的动作的基础。本章会讨论如何实现这些行为。

## 2.1 启动浏览器

在启动浏览器之前，我们还需要一个下载一个WebDriver，那么什么是WebDriver呢？假设我们需要找个猎人陪我们一起上山打猎，那么目前我们就已经找到了我们需要的猎人，但是，我们还缺一把猎枪，WebDriver的作用就和这把猎枪很像，它就是驱动浏览器运行的一个工具。

读者可以在seleniumhq网站的下载页面中查看Selenium官方承认（但并不都是Selenium官方开发）的所有第三方驱动。

谷歌浏览器WebDriver的官方下载地址请参见谷歌网站。

如果读者无法访问谷歌的下载页面，读者可以选用国内淘宝网站的镜像或者GitHub网站下载WebDriver。

在本书中主要采用谷歌浏览器做示范。当然，代码都是相通的，如果你掌握了技巧，那么你可以轻松写出其他浏览器所对应的代码。如果它们的WebDriver有什么区别，作者也会详细指出。

下载WebDriver并解压之后，可以发现里面的内容仅为一个chromedriver.exe文件。为了方便使用，我们需要将其解压在chrome的安装目录下。

C:\Program Files (x86)\Google\Chrome\Application\

然后，配置环境变量，右击“我的电脑”，选择“属性”并打开“属性”窗口。在“属性”窗口中，单击“高级系统设置”，在打开的“系统属性”窗口（见图2-1）中，单击 “高级”选项卡中的“环境变量”按钮。在打开的“环境变量”窗口（见图2-2）中，修改“系统变量”下面的Path的值为

;C:\Program Files (x86)\Google\Chrome\Application;

请注意，该行内容的前后各有一个分号。

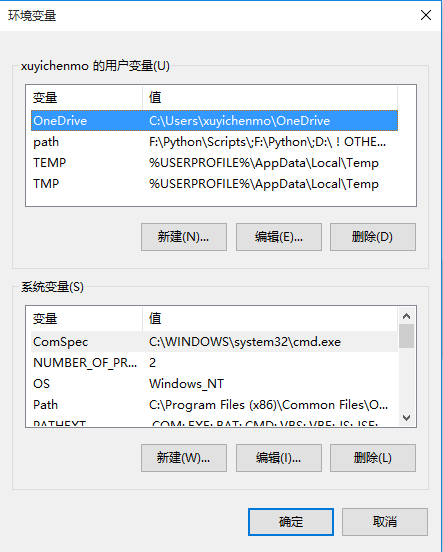


图2-1 “系统属性”窗口 图2-2 “环境变量”窗口同样，这里提供一种更为简单的方法，可以将chromedriver直接放到Python的安装目录下。在安装Python的时候已经配置过环境变量了，因此这样可以省去很多麻烦。

在Linux系统下，只需要将把下载好的chromedirver文件放在 /usr/bin 目录下就可以了。

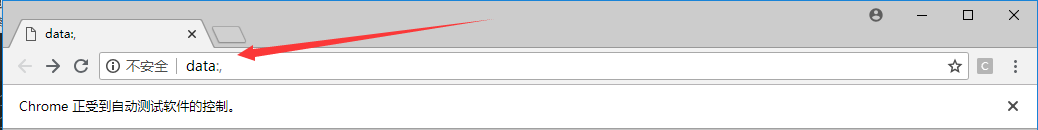
请注意的是，读者需要将浏览器版本和WebDriver版本对应，否则就会出现打开Chrome浏览器在地址栏中只显示“data:,”的情况，如图2-3所示。

图2-3 Chrome浏览器在地址栏中显示的“data:,”

安装与配置过程到此结束。

接下来，使用Python来写代码。下面给出一个示范。

#导入Selenuim库

from selenium import webdriver

#下面一行代码用于启动Chrome浏览器

driver = webdriver.Chrome()

在导入Selenium库后，仅仅用一行代码便可以打开Chrome浏览器。

driver = webdriver.Chrome()

类似地，通过以下这条命令可以打开Firefox浏览器。

driver = webdriver.Firefox()

## 2.2 打开页面

仅仅只打开浏览器，肯定不能满足我们的需求，我们还需要更进一步的操作。接下来，我们尝试打开一个页面。

#导入Selenuim库和键盘操作

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

#打开Chrome浏览器

driver = webdriver.Chrome()

#打开百度页面

driver.get("https://baidu.com")

当我们运行代码后，就会发现Chrome和以前有些不一样，在地址栏的下方，可以看到提示信息“Chrome正在受到自动测试软件的控制”，如图2-4所示。

图2-4 使用Chrome浏览器打开百度页面时显示的提示信息

运行结束后，我们成功地打开了百度页面。类似地，只要将driver.get("[url]")中的URL地址修改为我们想要打开的页面，那么就可以轻松地打开任意网站的地址（请不要忘记双引号）。

运行代码后打开的浏览器窗口默认并不是最大化的。有时候，我们并不想要这样，因此可以尝试改变这一默认值。在以下代码中，第一个参数值设置的是浏览器窗口的长，第二个参数值设置的是浏览器窗口的宽。

#设置打开浏览器窗口的长和宽

driver.set\_window\_size(800, 480)

通过另外一条命令，可以使浏览器窗口最大化。

#最大化

driver.maximize\_window()

## 2.3 浏览器基本操作

在了解完如何打开网页后，我们来看一看我们使用浏览的时候经常会使用到的几个基本操作命令——前进、后退、刷新和退出。

#导入Selenium库和time库

from selenium import webdriver

import time

#打开浏览器和百度页面

driver = webdriver.Chrome()

driver.get("https://baidu.com")

#在停留两秒后打开百度新闻

time.sleep(2)

driver.get("http://news.baidu.com")

#在停留两秒后后退

time.sleep(2)

driver.back()

#在停留两秒后前进

time.sleep(2)

driver.forward()

#在停留两秒后刷新

time.sleep(2)

driver.refresh()

#在停留两秒后关闭浏览器

time.sleep(2)

driver.quit()

细心的读者可能会注意到这里比前面多导入了一个time库，同时，在每个操作前都有一个time.sleep()，这条命令用于使程序停顿一定时间后再执行。之所以这样，不仅是为了让读者看清楚我们的操作，还为了给浏览器指定操作网页的时间。

我们再来看一看在这里使用的命令。

#前进

driver.back()

#后退

driver.forward()

#刷新

driver.refresh()

#退出

driver.quit()

这些都是比较常规的操作，读者可以尝试自己写一个小程序，多调用几次这些函数，来熟悉它们。

## 2.4 拖动窗口

在实际中，可能会碰到这种情况，我们打开了一个页面，但是这个页面比较长，我们需要拖动浏览器的滚动条到底部，然后跳转到下一步页面。为了应对这种情况，调用滚动条的相关的命令在Selenium库中是没有的，但我们可以通过使用JavaScript可以模拟操作。为了使用JavaScript，Selenium为我们提供了这条命令。

driver.execute\_script([JavaScript\_name])

我们再来考虑一下解决问题的思路，这是一行JavaScript代码，用于向下滑动到页面底部（如果你看不懂也没关系，毕竟我们使用的不是JavaScript）。

js="var q=document.documentElement.scrollTop=10000"

driver.execute\_script(js)

接下来，我们演示一下具体使用方法。

#导入Selenium库和time库

from selenium import webdriver

import time

#打开浏览器和Python官方网站

driver = webdriver.Chrome()

driver.get("https://baidu.com")

#停留两秒

time.sleep(2)

#执行JavaScript代码

JS="window.scrollTo(10000,document.body.scrollHeight)"

driver.execute\_script(JS)

#停留两秒后退出

time.sleep(2)

driver.quit()

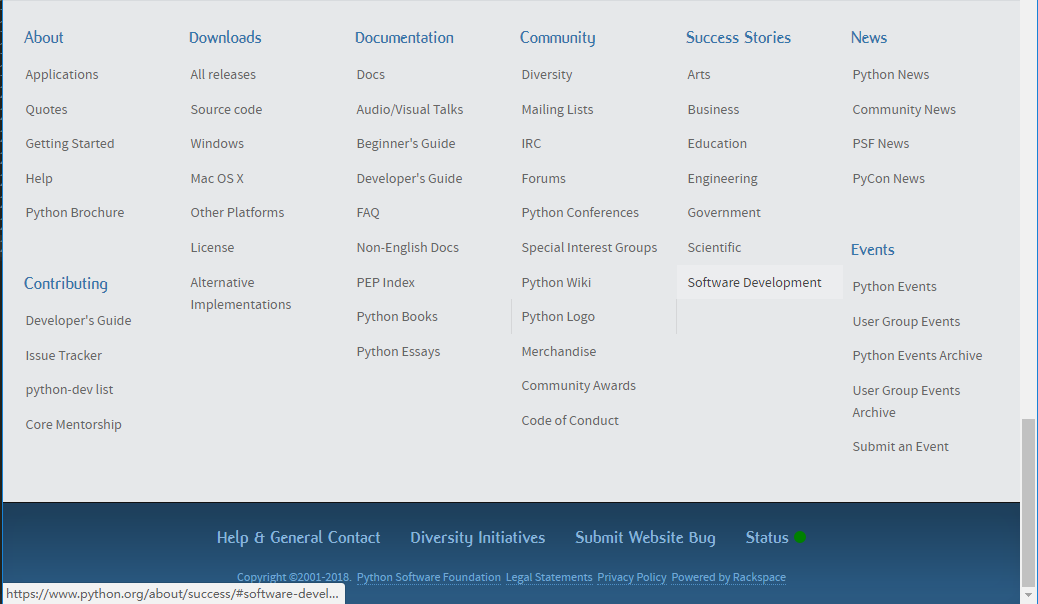
执行结果如图2-5所示。

图2-5 执行结果

已经成功跳转到页面的底部了，这时候，我们就可以放心地进行下一步的功能测试了。

对于某些版本的谷歌浏览器来说，这种方法并不能用，我们不得不采用寻找元素的方式，来进行页面的下滑。

#导入Selenium和time库

#ActionChains是一个用来模拟鼠标操作的库

from selenium import webdriver

import time

from selenium.webdriver import ActionChains

#以今日头条的首页为例

#利用将鼠标移动到最后一个元素的位置并模拟下拉

driver=webdriver.Chrome()

driver.get("https://www.toutiao.com/")

#保存现在的页面

driver.save\_screenshot("save\_1.png")

#模拟鼠标操作来实现下拉

ac=driver.find\_element\_by\_xpath("//ul[@infinite-scroll-disabled]/li[last()]")

#定位到指定元素

ActionChains(driver).move\_to\_element(ac).perform()

#停留2s，也就是给加载内容预留2s，然后第二次保存现在的页面

time.sleep(2)#

driver.save\_screenshot("save\_2.png")

运行代码并对比两次获取的截图后，就会发现窗口位置确实发生变化了。另外，这种方法对于各种浏览器都是有效的。

## 2.5 cookie的调用

对于cookie的调用方式，Selenium项目提供了成熟的解决方案。下面是一些常用的cookie调用方法。

* get\_cookies()：获得所有cookie信息。
* delete\_all\_cookies()：删除所有cookie信息。
* get\_cookie([name])：返回字典的key为[name]的cookie
* add\_cookie(cookie\_dict)：添加cookie。“cookie\_dict”指字典对象，必须有name和value两个值。
* delete\_cookie([name],[optionsString]) ：删除cookie信息。[name]是要删除的cookie的名称。第二个参数[optionsString]是cookie选项，目前支持的选项包括“路径”和“域”。

下面给出几个例子。

#导入Selenium、time和ActionChains库

from selenium import webdriver

import time

from selenium.webdriver import ActionChains

#打开并跳转到百度页面

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#这里通过查找元素实现搜索

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

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

#获取所有cookie

cookies=driver.get\_cookies()

#返回字典的key为的BAIDUID的cookie

cookie=driver.get\_cookie("BAIDUID")

print(cookies)

print(cookie)

#删除所有cookie信息。

driver.delete\_all\_cookies()

#退出

driver.quit()

请读者注意下面这段代码。

driver.get ("http://www.youdao.com")

# 向cookie的name和value中添加会话信息

driver.add\_cookie({"name":"testname\_1234567890"},{"value":"testvalue\_1234567890"})

看一下输入结果。

看一下输出结果。

[{'domain': '.baidu.com',

'httpOnly': False,

'name': 'H\_PS\_PSSID',

'path': '/',

'secure': False,

'value': '1451\_21078\_18560\_26350\_20930'},

{'domain': '.baidu.com',

'httpOnly': False,

'name': 'delPer',

'path': '/',

'secure': False, 'value': '0'},

{'domain': '.baidu.com',

'expiry': 3685852075.738426,

'httpOnly': False,

'name': 'BAIDUID',

'path': '/',

'secure': False,

'value': '4D078E74C7513305D07351785B50212A:FG=1'},

……

{'domain': '.baidu.com', 'expiry': xxxx, 'httpOnly': False, 'name': 'BAIDUID', 'path': '/', 'secure': False, 'value': 'xxxxx'}

返回的cookie是字典形式的。

下面看一看add\_cookie的用法，add\_cookie必须有name和value的值，那需要直接传递参数吗？作者通过实验验证了一下，并不是这样的。

请注意，name和value虽然是两个值，但需要放在一个字典中，不能分开写，否则会报错。报错提示我们需要两个参数，但我们传了3个参数，我们再来检查一下，确实是两个参数，这没错啊！难道是编译器编译错误？也不可能。

回想一下，我们在使用Python编写类的时候，一般默认是加一个Self参数的。很明显，问题出在这里，这个多出来的参数就是Self，因此才会提示我们给出的参数过多。

而如果我们像下面的代码这样使用，那么代码就可以正常运行了。

driver.add\_cookie({"name":"testname\_1234567890","value":"testvalue\_1234567890"})

## 2.6 使用多个窗口

在使用自动化测试工具的过程中，我们不得不面对需要打开一个新的窗口但当前的窗口暂时还不能关闭的情况。就像我们浏览网页的时候，总是希望多打开页面，到最后，地址栏上可能会有十几个页面。

因此，作为一个功能齐全的自动化测试工具，Selenium虽然自身并不提供这一功能，但是我们可以换一种思路来解决—— 通过JavaScript代码。

下面是作者提出的一种解决方法，供读者参考。

JS='window.open("https://www.sogou.com");

driver.execute\_script(JS)

下面给出一个示例。

#采用utf-8编码

from selenium import webdriver

import time

#打开浏览器，最大化窗口

driver=webdriver.Chrome()

driver.maximize\_window() #

driver.get("http://baidu.com")

#停留2s后打开搜狗浏览器进行搜索

time.sleep(2)

JS1='window.open("https://www.sogou.com");'

driver.execute\_script(JS1)

#停留2s后打开有道翻译

time.sleep(2)

JS2='window.open("https://fanyi.youdao.com/");'

driver.execute\_script(JS2)

#退出

#driver.quit()

在这个示例中，我们首先打开并且最大化窗口，在停留一段时间后，分别打开搜狗搜索和有道翻译，退出浏览器。

## 参考网站

* 历史版本Google浏览器驱动的下载，请参见googleapis网站。
* 历史版本firefox浏览器驱动下载，请参见mozilla网站。
* 历史版本chrome浏览器下载，请参见downloadastro网站。
* Selenium WebDriver 2.0包，请参见googleapis网站。

Chromedriver和支持的Chrome版本对照表如表2-1所示。

表2-1 版本对照关系

# 第3章 元素定位

在计算机领域中，页面的最基本组成单元为元素，要确定一个元素，我们需要特定的信息来说明这个元素的唯一特征。这与坐标系很类似，横坐标和纵坐标可以确定平面内唯一一个点。

本章将介绍Selenium的8种元素定位方式。通过学习本章与前两章，读者可以实现一个略有雏形的自动化程序。

## 3.1 元素定位概述

在读者开始学习本章之前，我们需要首先了解一下什么是HTML元素。简单来说，HTML元素就是HTML文件的一个基本组成单元，HTML元素涵盖的内容包括但不仅限于文字、图片、音频、动画和视频。

举个例子，一个HTML页面就相当于一个公司，我们要在一个庞大的公司中定位一个人是很困难的，因为我们无法记住公司中所有的人，所以需要一种特殊的方法——工牌来确定一个人所属的部门。而Selenium给我们提供的方法的作用就和工牌很类似。

Selenium提供了8种定位方式。

* 根据ID定位
* 根据name定位
* 根据Xpath定位
* 根据标签名定位
* 根据类名定位
* 根据CSS选择器定位
* 根据链接文本定位
* 根据部分链接文本定位

这几种定位方式很像通过不同的方面来说明一个人，比如，一个人在家里叫小明，而他在外面叫张明明，在班级中他又是班长，当某个人称呼他小明、张明明、班长的时候，我们都知道是他。

这8种定位方式在Python Selenium中所对应的方法如下。

* find\_element\_by\_id()
* find\_element\_by\_name()
* find\_element\_by\_xpath()
* find\_element\_by\_tag\_name()
* find\_element\_by\_class\_name()
* find\_element\_by\_css\_selector()
* find\_element\_by\_link\_text()
* find\_element\_by\_partial\_link\_text()

在元素定位的过程中，我们也会遇到很多困难，比如，前端设计师常常会使用很多方式来隐藏网页中的元素。常见的隐藏方式有以下几种。

* 将 opacity 设置为 0。
* 将 visibility 设置为 hIDden。
* 将 display 设置为 none 。
* 将 position 设置为 absolute然后将位置设置到不可见区域。

通常，这些方式也是为了防止网络爬虫而设置的。假设网站上一个position设置为absolote并且位于真实用户不可见区域的表单，那么当你提交了这个表单时，网站就有足够的证据判断你并不是真正的用户，紧接着服务器就会拒绝你的请求。

## 3.2 根据 ID 定位

HTML属性中的ID是很重要的，这些ID就相当于一个人的身份证。具有特定ID值的HTML元素可以被CSS（Cascading Style Sheet，层叠样式表）选择器和JavaScript脚本查找到，并执行某种特定的功能。

为了便于区分，在HTML页面中我们一般不会设置ID相同的HTML元素。另外，如果HTML元素ID重名，会有相当严重的后果。ID选择器具有唯一性。虽然重复的ID对应的CSS样式都会生效，但是如果用JavaScript去获取相应ID的元素，就会出错，因为取ID只能一个。

为了便于区分，在HTML页面中我们一般不会设置ID相同的HTML元素，否则会有相当严重的后果。

下面以百度搜索为例，来说明如何找到程序的ID。

下面以百度搜索为例，来说明如何根据ID定位元素。

下面是百度网站中搜索框的代码。

<input type="text" class="s\_ipt" name="wd" id="kw" maxlength="100" autocomplete="off">

重点在这里，id="kw"，我们就从这里知道了，百度网站中搜索框的ID为kw，以后我们就可以找到这个ID了。

再来看一下“百度一下”按钮的ID。

<input type="submit" value="百度一下" id="su" class="btn self-btn bg s\_btn">

通过上面这段代码我们可以看到，id="su"。接下来，我们进行一个实际操作。

#导入库

from selenium import webdriver

import time

#打开浏览器，打开百度

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#通过ID查找，然后输入，并单击

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

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

#退出浏览器

driver.quit()

运行结果如图3-1所示。

图3-1 通过ID在百度中搜索

代码成功运行，但这里的重点不是重复输入代码。下面介绍原理。在上面的代码中，，在进行导入库和打开浏览器等一系列常规操作后，首先查找id="kw"的搜索框，其次通过.send\_key()输入（.send\_key()是一个方法，意思是向搜索框发送特定的字符串，请不要忘记在特定字符串的前后要加上双引号）。

再次，再次查找ID="su"的搜索框，然后使用.click()方法进行单击（.click()同样也是一个方法，用于向某个特定的HTML元素发送确认请求）。

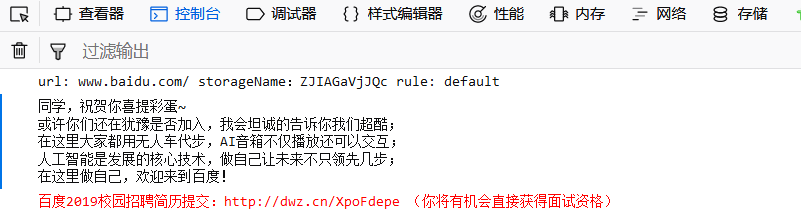
网页的源代码一般很长，我们一打开就会感到不知所措，这里凭什么找到搜索框对应的这一段代码？其实，一般学过前后端开发的人都会对浏览器中的开发者工具（Developer Tool）有所了解，我们在浏览器除了标签页外的任何页面中使用F12键就可以打开开发者工具了。开发者工具栏如图3-2所示。

图3-2 开发者工具栏

（编辑注：图中有错别字，请修改：坦诚的——坦诚地，做自己——后面请加逗号，不只——不止 ）

默认打开后显示的是“查看器”选项卡，但在“控制台”选项卡中也可能会看到这个网站的提服务供商想要给开发者看的一些信息，比如，图3-2中的百度招聘信息。

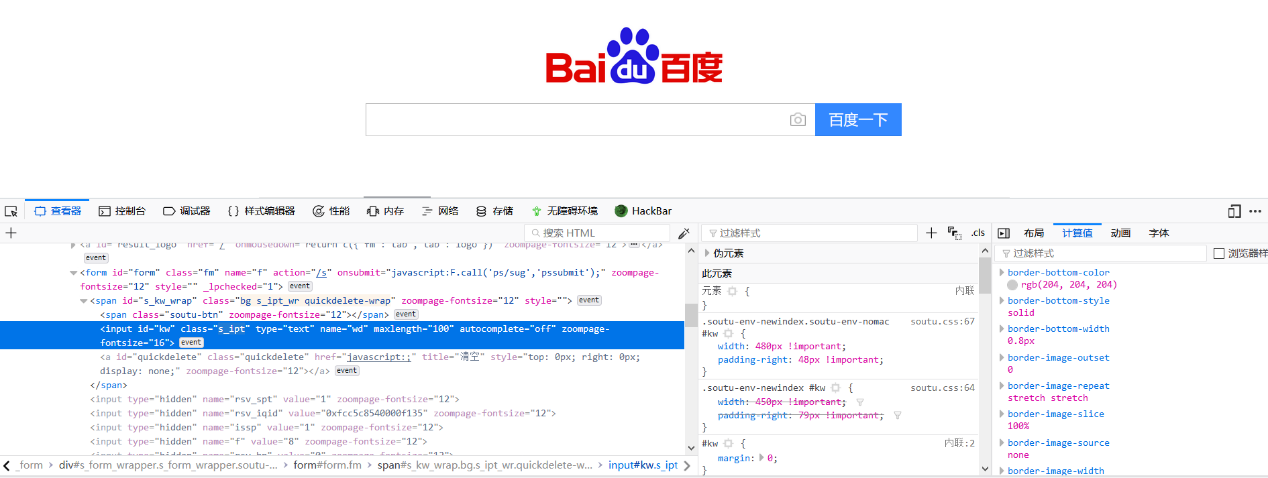
然后我们在开发者工具栏的左上角可以看到一个类似于鼠标箭头的图标，我们单击后就可以看到，选中的代码变为蓝色的，而我们的鼠标箭头移动到哪里并单击确认后，在下方就可以看到对应的源代码了如图3-3。

图3-3 元素定位器

最后在相应的代码行中，单击前方的三角形就可以展开代码，右击，从上下文菜单中选择“作为HTML编辑”，或者直接按F2键可以进入编辑状态。

## 3.3 根据 name定位

本节介绍另外一种元素定位方法——通过name定位。绝大部分做过Web开发的人都问过两个问题：元素的ID和name的区别在哪里？为什么有了ID还要有name呢？实际上， 前面已经解释过部分原因，ID就像是一个人的身份证号码，而name就像是他的名字，ID的值显然是唯一的，而name的值是可以重复的。

类似地，Selunium也有一个根据name查找元素的方法，即find\_element\_by\_name()。

我们用find\_element\_by\_name()来实现刚才的功能，再次查看前面的代码。

<input type="text" class="s\_ipt" name="wd" id="kw" maxlength="100" autocomplete="off">

这个HTML标签有个一个属性—— name="kw"。很明显，这个wd就是我们需要查找的。

我们通过下面这一行代码来实现相同的功能。

driver.find\_element\_by\_name("wd")

因为单击按钮所在的HTML标签并不存在name属性，所以我们无法通过查找name属性来找到这个元素。

修改：因为按钮“百度一下”所在的HTML标签并不存在name属性，所以我们无法通过查找name属性来找到这个元素。

同样，下面这段代码可以正常运行。

#导入库

from selenium import webdriver

import time

#打开浏览器，打开百度

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#通过name查找，然后输入

driver.find\_element\_by\_name("wd").send\_keys("selenium")

#通过id查找并单击

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

time.sleep(2)

#退出浏览器

driver.quit()

## 3.4 根据XPath定位

XPath是多种多样的，即便使用不同的方式书写，你也可以查找到相同的元素。比如，要查找百度的搜索框，就有下面7种写法。

* //\*[@id='kw']
* //\*[@name='wd']
* //input[@class='s\_ipt']
* /html/body/form/span/input
* //span[@class='soutu-btn']/input
* //form[@id='form']/span/input
* //input[@id='kw' and @name='wd']

现在，我们通过一种简单的方式来获取HTML——浏览器的开发者工具。

首先，按照前面的方式，打开浏览器的开发者工具，然后将其定位到搜索框上。这时，开发者工具应该已经定位到相应的代码上，选中这行代码（如果开发者工具的代码框被遮住，那么可以通过上下滑动来适当地展开，或者关掉弹出的Console和What’s New对话框）。然后右击，从上下文菜单中选择Copy→Copy XPath，我们就可以获取到由机器生成的XPath，如图3-4所示。

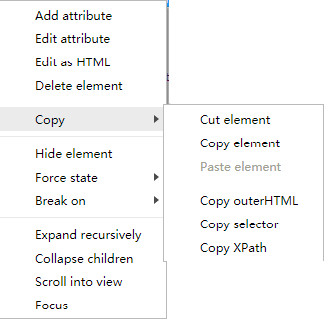


图3-4 获取Xpath

在获取到XPath后，我们使用XPath来查找元素。

#导入库

from selenium import webdriver

import time

#打开浏览器，打开百度

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#通过名称查找，然后输入

driver.find\_element\_by\_xpath(r'//\*[@id="kw"]').send\_keys("selenium")

#通过id查找并单击

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

time.sleep(2)

#退出浏览器

driver.quit()

和刚才有一点不一样，这里最好将XPath中的双引号和我们用来表明这是字符串的前后引号区分开。

driver.find\_element\_by\_xpath("//\*[@id="kw"]").send\_keys("selenium")

如果这样使用，编译器就会提示语法错误。

File "C:\Users\xuyichenmo\OneDrive\Documents\SELENIUM\test.py", line 10

driver.find\_element\_by\_xpath("//\*[@id="kw"]").send\_keys("selenium")

^

SyntaxError: invalid syntax

最好使用下面的写法。

driver.find\_element\_by\_xpath(r'//\*[@id="kw"]').send\_keys("selenium")

使用r’’（raw string，原始字符串常量）来转义，同时这个单引号和XPath的双引号区分开，避免混乱。

## 3.5 根据标签名定位

标签是HTML页面中一个重要的组成部分。假设我们正在建房子，标签的作用就相当于砖头，不同的标签名就相当于不同种类的砖头，它们有的大小存在差异，有的颜色存在差异。然而，即便同一种类的砖头，也会同时存在很多块。

我们以这个标签为例。

<em class="show-city-name" data-key="郑州"> 郑州： </em>

这里，em就是标签名，而后面的class="show-city-name" data-key="郑州"则是这个标签的属性，一个标签可以有多个属性，但只能有一个标签名。

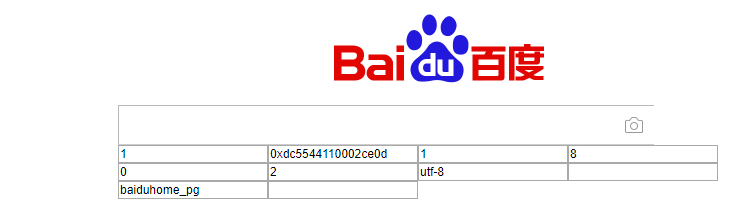
接下来，我们学习标签名定位。这种定位方法堪称最不靠谱的定位方法，因为在一个HTML页面中可能会有很多的相同的标签。比如，我们常见的百度搜索其实并不是只有一个搜索框（这些搜索框各有各的作用，虽然它们采用了Input标签，但是它们并不是真的用来搜索的），只是其他搜索框的属性设置为隐藏，用户无法看到，在这里将这些搜索框的Hidden属性删掉，如图3-5所示。

图3-5 百度中隐藏的搜索框

可想而知，如果我们采用标签名搜索这种不靠谱的方式，注定会失败。读者只需要知道存在这么一种定位方式即可，在实际的自动化测试实践中，我们用到这种方式的机会并不多，很多时候，我们都会采用其他几种方式。

虽然明知道会失败，但是我们还要进行一次元素定位。

#导入库

from selenium import webdriver

import time

#打开浏览器，打开百度

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#通过name查找，然后输入

driver.find\_element\_by\_tag\_name("input").send\_keys("selenium")

可能读者会问：为什么不写完呢？写到这里，这段代码已经必定会报错了，没有再往下写的必要了。我们看一下运行结果。

Traceback (most recent call last):

File "C:\Users\xuyichenmo\OneDrive\Documents\SELENIUM\test.py", line 10, in <module>

driver.find\_element\_by\_tag\_name("input").send\_keys("selenium")

File "F:\Python\lib\site-packages\selenium\webdriver\remote\webelement.py", line 479, in send\_keys

'value': keys\_to\_typing(value)})

File "F:\Python\lib\site-packages\selenium\webdriver\remote\webelement.py", line 628, in \_execute

return self.\_parent.execute(command, params)

File "F:\Python\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 320, in execute

self.error\_handler.check\_response(response)

File "F:\Python\lib\site-packages\selenium\webdriver\remote\errorhandler.py", line 242, in check\_response

raise exception\_class(message, screen, stacktrace)

selenium.common.exceptions.ElementNotVisibleException: Message: element not visible

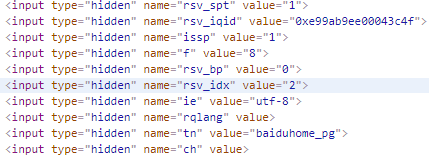
网页源代码中的对应部分如图3-6所示。

图3-6 隐藏的输入标签

元素不可见，而百度的搜索页面又设置了很多隐藏的输入标签，我们有理由推测Selenium找到的是一个隐藏的输入框。

## 3.6 根据类名定位

作为HTML标签的属性，Class规定元素的类名，绝大多数的class标签都指向CSS中的类。

可以同时给一个HTML标签附上多个类，比如，通过 <span class="left\_menu important">，同时将left\_menu和important赋给span标签（中间的空格叫作间隔符号，表示一个标签有多个class的属性名称）。

因为Selenium现在的版本已经不再支持复合类名（传入的参数是一个类名，而这里传入的是一个复合的类名），所以对于class="btn self-btn bg s\_btn"这样的复合类名，是无法通过类名来查找元素的。

那怎么解决呢？

方法一：删除其他的类名，只传入一个参数，比如，通过以下代码查找元素。

driver.find\_element\_by\_class\_name("s\_btn").click()

方法二：结合刚才学到的find\_elements\_by\_tag\_name()来解决这个问题，大致思路如下。

elements = driver.find\_elements\_by\_tag\_name('input')

for element in elements:

className = element.get\_attribute('class')

print(className)

if className == 'bg s\_btn':

subscript=elements.index(element)

elements[subscript].click()

这里使用的是find\_elements\_by\_tag\_name()，这是一个复数级别的查找，查找的是具有特定标签名的标签，这些标签会以一个列表的形式返回，但你无法看到具体内容。返回的列表如下。

[<selenium.webdriver.remote.webelement.WebElement

(session="5b11658be6d36431aec564cb8340b82d",element="0.07218954533142252-2")>,

<selenium.webdriver.remote.webelement.WebElement (session="5b11658be6d36431aec564cb8340b82d",element="0.07218954533142252-3")>, <selenium.webdriver.remote.webelement.WebElement (session="5b11658be6d36431aec564cb8340b82d",element="0.07218954533142252-4")>, <selenium.webdriver.remote.webelement.WebElement (session="5b11658be6d36431aec564cb8340b82d",element="0.07218954533142252-1")>]

列表中的内容以Selenuim自身存储定位元素的方式保存。因为这样的列表是难以直接使用的（在列表中添加、删除内容），所以我们考虑使用index() 函数来从列表中找出这个值的第一个匹配项的位置，然后将其赋值给一个变量。接着，从列表中查找这个变量值对应的内容，再使用click()方法。

针对方法一，写出以下程序。

#导入库

from selenium import webdriver

import time

#打开浏览器，打开百度

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#根据类名查找搜索框

driver.find\_element\_by\_class\_name("s\_ipt").send\_keys("selenium")

#根据ID查找搜索按钮

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

#退出

time.sleep(2)

driver.quit()

针对方法二，写出以下程序。

#导入库

from selenium import webdriver

import time

#打开浏览器，打开百度

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#根据类名查找搜索框

driver.find\_element\_by\_class\_name("s\_ipt").send\_keys("selenium")

elements = driver.find\_elements\_by\_tag\_name('input')

print(elements)

for element in elements:

className = element.get\_attribute('class')

print(className)

if className == 'bg s\_btn':

subscript=elements.index(element)

elements[subscript].click()

#退出

time.sleep(2)

driver.quit()

## 3.7 根据CSS选择器定位

在 CSS 中，选择器是一种模式，用于选择特定元素样式的模式。

类似于XPath定位，CSS选择器也有很多种定位元素的方式。以下均为定位百度搜索框的CSS选择器定位。

* "#kw"
* "html > body > form > span > input"
* "form#form > span > input"
* "span.soutu-btn> input#kw"
* ".s\_ipt"
* "[name=wd]"

这里讲解几种比较方便的书写方法。

* [name=wd]，选择所有name属性为wd的元素（虽然这个页面只有一个）。
* #kw，选择ID为kw的元素（ID是唯一的，只能选择一个）。
* a[src^="http"]，选择所有以http作为src开头的元素。
* "html > body > form > span > input"，从标签html中选出标签body，依次类推，直到选择input标签
* ".s\_ipt"，表示选择所有class属性为s\_ipt的标签，不要忽略了前面的点，前面的点才是关键的
* "span.soutu-btn> input#kw"这种是复合写法，首先查找到class属性中含有soutu-btn的span标签，然后从span标签的子元素中查找ID为kw的input标签。

同样，用不同的方式将刚才的代码重写一遍。

#导入库

from selenium import webdriver

import time

#打开浏览器，打开百度

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#根据类名查找搜索框

driver.find\_element\_by\_css\_selector("#kw").send\_keys("selenium")

driver.find\_element\_by\_css\_selector("#su").click()

#退出

time.sleep(2)

driver.quit()

因为搜索框和确定按钮的ID都唯一，所以我们可以直接采用#kw和#su的写法。

## 3.8 根据链接文本定位

虽然介绍了这么多定位方法，但在作者看来，根据ID和类名定位最方便。下面介绍的这种定位方法最实用。我们换一个目标，不再是搜索框和确认按钮，转到右上角的链接。

根据链接文本定位的是带超链接的文字。

与前面几种方法类似，只需要将文字放入参数的位置即可。

find\_element\_by\_link\_text(" [text] ")

代码如下。

#导入库

from selenium import webdriver

import time

#打开浏览器，打开百度

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#打开新闻

time.sleep(2)

driver.find\_element\_by\_link\_text("新闻").click()

driver.back()

#打开地图

time.sleep(2)

driver.find\_element\_by\_link\_text("地图").click()

driver.back()

#打开视频

time.sleep(2)

driver.find\_element\_by\_link\_text("视频").click()

driver.back()

#退出

time.sleep(2)

driver.quit()

## 3.9 根据部分链接文本定位

部分文字的定位主要用于网站的文字会经常变化但读者可以确定包含某个特定字符的情况下。当然，如果文本很长，而读者不想要输入全部的字符，就会用到这个定位方法。

find\_element\_by\_partial\_link\_text(" [text] ")

参数的字符串可以简单到只有一个字符。如果读者乐意输入全部的字符也是可以的，集合本身也是集合的子集。如果读者乐意每次都多输入几个字母，那么完全可以放弃上面的根据部分链接文本定位的方法。

这里的实现代码与上面的代码类似。

#导入库

from selenium import webdriver

import time

#打开浏览器，打开百度

driver=webdriver.Chrome()

driver.get("https://www.baidu.com/")

#打开新闻

time.sleep(2)

driver.find\_element\_by\_partial\_link\_text("新").click()

driver.back()

#打开地图

time.sleep(2)

driver.find\_element\_by\_partial\_link\_text("地").click()

driver.back()

#打开视频

time.sleep(2)

driver.find\_element\_by\_partial\_link\_text("视").click()

driver.back()

#退出

time.sleep(2)

driver.quit()

# 第4章 利器——PhantomJS

通常，从HTML代码中寻找所需要的数据确实是一个普遍的思路。然而，一旦面对十分复杂的页面，使用JavaScript渲染，这项工作就变得繁重而富有挑战性，需要我们逐个地分析后台请求。脚本式的操作又会干扰我们的正常操作。

所以，我们需要有一好用的工具来帮助我们像浏览器一样渲染JavaScript处理的页面。PhantomJS便是这样一个工具。PhantomJS是一个可以用于模拟用户手动摸索JavaScript渲染的一些结果的隐形浏览器。

## 4.1 PhantomJS

作为一个广泛而强大的网络数据采集框架，Selenium在网站自动化测试上得到了广泛的应用。遗憾的是，Selenium自身不携带浏览器，需要和Chrome等第三方浏览器一起使用。目前Selenium支持Chrome、Firefox、Opera等主流浏览器。如果你使用这些浏览器，你可以明显地看到一个个页面的打开，然后执行你设定的操作。

但我们不总是希望自己在使用Selenium编写的程序的时候什么都不干，不希望程序占用计算机的资源，这时候，你可能需要的是Selenium 和 PhantomJS。

PhantomJS便是这样一个工具，它有助于布局和元素多样化的网页。PhantomJS是一个无界面的并且可以通过脚本编程的WebKit浏览器引擎。它支持多种原生的Web 标准，包括DOM 操作、CSS选择器、Canvas、SVG和JSON等。

PhantomJS的主要应用场景是Web 测试、自动化页面访问、屏幕捕获和网络监控。

下面是 PhantomJS的开发者对这个项目的介绍——“Full web stack, No browser required. PhantomJS is a headless WebKit scriptable with a JavaScript API. It has fast and native support for various web standards: DOM handling, CSS selector, JSON, Canvas, and SVG.”（译文：完整的Web栈，不需要浏览器。 PhantomJS是一个可以用JavaScript API编写脚本的无头WebKit本。 它支持各种Web标准，包括DOM处理、CSS选择器、JSON、Canvas和SVG。）

## 4.2 下载与安装PhantomJS

可以从PhantomJS官方网站下载PhantomJS。单击PhantomJS官方网站下方的Install链接，会跳转到下载页面。该页面详细列举了PhantomJS在各种平台上的下载方法。

### 4.2.1 在Windows系统上安装PhantomJS

截至2018年2月4，PhantomJS在Windows上支持的最新版本为2.1.1。这里，建议读者使用最新版本的PhantomJS，官方文档中提到，如果在使用老版本时碰到一些难解的Bug，可以升级到最新版试试。

在Windows系统上安装PhantomJS的方法如下。

（1）直接下载phantomjs-2.1.1-windows.zip，并解压。

（2） 将bin文件夹中的可执行文件phantomjs.exe的路径添加到环境变量后(可能需要重启机器才能生效)。

提示：要把路径添加到环境变量中，右击“我的电脑”，选择“属性”并打开“属性”窗口。在“属性”窗口中，单击“高级系统设置”，在打开的“系统属性”窗口中，单击 “高级”选项卡中的“环境变量”按钮。在打开的“环境变量”窗口）中，修改“系统变量”下面的Path。如果读者采用的Windows7，需要注意的是在最末尾加上一个分号；如果采用的是Windows10系统，直接单击“新建”按钮即可。

### 4.2.2在Linux系统上安装PhantomJS

在PhantomJS官网的左上角有一个“fork me on GitHub”的超链接。通过此链接，读者可以直接找到PhantomJS位于GitHub的仓库，下载已经编译好的二进制文件安装包。截止2018年2月4日，PhantomJS在Linux系统上支持的最新版本为phantomjs-2-1-3。

在Linux系统上安装PhantomJS的方法如下。

（1）打开PhantomJS官网的下载页面，下载phantomjs-2-1-3-linux-x86\_64.tar.bz2。

（2）进入安装目录，解压二进制文件。

“cd /usr/local” “tar zxvf phantomjs-1.9.8-linux-x86\_64.tar.bz2”。

1. 创建软链接MySQL指向解压出来的文件夹，或将解压出来的文件夹重命名为PhantomJS。
2. ln -sf phantomjs-2-1--linux-x86\_64/bin/PhantomJS赋给PhantomJS执行权限。

由于WebKit模块中有数千个文件，因此由源代码编译PhantomJS会花费很长的时间，而且需要大量的磁盘空间。作者通过4个并行的进程进行编译，花费了超过20分钟的时间。这里建议读者采用另一种安装方式：直接下载二进制文件然后安装。待读者熟练了，可以尝试采用这种较为复杂的安装方式。

### 4.2.3 检验安装

读者可以采用以下的方式来检验是否安装成功。按下win + r组合键（win键是指键盘左下角的画着微软视屏图标的键位），输“cmd”，在弹出的窗口中输入PhantomJS –v，如果返回的是你下载的版本号，证明已经成功安装，否则，读者应返回检查上一步各项。

## 4.3 配置相应的WebDriver

WebDriver 和 Selenium 本是两个独立的项目，依靠不同的实现方式。Selenium 2 将WebDriver 和 Selenium结合，解决了原先难以解决的问题，极大地方便了使用和获取各种统计学数据，提供了PhantomJS可以实现的一些功能，将其纳入Python的可使用范围（PhantomJS是使用Java语言编程开发的）。

Selenium是Python的一个第三方自动化测试库。Selenium是测试库，它却非常适合用来写爬虫。而PhantomJS是其子包Webdriver下面的一个浏览器接口。除了PhantomJS浏览器之外，Webdriver还整合了Chrome、Firefox、IE等浏览器，并提供了操作这些浏览器的接口。

因PhantomJS是无头浏览器（headless browser），也称无界面浏览器，所以它不需要界面，同时占用的内存也相对较少，更适用于大规模的数据爬取（打开几十个Chrome进程爬取数据，真是内存的噩梦）。

简单并且容易上手的Selenium库，是爬动态网页的杀手级武器，但Selenium库对PhantomJS浏览器的支持还不是特别完善。当然，了解了存在的问题，并找到对应的解决方法，就能发挥PhantomJS强大的功能了。

大多数时候你需要的只是两行代码。

from selenium import webdriver

obj = webdriver.PhantomJS()

## 4.4 第一个PhantomJS小程序

如果你已经安装好了Selenium，下面通过一个简单的例子来阐明Selenium + PhantomJS的基本使用方法。在这个例子中，首先创建PhantomJS浏览器对象，然后打开Bing搜索，接下来打印出页面标题，接着在搜索框中输入字符串并截图，单击“搜索”并截图，打开搜索结果的第一个页面。整个示例的流程如图4-1所示。

在Selenium中PhantomJS的主要功能类似于其他的WebDriver，最大的特点在于使用时不会出现浏览器窗口弹出操作占用顶层窗口的情况，也不需要获取焦点。

可以把下面的Python代码手动输入到你的编辑器中（自己录入程序远比复制和粘贴效果好）。读者可以一行一行代码添加和删除注释，查看该行代码的运行效果。

在这里，读者学习的重点在于观察PhantomJS的实现效果和WebDriver支持的其他浏览器运行效果的异同点。

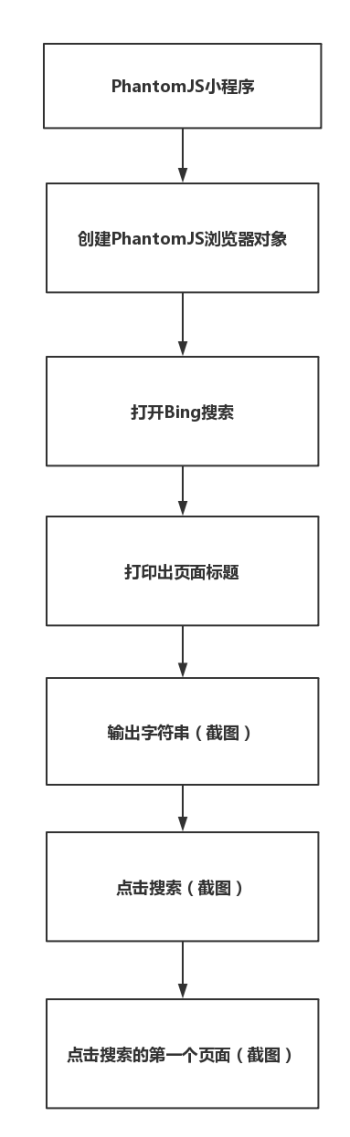


图4-1 示例的流程

编辑注：图4-1中的“点击”请改为“单击”，“打印出页码标题”请改为“输出页面标题”

# 导入 WebDriver

from selenium import webdriver

import time

# 要调用键盘按键操作，需要导入keys包

from selenium.webdriver.common.keys import Keys

# 调用环境变量指定的PhantomJS浏览器创建浏览器对象

driver = webdriver.PhantomJS()

# get方法会一直等到页面完全加载，

#然后才会继续执行程序，通常测试会在这里选择 time.sleep(2)

driver.get("http://cn.bing.com/")

#输出页面标题

title\_=driver.title

print(title\_)

# id="sb\_form\_q"是bing搜索输入框，输入字符串"SELENIUM自动化测试"

keywords = 'SELENIUM自动化测试'

driver.find\_element\_by\_id("sb\_form\_q").send\_keys(keywords)

#生成当前页面快照并保存

driver.save\_screenshot("bing\_1.png")

#单击"搜索"按钮

driver.find\_element\_by\_id("sb\_form\_go").click()

#生成当前页面快照并保存

driver.save\_screenshot("bing\_2.png")

#进入当前页面并单击第一项

driver.find\_element\_by\_xpath("/html/body/div/ol/li/h2/a").click()

#生成当前页面快照并保存，在这里由于没有页面重定位，仍停留在这个页面

driver.save\_screenshot("bing\_3.png")

#关闭浏览器

driver,quit()

在上述代码中，注意以下几点。

driver = webdriver.PhantomJS()语句默认调用环境变量指定的PhantomJS浏览器创建浏览器对象。如果没有在环境变量指定PhantomJS位置，这里应该指定为PhantomJS的安装路径，直接指向bin文件夹下的exe文件，使用的语句如下。

driver = webdriver.PhantomJS(executable\_path="【path】")

Selenium中有两种退出方法，分别是close方法和quit方法。区别在于使用driver.close()关闭driver后并不会清除临时文件中的WebDriver临时文件，而使用driver.quit()关闭浏览器后，会自动删除临时文件夹。

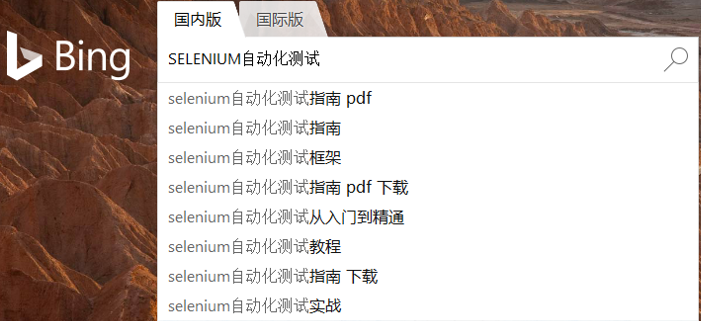
这里，借用selenium.webdriver.common.keys库来输入字符，测试效果如图4-2所示。

图4-2 输入字符的测试效果

然后，使用click()方法，搜索该字符串，测试效果如图4-3所示。



图4-3 搜索字符串的测试效果

接着，仍然使用click()方法打开第一个搜索结果，因为并没有重定位到新打开的页面，所以还显示搜索结果的页面，但搜索结果已经单击过了，如图4-4所示。

图4-4 搜索结果

## 4.5 通过time.sleep()慢下来

在做统计学采集的时候，你可能会遇到一些数据明明显示在浏览器里，却在源代码中找不到，并且抓取不出来的令人崩溃的问题，可能你向服务器提交了相当完美的表单却被拒绝，可能你的IP地址已经被查封。造成这些状况的可能原因之一就是过于频繁的访问，当你使用一个比正常的浏览频率快得多的频率来访问网址时，很可能你的IP地址会被网站查封。

建议你使用一行代码。

time.sleep(2)

这样做，有助于你避免许多道德与法律上的问题。虽然可能有些网站并没有足够的安全措施，但是不能为了自己获取数据而不惜拖垮网站。请牢记，你在本书上学习到的内容不应该在未经允许的情况下使用在任何一个网站中。

这里建议读者尽量把采集的时间放到夜晚，原因如下。

* 你完全没必要那么急躁。我们来做一个计算，假如你需要采集21600个页面，每一个页面你都使用2s来采集，那么12个小时完全够你采集到足够的数据。
* 网站访问的高峰期一般在白天，放到夜晚进行数据采集可以避免你的采集任务对网站的正常运行带来的负担。
* 没有一项任务可以在不停地查看程序运行状况的条件下完成。想想看，当你从睡梦中醒来时，在你的计算机中存放着你所需要的数据，这是一件多么惬意的事情。

## 4.6 PhantomJS的命令行参数

本节简单说明PhantomJS大部分的命令行参数，以方便读者后续查阅和使用。

* –help或-h, 用来列举所有可用的命令行选项，使用后立即运行。
* –version或-v，输出PhantomJS对应的版本号，立即执行。
* –debug=[true/false]，用来打开开关调试模式，开始调试后，程序运行时会输出额外的信息，当然，PhantomJS也支持[yes/no]。
* –config，用于指定json格式化的配置文件。
* –cookies-file=’[path]’，用来指定一个用来存储当前cookie信息的文本，路径应该是像“/path/to/cookies.txt”这样的路径。
* –disk-cache=[true/false]，指定是否启用硬盘缓存（在桌面服务的存储位置，在PhantomJS中该命令默认指定为false）。当然，PhantomJS也支持[yes/no]。
* –disk-cache-path=’[path]’，指定硬盘缓存的存储位置。
* –ignore-ssl-errors=[true/false],用于指定是否忽略ssl证书认证错误，例如，过期的证书或者自签名证书认证错误，默认为false。当然，PhantomJS也支持[yes/no]。
* –load-images=[true/false]，用于确定是否加载所有内联图像，默认为true。
* –output-encoding=encoding，设置用于终端输出的编码，默认为utf-8。
* –web-security=[true/false]，使用网络安全和禁止跨域（默认为true），也可以设置为[yes/no]。
* –ssl-protocol=[sslv3|sslv2|slsv1.1|tlsv1.2|any]，设置用于安全连接的ssl证书，默认为sslv3。
* --proxy-auth=username：password，指定用于代理身份验证的认证信息。
* –locao-storage-path=”[path]”，设置用于存储本地存储和webSQL内容的地址。

## 4.7其他问题

下面列举在Selenium + PhantomJS的组合使用过程中可能遇到的几个问题，为读者进一步完成Selenium自动化测试做好准备。

### 4.7.1.中文编码问题

在Python 2中使用Selenium的send\_keys（）输入中文会报错，其实在中文前面加一个u变成Unicode编码就能解决这个问题了。

### 4.7.2不同frame间的转换

可能读者可以在页面源代码中发现某些元素，但通过Selenium中的find\_element\_by\_xpath()、find\_element\_by\_id()等类似定位函数无法获得元素对象，还可能会出现“unable to find element with id 'kw'”的错误。也可能会存在一种特殊情况，元素被包裹在一个特定的frame中。

如果如此，那么你还需要首先使用以下代码。

Frame\_name= driver.find\_element\_by\_id(【Frame\_name】)

driver.switch\_to\_frame(Frame\_name)

需要注意的是，读者在切换到这个frame之后，只能访问当前frame的内容，如果想要回默认内容（默认的frame），需要使用Selenium中一个专用的函数driver.switch\_to\_default\_content()。

### 4.7.3 PhantomJS进程不会自动退出

主程序退出后，Selenium不能也无法保证PhantomJS也成功退出。这里建议读者最好手动关闭PhantomJS进程。下面一行代码可以帮助你解决这个问题。

driver.quit()

# 第5章 基于Python官网的实战

前面的章节介绍了Selenium中大部分与页面进行交互的操作。本章对前面的知识进行综合运用，进一步加深读者对Selenium的理解。

建议读者在学习的过程中运行本章的代码。如果只是单纯地看书，虽然进度比较快，但对于实际编程的效果并不好。

## 5.1 通过Selenium访问Python官网

在开始正式的实战前，我们先输入几行代码先创建一个基本的模具，为后续的编程奠定基础。

这里用到的内容主要是第2章的。涉及的内容包括导入库，启动浏览器，打开页面，退出。

为了易于输入和节省时间，在以后编写的Selenium程序中，我们将driver简写为dr。

#导入库

from selenium import webdriver

import time

#启动浏览器，打开页面

dr=webdriver.Chrome()

dr.get("https://www.python.org/")

#退出

driver.quit()

这里编写的内容可以说是绝大部分使用Selenium编写的程序的程式框架。如果你想要使用Selenium编写一个程序，那么你可以想都不用想，直接先输入这几行代码。

## 5.2 通过JavaScript修改Python官网的标题

JavaScript并不是我们学习的重点内容，其中一些关于浏览器的操作方式请参考网上的资料。这里给出修改页面标题和弹窗的JavaScript代码。通过这两段代码，让读者练习使用Selenium调用JavaScript。

#修改页面

JS1="document.title='xxxxxx';"

#弹窗

JS2=r"alert($(document).attr('title'));"

希望读者看到这里的时候，不要直接看下面的代码，先自己想一想应该怎么做，输入几行代码尝试运行一下。修改Python官网标题的流程如图5-1所展示。

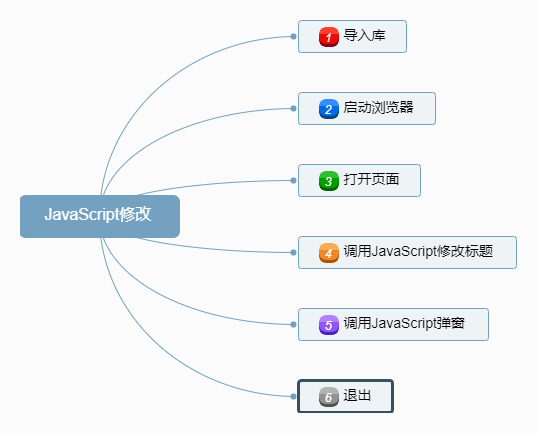
图5-1 修改Python官网标题的流程

图5-1中图字修改：JavaScript修改——通过JavaScript修改Python官网标题

具体的代码如下。

#导入库

from selenium import webdriver

import time

#启动浏览器，打开页面

dr=webdriver.Chrome()

dr.get("https://www.python.org/")

#修改标题

time.sleep(1)

JS1="document.title='xxxxxx';"

dr.execute\_script(JS1)

#弹窗标题

time.sleep(1)

JS2=r"alert($(document).attr('title'));"

dr.execute\_script(JS2)

#退出

driver.quit()

## 5.3 在搜索框搜索

本节中，我们练习一下第3章中的元素定位方式，在Python官网的搜索框进行一次搜索。首先，看一下搜索框的代码。

<input id="id-search-field" name="q" type="search" role="textbox"

class="search-field placeholder" placeholder="Search" value="" tabindex="1">

可以看到，搜索框的id为“id-search-field”并且唯一，我们通过id查找搜索框，然后向其中输入内容，并且查找“确认”按钮，然后单击。具体代码如下。

#导入库

from selenium import webdriver

import time

#启动浏览器，打开页面

dr=webdriver.Chrome()

dr.get("https://www.python.org/")

#修改标题

time.sleep(1)

JS1="document.title='xxxxxx';"

dr.execute\_script(JS1)

#弹窗显示现在的标题

time.sleep(1)

JS2=r"alert($(document).attr('title'));"

dr.execute\_script(JS2)

#搜索

time.sleep(1)

dr.find\_element\_by\_id("id-search-field").send\_keys("pycon")

dr.find\_element\_by\_id("submit").click()

#退出

driver.quit()

在这里，首先，通过get()方法跳转到页面[Python官网](http://www.python.org)。然后，通过执行JavaScriptdiam修改页面的标题。接着，使用alert方法运行一段可以弹出现在标题的代码。最后，通过分别查找id为“id-search-field”和“submit”的元素，向第一个元素中输入内容，单击第二个元素，实现搜索功能。

## 5.4 获取latest news部分

在本节中，基于5.3节中的代码，进行一些修改。这里通过一个截图获取关于pycon的最新消息。

调用Selenium中截屏功能的代码如下。

dr.save\_screenshot("save.png")

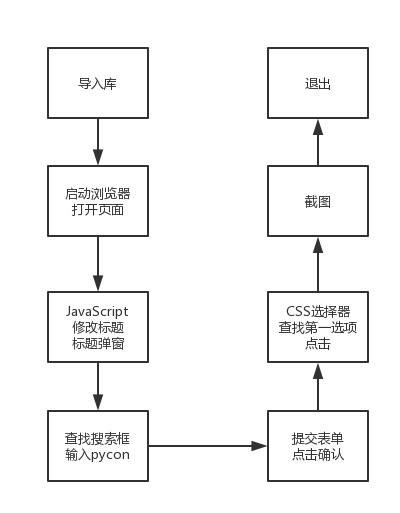
先整理一下现在已经实现的功能，具体流程如图5-2所示。

图5-2 流程

图5-2中图字修改： 启动浏览器——启动浏览器，

JavaScript修改标题 标题弹窗——通过JavaScript代码修改标题和标题弹窗

查找搜索框——查找搜索框，

提交表单，点击确认——提交表单，单击“确认”按钮

CSS选题器….单击——通过CSS选题器查找第一项并单

具体代码如下。

from selenium import webdriver

import time

#启动浏览器，打开页面

dr=webdriver.Chrome()

dr.get("https://www.python.org/")

#修改标题

time.sleep(1)

JS1="document.title='xxxxxx';"

dr.execute\_script(JS1)

time.sleep(1)

dr.find\_element\_by\_id("id-search-field").send\_keys("pycon")

dr.find\_element\_by\_id("submit").click()

#通过CSS样式查找

dr.find\_element\_by\_css\_selector(r'#content > div > section > form > ul > li:nth-child(1) > h3 > a').click()

#截图

dr.save\_screenshot("save.png")

#退出

driver.quit()

## 5.5 Selenium的等待

这里还要花点时间讨论一下Selenium的等待。使用等待是一个好习惯，可以避免你因为页面尚未加载完就执行下一步出现的错误。

Selenium的等待有很多种方式，除了在上面经常用到的time.sleep()以外，selenium项目自身还提供了两种等待方式，分别是显示等待和隐式等待。接下来，本节讨论这3种等待的区别。

time.sleep()这种等待方式也叫作强制等待.换句话说，就是不管浏览器是否加载完，我们都要等待特定的秒数。比如，作者和读者要一起出发去公园玩，那么作者提前到了，但读者必须等到9点才出发，即便读者已经到了，他们两个还是要等到某个时刻再出发。

隐式等待则和上一种方式有很大的不同。同样借用上面的示例，如果作者8:50到公园门口，那么只等读者10分钟。如果到了9点读者还没有来，作者就先自己进去了。如果读者在8：50到9点的时间内到了，那么作者和读者就立刻一起进去。

显式等待更加灵活，还以刚才的例子来说明，作者在公园等待读者，读者因为路上堵车，无法准时到达，于是给作者打了一个电话，作者就自己先进去了。

显式等待和隐式等待最大的区别在于，显式等待判断的并不是读者这个人是否到达，而是读者自身是否给出了某个信号。用在网页加载上也是一样的，隐式等待判断网页是否完全加载完，但可能在某些网站上某个JavaScript脚本的加载速度特别慢，但这个JavaScript脚本可能只是某个限制脚本，并不影响页面的浏览，这时候仍需要等到全部页面加载完才执行下一步的操作。而显式等待只需要判断某个特定的元素即可。

我们对同一个操作分别用3种等待方法实现。

使用强制等待的代码如下。

#导入库

from selenium import webdriver

import time

#打开浏览器和页面

dr = webdriver.Chrome()

dr.get('https://www.csdn.net/')

#输出当前页面地址

time.sleep(2)

print(dr.find\_element\_by\_link\_text('首页').get\_attribute('href'))

#退出

dr.quit()

强制等待不管浏览器是否加载完，程序都会等待2s，然后程序继续执行后面的代码，这种方法很适合在调试的时候使用。虽然强制等待在编程中很方便使用，但在运行的时候过于死板，十分影响程序的执行速度。

使用隐式等待的代码如下。

#导入库

from selenium import webdriver

#打开浏览器，设置隐式等待，打开页面

dr = webdriver.Chrome()

dr.implicitly\_wait(30) # 隐式等待，最长等30s

dr.get('https://www.csdn.net/')

#输出页面URL

print(dr.find\_element\_by\_link\_text('首页').get\_attribute('href'))

#退出

dr.quit()

特别说明一下，隐式等待设置一次，就会在整个程序运行过程中都起作用。只要设置一次隐式等待，不需要在整个程序中设置多次。

这是一个显式等待的例子，另外在使用显式等待的时候，读者还需要多导入以下库：

* selenium.webdriver.support.wait；
* selenium.webdriver.support；
* selenium.webdriver.common.by。

使用显式等待的代码如下。

#导入库

from selenium import webdriver

from selenium.webdriver.support.wait import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.by import By

#打开浏览器，设置隐式等待，打开页面

dr = webdriver.Chrome()

dr.implicitly\_wait(10)

dr.get('https://www.csdn.net/')

#使用显式等待

try:

WebDriverWait(dr, 20, 0.5).until(EC.presence\_of\_element\_located((By.LINK\_TEXT,u'首页')))

finally:

print(dr.find\_element\_by\_link\_text('首页').get\_attribute('href'))

#退出

dr.quit()

我们重点看一下这一行代码。

WebDriverWait(dr, 20, 0.5) .until(EC.presence\_of\_element\_located((By.LINK\_TEXT,u'首页')))

WebDriverWait()可选择的参数有4个，但一般情况下，我们只会用到其中3个。第一个参数设置打开的浏览器，第二个参数设置超时时间，第三个参数设置检查频率，因为这里第三个参数设置为0.5s，所以每隔0.5s检查一次。我们看一下WebDriverWait代码的完整版。

WebDriverWait(driver=self.dr,timeout=20,poll\_frequency=0.5,ignored\_exceptions=None)

后面一部分是until()语句。除了until()之外，还可以使用until\_not()。这两个选项的使用方法都差不多。语法格式都是(method, message=””)，message返回的是返回结果为0（即失败）时后弹出的信息。不同的是until检查的条件是直到返回值为True，until\_not检查的条件直到返回值为False。

向EC.presence\_of\_element\_located()传入的参数是一个元组，不过，也可以这样分行写。

#导入库

from selenium import webdriver

from selenium.webdriver.support.wait import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.by import By

#打开浏览器，设置隐式等待，打开页面

dr = webdriver.Chrome()

dr.implicitly\_wait(10)

dr.get('https://www.csdn.net/')

#定位条件

locator = (By.LINK\_TEXT,u'首页')

#使用显式等待

try:

WebDriverWait(dr, 20, 0.5).until(EC.presence\_of\_element\_located(locator))

finally:

print(dr.find\_element\_by\_link\_text('首页').get\_attribute('href'))

#退出

dr.quit()

可能细心的读者看到了，我们在使用显式等待的同时也设置了隐式等待。是的，它们可以同时用，但要注意，程序等待的最长时间取显示等待和隐式等待之中的最大值。

在代码的中间用到了EC（ExpectedCondition）。EC的方法其实不止这一种，还有许多，用法都和上面类似。EC的的方法如表5-1所示。

表5-1 EC的的方法

编辑注：表格请添加表头：方法名 说明

|  |  |
| --- | --- |
| alert\_is\_present | 判断页面上是否存在警告 |
| element\_to\_be\_selected | 判断某个元素是否选中 |
| element\_located\_to\_be\_selected | 判断被定位的元素是否选中 |
| element\_selection\_state\_to\_be | 判断某个元素的选中状态是否符合预期 |
| element\_located\_selection\_state\_to\_be | 同上，在上面的方法中传入element，在这个方法中传入locator |
| element\_to\_be\_clickable | 判断某个元素中是否可见并且启用 |
| frame\_to\_be\_available\_and\_switch\_to\_it | 判断该frame是否可以switch进去 |
| presence\_of\_all\_elements\_located | 判断是否至少有1个元素存在于dom树中 |
| presence\_of\_element\_located | 判断某个元素是否添加到了dom树里，添加到了dom树里并不代表该元素一定可见 |
| staleness\_of | 等某个元素从dom树中移除，注意，这个方法也返回True或False |
| title\_is | 判断当前页面的title是否符合预期 |
| title\_contains | 判断当前页面的title是否包含预期的字符串 |
| text\_to\_be\_present\_in\_element | 判断某个元素中的text是否包含预期的字符串 |
| text\_to\_be\_present\_in\_element\_value | 判断某个元素中的value属性是否包含预期的字符串 |
| invisibility\_of\_element\_located[[1]](#footnote-1) | 判断某个元素中是否不存在于dom树中或不可见 |
| visibility\_of\_element\_located | 判断某个元素是否可见，向该方法中传入locator |
| visibility\_of | 判断某个元素是否可见，向该方法中传入element |

# 

# 第6章 基于今日头条的实战

上一章的实战部分近似于复习课。接下来，我们进行一个稍微复杂一些的Selenium爬虫——基于今日头条的爬虫。

可能有读者就要问了，像今日头条这类网站的信息明明就在网站上，我们为什么还要通过爬虫去获取呢？在大数据时代，网站虽然可以满足人们对信息的共性需求，让人们看到，但网站仍然存在很大的不足，它无法针对每个人的需求进行数据的定制化操作。在工作中，如果我们需要收集特定的信息，而这些信息分布几百个不同的页面上，我们通过人工来收集信息不仅枯燥无味，还极其容易出现错误的数据分类、整理。

在本章中，我们首先通过百度热词来获取当前的实时热点，然后作为关键词在今日头条中搜索百度热词，最后保存相关信息。

## 6.1 通过Selenium访问百度热词

我们的目标网站为百度搜索风云榜。先打开百度搜索风云榜网站观察一下，我们的目标是实时热点，如图6-1所示。

图6-1 实时热点

为了获取元素，首先要进行元素定位。按F12键打开开发者工具，对这几条消息进行定位，右击，获取XPath。然后读者就会惊讶的发现，它的XPath都很类似，不同的只是中间一个叫作li的标签。这就意味着，我们只需要更改li标签对应的数字，就可以完成对所有XPath的定位。

//\*[@id="hot-list"]/li[1]/a[1]

//\*[@id="hot-list"]/li[2]/a[1]

//\*[@id="hot-list"]/li[3] /a[1]

//\*[@id="hot-list"]/li[4] /a[1]

//\*[@id="hot-list"]/li[5]/a[1]

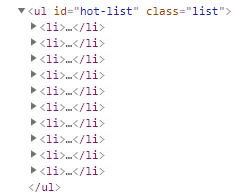
其实<ul>和<li>标签在HTML中是用来定义列表项目的，如图6-2所示。这些热词都在一个<ul>下面。在<ul>列表中，因为所有元素都是采用<li>标签进行并列的，所以它们的标签其实也很相像，主要的区别就是<li>标签的序号。

图6-2 <ul>和<li>标签

然后我们将XPath分割为3部分。首先是<li>标签序号前面的一部分，然后是<li>标签的序号，最后是后面的<a>标签。

xpath='//\*[@id="hot-list"]/li['+str(i+1)+']/a[1]'

可能细心的读者就会发现，我们在其中用了一个str(i+1)，为什么要+1呢？Python的序号是从0开始计数的，函数range(0,10)其实对应的数字就是1~9。

在整合完XPath后，我们通过XPath查找并获取对应的<a>标签的title属性。之所以获取title属性是因为百度热搜风云榜中的热词蕴含在其中。具体代码如下。

<a target="\_blank" title="华为首次超越苹果"

data="1|1" class="list-title

"href="http://www.baidu.com/baidu?cl=3&amp;tn=SE\_baiduhomet8\_jmjb7mjw&amp;rsv\_dl=fyb\_top&amp;fr=top1000&amp;wd=%BB%AA%CE%AA%CA%D7%B4%CE%B3%AC%D4%BD%C6%BB%B9%FB" href\_top="./detail?b=1&amp;w=%BB%AA%CE%AA%CA%D7%B4%CE%B3%AC%D4%BD%C6%BB%B9%FB">华为首次超越苹果</a>

既然已经知道了解决问题的方法了，那么我们就可以开始输入代码了。

#导入库

from selenium import webdriver

#打开浏览器和网页

dr=webdriver.Chrome()

dr.get('http://top.baidu.com/')

#整合 Xpath

for i in range(10):

xpath='//\*[@id="hot-list"]/li['+str(i+1)+']/a[1]'

print(dr.find\_element\_by\_xpath(xpath).get\_attribute('title'))

代码运行结果如下。

手指被可乐炸骨折

华为首次超越苹果

工商局调查拼多多

凌晨上班遇车祸

华夏李君殴打员工

重庆冉宇航被查

美军遗骸归国仪式

董明珠回应银隆

詹姆斯免费学校

18岁女孩被烧伤

## 6.2 通过Selenium搜索相关热词

刚才我们输出了获取到的关键词，如果我们需要调用这些内容，就不能再输出了。我们考虑将获取到的内容保存在字典中。具体代码如下。

hw\_dict=[]

value=dr.find\_element\_by\_xpath(xpath).get\_attribute('title')

hw\_dict.append(value)

如果仅仅要获取一条最热的新闻，那么这些代码完全足够了，但我们刚才获取到的新闻一共有10条，因此们需要将这些代码嵌套进for循环中，通过循环来保存。

既然我们已经获取到了关键词，下一步就是对关键词进行搜索了。在今日头条网页上进行一次搜索，我们先以selenium为关键词进行搜索，结果如图6-3所示。



图6-3 搜索selenium

先将页面的URL记录下来。

https://www.toutiao.com/search/?keyword=selenium

注意，在URL中同样有我们刚才搜索的关键词selenium，如果把关键词换为webdriver，页面的URL则变成了如下形式。

<https://www.toutiao.com/search/?keyword=webdriver>

很明显，我们搜索的内容根据访问的网址的关键字不同而不同（keyword=”xxx”的部分用于传递参数）。

通过以下代码，我们就可以这样进入新的页面了，直接搜索刚才获取到的关键词。

#导入库

from selenium import webdriver

dr=webdriver.Chrome()

dr.get('http://top.baidu.com/')

#建立空列表

hw\_dict=[]

#在列表中添加内容

for i in range(10):

xpath='//\*[@id="hot-list"]/li['+str(i+1)+']/a[1]'

value=dr.find\_element\_by\_xpath(xpath).get\_attribute('title')

hw\_dict.append(value)

#获取列表的第一个元素

keyword=hw\_dict[0]

#搜索第一个元素

URL='https://www.toutiao.com/search/?keyword=%s' %keyword

dr.get(URL)

运行结果如图6-4所示。



图6-4搜索热词的运行结果

至此，我们成功地实现了搜索。

## 6.3 获取第一条结果

在搜索结果结束了以后，我们模拟用户的操作—— 单击第一条新闻。

与刚才一样，查找元素，我们先看一下第一条结果能否触发单击操作的最下一层标签（这里选择的是文字内容）。具体代码如下。

<span class="J\_title" riot-tag="raw">真相细思极恐！

<em class="highlight">手指被可乐炸骨折</em> 事件原委解析和安全饮用<em class="highlight">可乐</em>方法</span>

然后我们通过开发者工具，获取它的XPath。

//\*[@id="J\_section\_0"]/div/div/div[1]/div/div[1]/a/span

学到这样，相信读者已经能够轻松地写出查找元素和模拟单击操作的代码。

但有一个问题值得我们考虑，如果按照我们刚才做的，我们再运行几次后会发现时好时坏，经常会有出错的情况，这是因为什么呢？程序运行的速度太快，网页跟不上，程序在查找第一个元素的时候，网页通常仍然尚未加载完，因此程序就会报错，等网页加载完了，程序已经退出了。

考虑到今日头条加载的内容较多，为了提高程序运行的效率以及准确率，我们使用在第5章中的显式等待方法。

虽然显式等待只需要一行代码，但是这里需要同时导入多个库。

from selenium.webdriver.support.wait import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.by import By

另外，因为显式等待涉及条件的判定，所以最终的代码如下。

#导入库，这里多加载了显式等待用到的库

from selenium import webdriver

from selenium.webdriver.support.wait import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.by import By

#打开浏览器和百度搜索风云榜

dr=webdriver.Chrome()

dr.get('http://top.baidu.com/')

#设置空列表

hw\_dict=[]

#在列表中添加内容

for i in range(10):

xpath='//\*[@id="hot-list"]/li['+str(i+1)+']/a[1]'

value=dr.find\_element\_by\_xpath(xpath).get\_attribute('title')

hw\_dict.append(value)

#搜索最热的关键词

keyword=hw\_dict[0]

URL='https://www.toutiao.com/search/?keyword=%s' %keyword

dr.get(URL)

#XPath拼接

XPath='//\*[@id="J\_section\_0"]/div/div/div[1]/div/div[1]/a/span'

#调用显式等待

try:

WebDriverWait(dr, 20, 0.5).until(EC.presence\_of\_element\_located((By.XPATH,XPath)))

finally:

dr.find\_element\_by\_xpath(XPath).click()

页面正常打开，我们可以放心地进行下一步的操作了。

## 6.4 定位元素——抓取内容

打开页面，获取页面的内容，并再次查看title所在的标签。

<h1 class="article-title">华为首次超越苹果，成全球第二大智能手机厂商</h1>

查看发布者和时间所在的标签。

<div class="article-sub"><!----> <span>36氪</span> <span>2018-08-01 07:20:30</span></div>

查看内容所在的标签。

<div class="article-content"><p>IDC最新报告显示，苹果二季度市场份额从去年同期12.1%降至11.8%，被华为超越。华为市场份额从去年同期11%增至15.8%，成为仅次于三星电子的全球第二大智能手机厂商。三星市场份额亦有所萎缩，从去年同期22.9%降至20.9%。这是苹果八年来首次跌落全球两大智能手机制造商位置。</p></div>

此外，句柄也是在Selenium爬虫中需要注意的地方。一个句柄就代表一个窗口，可能你打开了新的页面（弹出窗口），但是句柄还停留在旧的页面上，如果你这时候使用元素查找，毫无疑问，很难成功，因为你查找的是旧的页面。

可以使用如下代码来获取所有页面的句柄。

dr.window\_handles

获取到的所有句柄是一个列表，里面的内容按照你打开的页面的时间来进行排序，越晚打开的页面顺序越靠后。如果你查到[–1]，就代表你要找到刚刚打开的页面，切换句柄会用到这个函数。

dr.switch\_to\_window(all\_handles[–1])

传入的参数是你获取到的所有句柄列表中的一个元素。

因为后面还会重复用到获取信息的代码，所示我们编写一个getinfo()函数，来获取文章的发布方、发布内容和标题。

#导入库，这里多加载了显式等待用到的库

from selenium import webdriver

from selenium.webdriver.support.wait import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.by import By

import time

#提前定义了函数

def getinfo():

id\_=dr.find\_element\_by\_xpath('//\*/div[@class="article-sub"]/span[1]')

time\_=dr.find\_element\_by\_xpath('//\*/div[@class="article-sub"]/span[2]')

title\_=dr.find\_element\_by\_xpath('//\*/h1[@class="article-title"]')

text\_=dr.find\_element\_by\_xpath('//\*/div[@class="article-content"]')

item=[id\_.text,time\_.text,title\_.text,text\_.text]

return item

#打开浏览器和百度搜索风云榜

dr=webdriver.Chrome()

dr.get('http://top.baidu.com/')

#设置空列表

hw\_dict=[]

#获取最热的前10个关键词

for i in range(10):

xpath='//\*[@id="hot-list"]/li['+str(i+1)+']/a[1]'

value=dr.find\_element\_by\_xpath(xpath).get\_attribute('title')

hw\_dict.append(value)

#获取并搜索最热的关键词

keyword=hw\_dict[0]

URL='https://www.toutiao.com/search/?keyword=%s' %keyword

dr.get(URL)

#拼接XPath

XPath='//\*[@id="J\_section\_0"]/div/div/div[1]/div/div[1]/a/span'

#打开页面

try:

WebDriverWait(dr, 20, 0.5).until(EC.presence\_of\_element\_located((By.XPATH,XPath)))

finally:

dr.find\_element\_by\_xpath(XPath).click()

#给页面留足加载时间，同时也为了防止IP地址被禁

time.sleep(1)

#获取所有句柄

all\_handles = dr.window\_handles

#跳转到最后一个句柄，也就是刚刚打开的页面

dr.switch\_to\_window(all\_handles[-1])

#获取信息

info=getinfo()

print(info)

执行结果如下：

['36氪','2018-08-01 07:20:30', '华为首次超越苹果，成全球第二大智能手机厂商',

'IDC最新报告显示，苹果二季度市场份额从去年同期12.1%降至11.8%，被华为超越。华为市场份额从去年同期11%增至15.8%，成为仅次于三星电子的全球第二大智能手机厂商。三星市场份额亦有所萎缩，从去年同期22.9%降至20.9%。这是苹果八年来首次跌落全球两大智能手机制造商位置。']

## 6.5 存储数据

在前一节中，我们写的程序已经比较成熟了，但当我们需要分析大规模的数据时，这样输出的数据明显是不行的，我们还需要将数据按照统一的格式进行存储。

我们将文件以CSV格式存储，CSV格式的文件将会在下一章讲解到。读者在学完第7章后，再回顾本节，会有新的体验。

在开始存储数据之前，我们还需要考虑到一件事情。今日头条上关于最热新闻的报道肯定很多，不单单是一页就能加载完的，因此我们还需要一个能够模拟下拉的方法，来为我们加载排名靠后的内容。

让人愉快的是，Selenium已经为我们提供了模拟键鼠操作的方法。可以从ActionChains库中调用模拟键鼠操作的方法。

from selenium.webdriver.common.action\_chains import ActionChains

ActionChains库包含的关于鼠标的操作方法如下。

* 'click(on\_element=None)'，表示鼠标单击。
* 'click\_and\_hold(on\_element=None)'，表示鼠标单击后不松开。
* 'context\_click(on\_element=None)'，表示右击。
* 'double\_click(on\_element=None)'，表示双击。
* 'drag\_and\_drop(source, target)'，表示拖动到某个元素。
* 'drag\_and\_drop\_by\_offset(source, xoffset, yoffset)'，表示拖动到某个坐标后松开。
* 'release(on\_element=None) '，表示释放鼠标左键。
* 'move\_by\_offset(xoffset, yoffset) ' ，表示鼠标移动到某个坐标。
* 'move\_to\_element(to\_element) ' ，表示鼠标移动到某个元素。
* 'move\_to\_element\_with\_offset(to\_element, xoffset, yoffset) '，表示移动到距某元素左上方指定的位置。

ActionChains库包含的关于键盘的方法如下。

* 'key\_down(value, element=None) '，表示按下某个键盘上的键。
* 'key\_up(value, element=None) ' ，表示松开某个键。
* '[send\_keys(\*args)](https://seleniumhq.github.io/selenium/docs/api/rb/Selenium/WebDriver/ActionBuilder.html" \l "send_keys-instance_method" \o "#send_keys (instance method)) '，表示发送。
* 'perform()'，表示执行已经添加到生成器中的操作。

例如，如果你没有目标元素，只是单纯地想要在打开某个网页后按下某个键，那么可以这样写。

ActionChains(dr).key\_down(Keys.DOWN).key\_up(Keys.DOWN).perform()

但如果你还想要向某个表格里面写入表单，那么需要这样写。

elem=driver.find\_elements\_by\_id()

ActionChains(dr).key\_down(Keys.DOWN,elem).key\_up(Keys.DOWN,elem).perform()

这种写法叫作链式写法。此外，还有一种写法称作分布写法。

elem=driver.find\_elements\_by\_id()

ActionChains(dr).key\_down(Keys.DOWN,elem)

ActionChains(dr).key\_up(Keys.DOWN,elem)

ActionChains(dr).perform()

无论哪一种写法都是可以的，可以根据自己的偏好，选择对应的写法。

另外，这里用了这样一行代码。

for i in range(2000):

虽然这里i设置为0~1999，但使并不意味着需要这么长的时间，这是Python计数2000个需要的时间。这里使用和前一节不同的代码实现打开页面，前一节是打开新页面，切换句柄来获取内容。而这里在搜索之后，直接将获取到的所有URL写入一个列表，对于列表中的每个元素（URL）单独获取内容并写入。

为了防止中间个别页面因为内容不全而加载过慢，或者出现其他意外，我们使用try…except来处理异常，避免因为个别报错而停止采集数据。

#导入库

import csv

import time

from selenium import webdriver

from selenium.webdriver.common.action\_chains import ActionChains

from selenium.webdriver.common.keys import Keys

#把数据写入CSV函数

def write(item):

with open(r"F:\DOCUMENT\SELENIUM\第六章\jrtt.csv","a") as f:

writer=csv.writer(f)

try:

writer.writerow(item)

except:

print("error")

#获取信息函数

def getinfo():

id\_=dr.find\_element\_by\_xpath('//\*/div[@class="article-sub"]/span[1]')

time\_=dr.find\_element\_by\_xpath('//\*/div[@class="article-sub"]/span[2]')

title\_=dr.find\_element\_by\_xpath('//\*/h1[@class="article-title"]')

text\_=dr.find\_element\_by\_xpath('//\*/div[@class="article-content"]')

item=[id\_.text,time\_.text,title\_.text,text\_.text]

return item

#打开浏览器，设置隐式等待，打开页面

url="https://www.toutiao.com/search/?keyword=selenium"

dr=webdriver.Chrome()

dr.implicitly\_wait(2)

dr.get(url)

#给页面留下足够的加载时间

time.sleep(1)

#刷新足够的次数，保证加载到底

#这里循环次数要尽量大

#经过测试，刷新1600次左右可以下拉到页面底部

for i in range(2000):

#相当于一直按着DOWN键，下拉页面

ActionChains(dr).key\_down(Keys.DOWN). key\_up(Keys.DOWN).perform()

print(f'下拉已完成{i}次')

#再次留下加载的时间，因为无法确定最后刷新到了哪一个，所以不用显式等待

time.sleep(1)

url\_=dr.find\_elements\_by\_css\_selector('.title')

print(url\_)

#链接列表

url\_list=[]

for i in url\_:

url\_list.append(i.get\_attribute('href'))

print(url\_list)

#考虑到部分可能会出错，我们使用try…except来处理异常

for i in url\_list:

try:

dr.get(i)

time.sleep(0.5)

write(getinfo())

dr.get(url)

print('已完成写入')

except:

print("error")

print('all\_done')

这里需要说明的是下面这行代码。

ActionChains(dr).key\_down(Keys.DOWN). key\_up(Keys.DOWN).perform()

这行代码的功能相当于我们一直持续按着键盘上的DOWN键。

接下来，你就可以惬意地泡一杯咖啡，让程序运行一会儿。如果读者不想打开浏览器器窗口，那么可以用到第4章讲解到的PhantomJS来进行网络数据采集。

输出的CSV文件读者可以直接用Excel表格打开。

# 第7章 数据的编解码和处理

如果你使用API的话，那么你很可能会喜欢API中那些已经处理好的数据。遗憾的是，在实际工作中，API的使用次数总是少的，大多数时候，都需要我们自己来进行数据的挖掘、整理和分析。

既然和互联网打交道，就少不了各种数据类型之间的转换。需要数据类型转换的根本原因是因为计算机的保存方式和目的不同，不同的计算机对不同类型的数据有不同的保存方式，比如，有的数据类型是为了节省存储空间，有的则为了使用方便等。这些数据的编解码问题往往比较麻烦。

本章还会介绍几种数据存储方式，虽然在命令行或者你使用的编译器中显示数据很炫酷，但随着数据越来越多，这种方式总归是不行的，就好像你可以轻松打开一个包含一篇文章的txt文件，但是无法立刻打开一部几百万字的小说。

## 7.1 读写CSV文件

在上一章的结尾，我们就用到了CSV文件。当时我们使用CSV文件进行数据存储，我们在这里将会学习CSV文件的读取和应用。当然，CSV文件只是一种简单的数据存储方法，还可以使用较为复杂的关系数据库和非关系数据库存储数据。

顾名思义，CSV（Comma-Separated Value，逗号分隔值）就是采用逗号进行分隔的值。可以直接在txt文件中编辑CSV，这也意味着它是一个纯文本文件（只能包含数字和文字，不能保存图片、视频的其他格式）。

Python的CSV库主要包含两个主要用法—— Reader和Writer，分别代表读和写。

假设你有一份city.csv文件（读者可以在本书代码的托管网站上下载），文件的类型如下。

城市ID,城市名,城市名（英文）,国家,国家CODE

SP91FD79GFQU,安道尔,Andorra la Vella,安道尔,AD

THM3PVJHX1G2,Ar Ruways,Ar Ruways,阿联酋,AE

THQEJNH8EJWT,阿布扎比,Abu Dhabi,阿联酋,AE

THR9SGGWEJYQ,艾因,Al Ain,阿联酋,AE

THRRDQPEMRVW,迪拜,Dubai,阿联酋,AE

THRXZEE9YZJZ,Adh Dhayd,Adh Dhayd,阿联酋,AE

THX2K8PMN3P2,夏尔迦,Sharjah,阿联酋,AE

THX2T3YW6N1Z,Ajman,Ajman,阿联酋,AE

THX92JYJ35PD,Umm al-Quwain,Umm al-Quwain,阿联酋,AE

CSV文件的读取很简单，你可以这样子使用，但你需要注意的是，你想要读取的文件包含的内容可能千奇百怪。因为网络上的文本质量参差不齐，错误的拼写、混乱的换行、空格以及混用的标点符号等都是我们处理数据时遇到的大问题，部分字符可能甚至在utf-8编码中不存在，所以你需要把error参数设置成ignore模式（意为忽略可能出现的编码错误）。

#导入库

import csv

#打开文件

file=open('city.csv','r', encoding='utf-8',errors='ignore')

csv\_file = csv.reader(file)

#这里通过遍历元素来逐行输出

for i in csv\_file:

print(i)

print(line)

运行结果如下。

['\ufeff城市ID', '城市名', '城市名（英文）', '国家', '国家CODE']

['SP91FD79GFQU', '安道尔', 'Andorra la Vella', '安道尔', 'AD']

['THM3PVJHX1G2', 'Ar Ruways', 'Ar Ruways', '阿联酋', 'AE']

['THQEJNH8EJWT', '阿布扎比', 'Abu Dhabi', '阿联酋', 'AE']

['THR9SGGWEJYQ', '艾因', 'Al Ain', '阿联酋', 'AE']

['THRRDQPEMRVW', '迪拜', 'Dubai', '阿联酋', 'AE']

['THRXZEE9YZJZ', 'Adh Dhayd', 'Adh Dhayd', '阿联酋', 'AE']

['THX2K8PMN3P2', '夏尔迦', 'Sharjah', '阿联酋', 'AE']

['THX2T3YW6N1Z', 'Ajman', 'Ajman', '阿联酋', 'AE']

……..

既然有读取，就一定有写入。下面给出一个示例，写入的数据内容如下。

header=["name","gender","score"]

file\_1=["zhang\_san","male","100"]

file\_2=["Li\_si","male","80"]

示例代码如下。

import csv

# 要写入的文件

header=["name","gender","score"]

file\_1=["zhang\_san","male","100"]

file\_2=["Li\_si","male","80"]

# 创建

# 如果不加,newline=””会多一行空行

csvFile = open("writer.csv","w",newline='')

writer = csv.writer(csvFile)

# 写入的内容都以列表的形式传入函数

writer.writerow(header)

writer.writerow(file\_1)

writer.writerow(file\_2)

#关闭文件

csvFile.close()

写入后，文件的内容如下。

name,gender,score

zhang\_san,male,100

Li\_si,male,80

当然，如果你要对数据进行转储和处理，那么你完全可以从另一个CSV文件中读取，再进行内容的整理，组合后重新写入一个新的CSV文件。

在第7章之前，我们学习和处理的数据大部分是有规则的、简单的并且轻量级，但是随着数据规模的增大，我们迫切创建一个便于进行数据管理的数据集。而CSV刚好符合我们的要求，因此你完全可以考虑用CSV创建一个轻量级的数据库。

## 7.2 读写JSON文件

JSON（JavaScript Object Notation，JavaScript对象标记）数据也是一种形式。JSON官方是这样介绍JSON的：“一种轻量级的数据交换格式，易于人阅读和编写，同时也易于机器解析和生成”。采用完全独立于语言的文本格式的JSON，是进行数据交换的理想格式。

JSON基于一种 “名称：值”的集合，这和Python的字典格式非常类似。其实是因为因为JSON沿用了类C语言家族（包括C、C++、Java、Python和Perl等）的习惯。如果读者使用过API，那么你会惊讶地发现，绝大部分API返回的数据都是JSON格式的。

其实JSON格式和下面要讲解的XML格式有很大关联，它们的关系更像是老牌强者和后起之秀。本来XML最独大，后来JSON逐渐占了上风。

和XML相比，JSON具有以下特点，因此很多项目和网页的内容传输都使用JSON。

* JSON内容简洁。
* JSON采用压缩传输技术来节省宽带。
* 沿用了类C语言家族的习惯，适应的浏览器众多。
* JSON格式简单，易于编写和解析。

Python的库中同样有专门针对JSON格式的，并且这个库是Python自带的，完全不必耗费时间去安装它。

JSON库是主要包含4个方法，分别是loads()、dumps()、load()和dump()。其中load()和dump()与文件的读写相关。

dumps()方法将dict转化成str格式，loads()方法将str转化成dict格式。下面是dumps()和loads()方法的使用示例。

#导入库

import json

#这是我们提供的JSON格式的示例数据

json\_data={

"results": [

{

"location": {

"id": "WX4FBXXFKE4F",

"name": "北京",

"country": "CN",

"path": "北京,北京",

"timezone": "Asia/Shanghai",

"timezone\_offset": "+08:00"

}

}

]

}

b=json.dumps(json\_data)

print(b,type(b))

c=json.loads(b)

print(c,type(c))

运行结果如下。

{"results": [{"location": {"id": "WX4FBXXFKE4F", "name": "\u5317\u4eac", "country": "CN", "path": "\u5317\u4eac,\u5317\u4eac", "timezone": "Asia/Shanghai", "timezone\_offset": "+08:00"}}]} <class 'str'>

{'results': [{'location': {'id': 'WX4FBXXFKE4F', 'name': '北京', 'country': 'CN', 'path': '北京,北京', 'timezone': 'Asia/Shanghai', 'timezone\_offset': '+08:00'}}]} <class 'dict'>

我们可以相比两次输出的内容，但是最后附加的格式大不相同。在第一个文件编译器中，<class 'str'>用于说明第一个文件是str格式的。而第二个文件返回的却是<class 'dict'>，这告诉我们它是一个字典，虽然在语言和逻辑上它们是等价的，但在编程语言中你难以对一个字符串做类似于字典的操作。

此外，dump()和load()的作用与dumps()以及loads()的作用相同，只是dump()和load()把文件的转换与数据的读写结合起来了

#导入库

import json

#JSON数据

json\_data={

"results": [

{

"location": {

"id": "WX4FBXXFKE4F",

"name": "北京",

"country": "CN",

"path": "北京,北京",

"timezone": "Asia/Shanghai",

"timezone\_offset": "+08:00"

}

}

]

}

#打开文件

f = open('test.txt', 'w')

print(type(f))

json.dump(json\_data, f)

#tip 1

f.close()

#先读取再关闭

f=open('test.txt', 'r')

print(f.read())

f.close()

#JSON读取test.txt的内容

f=open('test.txt', 'r')

print(json.load(f))

f.close()

运行结果如下。

<class 'dict'>

{"results": [{"location": {"id": "WX4FBXXFKE4F", "name": "\u5317\u4eac", "country": "CN", "path": "\u5317\u4eac,\u5317\u4eac", "timezone": "Asia/Shanghai", "timezone\_offset": "+08:00"}}]}

{'results': [{'location': {'id': 'WX4FBXXFKE4F', 'name': '北京', 'country': 'CN', 'path': '北京,北京', 'timezone': 'Asia/Shanghai', 'timezone\_offset': '+08:00'}}]}

效果不错，第一次和第二次都有输出，这说明刚才的JSON数据确实通过dump()方法写入了文本。

可能读者会问：“为什么要重复多次关闭和打开呢？这不纯粹是在刁难人吗？”原因有两个。一方面，我们必须关闭一次。在没有关闭的时候，我们的操作是在内存中进行的，还没有写入文本，如果这时候你用f.read()去读取，你会什么也读取不到。

另一方面，不可以同时使用f.read()和json.load(f)，否则会出现以下错误。

json.decoder.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

这种错误和Windows操作系统下的当前文件已经被其他程序占用很类似，f.read()调用后，json.load(f)方法就会什么都获取不到。

## 7.3 将字典转化为XML

XML （eXtensible Markup Language，可扩展标记语言）是一种灵活的信息格式，规定了一系列编码转换的规则，它的设计理念是让信息可以同时被人和机器阅读。虽然XML文件占用的空间比单纯的二进制文件要多很多，但是这种可扩展标记语言更容易理解和掌握，它的这种明了性使其易于在任何语言、程序中读写数据。和HTML相比，XML继承了通用标准标记语言的的大部分功能，技术实现却要简单得多，但和前面提到的JSON格式来比，XML格式的数据在存储空间的有效比重上不太占优势。

那么，什么是XML呢？我们来看一个简单的XML例子。

<?xml version="1.0" encoding="utf-8"?>

<test>

<name>zhang\_san</name>

<gender>female</gender>

<phone\_number>01234567890</phone\_number>

<note>none</note>

</test>

它的格式和HTML很相像，大标签中有小标签，以此来对文本进行不同的标记和说明。

在解析XML的时候，我们主要会用到两个方法—— Element和tostring。前者用于创建一个XML文件，后者则用于将XML文件转换为可以被人类读取、理解的字符形式的代码。

在这个示例中，我们首先提供了一个字典，然后写了一个可以普遍使用的转换函数，再将字典转换为XML，接下来输出XML。第一次输出程序返回的是一个内存地址，这里的内容存储在这里。第二次转换为字符串输出，接下来我们对最外层设置了一个属性，并再次输出。

from xml.etree.ElementTree import tostring

from xml.etree.ElementTree import Element

dict = { "name":"zhang\_san", "gender":"female", "phone\_number":"01234567890","note":"none"}

#写一个转换函数

def convert(tag\_name,dict):

#建立一个XML

elem = Element(tag\_name)

#通过双值循环取出字典中的键和值

for key,value in dict.items():

#输出

print(key,":",value)

#相当于为键创建一个标签

xml\_elem=Element(key)

#将键的值赋给这个标签

xml\_elem.text=str(value)

#添加到XML文件中

elem.append(xml\_elem)

return elem

#字典转换

dict\_convert = convert("test", dict)

print("-------------------------------------------")

print(dict\_convert)

#转换为可读的标准XML

print(tostring(dict\_convert))

#设置最外层标签的属性

dict\_convert.set('id\_number','01234567890')

#再次输出

print(tostring(dict\_convert))

输出结果如下。

name : zhang\_san

gender : female

phone\_number : 01234567890

note : none

-------------------------------------------

<Element 'test' at 0x02C5DED0>

b'<test><name>zhang\_san</name><gender>female</gender><phone\_number>01234567890</phone\_number><note>none</note></test>'

b'<test id\_number="01234567890"><name>zhang\_san</name><gender>female</gender>

<phone\_number>01234567890</phone\_number><note>none</note></test>'

0x02C5DED0这是一个十六进制的内存地址，而第二次输出和第三次输出的区别就在于id\_number="01234567890"，这是设置id后修改的内容。

另外一点需要说明一下。可以看到，两次输出的内容前面有一个b’，这说明输出的字符是以字节码形式输出的。

## 7.4 解析XML

有一个知识点需要读者在学习本节的内容之前了解。和HTML类似，XML在开标签中紧跟着标签名后添加的内容表示为这个标签的属性。

解析XML和上面用到的库大致相同却又略有差异，这里需要用到ElementTree库。通过以下代码导入这个库。

from xml.etree import ElementTree as et

按照常规方式导入库后，我们看一看这里的几种方法。

>>>xml\_string=b'<test><name>zhang\_san</name>

<gender note="transsexuals">male</gender>

<phone\_number>01234567890</phone\_number><note>none</note></test>'

>>> root=et.fromstring(xml\_string)

>>> root.tag

'test'

>>> root.attrib

{}

这里首先用到了和上面相同的XML字符，然后用到了formstring()方法。

Python的XML库以两种方法读取文件。其中一种就是我们刚才使用的fromstring()方法，顾名思义，它用于从字符串中读写XML。此外，还有一种用来从文件中读取的方法。

root = ElementTree.parse(r"D:/test.xml")

我们刚才用到了root.tag，它用来获取根节点。root.attrib则可以获取根节点的属性，因为这里的根节点并没有定义属性，所以在这里输出为一个空的字典。

用多个例子来证明，这样更有说服力。下面这个例子还会用到.attrib方法。

>>> for i in root:

print(i.tag,i.attrib)

name {}

gender {'note': 'transsexuals'}

phone\_number {}

note {}

root返回的是一个字典（但你不能直接使用，比如，你在IDLE中输入root之后，返回的不是一个字典，而是一个内存地址）。如果你遍历字典，字典中存储的内容不能直接通过指定字典中的序数来得到。同样，你需要使用xml的.text方法，才可以获取具体的内容。

>>> for i in root:

print(i)

<Element 'name' at 0x000000541846F9F8>

<Element 'gender' at 0x000000541846FAE8>

<Element 'phone\_number' at 0x000000541846FBD8>

<Element 'note' at 0x000000541846FB88>

>>> for i in root:

print(i.text)

zhang\_san

male

01234567890

none

在第一次遍历时，我们只获取到了这些元素存储的内存地址。使用了.text方法后，在第二遍遍历时，就输出了相当于字典中value的内容。当然，如果要组合成类似于字典的key:value形式，可以使用i.tag，”:”，i.text。

这里用不同的方式来将XML文件转为键值对的形式，这里的重点不在于输出的内容和形式，而在于我们实现的思路。

#导入库

from xml.etree import ElementTree as et

#定义字符串

xml\_string=b'<test><name>zhang\_san</name><gender>female</gender><phone\_number>01234567890</phone\_number><note>none</note></test>'

#从文本从获取XML

root=et.fromstring(xml\_string)

#instance 1

print("\*\*\*\*\*====================================\*\*\*\*\*")

#列表形式输出的内存地址

for i in list(root):

print(i.tag,i.text)

#instance 2

print("\*\*\*\*\*====================================\*\*\*\*\*")

test\_person=root.getiterator("test")

#用来输出内容

for i in test\_person:

for j in i:

print(j.tag,j.text)

#instance 3

print("\*\*\*\*\*====================================\*\*\*\*\*")

#返回的是一个列表

find\_node2=root.findall("name")

print(find\_node2[0],find\_node2[0].tag,find\_node2[0].text)

#instance 4

print("\*\*\*\*\*====================================\*\*\*\*\*")

#查找获得的第一个

find\_node=root.find("name")

print(find\_node,find\_node.tag,find\_node.text)

我们来分析一下运行结果，例子1，2都是的输出结果都是一样的

\*\*\*\*\*====================================\*\*\*\*\*

name zhang\_san

gender female

phone\_number 01234567890

note none

list()用于将我们读取的root来转化为一个列表，而在第二个例子，我们通过迭代getiterator()然后使用for循环，实现的效果和直接使用list()函数的作用是相同的

例子3和例子4的输出结果也是一样的

<Element 'name' at 0x0000001A94F8B8B8> name zhang\_san

在例子3中，findall()方法返回的是一个例子，而在例子4中find()方法返回的是第一个查找到的结果，所以我们从中取出第一个元素和查找第一个元素一样。

## 7.5 Base64编码

Base64的名字在一定程度上解释了它的形式，Base64基于64个可打印字符来表示二进制数据。64是怎么来的呢？Base64编码中每6位是一个单元，而26等于64。

Base64主要应用于对网络上8位字节码传输的编码（如电子邮件），也部分应用于加解密（严格意义上不算加密，因为它的可逆性，只是采用Base64后对人类来说不可读）。另外，曾经比较火爆的两款下载软件—— 迅雷、QQ旋风使用的专用链接也采用了Base64编码。

Python中提供了一种Base64编码（这是内置库）的解决方案，在Python中，你使用Base64编解码就像print()那么简单。

Base64库只有两个主要函数—— encode()和decode()。下面给出一段使用encode()和decode()函数的示例代码。

#导入库

import base64

#定义字符串

string=b"admin"

#encode表示编码，decode表示解码

encoded=base64.b64encode(string)

decoded=base64.b64decode(encoded)

#输出

print(encoded)

print(decoded)

输出结果如下。

b'YWRtaW4='

b'admin'

## 7.6词性分析、统计分析NLTK

NLP(Natural language processing，自然语言处理)是一门涵盖语言学、计算机科学、人工智慧等领域的科学，一种研究人与计算机之间如何用自然语言进行通信的理论方法。简单来说，NLP就是开发能够理解人类语言的应用程序或服务。

这里讨论一些自然语言处理的例子，如分词、分句、词性划分、理解匹配词的同义词。当然，这并不是NLP能做的所有事情，读者只需要掌握常用的方法。

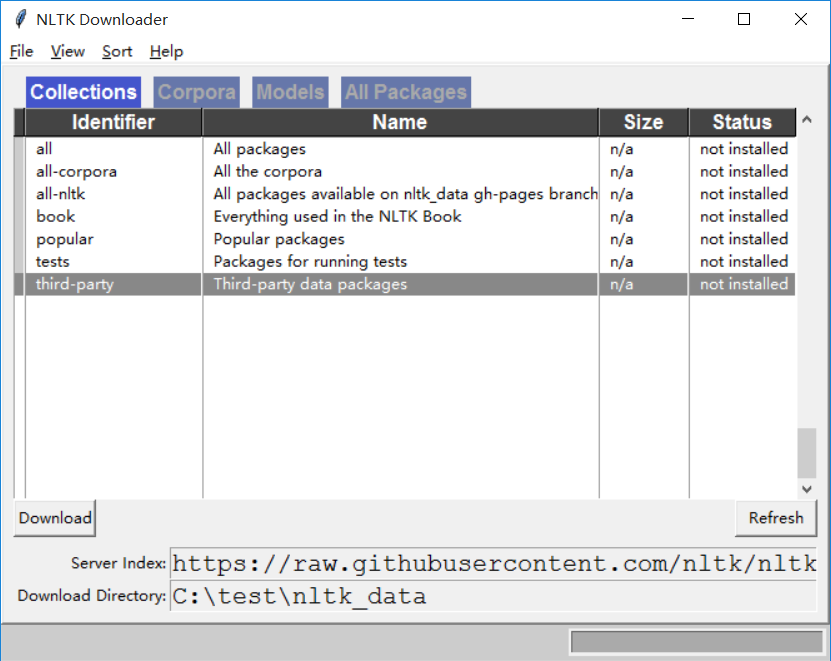
NLTK（Natural Language ToolKit，自然语言工具包）在用 Python 处理自然语言的工具中处于领先的地位，同时，它基于Python构建。它提供了 WordNet 这种方便处理词汇资源的接口（拥有超过50个语料库和词汇资源），还有分类、分词、除茎、标注、语法分析、语义推理等类库。最让人惊喜的一点是，它基于Apache 2.0开源协议免费发布。

NLTK是一个第三方包，也是在NLP领域中最常使用的一个Python库。需要通过pip或者其他的包管理器或者源代码安装NLTK。然而，仅仅安装NLTK库，它的大部分方法仍然无法使用。在NLTK中，自带了一个函数download()。通过以下代码会打开一个类似于图7-1的界面。

import nltk

nltk.download()

所以，可以使用这个方法来下载它绝大部分的库。遗憾的是，它的下载速度非常慢，如果你想要下载完所有内容，那么你可能需要一到两天，甚至更长的时间，并且在下载时还可能会碰到传输失败等问题。

图7-1 通过nltk.download()打开的界面

如果不想重复打开这个页面（作为一个极简主义者，作者还是觉得使用命令更加炫酷），那么你可以用如下命令。

nltk.download('[包名]')

下载完成之后，需要将下载的内容存放在这些位置之一。

- 'C:\\Users\\xxxxx\nltk\_data'

- 'C:\\nltk\_data'

- 'D:\\nltk\_data'

- 'E:\\nltk\_data'

#后三个盘符、目录随着你的Python安装目录而改变

- 'F:\\Python\\nltk\_data'

- 'F:\\Python\\share\\nltk\_data'

- 'F:\\Python\\lib\\nltk\_data'

- 'C:\\Users\\xxxxx\\AppData\\Roaming\\nltk\_data'

另外，部分包是压缩包，需要解压。在解压时，记得选择解压到当前文件夹，而不是解压到以文件夹名字命名的新建文件夹，否则会多一层目录，而导致NLTK无法调用nltk\_data。

如果你很自信地认为已经整理好nltk\_data并且能够正常使用，那么你可以使用以下命令来确认。

from nltk.book import \*

返回的结果大致如下。

\*\*\* Introductory Examples for the NLTK Book \*\*\*

Loading text1, ..., text9 and sent1, ..., sent9

Type the name of the text or sentence to view it.

Type: 'texts()' or 'sents()' to list the materials.

text1: Moby Dick by Herman Melville 1851

text2: Sense and Sensibility by Jane Austen 1811

text3: The Book of Genesis

text4: Inaugural Address Corpus

text5: Chat Corpus

text6: Monty Python and the Holy Grail

text7: Wall Street Journal

text8: Personals Corpus

text9: The Man Who Was Thursday by G . K . Chesterton 1908

与其称为书，不如称为数据集更合适， text1~9代表一个超长字符串的文本。例如，text1（原文在数据包下载目录下的gutenberg.zip中的melville-moby\_dick.txt）是一本名为 *Moby Dick*的书，而text6则是1975年的一部讽刺喜剧。

它们主要用于分析和测试。在本节中，我们将主要使用text1和text3来进行自然语言处理。

首先，看以下命令。

>>> text1.concordance('different')

该命令用于从这一大串字符串中查找包含different这个单词的语句，返回的结果如下。

among many others we spoke thirty **different** ships , every one of which had had

and , now soft with entreaty . How **different** the loud little King - Post . " Si

f at the hip , now , it would be a **different** thing . That would disable him ; b

have thought that in him also two **different** things were warring . While his on

herein are several pictures of the **different** species of the Leviathan . All the

e . The other engraving is quite a **different** affair : the ship hove - to upon t

tents , to Queequeg it was quite a **different** object . " When you see him ' quid

要统计标点符号和单词出现的频率，下面这条命令在进行数据处理的时候相当常用。

>>> from nltk import FreqDist

>>> fdist=FreqDist(text1)

>>> fdist.most\_common(10)

[(',', 18713), ('the', 13721), ('.', 6862), ('of', 6536), ('and', 6024), ('a', 4569), ('to', 4542), (';', 4072), ('in', 3916), ('that', 2982)]

>>> fdist.plot(10)

运行结果如图7-2所示。

我们分析一下这几行代码。第1行按照常规导入要使用的库。第2行使用FreqDist()方法来格式化text1，将其排列为一个按照分割后单词（或者标点）及其使用频率分组的列表。在第3行中，fdist获取的内容是一个列表，列表中每个元素都是一个元组，第一个值为标点或单词，第二个为其出现的频率。

most\_common([数字])用于获取这个字典最前面的10个字符，也即最常用的单词。

fdist.plot([number])则用于指定将前多少个最高频单词在文本中出现的位置绘制成图（在用到这个绘图功能之前，需要首先安装NumPy 和Matplotlib 包）

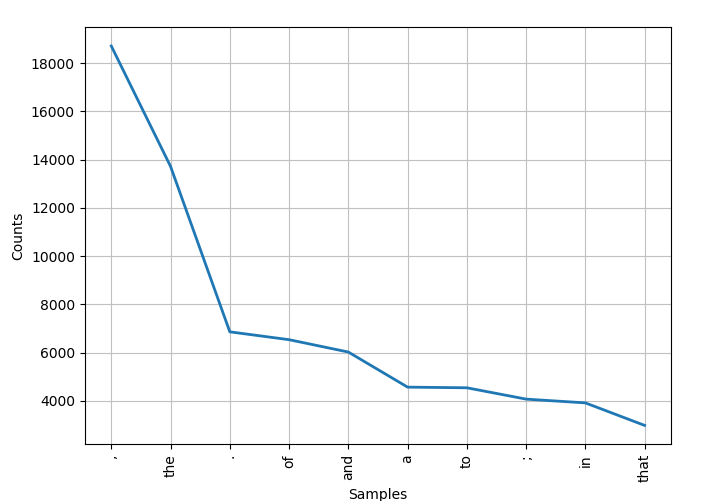


图7-2 统计高频单词（或者标点）的运行结果

另外，如果使用下面这条命令，可以查看这些单词在文本中的分布位置。

text4.dispersion\_plot(['citizens','democracy ', 'freedom ', 'duties', 'America'])

运行结果如图7-3所示。*x*轴代表全部文本，每一个蓝色的短纵列都代表一次出现。

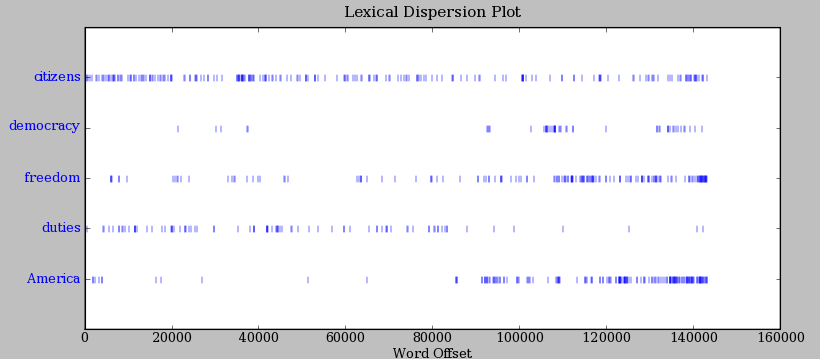


图7-3 单词在文本中的分布位置

除了简单的文本统计之外，NLTK还提供了许多的分析方法。下面给出一段示例代码。

>>> text2.common\_contexts(["monstrous", "very"])

a\_pretty is\_pretty am\_glad be\_glad a\_lucky

这个函数会给我们两个单词具有相同上下文的单词，给定的两个参数必须首先用中括号括起来，然后再用小括号括起来。

还有一个功能类似的函数。

>>> text1.similar("different")

all various known that but much there great had on other noble can

over round dead then soon small pointing

text1.similar([文字])可以告诉我们和我们指定的字符具有相同作用（并不是它们的语义相同）的单词。比如，我们经常使用beautiful来形容girl，于是这个函数可能就会给我们返回handsome（同样用来形容girl）

NLTK中还有一个用于分句的命令。

>>> sent\_tokenize(text)

['One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin.', 'He lay onhis armour-like back, and if he lifted his head a little he could see his brown belly, slightly domed and divided by arches into stiff sections.', 'The bedding was hardly able to cover it and seemed ready to slide off any moment.', 'His many legs, pitifully thin compared with the size of the rest of him, waved about helplessly as he looked.']

如果是很规范的文本，我们直接根据标点符号进行断句就可以了，为什么还要使用这个命令呢？

那么像"Hi Mr. Liu, how are you? Blessing for you"这种文本呢？显然，称谓之间的标点符号是不可以把它单独分割成一个句子的，这时候NLTK的分句用法就会展现出它强大的功能。

>>> test\_text="Hi Mr. Liu, how are you? Blessing for you"

>>> sent\_tokenize(test\_text)

['Hi Mr. Liu, how are you?', 'Blessing for you']

除此以外，还有一个比较实用的功能，generate()函数可以用于生成类似某种文本风格的字符，在语义、逻辑上可以文本保证大体通顺。但稍稍有些遗憾，我们可以从下面的源代码中（Python目录下\Lib\site-packages\nltk中test.py的第492行）看到这个功能在新版本的NLTK中不再支持，所以如果你想使用，那么你可以安装2.0版本的NLTK（一种简单方法是通过pip指定安装的版本）。

def generate(self, words):

"""

Issues a reminder to users following the book online

"""

import warnings

warnings.warn('The generate() method is no longer available.', DeprecationWarning)

但我们还是简单说一下这个函数的用法。如果使用NLTK自带的text1，那么要生成和text1相同风格的字符，只需要输入以下代码。

text1.generate()

接下来，介绍在进行数据处理的时候几个常用的方法。

* set([string])，用于获取去重后文本中包含的字符。
* sorted，用于对文本进行排序，从各种字符开始，然后按照字母表排序，下面是一个例子。

['!', "'", '(', ')', ',', ',)', '.', '.)', ':', ';', ';)', '?', '?)', 'A', 'Abel', 'Abelmizraim', 'Abidah', 'Abide',

'Abimael', 'Abimelech', 'Abr', 'Abrah', 'Abraham', 'Abram', 'Accad', 'Achbor', 'Adah',.....,'yonder', 'you', 'young', 'younge', 'younger', 'youngest', 'your', 'yourselves', 'youth']

* len()，用于获取文本中元素或者字符的个数，可以和其他方法组合起来使用，比如以下代码。

#获取不重复单词（标点）个数

>>> len(set(text3))

2789

对于上面获取到的内容进行一些简单的四则运算。下面这行代码可以统计每个词出现的频率。

>>> print(len(text3)/len(set(text3)))

16.050197203298673

另外，通过如下代码，可以获取文本中超过15个字符的单词。

>>> text\_set=set(text1)

>>> long\_words=[i for i in text\_set if len(i)>15]

>>> print(long\_words)

['circumnavigation', 'indispensableness', 'simultaneousness', 'characteristically', 'subterraneousness', 'supernaturalness', 'circumnavigations', 'Physiognomically', 'hermaphroditical', 'irresistibleness', 'uninterpenetratingly', 'uncomfortableness', 'uncompromisedness', 'preternaturalness', 'responsibilities', 'cannibalistically', 'circumnavigating', 'comprehensiveness', 'CIRCUMNAVIGATION', 'undiscriminating', 'superstitiousness', 'physiognomically', 'apprehensiveness', 'indiscriminately']

下面这种使用方法可以用来统计出现次数超过150的单词或者标点（限于篇幅原因，最后的输出结果有删减）。

>>> fdist=FreqDist(text1)

>>> freq\_word=[i for i in fdist if fdist[i]>150]

>>> freq\_word

['by', '.', '(', 'a', 'to', 'The', '--', 'in', ',', 'and', ';', 'I', 'see', 'him', 'now', 'He', 'was', 'ever', 'his', 'old', 'with', 'all', 'the', 'of', 'world', 'it', '"', 'you', 'hand', 'them', 'what', 'whale', '-', 'is', 'be', 'our', 'out', 'through', 'which', 'yet', 'Captain', 'deck', 'good', 'last', 'cried', 'know', 'Queequeg', 'Pequod', 'Ahab', 'Starbuck', 'Stubb']

下面这种使用方法会输出文本中频繁出现的双连词。

>>> text1.collocations()

Sperm Whale; Moby Dick; White Whale; old man; Captain Ahab; sperm

whale; Right Whale; Captain Peleg; New Bedford; Cape Horn; cried Ahab;

years ago; lower jaw; never mind; Father Mapple; cried Stubb; chief

mate; white whale; ivory leg; one hand

WordNet是一个基于认知语言学并且功能很强大的英语词典，由普林斯顿大学的心理学家、语言学家和计算机工程师联合设计。我们完全可以用它来获取相关单词的解释和例子。

#获取一个词和它的变形的语义

syn = wordnet.synsets("pain")

#获取定义

print(syn[0].definition(),"\n")

#获取例子

print(syn[0].examples())

通过syn会得到一个列表，在这个列表中，会有很多释义，我们仅仅选择最广泛、最实用的那个。

## 7.7 散列表

散列算法和一般算法最大的区别在于一般的加密算法是一一映射（对特定值都进行某种变换，每个数有且仅有一个对应的函数值，每个函数值有且仅有一个对应的数，请参考函数的一一映射），因此理论上散列算法是可逆的。散列算法的定义域是一个无限集合，而值域（输出位数固定）是一个有限集合，将无限集合映射到有限集合，根据“鸽笼原理”(Pigeonhole principle)，每个散列结果都存在无数个可能的目标文本（请参考函数的满射）。因此，散列不是一一映射的，符合是不可逆的。

散列算法是一类算法的统称，符合上面两个定义的映射仅仅可以叫作散列算法和加密算法，但未必是好的散列算法和加密算法。好的散列算法和加密算法往往要求它很难发生文本碰撞（即输入的数不同，而输出的函数值相同）。

好的散列算法应该对于内容的改变极其敏感，即使输入有很小的改动，如1亿个字符中变了1个字符，那么结果应该截然不同。这就是为什么散列算法可以用来检测软件的完整性。

散列算法有很广泛的应用，加密常用的MD5（Message-Digest Algorithm 5，信息摘要算法5）、SHA（Secure Hash Algorithm，安全散列算法）都是散列算法。很多网站的密码都是采用散列值加密的。当你忘记密码的时候，网站并不是将你忘记的密码发送给你，而是让你重新修改密码，然后让你用这个新密码登录。这是因为你在注册时输入的密码的散列值存储在数据库里，而散列算法不可逆，所以即使通过服务器也不可能复原你的密码，而只能重置密码。

散列算法领域中加密常用的算法是SHA1和MD5。我们用Python的hashlib库实现这两种加密。

#导入散列算法库

import hashlib

#使用MD5加密

md5\_1 = hashlib.md5()

md5\_1.update(b'I’m using md5 in python')

print(md5\_1.hexdigest())

#如果数据量很大，可以分块多次调用update()，最后计算的结果是一样的

md5\_2 = hashlib.md5()

md5\_2.update(b'I’m using md5')

md5\_2.update(b' in python')

print(md5\_2.hexdigest())

#使用SHA1加密

sha1 = hashlib.sha1()

sha1.update(b'I’m using sha1 in python')

print(sha1.hexdigest())

输出结果如下。

ef262faba34b6ad650c6e29d23622ce0

ef262faba34b6ad650c6e29d23622ce0

7fbb6478e03e9a49f2fdd6e2744a39d08290b168

Update([字符])用来上传我们需要加密的字符串，hexdigest()用于计算结果值。

第一次，直接一次性上传字符。第二次，分布上传，这种上传方式适用于数据较大的情况。另外要注意的一点是，在散列算法中空格也是有效的，因此你需要注意你是否忽略了某个空格而导致输出值不同。第三次，我们采用SHA1加密，同样，这种加密方式也有分布上传。

那么我们怎么使用散列呢？如果数据仅仅用作比较验证，不需要还原成明文形式，那么使用散列算法加密；如果数据在以后需要被还原成明文，采用可以其他加密算法。

## 7.8 布隆过滤器

1970年由布隆提出的布隆过滤器是一个随机映射函数，可用于快速检验一个元素是否在一个集合中。布隆过滤器的空间效率和查询速度都是一般的查询算法可以比较的（布隆过滤器的实现基于散列算法），因此在内容查询上具有很大的优势。高速度查询的同时也带来一定的缺点，它存在一定的识别率和错误率（非常低）。可以说，布隆过滤器用错误率来换取时间和空间。

在处理超大型数据时，布隆过滤器是必需的。如果你如果用旧的一个一个对比的方式来查看某个元素是否在1TB的数据中，那么无论你的计算机/服务器配置多高，你都会至少卡顿几秒。

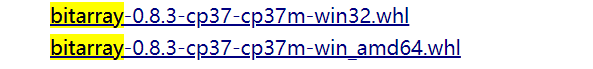
我们需要一个叫作pybloom\_live的库，你在使用这个库的时候，可能会提示你需要下载庞大的VC运行库，提示如下。

Microsoft Visual C++ 14.0 is required.

Get it with "Microsoft Visual C++ Build Tools":

http://landinghub.visualstudio.com/visual-cpp-build-tools

但你完全可以避免下载这个庞大的运行库，pybloom\_live中出现的这个问题是因为缺少一个叫作bitarray（这个库基于CV++14）的库，通过搜索[Unofficial Windows Binaries for Python Extension Packages](https://www.lfd.uci.edu/~gohlke/pythonlibs/)即可在加利福尼亚大学官网下载相应的whl文件。别忘了，根据自己下载的Python版本进行文件选择，例如，cp37表示Python3.7版本，我们应该选择如图7-4所示的两个文件。

图7-4 bitarray版本

然后，通过以下命令就可以成功安装pybloom\_live了。

pip install bitarray-0.8.3-cp37-cp37m-win\_amd64.whl

我们基于pybloom\_live实现布隆过滤器。具体代码如下。

#导入库

import os

from pybloom\_live import BloomFilter

#数据库文件

animals = ['dog', 'cat', 'giraffe', 'fly', 'mosquito', 'horse', 'eagle',

'bird', 'bison', 'boar', 'butterfly', 'ant', 'anaconda', 'bear',

'chicken', 'dolphin', 'donkey', 'crow', 'crocodile','testadd']

#判断文件是否存在

#如果文件存在，就读取，否则，创建文件

is\_exist = os.path.exists('test.blm')

if is\_exist:

bf = BloomFilter.fromfile(open('test.blm', 'rb'))

#如果没有该文件，则创建bf对象，最后保存文件

else:

bf = BloomFilter(20000, 0.001)

#如果存在，则跳过；如果不存在，则写入

for i in range(10):

if i in bf:

print('pass')

pass

else:

print('add %s' %i)

bf.add(i)

bf.tofile(open('test.blm','wb'))

#判断是否存在

for i in range(20):

if i in bf:

print("written")

else:

print("unwritten")

运行结果如下。

add 0

…….

add 8

add 9

written

……

written

written

unwritten

……

unwritten

unwritten

pybloom\_live的用法比较简单。下面介绍读取和创建文件的方法。

通过以下代码从文件中读取。

BloomFilter.fromfile(open('test.blm', 'rb'))

通过以下代码创建新的布隆文件。

BloomFilter(20000, 0.001)

创建容量为20000，错误率为0.001的布隆过滤器。

要添加元素到文件中，可以使用bf.add(i)。

要把元素写入文件中，可以使用bf.tofile(open('test.blm','wb'))。

再来考虑一下刚才写的代码

我们在第一次运行时插入前10个值，后来查询后20个值，所以对于第11~20个值会输出unwritten，而对于前10个值输出written。

如果再次运行，你就会发现里面的add全部变为pass了。

pass

…….

pass

pass

## 7.9 关系数据库MYSQL

接下来，介绍关系数据库。数据库是一种比较古老的UNIX持久化存储，基于文件，采用类似于字典的对象接口实现。

很多情况下, SQL 已经成为 “数据库” (database) 的一个同义词。实际上，SQL是 Strctured Query Language 的首字母缩写，而并非指数据库技术本身。 更确切地说， 它所指的是从RDBMS (Relational Database Management System，关系数据库管理系统) 中检索数据的一门语言。

### 7.9.1安装MySQL

MySQL也属于RDBMS 中的一员。

数据以一定方式存储在一起，数据库实现了数据共享，通常使用文件系统作为基本的持久化存储工具。文件系统有尽可能小的冗余度，避免了重复输入信息，甚至提高了数据存储的效率和准确度。

大多数数据库提供了GUI（Graphical User Interface，图形用户界面）。可以直接在GUI中创建表、添加表，虽然这可能更方便一些，但是直接使用命令行更方便我们理解数据库。

接下里，以MySQL为例子，介绍关系数据库。读者可以在MySQL网站下载Windows系统下的MySQL数据库。

下面说明几个需要注意的事项。其他情况下读者选择默认值就可以了。建议读者选择约300MB的离线安装包，15MB的在线安装包在安装过程中还需要联网和下载，这一过程需要的时间远远超过直接在官方网站上下载的时间。

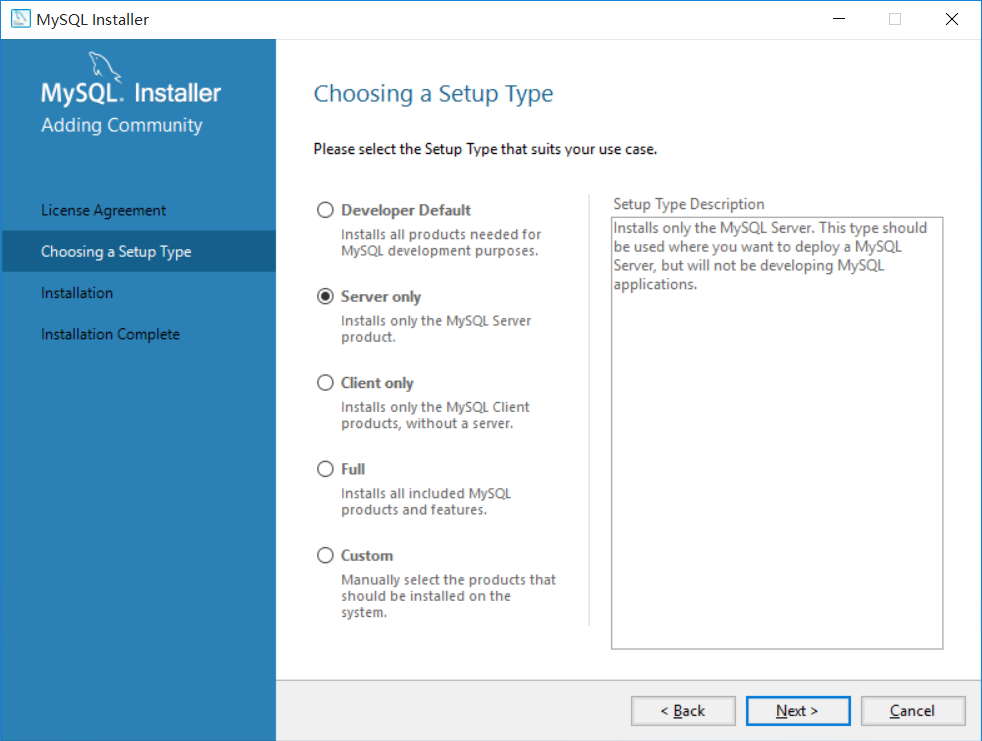
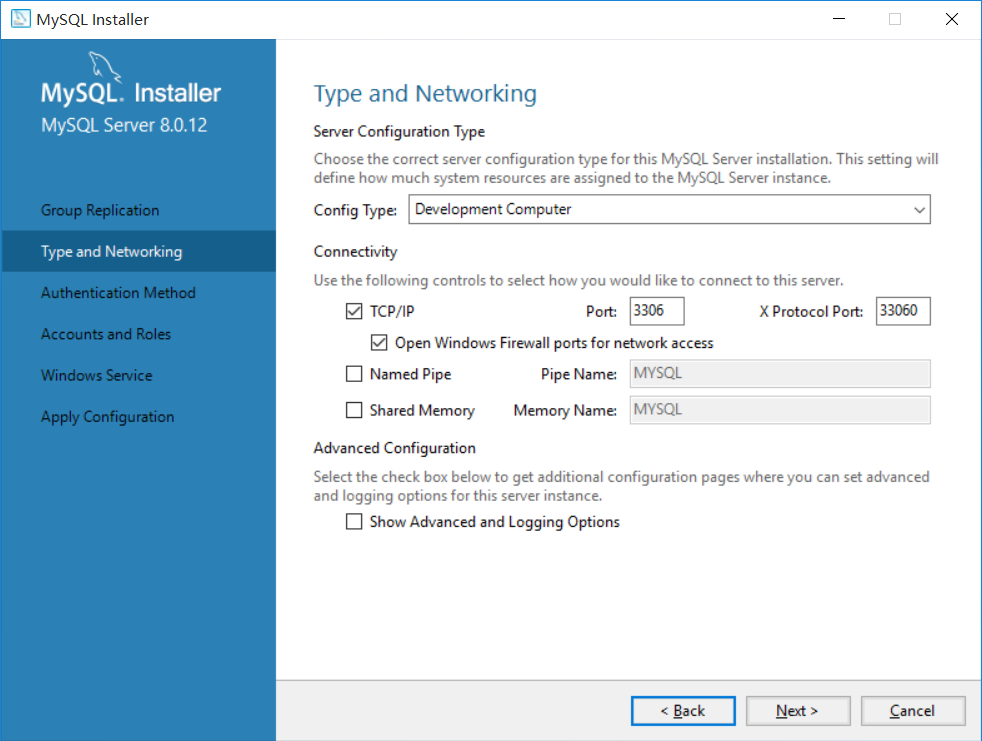
首先，关于安装方式，如图7-5所示，建议在安装类型中选择Server only单选按钮（我们暂时只需要在测试中使用），这样你可以避免下载一系列和数据库相关的软件。

图7-5 选择安装类型

然后，类型和网络的配置如图7-6所示，在这一步保持默认设置即可。

图7-6 类型和网络的配置

最后，选择使用认证方式（见图7-7），使用强加密验证或传统的方式。这里选择两个单选按钮中的任意一个均可，但选择第一个单选按钮存在安装失败的风险，可能会存在不支持的情况。

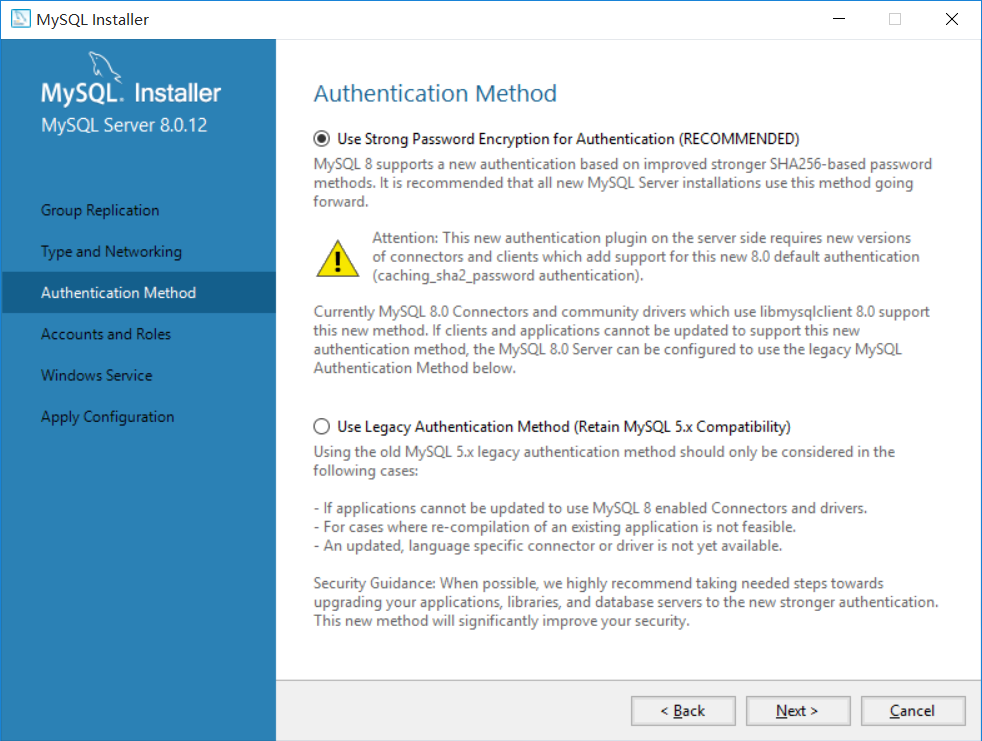
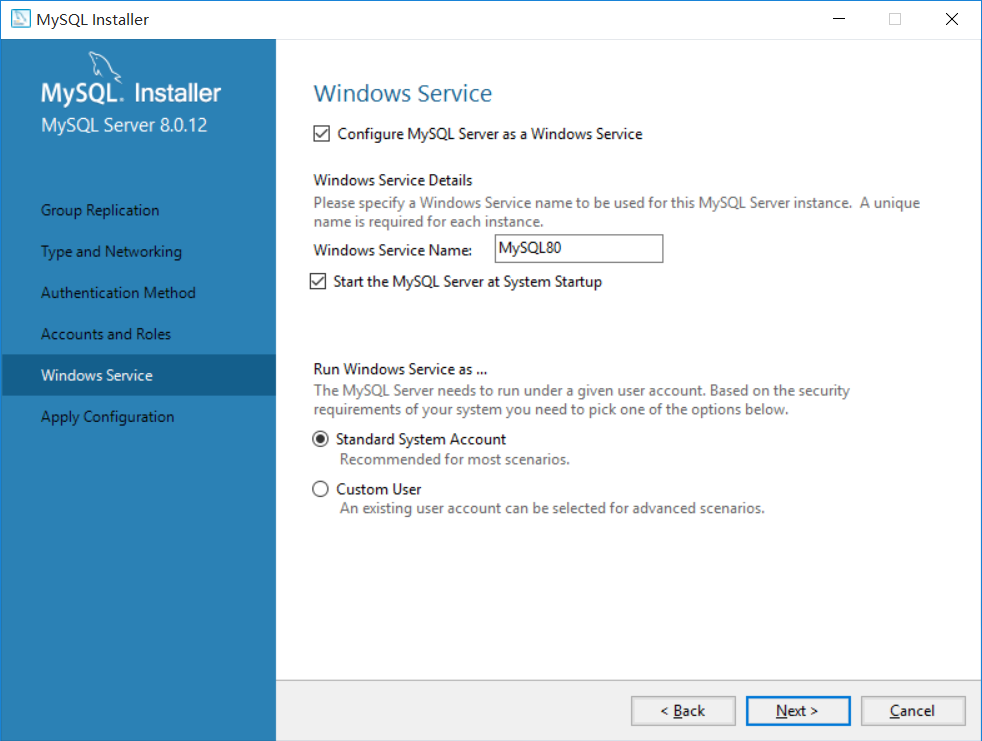


图7-7 选择认证方式

在图7-8所示界面中， Configure MySQL Server as a Windows Service复选框表示把MySQL服务器作为一个系统服务来运行， Start MySQL Server at System Starup复选框表示开机启动MySQL服务器。Run Windows Service as选项用于设置MySQL作为系统服务运行的身份，Standard System Account单选按钮表示标准定制账户，Custom User单选按钮表示定制用户。

图7-8 配置MySQL服务器

建议勾选Configure MySQL Server as a Windows Service复选框，取消勾选Start MySQL Server at System Starup复选框，选择Standard System Account单选按钮。

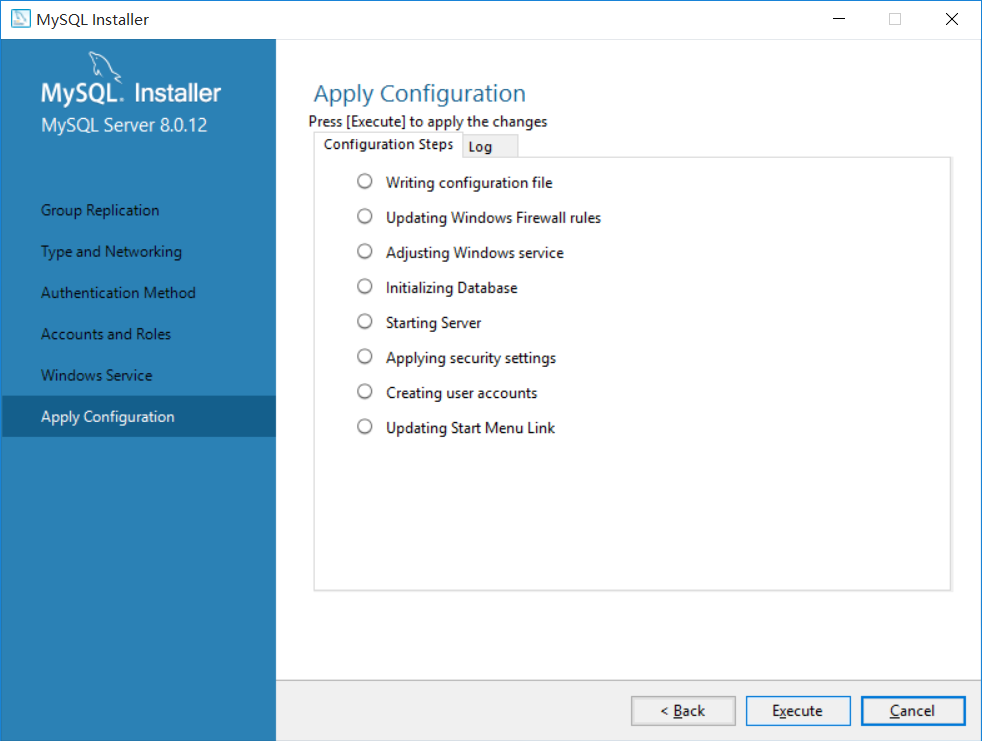
这是最后的配置步骤，在这一步单击Execute按钮，等待执行结束后，MySQL的安装步骤就完成了（见图7-9）。

图7-9 配置完成

在安装完成了以后，可以通过这条命令来启动MySQL的命令行或者直接通过启动栏的图标打开。

mysql -uroot -p

在Windows系统下，MySQL中的数据库和表名都不区分大小写（MySQL在存储和查找的时候，将所有的数据库和表名一致转换为小写字母）。而在Linux系统中，MySQL中的数据库和表名区分大小写。

### 7.9.2 MySQL常用命令

建议读者在使用的时候，提前设置好一定的规则，比如，统一使用大写加数字或者统一使用小写加下划线。如果读者使用Windows系统，为了减少使用大小写切换的次数，把命令写成小写形式也是没有关系的。

通过以下命令，创建数据库。

mysql> CREATE DATABASE test;

Query OK, 1 row affected (0.09 sec)

从字面上理解即可，这里创建了一个名为test的数据库。

通过以下命令**，**使用数据库。

mysql> USE test；

Database changed

在调用某个数据库的内容之前，需要先使用USE命令来切换到这个数据库。

通过以下命令，删除数据库。

mysql> DROP DATABASE test;

Query OK, 0 rows affected (0.02 sec)

删除数据库的同时，数据库中的所有表和数据也会从系统中移除，无论任何时候，错误使用这条命令都是很严重的。

有一点需要提醒读者，在MySQL中，命令以分号结尾，如果命令结束，却没有使用分号，可能会让你多行式输入命令。下面给出一段示例代码。

mysql> DROP DATABASE test

-> ;

Query OK, 0 rows affected (0.02 sec)

通过以下命令，查看数据库。

mysql> SHOW DATABASES;

这条命令用来查看当前已经有的数据库，效果如图7-10所示，前4个数据库都是MySQL默认存在的，如果你不了解，最好不要对它们做任何更改。如果要和这里的输出内容相同，那么需要重复创建test数据库。

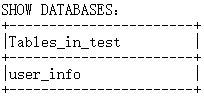


图7-10 查看数据库

表在数据库中用单词TABLE来表示，**要创建表，可以使用以下命令。**

mysql> CREATE TABLE user\_info(log\_in VARCHAR(10),id INT,password INT);

Query OK, 0 rows affected (0.17 sec)

在这个语句中，创建了一个有3个参数的表，第一个参数设置为不超过10个字符的字符串，第二个参数user\_id和第三个参数pro\_id都设置为整数。

然后向其中加入信息。

INSERT INTO user\_info VALUES("li\_si",0,123456);

每一个参数都和刚才设置的参数对应。这里建议读者多插入几条数据，不仅是为了练习，更是为了方便我们执行后面的操作。

要查看表，可以使用以下命令。

mysql> SHOW TABLES;

要查看数据，可以使用以下命令。

SELECT \* FROM user\_info

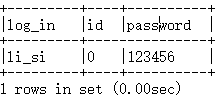
结果如图7-11所示。

图7-11 查询user\_info表的结果

这条命令有很多种用法，可以和其他的命令互相结合。比如，想要实现条件化查询，可以使用以下几种方法。

#在user\_info表中查询所有id小于10的账户

SELECT \* FROM user\_info WHERE id<10;

#在user\_info表中查询所有密码等于password的账户

SELECT \* FROM user\_info WHERE password=123456;

#在user\_info表中查询所有log\_in=li\_si的账户

SELECT \* FROM user\_info WHERE log\_in="li\_si";

WHERE语句之后可以跟设置的任意参数，然后对其进行比较，并返回符合条件的数据。

要更新表中的数据，可以使用以下命令。

mysql> UPDATE user\_info SET id=1 WHERE id=0;

Query OK, 1 row affected (0.09 sec)

Rows matched: 1 Changed: 1 Warnings: 0

这行语句的使用格式如下。

UPDATE [表名] SET [重设参数] WHERE [查询条件]

还有其他不同的形式。

#设置password=123456的账户的用户名为test

UPDATE user\_info SET log\_in="test" WHERE password=123456;

#将id=1的账户的密码设置为123456

UPDATE user\_info SET password=12345678 WHERE id=1;

要删除行，可以使用以下命令。

DELETE FROM user\_info WHERE id=1;

DELETE语句和SELECT语句很像，都可以使用WHERE语句进行条件化查询，但DELETE语句与之相比没有符号“\*”。

另外，如果要删除数据库中的所有数据，比如，游戏中进行一个删档测试，那么你会用到以下命令。

DELETE FROM user\_info;

如果要在删除所有数据的同时删除掉表，那么可以直接使用以下命令删除表和其中的所有数据。

DROP TABLE user\_info;

在对SQL语句有了基本的理解之后，我们来实现Python和MySQL的交互，这需要借助pymysql这个第三方库。可以使用pip来安装pymysql。

pip install pymysql

或者可以在网络下载这个库的源代码，然后执行这条命令。

python setup.py install

下面通过代码来展示怎样使用pysql来进行MySQL和Python的交互。

示例代码如下。

#导入模块

import pymysql

#连接数据库

conn=pymysql.connect(host='127.0.0.1', user='root', password='password', database='test', port=3306, charset='utf8')

cur=conn.cursor()

print("\*\*\*\*\*====================================\*\*\*\*\*")

# 使用 execute()方法执行 SQL 查询

# 使用 fetchone() 方法获取单条数据.

version="SELECT VERSION()"

cur.execute(version)

data = cur.fetchone()

print ("Database version : %s " % data)

print("\*\*\*\*\*====================================\*\*\*\*\*")

select="SELECT \* FROM user\_info"

exe\_select=cur.execute(select)

#现在获取所有数据

all\_data=cur.fetchall()

print(all\_data)

#向下移动一行,绝对值从数据输出的上一行开始

#mode默认值为相对

cur.scroll(1,mode='absolute')

data1=cur.fetchone()

print(data1)

#向上移动两行

#因为游标现在相当于在输出的所有数据的下一行

cur.scroll(-2,mode='relative')

data2=cur.fetchone()

print(data2)

# 关闭

cur.close()

# 关闭数据库连接

conn.close()

输出结果如下。

Database version : 8.0.12

\*\*\*\*\*====================================\*\*\*\*\*

(('li\_si', 0, 123456), ('zhang\_san', 1, 12345678))

('zhang\_san', 1, 12345678)

('li\_si', 0, 123456)

pymysql.connect()语句的参数比较多，下面讲解这个命令的几个参数。

* host：连接的mysql主机，本机设置为'localhost'或者'127.0.0.1'。
* port：连接的mysql主机的端口，默认是3306。
* db：需要连接的数据库的名称。
* user：用户名。
* password：密码。
* charset：编码方式，默认是'gb2312'。

运行以下命令。

cur=conn.cursor()

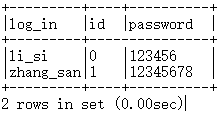
结果如图7-12所示。

图7-12 运行conn.cursor()查询的结果

首先，使用这条命令来连接并创建一个游标或者一个类似于游标的对象。

cur.execute()

execute()函数可以用来执行SQL语句。

cur.scroll(1,mode='relative')

scroll()用来移动创建的游标。这条命令默认是在相对模式下使用的。如果你使用完一条命令后，那么游标是默认在所有输出的下一行，命令输入框上（所以我们第三次输出scroll()命令的第一个参数为-2）。

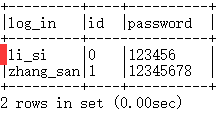
当然，如果要使用绝对模式，那么需要指定mode。当使用绝对模式的时候，光标相当于从第一行的输出数据开始，大致在图7-13加重标记的位置。

图7-13 光标的位置

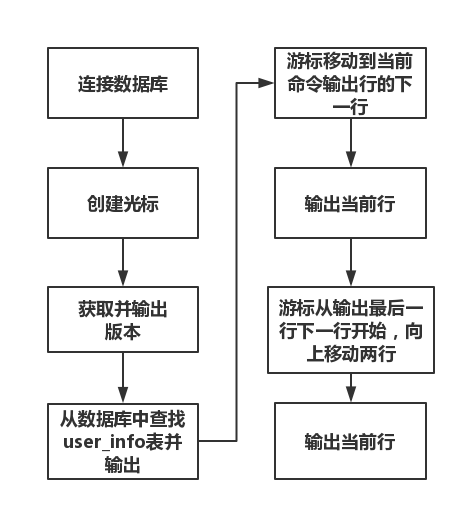
Scroll()命令的第一个参数表示我们要移动的方向和大小。如果是正数，那么游标向下移动；如果是负数，那么游标向上移动。

获取输出有两个命令，分别是fetchone()和fetchall()。fetchone()命令获取游标当前所在行的命令，而fetchall则获取所有输出的数据。

在最开始我们输出账户、密码、需要连接的数据库、端口来连接到数据，然后我们创建一个游标，接着执行SQL命令，获取数据，对所有数据进行查询并输出。因为当游标的参数为绝对值的时候，默认是在输出数据的第一行，所以我们向下移动一行获取到的是第二条数据。然后我们进行第三次数据，从输出最后一行的下一行开始，再向上移动两行，获取到的是第一条数据。

再来看一看数据增加、删除、修改，因为这些操作涉及数据库的修改，所以这些操作很重要。MySQL的执行机制与Git很像，需要我们在完成操作后使用commit()提交修改。

整个流程如图7-14所示。

图7-14 流程

其实读者在使用的时候只需要在所有代码的最后使用commit提交一次即可，这里这样书写仅仅是因为习惯问题。

#导入库

import pymysql

#创建数据库

conn=pymysql.connect(host='localhost',port=3306,db='test',user='root',passwd='password',charset='utf8')

cur=conn.cursor()

#查询原数据------------------------------------------------

select="SELECT \* FROM user\_info"

cur.execute(select)

all\_data=cur.fetchall()

print(all\_data)

print("\*\*\*\*\*====================================\*\*\*\*\*")

#插入数据

insert="INSERT INTO user\_info VALUES('li\_liu',2,123456789);"

count=cur.execute(insert)

conn.commit()

#输出插入后的数据

select="SELECT \* FROM user\_info;"

cur.execute(select)

all\_data=cur.fetchall()

print(all\_data)

print("\*\*\*\*\*====================================\*\*\*\*\*")

#删除数据

delete='DELETE FROM user\_info WHERE log\_in="wang\_wu";'

count=cur.execute(delete)

conn.commit()

#输出删除后的数据

select="SELECT \* FROM user\_info;"

cur.execute(select)

all\_data=cur.fetchall()

print(all\_data)

print("\*\*\*\*\*====================================\*\*\*\*\*")

#更改

update='UPDATE user\_info SET log\_in="im changed" WHERE id=0;'

count=cur.execute(update)

conn.commit()

#输出更改后的数据

select="SELECT \* FROM user\_info;"

cur.execute(select)

all\_data=cur.fetchall()

print(all\_data)

print("\*\*\*\*\*====================================\*\*\*\*\*")

#关闭

cur.close()

# 关闭数据库连接

conn.close()

输出的数据如下。

(('li\_si', 0, 123456), ('wang\_wu', 1, 12345678))

\*\*\*\*\*====================================\*\*\*\*\*

(('li\_si', 0, 123456), ('wang\_wu', 1, 12345678), ('li\_liu', 2, 123456789))

\*\*\*\*\*====================================\*\*\*\*\*

(('li\_si', 0, 123456), ('li\_liu', 2, 123456789))

\*\*\*\*\*====================================\*\*\*\*\*

(('im changed', 0, 123456), ('li\_liu', 2, 123456789))

\*\*\*\*\*====================================\*\*\*\*\*

我们第一条SQL语句（查询语句不计算在内）向其中添加了li\_liu（其实正确的顺序应该是zhang\_san 、li\_si 、wang\_wu 、li\_liu）的数据，然后对数据进行输出。第二条SQL语句删除了wang\_wu的数据。第三条语句修改了li\_si的log\_in参数，li\_si被替换成了im changed。

同样，pymysql库还有一个用于返回的操作——rollback()命令。在没有最终提交之前，通过这条命令可以回退一切操作。下面给出一段示例代码。

>>> import pymysql

>>> conn=pymysql.connect(host='localhost',port=3306,db='test',user='root',passwd='password',

charset='utf8')

>>> cur=conn.cursor()

>>> insert="INSERT INTO user\_info VALUES('li\_liu',2,123456789);"

>>> select="SELECT \* FROM user\_info"

>>> cur.execute(select)

2

>>> all\_data=cur.fetchall()

>>> print(all\_data)

(('im changed', 0, 123456), ('li\_liu', 2, 123456789))

>>> cur.execute(insert)

1

>>> cur.execute(select)

3

>>> all\_data=cur.fetchall()

>>> print(all\_data)

(('im changed', 0, 123456), ('li\_liu', 2, 123456789), ('li\_liu', 2, 123456789))

>>> conn.rollback()

>>> cur.execute(select)

2

>>> all\_data=cur.fetchall()

>>> print(all\_data)

(('im changed', 0, 123456), ('li\_liu', 2, 123456789))

这里在Python的交互式IDLE下演示了rollback()操作。初次执行查询语句，表中只有两条数据。接着向其中插入一条数据，再次查询，表中已经变为3条数据。执行rollback()后，最后一次查询，返回的数据与刚开始的数据一模一样。

## 7.10非关系数据库

常见的非关系（Not Only SQL，NoSQL）数据库包括以下几个。

* MongoDB
* CouchDB
* Redis
* Big Table

非关系数据库的本意其实是不仅仅SQL，作者认为，翻译为超关系数据库或者其他的更合适。事实上，非关系数据库后来确实是朝着非关系型的方向发展的（这一点和本意有所不同）。

NOSQL是基于键-值对的，键-值对可以视为表中主键和值的对应关系，不需要经过SQL层的解析，所以NOSQL的性能非常高。因为数据之间没有耦合性，所以非常容易水平扩展。关系型数据的优复杂查询功能较弱。

关系数据库和非关系数据库还有一点商业上的区别。大部分的关系数据库都是商业运行的，这就意味着如果你使用关系数据库，你需要为之支付一定的费用。虽然MySQL是免费的，但是MySQL需要一系列复杂的配置才能投入运行。

对于小型化创业公司，非关系型数据是一个不错的选择。

有了关系型数据的基础，非关系数据库使用的查询语言大部分和SQL相似，但非惯性数据的种类又略有不同，这里就不再详细说明。

## 7.11总结

在你刚刚上手爬虫的时候，上手会非常快，但紧接着，在对各种网页进行数据采集的时候，你就会发现，这些编码问题占据了编程的绝大部分时间。数据编码和数据库从来不是与网络爬虫分离的。相反，它们关系紧密，不可分割。

一旦踏上了编程的道路，如果不把编码问题搞清楚，那么你随时有可能在编码上碰壁，因为总有一天它会在你意想不到的地方造成严重的错误。

编码问题可能占据了在编程中的大部分时间。

上面讲的编码更多偏向于不同数据格式的解析、加/解密he 存储，而网页的编码和上面讲的编码又有所不同。很多国内网站采用GBK编码，而国际站采用UTF8编码。这两种编码在进行转换时就可能会出现部分错误。编码的混乱，很大程度上导致了读者书写爬虫的出错。作者在刚刚学习爬虫的时候，遇到过ASCII、ANSI、GB2312、GBK、UTF-8、UTF-16和ISO 8859-1这些编码，几乎被它们搞得焦头烂额，明明数据就在那里，但你就是无法转码和存储。

只有充分发挥不屈不挠的精神，和解决了编码问题，读者才有可能少走很多弯路。

至此，读者已经学习到了本书的大部分内容。但看完书是一回事，真正会用又是另一回事，不经过实践，难免纸上谈兵。

对于要学的技能，把理论内容看上千百遍都未必能熟练运用，而要经过真实的实践。对于要开发的项目，在心中想100遍都未必能有个完美的方案，而要真正动手去做。

接下来，我们将会对以上内容进行通汇贯通。通过两个项目的实践，加深读者对以上所学的内容的理解。

**第8章 基于猫眼电影的实战**

本章通过一个假设的项目需求，详细讲解从了解客户需求到网站分析、处理的流程，并在分析中逐步引入代码。本章将会帮助读者有效提高自己的业务能力。

## 8.1 网站分析

本书中的第三个实战项目基于猫眼电影。猫眼电影是国内一个有名的电影评论网站，性质和评分机制与豆瓣很相似。如果你是一个电影迷，那么你一定听说过这个网站。

假设读者接到客户的需求通知单，客户是一位自媒体运营人员，因为自媒体推广内容和电影相关，所以需要我们写一个程序来获取当前评价最高的电影榜单，为客户的公众号提供参考。

作为乙方，我们首先要选择一个获取数据的对象，然后我们将选择猫眼电影作为目标。

在对网站进行数据采集之前，我们先要看一下网站是否禁止我们的这种行为。打开猫眼电影的robots.txt看一看。网站一般都存在robots.txt，用于告诉爬虫和搜索引擎（实际上也是爬虫）哪些页面可以获取，哪些页面不可以获取。因为猫眼电影的robots.txt并没有声明不可以采集的目录（网站的robots.txt内容和网址分别如图8-1和图8-2所示），所以我们就可以放心地进行采集，但读者最好在夜间进行数据采集，夜间的访问量小，会减少对网站的正常运行的影响。

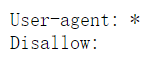


图8-1 robots.txt内容 图8-2 robots.txt网址

我们采集猫眼电影的TOP100榜单。客户不是专业人员，但作为专业人员，我们要对客户的需要进行扩展。首先，我们分析电影榜单中的一部示例电影，如图8-3所示。

图8-3 电影榜单中的一部示例电影

这里有几个要获取的要素：霸王别姬，这是电影名；主演，这也是要获取的内容；上映时间和电影的受欢迎程度并不存在具体的关联，不在我们考虑的范围内；电影的评分，客户肯定优先参考评分。但有了这些要素还远远不够，我们还需要通过以下代码获取电影的内容介绍。

<a href="/films/1203" title="霸王别姬" data-act="boarditem-click" data-val="{movieId:1203}">

霸王别姬</a>

我们考虑用XPath来进行内容的获取，找到共同规律后使用for循环拼接。先看一下XPath的规律。

//\*[@id="app"]/div/div/div[1]/dl/dd[1]/div/div/div[1]/p[1]/a

//\*[@id="app"]/div/div/div[1]/dl/dd[2]/div/div/div[1]/p[1]/a

//\*[@id="app"]/div/div/div[1]/dl/dd[3]/div/div/div[1]/p[1]/a

发现了没有？其实大部分内容都相同，不同的是dd[]中的数字。一个页面上有10条数据，并且数字是从1开始编号的，而我们平常在Python中的range()函数是从0开始编号的，因此我们考虑使用range(1,11)，即从1计数到10。

因为内容都在title标签中，所以可以使用get\_attribute(“title”)方法来获取title属性中的文字。

再来看一看导演这部分怎么获取。

<p class="star">

主演：张国荣,张丰毅,巩俐

</p>

这里文字部分被包裹在标签中。我们考虑使用前几章学过的.text方法，通过元素查找找到元素的位置，然后获取标签中的文字。

<p class="score">

<i class="integer">9.</i>

<i class="fraction">5</i></p>

关于评分的部分略微有些复杂。猫眼电影的整数部分的小数点在一个标签中，而小数部分又在一个标签中，但不管是整数部分还是小数部分，都在一个属性为score为p标签下。

再来看一下主演和评分部分的XPath。

主演部分的XPath如下。

//\*[@id="app"]/div/div/div[1]/dl/dd[1]/div/div/div[1]/p[2]

//\*[@id="app"]/div/div/div[1]/dl/dd[2]/div/div/div[1]/p[2]

//\*[@id="app"]/div/div/div[1]/dl/dd[3]/div/div/div[1]/p[2]

评分部分的XPath如下。

//\*[@id="app"]/div/div/div[1]/dl/dd[1]/div/div/div[2]/p

//\*[@id="app"]/div/div/div[1]/dl/dd[2]/div/div/div[2]/p

//\*[@id="app"]/div/div/div[1]/dl/dd[3]/div/div/div[2]/p

我们可以用和刚才一样的XPath拼接方法来处理，并通过循环获取数据。但一页结束后，我们怎么进入下一页呢？

再来分析一下猫眼电影的TOP100榜单的页面URL。

https://maoyan.com/board/4?

https://maoyan.com/board/4?offset=10

<https://maoyan.com/board/4?offset=20>

<https://maoyan.com/board/4?offset=30>

为什么只有第一个页面不同？可以看到，每个页面的offset在前一个页码的基础上增加了10，offset刚好对应页面的数据条数。但第一个页面又有所不同，没有offset参数。

我们进行一个大胆的猜测，第一个页面也有offset，只是offset的值为0，因此没有

显示。

实践出真知，直接访问检验一下。确实是这样，加入offset参数后，我们仍然能进入第一个页面。

## 8.2 最初的测试代码

首先，导入库。导入的库是本书主要使用的Selenium项目和Time模块。使用Time模块，而不使用显式等待，主要是为了让爬虫更像一个人，而不是因为爬虫过快的访问给浏览器带来困扰。

然后，我们从整体上来看TOP100榜单。因为每个页面有10条数据，所以我们需要构造10个页面链接。当然，根据个人爱好，把每个页面访问一遍把URL复制下来也可以。

这里使用列表，然后进行URL拼接，并写入列表中。这样列表中一共有10个元素（URL）。

接下来，再次使用FOR循环，对列表中的每个页面都通过XPath获取标题、分数、演员。最后，通过打开页面，获取电影的内容介绍（流程如图8-4所示）。

#导入库

from selenium import webdriver

import time

#序号列表

numbers\_list=[0,10,20,30,40,50,60,70,80,90]

url\_list=[]

for i in numbers\_list:

url="https://maoyan.com/board/4?offset=%d" %i

url\_list.append(url)

print("获取地址:%s" %url)

dr=webdriver.Chrome()

#循环调用URL

for i in url\_list:

dr.get(i)

time.sleep(1)

for j in range(1,10):

#获取演员

xpath2='//\*[@id="app"]/div/div/div[1]/dl/dd[%d]/div/div/div[1]/p[2]' %j

actor=dr.find\_element\_by\_xpath(xpath2).text

#获取分数

xpath3='//\*[@id="app"]/div/div/div[1]/dl/dd[%d]/div/div/div[2]/p' %j

score=dr.find\_element\_by\_xpath(xpath3).text

#获取标题

xpath1='//\*[@id="app"]/div/div/div[1]/dl/dd[%d]/div/div/div[1]/p[1]/a' %j

title=dr.find\_element\_by\_xpath(xpath1).get\_attribute('title')

#获取电影介绍

dr.find\_element\_by\_xpath(xpath1).click()

xpath4='//\*[@id="app"]/div/div[1]/div/div[2]/div[1]/div[1]/div[2]/span'

introduce=dr.find\_element\_by\_xpath(xpath4).text

print('电影标题：%s' %title)

print("主要演员：%s" %actor)

print("分数：%s" %score)

print(introduce)

dr.get(i)

#关闭

dr.close()

输出如下所示。

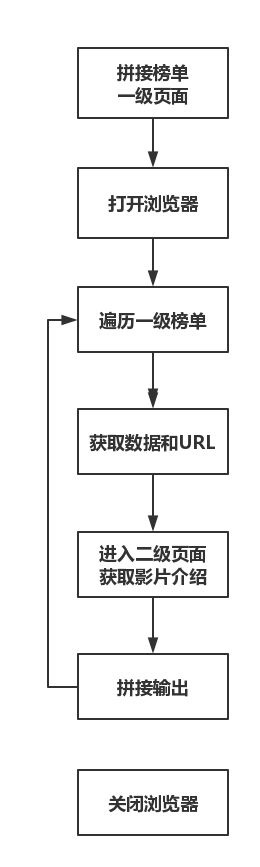
电影标题：罗马假日

主要演员：格利高里·派克,奥黛丽·赫本,埃迪·艾伯特

分数：9.1

欧洲某国的安妮公主（奥黛丽·赫本 饰）到访罗马，国务烦身，但她又厌倦繁文缛节。一天晚上，身心俱疲的她偷偷来到民间欣赏夜景，巧遇报社记者乔（格利高里·派克 饰）。二人把手同游，相当快乐。公主更是到乔的家中作客并在那过夜。不料乔无意中发现了公主的真实身份，他决定炮制一个独家新闻，于是乔和朋友、摄影师欧文（埃迪·艾伯特 饰）一起带公主同游罗马，并且偷拍了公主的很多生活照。然而，在接下来与公主的相处中，乔不知不觉恋上了公主。为了保护公主的形象，乔只能忍痛抛弃功成名就的良机，将照片送予了公主。安妮公主在经历了罗马一日假期后，反而体验了自己对国家的责任，毅然返回了大使馆，为了本身的责任而果断抛弃了爱情。

得到的数据符合我们的预期。我们回顾一下本程序的流程（见图8-4）。

图8-4 流程

## 8.3进一步完善之后的测试代码

既然能够输出了，我们就再对刚才的代码进行一次修改，将其写入CSV文件中。

为什么采用CSV，而不使用数据库呢？首先，我们的数据比较少，没有大型的存储需求。然后，对于小型的项目来说，我们创建数据库，创建表，然后写入、查询和检验，这一套流程耗费的时间和精力可能可能比我们写爬虫本身耗费的时间还长。数据库很强大，但应用在这种场景下就有些大材小用了。

修改之后的源代码如下。

#导入库

from selenium import webdriver

import time

import csv

这里进行URL拼接

numbers\_list=[0,10,20,30,40,50,60,70,80,90]

#写入URL字典中

url\_list=[]

for i in numbers\_list:

url="https://maoyan.com/board/4?offset=%d" %i

url\_list.append(url)

print("获取地址:%s" %url)

#打开浏览器

dr=webdriver.Chrome()

#打开CSV文件

csv\_file=open("writer.csv","w+",newline='',encoding='utf-8-sig')

writer=csv.writer(csv\_file)

#访问一级页面

for i in url\_list:

dr.get(i)

time.sleep(1)

print("正在链接为%s的页面" %i)

#循环嵌套，再次获取每个页面中的链接

for j in range(1,11):

#从一级页面获取信息

#获取演员

xpath2='//\*[@id="app"]/div/div/div[1]/dl/dd[%d]/div/div/div[1]/p[2]' %j

actor=dr.find\_element\_by\_xpath(xpath2).text

#获取分数

xpath3='//\*[@id="app"]/div/div/div[1]/dl/dd[%d]/div/div/div[2]/p' %j

score=dr.find\_element\_by\_xpath(xpath3).text

#获取标题

xpath1='//\*[@id="app"]/div/div/div[1]/dl/dd[%d]/div/div/div[1]/p[1]/a' %j

title=dr.find\_element\_by\_xpath(xpath1).get\_attribute('title')

#访问二级页面

#获取电影介绍

dr.find\_element\_by\_xpath(xpath1).click()

xpath4='//\*[@id="app"]/div/div[1]/div/div[2]/div[1]/div[1]/div[2]/span'

introduce=dr.find\_element\_by\_xpath(xpath4).text

#拼接

text=[]

text.append(title)

text.append(actor)

text.append(score)

text.append(introduce)

#写入CSV文件中

writer.writerow(text)

print("-----正在获取第%s条" %j)

#返回初始页面

dr.get(i)

#最后输出

print("all\_done")

dr.close()

程序的输出结果应该如下。

获取地址:https://maoyan.com/board/4?offset=50

获取地址:https://maoyan.com/board/4?offset=60

获取地址:https://maoyan.com/board/4?offset=70

获取地址:https://maoyan.com/board/4?offset=80

获取地址:https://maoyan.com/board/4?offset=90

正在链接为https://maoyan.com/board/4?offset=0的页面

-----正在获取第1条

-----正在获取第2条

-----正在获取第3条

-----正在获取第4条

-----正在获取第5条

-----正在获取第6条

当程序运行完之后，我们直接以Excel表格的形式打开数据集。最终文件中的数据集如图8-5所示。



图8-5 数据集

说实话，当你写好程序之后，出去小憩一下，等你回来之后，发现所有的数据都出现在那里，真的是一件令人惬意的事情。我们在打开CSV文件的时候使用到了UTF-8-SIG编码。对于大部分读者来说，这么编码可能很新，甚至是第一次看到。

为什么要使用UTF-8-SIG编码？UTF-8-SIG编码即UTF-8 with BOM。我们如果将UTF-8-SIG换为UTF-8，就会发现我们存储的CSV文件虽然可以通过记事本打开，但如果用Excel打开就会乱码，为什么呢?

因为Excel表格表示读取CSV文件的时候是通过读取文件头上的BOM（Byte Order Mark，字节顺序标记）来识别编码的，并且按照Unicode编码来识别没有BOM的文件。注意，BOM是一种文件头部协议，存储在文件头部，用于标识文件编码。

然而，我们生成的CSV文件没有BOM，Excel自动按照Unicode编码读取，文件的编码却不是Unicode，就会出现编码问题。如前所述，作为开发人员，必须彻彻底底地弄清楚编码问题，甚至有时候要推翻自己原先认为对的东西。

# 第9章 基于淘宝商品的实战

本章以淘宝商品为例子，阐述如何应对复杂网站的页面数据采集，展示较为新颖的实现方式，解决读者因难以下手而犹豫不决和无法直接应用的问题。

在本章中，我们利用Selenium抓取淘宝商品，得到淘宝某一关键词对应商品的名称、价格、店铺名称等信息。

## 9.1 准备工作

在本次爬取的过程中，我们会使用到一个叫作pyquery的库。这个库依赖于lxml，而lxml的安装依赖于Visual C++ 14。在pip安装的时候，如果在建立lxml.etree扩展的时候失败了，就会提示我们需要安装Visual C++ 14。

building 'lxml.etree' extension

error: Microsoft Visual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build Tools": http://landinghub.visualstudio.com/visual-cpp-build-tools

所以，直接使用pip很可能会报错，建议在加利福尼亚大学官网搜索[Unofficial Windows Binaries for Python Extension Packages](https://www.lfd.uci.edu/~gohlke/pythonlibs/)，并下载wheel文件，然后通过pip安装wheel文件，来避免这些麻烦。

这里在爬虫淘宝的时候，有一个地方可能需要更改一下：使用Chrome浏览器来进行测试。然而，在爬取过程开启一个Chrome浏览器确实不太方便，因此这里可以使用第4章介绍的PhantomJS无头浏览器。为此，只需要将WebDriver后面的Chrome()改成PhantomsJS()即可。

然后还有Python的内置库urllib，该库是一个可以对URL进行专业化处理的组件集合。如果读者以前写过其他爬虫，那么一定使用过这个库。没有使用过的读者可以了解一下这个库是干什么的，但不必深入了解，我们仅仅会用到其中的一个方法。

## 9.2 流程简述

首先，我们看一下淘宝的链接组成。比如，我们在其中输入type c关键词，淘宝的链接是这样的。

<https://s.taobao.com/search?q=type+c&type=p&tmhkh5=&spm=a21wu.241046-us.a2227oh.d100&from=sea_1_searchbutton&catId=100>

而我们在其中输入usb关键词，淘宝返回的链接如下。

<https://s.taobao.com/search?q=usb&imgfile=&js=1&stats_click=search_radio_all%3A1&initiative_id=staobaoz_20180805&ie=utf8>

所以，我们完全有理由推测，淘宝搜索将我们搜索的内容放入链接中，通过返回的链接不同来确定显示的内容不同。

其实，这个链接的参数能够说明很多内容。比如，参数q代表我们要查询的内容， imgfile参数和图片搜索相关，initiative\_id参数包含访问的时间，ie参数说明使用的编码，诸如此

类等。

淘宝的页面内容很多，虽然大多数时候服务器返回的速度很快，但不排除部分时候下载有些内容特别慢，影响我们爬虫的速度。然而，判断页面是否已经加载完的标准是我们需要的内容是否显示，而不是必要等到所有的内容都加载完。

结合本书前面讲述的显式等待和隐式等待，在淘宝网站中需要显示的内容如图9-1所示。

图9-1 在淘宝网站中需要显示的内容

所以，我们只需要判断这个内容所属的标签完全加载就可以了。当然，这里是所有商品的内容。

然后，我们考虑使用下面的CSS选择器路径来判断所有商品是否加载完。

.m-itemlist .items .item

获取到的商品很多，我们使用这个CSS选择器获取到的是一个列表。

all=dr.find\_elements\_by\_css\_selector(".m-itemlist .items .item")

for i in all:

……..

如果我们使用Selenium获取详细信息，需要对商品价格、商店、商家等信息重复写多次CSS选择器，然后书写多个循环调用。为了避免重复这些的相同或相近的内容，不采用find\_elements\_by\_css\_selector()，而使用一个全新的pyquery库来对页面进行解析，效果和上面的代码的是一样的，但更加方便一些。

接下来，通过以下代码导入库并将其简写为PQ。

from pyquery import PyQuery as PQ

接下来，通过以下代码解析页面源代码。

html=dr.page\_source

pq\_file=PQ(html)

接下来，在页面中使用CSS选择器#mainsrp-itemlist .items .item来查找所有的内容。

all=pq\_file('#mainsrp-itemlist .items .item').items()

接下来，分别选择下级中class属性为.pic .img的标签（示例标签如下），获取这个标签的data-src属性，也就是图片路径。

<img id="J\_Itemlist\_Pic\_566741234264" class="J\_ItemPic img"

src="//gsearch2.alicdn.com/img/bao/uploaded/i4/i4/2616970884/xxx.jpg\_180x180.jpg\_.webp"

data-src="//gsearch2.alicdn.com/img/bao/uploaded/i4/i4/2616970884/xxx.jpg"

alt="2018新款 Apple/苹果 9.7英寸iPad 智能平板电脑 正品国行新品">

类似地，我们将商品的一些信息拼凑成列表。

product = [i.find('.pic .img').attr('data-src'),

i.find('.price').text(),

i.find('.deal-cnt').text(),

i.find('.title').text(),

i.find('.shop').text(),

i.find('.location').text()]

内容部分结束了。我们还需要考虑多页面采集问题，页数可以采用循环来实现。而要对指定页数的页面进行采集，我们需要进行页面的跳转。页码跳转实现起来并不是很复杂，只需要找到相应的搜索框，然后进行输入相应的数字即可。

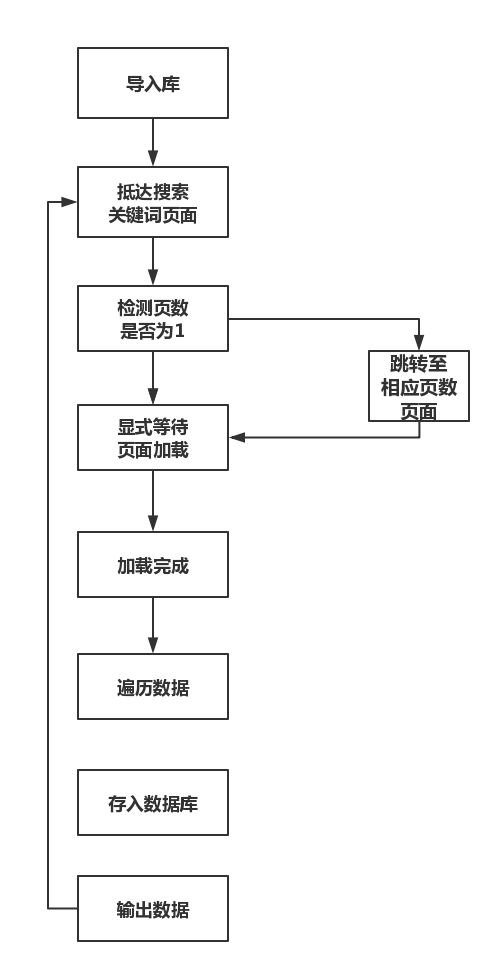
页码跳转的实现流程如图9-2所示。

图9-2 页码跳转的实现流程

## 9.3 代码解读

在理解了思路和实现流程之后，我们就可以尝试抓取淘宝网站的数据。最终我们编写的功能代码如下。

#导入库

from selenium import webdriver

from selenium.common.exceptions import TimeoutException

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.support.wait import WebDriverWait

from urllib.parse import quote

from pyquery import PyQuery as PQ

#提取商品数据

def get\_products():

#获取页面源代码

html=dr.page\_source

#解析页面

pq\_file=PQ(html)

all=pq\_file('#mainsrp-itemlist .items .item').items()

for i in all:

product = [i.find('.pic .img').attr('data-src'),

i.find('.price').text(),

i.find('.deal-cnt').text(),

i.find('.title').text(),

i.find('.shop').text(),

i.find('.location').text()]

print(product)

#打开浏览器

dr=webdriver.Chrome()

wait=WebDriverWait(dr,10)

keyword='iPad'

def scratch\_page(page):

#输出页面

print('正在爬取第',page,'页')

#防止因为某个页面失败而导致爬虫停止

try:

#跳转关键词链接

url='https://s.taobao.com/search?q='+quote(keyword)

dr.get(url)

if page > 1:

#检测输入框

input=wait.until(

EC.presence\_of\_element\_located((By.XPATH, '//\*[@id="mainsrp-pager"]/div/div/div/div[2]/input')))

#检测提交框

submit = wait.until(

EC.element\_to\_be\_clickable((By.XPATH, '//\*[@id="mainsrp-pager"]/div/div/div/div[2]/span[3]')))

#清空输入

input.clear()

#输入数字

input.send\_keys(page)

#单击

submit.click()

#等待页面加载出来

wait.until(

EC.text\_to\_be\_present\_in\_element((By.CSS\_SELECTOR,'#mainsrp-pager li.item.active > span'), str(page)))

wait.until(EC.presence\_of\_element\_located((By.CSS\_SELECTOR, '.m-itemlist .items .item')))

get\_products()

except:

#重试

scratch\_page(page)

def main():

scratch\_page(1)

if \_\_name\_\_ == '\_\_main\_\_':

main()

我们使用quote()函数来将keyword编码成符合URL规范的内容，包含特殊字符（包括汉字）的字符串会被替换成%xx的格式，只有字母、数字、和'\_.-'不会被转换。一般来说，这个函数是用来格式化URL地址的，这个函数包含一个可选参数来指定不格式化的内容。

当编译一个源代码文件的时候，Python编译器会按照顺序执行里面的所有内容，但在执行代码之前，它会先定义一个特定的变量”\_\_name\_\_”。如果代码是作为主程序运行的（读者主动运行这个源文件，而不是当作库导入源代码），会把”\_\_main\_\_” 赋予”\_\_name\_\_”；如果是当作库导入的，那么会把这个源文件的名字赋予”\_\_name\_\_”。

定义一个main()函数，只在主动运行的时候，才运行main()函数。

当运行程序的时候，会返回以下结果。

正在爬取第 1 页

['//g-search1.alicdn.com/img/bao/uploaded/i4/imgextra/i1/13022581/TB2s3yWkbZnBKNjSZFrXXaRLFXa\_!!0-saturn\_solar.jpg', '¥\n2988.00', '7323人付款', '赠电动牙刷【2年保修】Apple/苹果\niPad\n2018款 9.7英寸平板电脑', '绿森数码官方旗舰店', '浙江 杭州']

['//g-search1.alicdn.com/img/bao/uploaded/i4/i4/2616970884/TB1HFkEHHGYBuNjy0FoXXciBFXa\_!!0-item\_pic.jpg', '¥\n2488.00', '12679人付款', '2018新款 Apple/苹果 9.7英寸\niPad\n智能平板电脑 正品国行新品', '苏宁易购官方旗舰店', '江苏 南京']

['//g-search2.alicdn.com/img/bao/uploaded/i4/i2/1669409267/TB1mX4wmAomBKNjSZFqXXXtqVXa\_!!0-item\_pic.jpg', '¥\n2268.00', '12957人付款', '[12期分期][两年质保]Apple/苹果\niPad\n2018款 9.7英寸wifi新款平板电脑32G/128G 正品国行新品授权旗舰店', '卓辰数码旗舰店', '浙江 杭州']

['//g-search3.alicdn.com/img/bao/uploaded/i4/i2/446381428/TB2fxUNEpmWBuNjSspdXXbugXXa\_!!446381428.jpg', '¥\n1858.00', '398人付款', 'Apple/苹果\niPad\nmini4 wifi 7.9英寸平板电脑 2018新款\niPad\n迷你4', '旺隆数码', '广东 深圳']

内容可以重新输出。第一步已经完成了，接下来，我们考虑将数据存储到数据库中。毕竟，这次的数据已有些规模了。另外，我们的目的是对前面所学的内容进行融会贯通，实践的范围越大越好。

这次多了一个pymysql库。首先，连接数据库。注意，读者在这里连接的数据库是tb。在运行这段代码之前，你需要首先使用SQL语句新建一个数据库，然后在数据库中创建一个表（作者把它命名为info）。表中的各个字段限制总是大一些为好，你无法总是料到可能出现的超级长的URL或者商品标题。

另外，我们更换一个关键词，从抓取淘宝上关于iPad的信息转战到type C接头，既为了避免数据的重复，也为了检验代码的可复用性。

这里建立的info表有6个字段，和下面的代码相对应，第一个URL字段的VARCHAR()中的数字要设置得大一些。

['//g-search1.alicdn.com/img/bao/uploaded/i4/imgextra/i3/14414497/xxx.jpg',

'¥\n79.00', '496人付款', 'type-c扩展坞拓展usb-c转hdmi+vga转接头转接口华为mate10/P20小米苹果笔记本电脑macbookpro雷电3转换器', '数码专营店', '上海']

我们将上面的数据定义为变量product，然后使用SQL的insert方法将其插入数据中(在这里，因为代码略微有些长，所以采用空格+“\”+换行符的格式，表示当前这一行代码将延续到下一行，两行代码将拼接并且可以视为一行代码。

insert="INSERT INTO info VALUES('{0}', '{1}', '{2}', '{3}', '{4}', '{5}')" \

.format(product[0],product[1],product[2],product[3],product[4],product[5])

最终代码如下。

#导入库

from selenium import webdriver

from selenium.common.exceptions import TimeoutException

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.support.wait import WebDriverWait

from urllib.parse import quote

from pyquery import PyQuery as PQ

import pymysql

#连接数据库

conn=pymysql.connect(host='localhost',port=3306,db='tb',user='root',passwd='password',charset='utf8')

#设置游标

cur=conn.cursor()

#因为要重复使用，所以定义成函数

def mysql\_save(product):

#提示1

insert="INSERT INTO info VALUES('{0}', '{1}', '{2}', '{3}', '{4}', '{5}')" \

.format(product[0],product[1],product[2],product[3],product[4],product[5])

cur.execute(insert)

conn.commit()

print(product)

#提取商品数据

def get\_products():

#获取页面源代码

html=dr.page\_source

pq\_file=PQ(html)

all=pq\_file('#mainsrp-itemlist .items .item').items()

for i in all:

#拼接成列表形式

product = [i.find('.pic .img').attr('data-src'),

i.find('.price').text(),

i.find('.deal-cnt').text(),

i.find('.title').text(),

i.find('.shop').text(),

i.find('.location').text()]

#存入MySQL

mysql\_save(product)

#打开浏览器

dr=webdriver.Chrome()

wait=WebDriverWait(dr,10)

keyword='type c'

#抓取页面

def scratch\_page(page):

#输出正在爬取的页面

print('正在爬取第',page,'页')

#防止因为某个页面失败而导致爬虫停止

try:

#跳转关键词链接

url='https://s.taobao.com/search?q='+quote(keyword)

dr.get(url)

#检测页码是否大于1，然后跳转

if page > 1:

#检测输入框

input=wait.until(

EC.presence\_of\_element\_located((By.XPATH, '//\*[@id="mainsrp-pager"]/div/div/div/div[2]/input')))

#检测提交框

submit = wait.until(

EC.element\_to\_be\_clickable((By.XPATH, '//\*[@id="mainsrp-pager"]/div/div/div/div[2]/span[3]')))

#清空输入

input.clear()

#输入数字

input.send\_keys(page)

#单击

submit.click()

#等待页面加载出来

wait.until(

EC.text\_to\_be\_present\_in\_element((By.CSS\_SELECTOR,'#mainsrp-pager li.item.active > span'), str(page)))

wait.until(EC.presence\_of\_element\_located((By.CSS\_SELECTOR, '.m-itemlist .items .item')))

get\_products()

except:

#重试

scratch\_page(page)

end\_page=101

def main():

for i in range(1, end\_page):

scratch\_page(i)

if \_\_name\_\_ == '\_\_main\_\_':

main()

在“提示1”的位置，使用{0}.format()这样的方法来进行字符替换。还有另一种进行字符替换的方法——%s,%[str]，前面的%s是格式符，代表字符串，[str]表示需要格式化的内容。

而format()的基本语法是使用{}来代替%这类冗杂的语法（因为字符串、数字等对应的百分号后跟不同的字母，需要单独记忆），在{}中加入的数字表示顺序，依然从0开始。当然，根据个人爱好，按照默认顺序也可以。

"{} {}".format("hello","selenium")

输出的结果和本节第一段代码的运行结果相同。

最终获取的部分数据如图9-3所示。

图9-3 获取的部分数据

## 9.4 可配置项

当读者已经能够熟练掌握MySQL的命令之后，就不再推荐使用命令行了，因为MySQL的命令行仍有部分不足。比如，当表的内容过长时，命令行工具就无法整齐排列，可读性极差。

MySQL有很多GUI（Graphical User Interface，图形用户界面），建议读者使用它们。比如，作者正在使用的MySQL WORKBENCH是MySQL官方推荐的一个图形化工具，可以在官网上下载此工具。

# 第10章 单元测试

顾名思义，单元测试指的是对软件中的最小可测试单元进行检查和验证。本章将会注重讲解有关单元测试的内容。本章解决单元测试是什么、为什么要写单元测试、应该怎么写单元测试这三个难题，帮助读者评估在自己的项目中是否要使用单元测试。

## 10.1 为什么要写单元测试

程序员的智慧是有限的，但系统的复杂度是无限大的。随着系统复杂性的增加，你必须依靠其他工具来帮助你减少问题。

假设你在一家IT公司工作，然后使用了3万行左右的代码实现了超级复杂的应用软件。这个软件有300个函数，70个互相嵌套的调用，并且代码风格很好，你对自己书写的代码有着清晰的认识。

但当过了一两年的时间，这个应用程序突然出现了某个Bug或者客户要求你为其增加一个新的功能，你无法保证自己一定知道这个Bug出现在哪个部分，或者新增的功能导致了哪一部分出现了意料不到的Bug。

单元测试的作用更多地体现在代码的维护上，而不是在代码的编写上。

有人可能会想，我们打开程序界面，一个一个按钮单击，不就知道功能是否实现了吗？何必要浪费时间再写一个单元测试呢？

实际上，这么想确实没有错，并且在早期确实有一部分公司是这么做的。但随着项目越来越庞大，我们可能对照检查表逐个排查，加上程序运行过程中的切换动画，等检查完一遍，很可能一个上午就过去了。

单元测试的理念是将程序模块化，一个模块一个模块地进行检查，并且使用计算机来执行测试，而不是手动测试或调试，这样自动化的测试不但节约了时间和成本，而且降低了错误率。

## 10.2 怎样写单元测试

前面提到，单元测试是对软件中的最小可测试单元进行检查和验证，这体现的是分而治之的思想。我们在写单元测试的时候，一般会针对每一个方面的功能单独写一个测试，把这些测试封装到一个继承自unittest.TestCase的类中。

假如我们要对一个图书馆书库管理系统进行测试，那么要求存入的书名不能为空。另外，我们还需要测试用户的借书卡内的余额必须为一个浮点型数据且大于零。

单元测试代码一般是和生产代码分开的，保存在独立的目录中，使用的时候导入待测试的代码。

继承自unittest.TestCase的类的函数在书写上有以下规范。

* 使用setup()函数开始单元测试，使用tearDown()函数结束单元测试
* 所有以test\_开头的函数都会当作单元测试执行，其余的函数都会忽略（除了setup()和tearDown()以外）
* 编写不同类型的断言语句来让程序测试成功或者失败。

一般来说，编写的单元测试要求能够覆盖常用的输入组合、异常、边界条件。另外，单元测试的作用只是排除一些容易发生的浅层次的Bug，即便生产代码通过了单元测试代码，并不意味着没有Bug了。

在单元测试中，常用的断言有以下几种。

* assertEqual(self,[assert 1],[assert 2])
* assertTrue([bool])
* assertRaises([exception])

## 10.3 单元测试的演示

unittest是Python的标准库，所有未经修改过的Python版本都可以直接使用，不需要进行安装。

下面的代码没有对具体的内容进行测试，仅仅用于演示。

#导入库

import unittest

#继承类，编写单元测试

class calc(unittest.TestCase):

#初始化函数

def setUp(self):

print("unittest start")

#结束函数

def tearDown(self):

print("part of unittest end")

#具体测试部分

def test\_add(self):

result=1+1

self.assertEqual(2,result)

def test\_minus(self):

result=2-1

self.assertEqual(1,result)

#定义主函数

def main():

unittest.main()

if \_\_name\_\_ == '\_\_main\_\_':

main()

输出结果如下。

part of unittest start.

part of unittest end

part of unittest start

part of unittest end.

----------------------------------------------------------------------

Ran 2 tests in 0.000s

OK

我们首先从unittest.TestCase继承类，然后定义初始化和结束的函数（没有实际内容，仅仅是一个print()函数）。接下来，写两个测试部分的样例，self.assertEqual()用于判断参数中的两个值是否相等。

unittest.main()用于开始执行单元测试，setUp()函数和tearDown()函数在每个测试的开始和结束的时候都会运行一次。如果要仅仅在类开始或结束的时候运行一次，可以考虑使用setUpClass()和tearDownClass()，setUpClass()会在类中所有test开始之前执行一次，而tearDownClass()会在类中在所有test结束之后执行一次。

另外，请注意，unittest.TestCase中的函数和断言的名称与平常的命名规范不太相同，在这个测试实例中一般格式为第一个单词全部小写，第二个单词首字母大写。

使用\_\_name\_\_ == '\_\_main\_\_'是为了使程序只在自我运行的时候才会直接开始测试。如果通过其他文件导入，则导入的只有一个单元测试类。

下面定义一个断言错误的代码。

def test\_wrong\_minus(self):

result=2-1

self.assertEqual(2,result)

断言错误会格式化地告诉我们进行了多少个测试，多少个测试出现错误，并且很清楚地说明错误原因，方便我们进行后期的Bug排查。

======================================================================

FAIL: test\_wrong\_minus (\_\_main\_\_.calc)

----------------------------------------------------------------------

Traceback (most recent call last):

File "C:\Users\xxxxxx\OneDrive\Documents\xxxxxx\assert.py", line 16, in test\_wrong\_minus

self.assertEqual(2,result)

AssertionError: 2 != 1

----------------------------------------------------------------------

Ran 3 tests in 0.000s

FAILED (failures=1)

在我们进行断言的时候，一种重要的方式就是要求执行的程序抛出指定类型的error。下面这个例子中，因为字典中没有test3，所以会抛出一个KeyError。

d={"test1":1,"test2":2}

with self.assertRaises(KeyError):

value=d[”test3”]

## 10.4 单元测试样例

现在假设我们在为一个图书馆的书籍管理系统进行测试，检验其安全性。

上一次开发者实现的其中一个功能如下，它传入一个字典作为参数（实际上是从字典中继承的）。

class book(dict):

def \_\_init\_\_(self, \*\*kw):

super().\_\_init\_\_(\*\*kw)

def \_\_getattr\_\_(self, key):

try:

return self[key]

except KeyError:

raise AttributeError(r"Dict object has no attribute '%s' " % key)

def \_\_setattr\_\_(self, key, value):

self[key] = value

#按照页数分类

def get\_page(self):

if self.page <= 60:

return "短篇"

elif self.page <= 150:

return "中篇"

elif self.page > 150:

return "长篇"

构造函数中的\*\*kw也叫作关键字参数，而关键字参数允许我们传入小到零个、大到无穷个含名称的参数。也就是说，使用\*\*kw传入的参数会被当作一个字典来处理，参数名会被当作关键字，而参数值会被当作关键字的值。

可能读者对于类中的第二个函数感到困惑。

def \_getattr\_(self, key):

\_getattr\_\_其实是Python中的一个内置函数，可以用来返回未知属性。当调用不存在的属性时，Python尝试调用\_\_getattr\_\_(self，'key')来获取属性并返回键。因为传入了一个字典，所以self[key]其实就是像字典中取值。

\_\_setattr\_\_(self, key, value)也是一个内置函数。当尝试给类的实例属性赋值时，就会调用这个方法。下面给出一段示例代码。

>>> book.note="a brilliant book"

>>> book.note

'a brilliant book'

使用这种方法向字典中添加的元素和值（除了字典中最初的元素和值之外）都会被添加到一个叫作\_\_dict\_\_的属性中。另外，这个属性原来并不为空，从末尾位置开始添加。

>>> print(book.\_\_dict\_\_)

{'\_\_module\_\_': '\_\_main\_\_', '\_\_init\_\_': <function book.\_\_init\_\_ at 0x03415660>, '\_\_getattr\_\_': <function book.\_\_getattr\_\_ at 0x034156A8>, '\_\_setattr\_\_': <function book.\_\_setattr\_\_ at 0x034156F0>, 'get\_page': <function book.get\_page at 0x03415738>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'book' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'book' objects>, '\_\_doc\_\_': None, 'note': 'a brilliant book'}

我们用单元测试来检查其稳定性以及它是否可以正常使用。

isinstance([object],[class])也是一个内置函数。如果该函数的前一个参数（一个实例）是后一个参数（一个类）的实例，返回的bool变量值为True；否则，返回的bool变量值为False，因为这里的instance是book的一个实例，而book类继承自dict，所以instance也可以算dict的实例。

from library\_book import book

import unittest

class book\_test(unittest.TestCase):

global instance

instance=book(name="Harry Potter",page=350)

def test\_init(self):

self.assertEqual(instance.name,"Harry Potter")

self.assertEqual(instance.page,350)

self.assertTrue(isinstance(instance, dict))

def test\_key\_add(self):

instance.key1='value1'

self.assertTrue('key1' in instance)

self.assertEqual(instance.key1, 'value1')

def test\_attr\_add(self):

instance["key2"]="value2"

self.assertTrue('key2' in instance)

self.assertEqual(instance.key2, 'value2')

def test\_key\_error(self):

with self.assertRaises(KeyError):

instance['empty']

def test\_attr\_error(self):

with self.assertRaises(AttributeError):

instance.empty

def main():

unittest.main()

if \_\_name\_\_ == '\_\_main\_\_':

main()

首先我们从刚才写的文件中导入相应的代码。因为这里将刚才实现的book存放到library\_book.py中，并且将library\_book.py和刚才新建的单元测试代码存放在同一个目录下，所以可以直接导入（Python的包其实就是Python的代码文件，不过导入采用的是另一种实现形式）。

from library\_book import book

这里在所有函数的外层定义了一个instance变量，用global声明它是全局变量，在这个类中的所有函数都可以使用instance变量。需要注意的是，实例只需要加载一次，而不是在每个测试的开头都要重复加载。

我们一共测试了5个部分，这也是最常见的单元测试需要检查的部分。

首先，测试的是字典是否正确解析。我们将采用assertEqual断言的方式检查实例化对象的值是否是我们想要的值。

然后，我们检验了两种向字典中添加关键字和值的方式是否能够正常使用。

最后，我们检验了是否可以正常抛出错误，这种行为经常检查是否可以人为（恶意用户）地生成Bug。

那么这个unittest.main()是什么呢？我们来看一下下面的代码，实际上这个unittest.main()的执行过程和它是几乎相同的。首先找出所有以test\_开头的函数，然后将其添加到TestSuite中，最后后使用run()方法执行TestSuite中的所有内容。

test\_runner=unittest.TextTestRunner()

test\_suit=unittest.TestSuite()

test\_suit.addTests(map(book\_test,["test\_init","test\_key\_add","test\_attr\_add","test\_key\_error","test\_attr\_error"]))

test\_runner.run(test\_suit)

map函数以映射的方式对对列表中的每个元素调用book\_test方法，其中iterable参数可以有多个，但function参数只能传入一个。

map(function, iterable, ...)

如果读者的代码量比较小，很有可能你写的单元测试代码比生产代码本身更加冗杂，这也是很多人并没有感觉到单元测试的重要性并且厌恶写单元测试的原因。

如果读者对自己写的代码有清晰的了解，并且确定自己在进行代码维护（或者不需要维护）的时候可以在短时间内回想起代码的作用，那么其实单元测试并不是必需的。

实际上，当研发一个技术难度比较高的网络项目时，前端和后端的生产代码一般分离化处理，以降低修复了1个Bug却出现3个Bug这种情况的概率。

## 10.5 Selenium的单元测试

在网页测试的领域，传统的爬虫技术（urllib、BS4和requests）传统一般难以处理Ajax和JavaScript，但实际上，绝大部分前沿的网站都会采用这些技术框架来提高安全性或改善用户体验等诸多目的。

幸运的是，我们有Selenium。其实，Selenium项目本身是用来进行网络测试的，只是后来在网络爬虫中的应用反而越来越多。

Selenium项目的测试方法和内置库的unittest语法还有些不太一样。Selenium项目不需要单元测试是类的一个函数，并且生产代码和测试代码是可以混在一起。Selenium单元测试更多地和Python本身的断言结合起来。这种断言不需要括号，但要求采用空格分开。如果测试通过，不会有任何提示，而当测试失败的时候会返回提示信息。

assert "test" in driver.title

它的单元测试更像是闯关。如果不满足某个条件，就不会再向下执行，而不是像unittest会在所有的测试代码执行完毕之后返回错误的数量和详细信息。Selenium的单元测试也并没有提供一个完整的测试框架，仅仅是对网页上元素内容的验证。不过，这对我们来说已经足够了。请看下面一段代码。

from selenium import webdriver

dr=webdriver.Chrome()

dr.get("https://en.wikipedia.org/wiki/Python")

assert "Python" in dr.title

dr.close()

在运行这段代码的时候，如果没有意外，读者应该可以打开维基百科中一个关于Python的界面，紧接着会关闭这个界面。

如果我们将“Python”换成其他字符，并且在断言检查的时候字符不存在，则会直接能返回错误，告诉我们在检查这一部分的时候失败了。

Traceback (most recent call last):

File "C:\Users\xuyichenmo\OneDrive\Documents\SELENIUM\第十章\selenium\_unittest.py", line 6, in <module>

assert "python" in dr.title

AssertionError

实际上，我们在使用Selenium对页面进行操作的时候，比如，获取某个元素的文本，单击某个按钮，就是在对页面执行单元测试。页面只需要能够执行一个正常用户所有可能的操作，然后在中间使用assert断言进行阻隔，确保中间某一步不会出错。比如，如果获取到的文本内容为空，那么有以下断言。

assert string is not None

然后对中间出现的错误进行统一的处理，使用try…except方法来避免因为某一步的出错而造成全部测试代码的失败。

在如下代码中，我们尝试从字典中取出关键字test对应的值，但字典中没有这个关键字，当我们执行代码的时候，虽然出现了错误，却可以正常执行。

dic= {'Michael': 95, 'Bob': 75, 'Tracy': 85}

try:

d=dic['test']

except KeyError as e:

print(e)

## 10.6 美化报告

HTMLTestRunner是一个基于unittest的第三方库，由Wai Yip Tung开发。

为什么要使用HTMLTestRunner呢？unittest在命令行中输出的测试结果也许开发者可以看懂，但这样的报告无法直接供相关人员使用，因此，我们需要一种界面更加友好的测试报告来告诉我们测试的详细结果和技术部分。

几乎绝大部分的Python第三方库都可以直接使用pip install [package name]来安装，但HTMLTestRunner略微有些不同。

虽然可以使用pip来安装它，但安装的命令如下。

pip install html-testRunner

安装的包是HtmlTestRunner，我们需要的是HTMLTestRunner，虽然看似只有大小写的区别，但是我们无法使用这个库。

因此读者需要到Wai Yip Tung的个人网站上下载HTMLTestRunner（在撰写本书的时候，最新版本为0.8.2，并且是针对Python 2开发的）。

在下载完成文件后，将其保存到自己Python安装目录下的\Lib\site-packages目录下。

如果使用的是的Python 2.*x*版本，应该保存完文件后立刻就可以导入库了；如果使用的是Python 3.*x*版本，则还需要做一定的修改。

No module named 'StringIO'

这些修改主要是由于StringIO库和io库（Python 3中使用io.StringIO库）的不同，以及Python 2和Python 3的编码问题造成的。

下面列出具体的修改。

* 在第94行中，把import StringIO 修改为 import io。
* 在第539行中，把self.outputBuffer = StringIO.StringIO()修改为self.outputBuffer = io.StringIO()。
* 在第642行中，把if not rmap.has\_key(cls)。修改为if not cls in rmap。
* 在第766行中，把uo = o.decode('latin-1') 修改为uo = e。
* 在第775行中，把ue = e.decode('latin-1') 修改为ue = e。
* 在第631行中，把print >> sys.stderr, '\nTime Elapsed: %s' % (self.stopTime-self.startTime) 修改为print(sys.stderr, '\nTime Elapsed: %s' % (self.stopTime-self.startTime))。

在修改过后，如果没有意外，应该可以正常导入，而不出错。

import HTMLTestRunner

接下来，编写代码。

#导入库

import HTMLTestRunner

import unittest

#测试文件所在的独立目录

directory=r"[dir]"

#遍历测试文件

def createsuite():

#创建unittest实例

test\_suit =unittest.TestSuite()

#从文件夹目录中查找需要测试的实例 discover=unittest.defaultTestLoader.discover(directory,pattern='\*\_unittest.py',top\_level\_dir=None)

#计数器

counter=0

for i in discover:

counter+=1

print("已发现{}个测试实例需要执行".format(counter))

for j in i:

#向其中添加测试实例中添加测试单元

test\_suit.addTests(j)

#返回

return test\_suit

#定义执行函数

def main():

#输出HTML文件的位置

filename="E:\\result.html"

#以二进制方式打开文件

f=open(filename,'wb')

#使用HTMLTestRunner运行

runner=HTMLTestRunner.HTMLTestRunner(stream=f,title='测试报告',description='实例执行情况：')

#执行

runner.run(createsuite())

#关闭文件流

f.close()

#只在主动运行的时候才执行

if \_\_name\_\_ == '\_\_main\_\_':

main()

首先，导入unittest和HTMLTestRunner库。然后，定义一个目录（目录文件是我们需要测试的所有文件）。

使用unittest.defaultTestLoader.discover()方法对目录中所有的可执行测试进行探测。第一个参数是需要探测的目录。第二个参数是文件名筛选模式（比如，我们使用\*\_unittest.py表示所有名称以\_unittest结尾的Python文件）。第三个参数设置测试模块的顶层目录，在这里我们没有把顶层目录设置为None值，默认值为None。

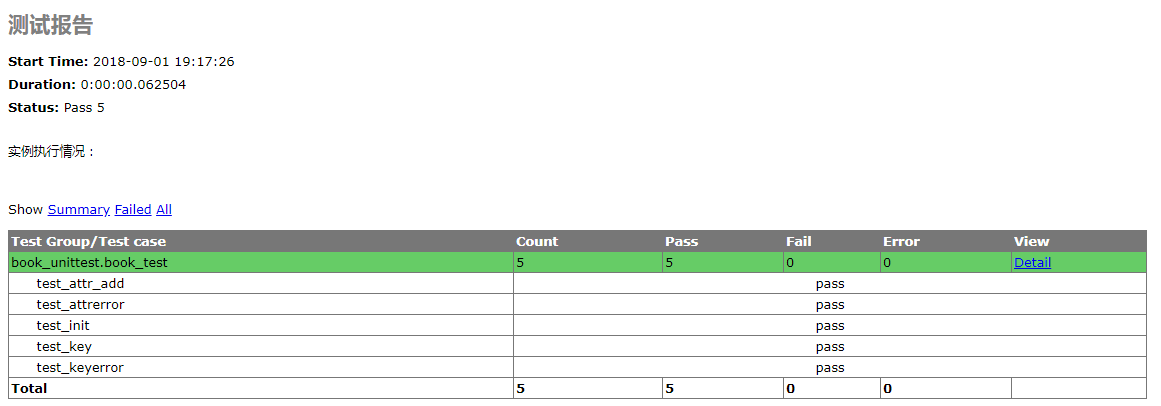
接着，我们使用循环对返回的列表中的每一个元素进行调用，使用test\_suit.addTests()方法将其添加到待测试的单元测试列表中。

最后，main()函数定义了执行待测试的单元测试列表，并且使用以下代码定义了只有在主动运行的时候才执行main()函数。

if \_\_name\_\_ == '\_\_main\_\_':

main()

如果运行无误，应当会在我们定义的目录下产生一个HTML文件。这个HTML文件可以直接使用浏览打开，网页中的unittest报告如图10-1所示。

图10-1 unittest报告

# 第11章 多线程

在最早的程序执行流程中，所有的代码文件必须按照顺序依次执行。这种方式效率极低，必须前面的执行完，后面的才会执行，而且不能充分发挥CPU多核的优点。多核多线程的实际能力才代表计算机整机配置的实际标注，这也就是我们通常说的程序优化。否则，即便计算机的实际配置很高，实际运行体验也很糟。

多线程并行执行有以下几个优点。

* 使用多线程可以明显提高程序的响应速度。
* 充分利用CPU和内存全部能力。
* 将程序分块处理，方便程序的开发和维护。
* 可以设置程序的优先级来优化响应速度。
* 多线程花费小，切换快。
* 不同的测试实例之间数据通信方便。

## 11.1 什么是进程线程

每个程序执行的时候，都会创建一个新的进程。这个进程包括程序本身和程序使用到的所有内存与系统资源，而每一个进程都会拥有一个或以上的线程。对于进程来说，线程是共享进程中的内容的。

读者可以将其想象成绳子，每一根绳子都基本上由多条小绳子交错缠绕而成。在每条小绳子的粗细和长度不变化的情况下，小绳子越多，这根大绳子就越粗壮，实际的承受能力就越强。

多线程操作的前提就是程序允许我们创建多个并行的线程，并且这些线程各自具有一定的独立性，允许它们完成自己的任务。这样来实现功能，可以提高CPU和内存的利用率。在一个线程不得不陷入等待的时候，CPU可以转而运行其他的线程，比如，前端设计师在进行网页的设计的时候，可以将耗时的操作放在子线程中，而将UI（User Interface，用户界面）放在主线程中，来提高用户的使用体验。

## 11.2 单线程，多线程对比

单线程和多线程有各自的优缺点，这里挑选几个来进行一个简单的对比，方便读者确定自己需要在哪些环境中使用多线程，而再哪些环境中则完全没有必要。

* 在程序响应速度上，多线程具有一定的优势。多线程可以同时处理并行任务，多线程具有单线程无可比拟的优点。即便采用多进程的单线程任务和单进程的多线程任务进行对比，线程消耗的资源也少于进程。
* 在内存占用上，单线程具有优势。如果仅仅对比单进程下的单线程和多线程，那么单线程任务在内存占用多面具有一定的优势，多线程反而消耗的内存资源更少。这是因为线程也需要消耗资源，线程越多，程序越容易卡顿。
* 在程序出错的可能性上，单线程占据优势。多线程任务由于线程过多会导致控制过程较为复杂，最终反而会导致许多原本不会发生的Bug。比如，线程之间对共享资源的访问会存在相互影响的情况，这是不得不面对的，我们必须花费更多的精力来处理这些问题。
* 在运行效率上，多线程具有优势。这很像我们在小学做过的数学题，假设烤面包需要10min，刷牙需要5 min，洗脸需要5 min，请指出如何统筹安排花费的总时间最短。单线程处理的情况就相当于我们傻傻地等待面包烤完之后再进行刷牙洗脸，而多线程可以在烤面包的同时刷牙洗脸，来提高程序的运行效率，但这两种能够处理方式的结果是一致的。

## 11.3 实际体验

首先，我们书写一个单进程单线程的输出版本。

from time import ctime

import time

def test1():

for i in range(3):

print("this is test1 {}".format(ctime()))

time.sleep(1)

def test2():

for i in range(3):

print("this is test2 {}".format(ctime()))

time.sleep(1)

if \_\_name\_\_ == '\_\_main\_\_':

test1()

test2()

print("done {}".format(ctime()))

输出是这样的。

this is test1 Sun Aug 19 23:46:39 2018

this is test1 Sun Aug 19 23:46:40 2018

this is test1 Sun Aug 19 23:46:41 2018

this is test2 Sun Aug 19 23:46:42 2018

this is test2 Sun Aug 19 23:46:43 2018

this is test2 Sun Aug 19 23:46:44 2018

done Sun Aug 19 23:46:45 2018

这说明程序依次执行了test1()和test2()函数，我们先来计算一下程序输出的时间，以便和下面的多线程版本进行对比，从8月19日23:46:39 到8月19日23:46:45，一共花费了6s的时间。

然后，我们再写第二个版本的多线程程序。

from time import ctime

import time

import threading

def test1():

for i in range(3):

print("this is test1 {}".format(ctime()))

time.sleep(1)

def test2():

for i in range(3):

print("this is test2 {}".format(ctime()))

time.sleep(1)

threads = []

t1 = threading.Thread(target=test1)

threads.append(t1)

t2 = threading.Thread(target=test2)

threads.append(t2)

if \_\_name\_\_ == '\_\_main\_\_':

for t in threads:

t.setDaemon(True)

t.start()

print("done {}".format(ctime()))

第二次的输出如下。

this is test1 Sun Aug 19 23:49:11 2018

done Sun Aug 19 23:49:11 2018

this is test2 Sun Aug 19 23:49:11 2018

虽然不到一秒钟就运行完了程序，但为什么输出只有3行？输出变得这么少，应该是出了什么问题。

然而，当我们向其中加入一行代码的时候，输出就会恢复正常了。

if \_\_name\_\_ == '\_\_main\_\_':

for t in threads:

t.setDaemon(True)

t.start()

t.join()

print("done {}".format(ctime()))

这时候输出就在交错进行。

this is test1 Sun Aug 19 23:51:41 2018

this is test2 Sun Aug 19 23:51:41 2018

this is test1 Sun Aug 19 23:51:42 2018

this is test2 Sun Aug 19 23:51:42 2018

this is test1 Sun Aug 19 23:51:43 2018

this is test2 Sun Aug 19 23:51:43 2018

done Sun Aug 19 23:51:44 2018

想必读者已经猜到了，是的，t.join()这一行代码用于锁定主线程，直到所有子线程执行完毕。而如果去掉这一行，那么很有可能主线程已经执行完，而子线程还未执行完，程序就已经退出了。

我们做一个简单的四则运算，从8月19日23:51:41到8月19日23:51:44，程序一共运行了3s，比上面的6s在效率上提高了50%。

在运行的时候，输出的数据可能是有些混乱的，有可能本应该在两行的输出，却挤到一行中。这时因为我们使用的time.sleep()的运行时间并不是严格的1s，它存在毫秒级别的误差。当程序运行的时候，很有可能上一个线程的休眠的时间长一些，而下一个线程休眠的时间略微短一些，这就会导致输出时两者的内容杂乱在一起写入缓冲区中，所以输出的数据格式就会凌乱。

实际上，当我们实例化一个thread对象以后，有这几个方法可用。

* start()，用来开始线程的活动。

这个方法对于每个线程最多只能调用一次，它在一个单独的控制线程中调用对象的run()方法。如果在同一个对象中重复多次调用，那么这个方法会抛出一个RuntimeError错误。

* run()，用来表示线程的活动。

可以在子类中重写此方法。标准run()方法以传递给对象构造函数的可调用对象作为目标参数(如果有)，分别使用args和kwargs参数中的顺序参数与关键字参数。

* join([timeout=None])，等待直到所有的子线程结束或者出现超时错误。

如果读者指定timeout参数的值，读者传入的参数应该是一个浮点数，以秒为单位来指定超时时长。

另外，因为join()返回的值总是None，读者必须调用alive()函数在join()函数之后来判断是否超时。如果子线程仍然存活，会抛出一个超时错误。

* name

仅用于识别目的的字符串。它没有语义。多个线程可能被赋予相同的名称。初始名称由构造函数设置。

* getName()，用来获取name的值。
* setName()，用来设置name的值。
* ident

这是线程的“线程标识符”，如果线程尚未启动，则为空。这是非零整数。看到get\_ident()函数。当线程退出并创建另一个线程时，可以回收线程标识符。即使线程已经退出，标识符也是可用的。

* is\_alive()

用于判断线程是否存在，这个方法在run()方法开始之前返回True，直到run()方法终止之后才返回False。

* daemon

一个布尔值，表示线程是否是守护线程。这必须在调用start()之前设置，否则会引发运行时错误。它的初始值继承自创建的线程，主线程不是守护线程，因此在主线程中创建的所有线程默认不是守护线程。

下面解析一下守护线程的概念。daemon的值默认为False，这就意味着主线程结束时会检测该子线程是否结束。如果该子线程还在运行，则主线程会等待它完成后再退出，但如果这个daemon值设置为true，在主线程退出的时候这些子线程也会同时退出。当没有存活的非守护进程线程时，整个Python程序退出。

另外有一点需要说明一下，一个线程可以多次使用join()方法

如果尝试使用join()函数连接当前线程，join()将引发一个运行时错误，因为这会导致死锁（两个线程互相等待，相当于围棋中的循环劫）。在线程启动之前连接线程也是一个错误，并且试图这样做会引发相同的异常。

## 11.4 threading库中常用的函数

threading库提供了非常多的函数，但对大部分人来说，完全弄清楚这些函数是不现实的。表11-1汇总了threading库中常用的函数。掌握了这些函数，相信读者便可以处理绝大部分问题了。

表11-1 threading库中的常用函数

|  |  |
| --- | --- |
| threading.active\_count() | 返回Thread当前活动的对象数，返回的计数等于enumerate()返回的列表的长度 |
| threading.current\_thread() | 返回当前Thread对象，对应于调用者的控制线程。如果未通过模块创建调用者的控制 线程，则返回具有有限功能的虚拟线程对象 |
| threading.get\_ident() | 返回当前线程的“线程标识符”” |
| threading.enumerate() | 以列表形式返回当前活动的所有对象 |
| threading.main\_thread() | 返回主线程，一般情况下，这个主线程都是启动Python解释器的线程 |
| threading.TIMEOUT\_MAX | 返回最长超时时间 |
| threading.stack\_size([number]) | 返回创建一个线程所需要的最小栈的大小，如果指定了一个参数，那么会将这个参数的值设置为最小栈的大小 |
| threading.setprofile([func]) | 对从threading库启动的所有线程配置文件 |
| threading.settrace([func]) | 对从threading库启动的所有线程配置一个跟踪函数 |

## 11.5 锁的概念

在开始之前，首先要理解原语的概念。内核或函数称为原语，指的是一段用机器指令编写的完成特定功能的程序，程序在执行的过程中是不可以中断的，比如，我们在前面学习的SMTP，当我们发送一封邮件的时候，如果程序不可以接收另一封邮件，它必须等到我们发送邮件这个任务结束之后才能开始接收邮件。

同步则是指两个截然不同但相关的概念（进程同步和数据同步）中的一个。进程同步是指多个进程要轮流操作完成某个特定的指令，而数据同步则是指保持数据集的多个副本的一致性。进程同步原语通常用于实现数据同步

“加锁对象”是最低级别的同步原语，由线程的扩展模块threading实现。一个加锁对象由两个状态——加锁和释放。

这两个状态分别对应Lock.acquire()和Lock.release()。

其中，acquire()函数在开始之前会首先检测对象的状态。如果对象尚未加锁，那么会直接加上一把锁；如果对象已经加锁了，那么会等到这个对象已经加过的锁释放之后，才会加上属于自己的锁。release()函数只能用于用于加锁对象，将对象的状态由加锁转为释放，如果读者尝试释放一个释放状态的对象，程序会返回一个RuntimeError错误。

怎么理解呢？就好像很多人在比赛百米赛跑，每个线程都是参赛选手，终点处只有一瓶水，当口令一吹响之后，所有的线程都会朝着那瓶水前进，但只有一个线程能够得到那瓶水，如果这个线程自己不需要了，并且水含有剩余，那么它可以把这瓶水送给其他进程。

acquire()函数还有一个[blocking=1]的默认参数。当blocking的值为1的时候，以阻塞方式获得锁，必须要等到锁释放才能加锁；当blocking的值为0的时候，如果想要加锁一个加锁对象，那么会直接返回False。

我们分别对于加锁和未加锁状态书写一段相同的代码。

import threading

import time

class thread\_test (threading.Thread):

def \_\_init\_\_(self,threadname):

threading.Thread.\_\_init\_\_(self,name = threadname)

def run(self):

global number

number = number + 1

time.sleep(1)

print("the value of number is {}".format(number))

#调用lock的release方法

#实例化类

threadlist = []

for i in range(5):

t = thread\_test (str(i))

threadlist.append(t)

number=0

for i in threadlist:

i.start()

我们首先导入threading模块和time模块，然后通过继承threading.Thread来创建类。初始化方法中同样调用threading.Thread的初始化方法，接着重载run()方法，number的计数自增，然后停止1s，停止时间是为了让程序获取到的值相同，然后输出。

在函数之外，我们创建了一个threadlist的列表，通过循环实例化thread\_test()类，以多线程方式执行。

the value of number is 5

the value of number is 5

the value of number is 5

the value of number is 5

the value of number is 5

接下来，再来看当我们使用锁时的运行情况。

import threading

import time

class thread\_test(threading.Thread):

def \_\_init\_\_(self,threadname):

threading.Thread.\_\_init\_\_(self,name = threadname)

def run(self):

global number

lock.acquire()

number = number + 1

time.sleep(1)

print("the value of number is {}".format(number))

lock.release()

lock = threading.Lock()

threadlist = []

for i in range(10):

t = thread\_test(str(i))

threadlist.append(t)

number=0

for i in threadlist:

i.start()

lock.acquire()和lock.release()之间的是我们需要加锁的部分。

这次程序的运行时长会稍微长一些，因为我们将time.sleep()放入到了加锁的状态中。也就是说，这一段代码是不能够并行的。另外。我们设置的全局变量的结果改变了。

the value of number is 1

the value of number is 2

the value of number is 3

the value of number is 4

the value of number is 5

## 11.6 主程序是线程还是进程

我们通过实例化threading来实现多线程运行，因此可以认为这个threading的作用类似于我们的程序入口点，它究竟是线程还是进程呢？实际上，在现有的操作系统中，对外来说，这个程序就是一个独立的进程。而对内来说，可以将其理解为线程，就是一个进程中包含这一个线程，而这个线程对于它的子线程来说又是一个进程。

因此，无论读者怎样理解，都是对的，完全没有必要再这个问题上纠结。

## 11.7 更多解决方案

实际上，Python对于多线程编程提供了多种解决方法，包含thread库、threading库、queue库和multiprocessing库。

thread库在Python3.*x*中更名为\_thread。\_thread和threading都涉及线程创建与管理，\_thread涉及的更多内容是底层方面的，很多内容都需要读者自己手动完成，并且这些内容大多不全面（比如，没有守护线程的概念）。在同等需求的情况下，推荐使用threading库。简单来说，当提出了更高的要求的时候，读者可以尝试使用\_thread库来编写一个属于自己的多线程库。

queue库用于创建一个队列数据结构，这些数据可以在多线程之间共享。

multiprocessing库的效果比threading库更好，使用multiprocessing可以实现真正的多核和多核派生进程，并且这个库也提供了一些在共享任务的进程间传输数据的解决方案，实现的是对进程安全的队列。

## 11.8 基于多线程和Selneium的示例

实际上，在维护一个大型爬虫的时候，基本实现思路一般是获取URL，并且采用布隆过滤器判断是否已经爬取过。如果已经爬取过，则直接跳过；如果没有爬取过，则对该URL进行爬取并且记录存储。

假设URL\_spider和selenium\_spider是我们已经实现的两部分功能代码，其中一个采用布隆过滤器实现URL筛查并获取用户ID，另一个用于获取页面中需要的信息。

通常在实际使用中，对于大型的爬虫，我们都会采用将代码写在不同的Python文件当中。这样不但起到了整理项目代码架构的作用，而且方便我们调用相应函数。

一份多线程参考代码如下。

from URL\_spider import url\_spider

from selenium\_spider import sele\_Spider

from threading import Thread

class multi\_spider:

#定义变量、列表

user\_list=None

thread\_list=[]

#初始化变量

def \_\_init\_\_(self, userList):

self.userList=userList

#对页面爬取

def sele\_Spider(self,nick):

url=url\_spider.construct\_url()

ID=url\_spider.search\_id(url)

sele\_spider=sele\_Spider(ID)

sele\_spider.search(nick)

#开启多个线程

def muti\_hreads(self):

#对于列表中的每一个用户，都放入线程队列

for nick\_name in self.userList:

t=Thread(target=self.sele\_Spider,args=(nick\_name,))

self.thread\_list (t)

for threads in self.thread\_list:

threads.start()

在上述代码中，我们用一个类实现多线程，类中包含的每一个函数都承担着多线程的一部分功能。

其中需要注意的是，下面的循环用于将需要爬取的内容加入多线程队列中。

for nick\_name in self.userList:

t=Thread(target=self.sele\_Spider,args=(nick\_name,))

self.thread\_list (t)

我们使用下面的循环开始爬取多线程队列中的每一个内容。

for threads in self.threadList:

threads.start()

## 11.9 GIL

Python虚拟机（也称为解释器主循环）和CPython版本有关，控制着Python代码的执行。由于GIL（Global Interceptor Lock，全局解释器锁）的存在，它控制线程对Python虚拟机的访问，导致同一时间只运行一个线程。这并不是说GIL是不好的。事实上，Python的设计理念就要求解释器的主循环中只有一个线程单独执行。也就是说,在任何给定的时刻,只有一个线程执行。

通过使对象模型（包括重要的内置类型，如字典）隐式地对并发访问安全，可以简化CPython实现。锁定整个解释器使解释器更容易实现多线程化，而多处理器机器提供的并行性则大大降低。

锁定GIL会在一定程度上方便多线程化，但也会降低多处理器机器的并发性。所以，总的来说，大部分时候，克服这个性能问题将使实现更加复杂，因此维护成本更高。

GIL 本意是用来保护解释器和环境变量的。如果去掉 GIL，就需要多个更巧妙的锁对解释器的众多全局状态进行保护或者采用 Lock-Free 算法。但无论哪一种方法，要做到多线程安全都会比单使用 GIL 一个锁要难得多。对 Python 社区来说，有这么多第三方的扩展也依赖 GIL无异于作茧自缚。

再回到本章开始提到的小学数学题，我们使用threading实现的多线程更像是一个人高效率地完成多项任务，而multiprocessing库实现的效果则是一个人分成多个人，但也有一定的缺憾，这些人是共用一个大脑的，如果人多了，反而大脑会不够用。

**第12章 发送邮件**

本章讨论程序反馈最低廉且合理的方式，通过在联网状态下自动化发送邮件，实现服务器（程序）无人值守情况下的报备以及记录，帮助管理人员及时发现并且处理服务器在运行的时候出现的错误。

## 12.1 通信的选择

当我们运行自动化测试程序并且无人值守的时候，需要一种通信方式来告诉我们程序运行到了哪里，但这里为什么选择电子邮件这种古老的方式呢？原因其实很多。首先，最主要的是考虑到资费的原因，电子邮件目前来说可以说是少有的低成本的方式，如果读者选用第三方服务器，成本可以降到零。其次，我们需要考虑人的需求，正常人肯定不希望在熟睡的时候接到一个预先设定好的数字电话，提醒你去修复Bug。然后，再回到现实，企业邮件作为企业进行国内外事务、商务交流的基本途径之一，其安全性、稳定性有较高的保障。

在开始使用Python收发电子邮件之前，我们先来了解一下电子邮件的确切标准。互联网电子邮件标准收录在RFC2822中，RFC是一系列以编号排定的文件，收录了绝大部分的互联网通信标准。根据RFC的定义，“电子邮件由头字段以及可选的正文组成”，而在RFC2821（关于SMTP协议）中收到邮件的每一个MTA（Message Transfer Agent，消息传输代理），都必须在标头顶端加上自己的“Received:”字段，以描述该封邮件传递过程中的信息。

最理想的情况是发送机器和目标机器在同一局域网内，这样可以直接由发送机器送达到接收机器。实际情况却没有这么顺利，在一封电子邮件送达的过程中，需要多台计算机作为中转机器来负责邮件的路由、队列处理，直到最终送达。如果你打开一封邮件的头文件，你就会惊讶地发现，一封邮件送达之后，其实已经贴上了MTA的很多印记。

SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）正是因为可以帮助每台计算机在发送或中转信件时找到下一个目的地，所以它被称为现在电子邮件的基础。恰巧，Python支持SMTP，可以直接使用SMTP来实现电子邮件的发送。

这里使用QQ邮箱的POP3/SMTP服务来实现通过Python发送邮件。

## 12.2 获取授权码

 QQ邮箱对于第三方收发有着较严格的限制，在正式实现代码之前，我们还需要打开QQ的POP3/SMTP协议。建议读者登录自己的QQ邮箱，然后打开设置页面，跳转到“开启服务”这一栏。这里第一行的POP3/SMTP服务因为已经开启过了（见图12-1），所以显示的是“已开启|关闭”，读者看到的内容应该是和下面几个协议的状态相同。读者开启协议之后，保存即可。

图12-1 开发POP3/SMTP服务

POP（Post Office Protocal）协议用来实现下载邮件，最新版本是第3版，所以一般称为POP3

在开启协议之后，还需要获取一个授权码，来供程序使用。在刚才打开的设置界面的协议下方有一个生成授权码的按钮，如图12-2所示。

图12-2 生成授权码

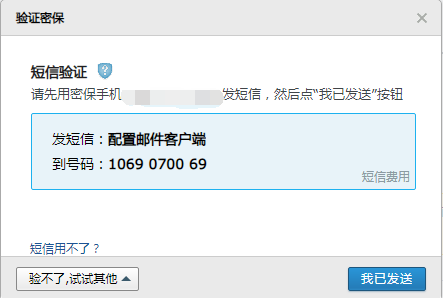
如果读者绑定了密保手机，在这一步还会要求使用密保手机发送一条短信来验证，如图12-3所示。

图12-3 发送短信

在验证通过之后，我们就可以拿到授权码了（如图12-4蓝框中的马赛克部分）。

图12-4 授权码

我们将授权码复制下来，在接下来的代码编写部分还会用到。

## 12.3 发送邮件

Smtplib是Python常用的库。这个库定义了一个SMTP客户端会话对象，可用于将邮件发送到具有SMTP或ESMTP侦听器的任何互联网上的计算机。

读者可以在RFC 821与RFC 1869中查看有关简单邮件传输协议和SMTP服务扩展的内容。使用以下代码创建实例。

smtplib.SMTP(host='', port=0, local\_hostname=None, [timeout, ]source\_address=None)

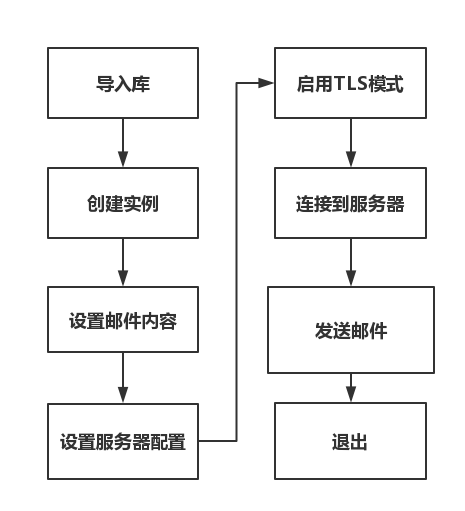
创建的实例分别有以下几个参数。

* host是需要连接的主机，
* port是连接主机使用的端口。
* 可选的 timeout参数用于指定停止操作：如果尝试连接10s后无反应（以秒为单位），那么停止连接；如果这个参数未指定，将使用全局默认超时设置。
* local\_hostname指的是如果指定了这个参数，那么者在使用HELO/EHLO command进行初始化的时候，这个参数的值就会被当作本地主机的FQDN（Fully Qualified Domain Name，全限定域名，FQDN使用“.”将主机名和域名连接起来，如果主机名为me，使用的域名为helloworld.com，那么FQDN就是me.helloworld.com，而邮件服务器的连接符是@，这也是邮箱地址的由来）。
* 可选参数source\_address用来将SMTP实例绑定到具有多个网络接口的计算机的源地址上。SMTP的常用方法如表12-1所示。

表12-1 SMTP的常用方法

|  |  |
| --- | --- |
| sendmail([from],[to],[msg]) | 将邮件[msg]（用列表或者元组来表示）从[from]发送到[to] |
| login([user],[password]) | 使用用户和授权码来登录服务器 |
| ehlo() | 初始化SMTP或ESMTP会话 |
| helo() | 同上 |
| starttls(keyfile=None,certfile=None) | 启用TLS模式，默认情况下，keyfile和certfile为空。如果指定keyfile和certfile，就使用这两个文件来创建安全套接字 |
| set\_debuglevel([level]) | 为服务器通信设置调试级别 |
| quit() | 关闭连接并退出 |

整体实现流程如下。

图12-5 发送邮件的流程

Python的email模块包含许多有用的电子邮件格式化函数。MIMEText对象为负责底层实现的MIME(Multipurpose Internet Mail Extensions，多用途互联网邮件扩展类型)协议传输创建了一个空消息，它最终通过高级SMTP协议发送。MIMEText对象MSG包含电子邮件地址、消息正文和主题，Python使用它们来创建正确格式化的消息。smtplib模块用于建立关于服务器连接的信息。

在这里用到了Python的smtplib 和 email两个包来发邮件。我们看一下程序。

#编码是utf-8

import smtplib,time

from email.message import Message

from time import sleep

import email.utils

smtpserver = 'smtp.qq.com'

username = '[your QQ ID]@qq.com'

password = '[your password]'

time = email.utils.formatdate(time.time(),True)

def Sendmail(from\_addr, to\_addr, msg):

message = Message()

message['Subject'] = msg['subject']

message['From'] = from\_addr

message['To'] = to\_addr

message.set\_payload(msg['content'] + '\n' + time)

m = message.as\_string()

sm = smtplib.SMTP(smtpserver,port=587,timeout=20)

sm.ehlo()

sm.starttls()

sm.ehlo()

sm.login(username,password)

sleep(2)

sm.sendmail(from\_addr,to\_addr,m.encode('utf8'))

sleep(2)

sm.quit()

return True

if \_\_name\_\_ == '\_\_main\_\_':

from\_addr = '2970359310@qq.com'

to\_addr = '2970359310@qq.com'

msg = {'subject':'这是测试','content':'测试'}

Sendmail(from\_addr,to\_addr,msg)

print("done")

如果出现以下类似的错误，那么证明读者在获取授权码的某个步骤出错了。

smtplib.SMTPAuthenticationError……

如果一切无误，读者应该可以看到程序输出done并且收到邮件。

下面这条命令是一个复合命令。首先，使用time.time()来获取时间轴（输出的时间格式像这样，1534678628.0146534表示的是从1970年1月1日0时0分0秒起至当前时间,经过的浮点秒数）。然后使用email.utils.formatdate()函数来将其格式化为标准形式的时间——Sun, 19 Aug 2018 13:24:36 +0800。

time = email.utils.formatdate(time.time(),True)

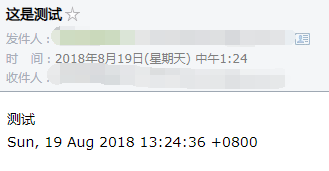
我们打开邮箱来检查一下，发现邮件已经成功发送了（见图12-6）。

图12-6 邮件发送成功

# 第13章 Selenium IDE

Selenium IDE和Katalon Recorder是两个简单且方便录制自动化脚本的工具，功能存在相同之处，也存在部分差异。本章将通过结束这两个录制工具为读者提供一条功能化的测试之道，以快捷地生成自动化脚本。

## 13.1安装SeleniumIDE

Selenium IDE是一个官方的插件，有Chrome和FireFox两个版本。Selenium IDE可以用来记录并且重复执行用户和浏览器的交互操作。使用这个编辑器可以创建一些简单的脚本和测试文件。一些比较版本比较新的FireFox可能无法使用Selenium IDE（其原因是Selenium IDE官方并未适配到最新版本），这里提供一个FireFox历史版本下载的网站（请参见FireFox官方网站）。

在这里，我们转向浏览器，不再主要使用Chrome浏览器。以FireFox浏览器作为例子，不仅是因为Selenium IDE支持FireFox，更重要的是，本章将会讲解并使用一个比官方版本插件更好的第三方插件。

下面介绍安装Selenium IDE的两种方法。

方法一是从Selenium的网站下载扩展组件文件。和之前的页面位置相似，在Selenium官方网站的下载页面上下载Selenium IDE，如图13-1所示。然后单击下载的文件安装Selenium

IDE。

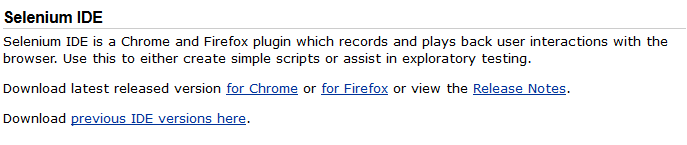


图13-1 下载Selenium IDE

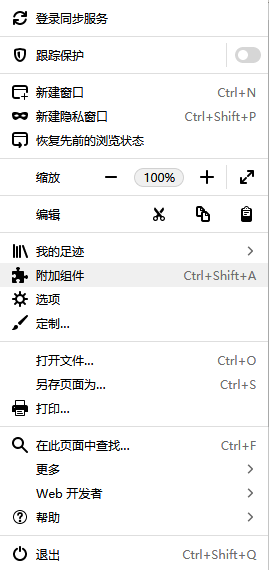
方法二如下。在Firefox浏览器中，单击 Fx57Menu图标，然后在弹出来的界面，选择“附加组件”（见图13-2），或者直接使用快捷键Ctrl+Shift+A（这个快捷键和QQ的截图快捷键是相同的，可能会造成按键冲突的问题）。

图13-2 选择“附加组件”

在右上角的输入框中输入Selenium，然后单击搜索按钮，在图13-3中即可看到需要的插件。

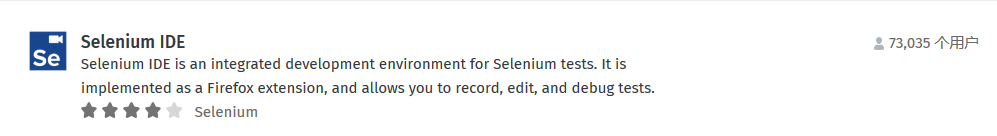


图13-3 搜索结果

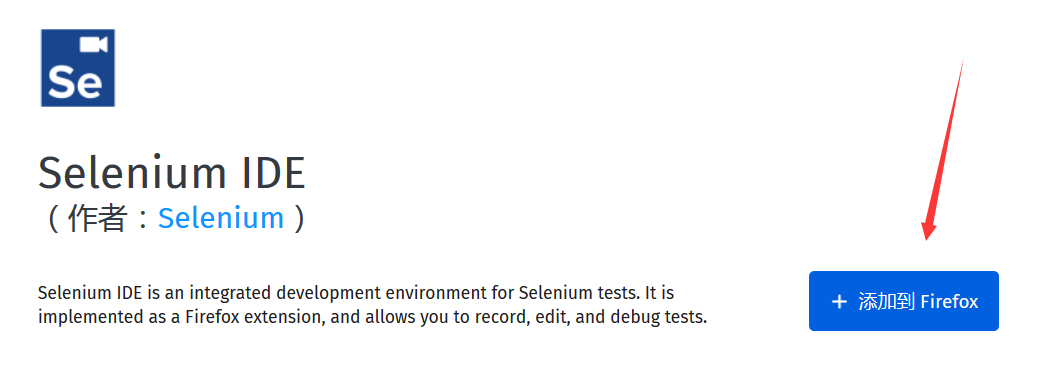
搜索结果中排名第一的插件就是我们需要的插件，单击该插件并展开，然后单击“添加到Firefox”按钮，如图13-4所示。之后，只需要静静等待插件安装完成即可，这可能会花费十几秒钟的时间。

图13-4 Selenium IDE插件的页面

这里建议读者同样安装排名第二的Katalon Recorder (Selenium IDE for FF55+)，我们会在稍后使用到这个插件。

## 13.2 Selenium IDE

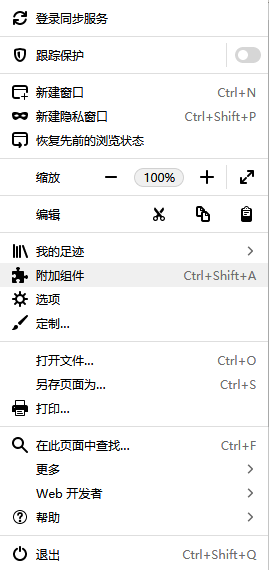
当两个插件都安装完成以后，从难到易，我们先从官方版本的Selenium IDE来开始学习。

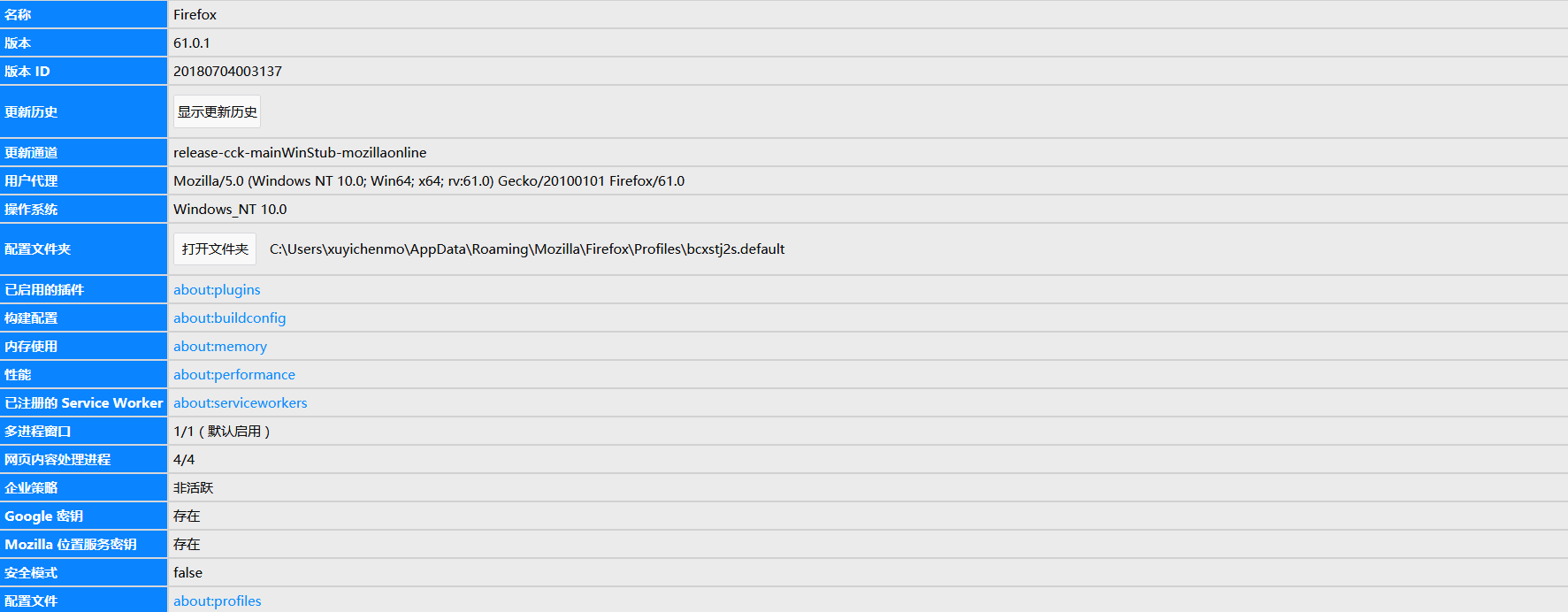
有一点需要提前说明，非常遗憾，在FireFox55之后的版本中，Selenium IDE的功能反而越来越少（这也是Katalon Recorder 中Selenium IDE for FF55+的由来，FF55+就是FireFox浏览器55以后的版本）。

原因有这几个。第一，Selenium项目是一个并不是一个商业化公司，项目开发者没有时间和精力来推动Selenium IDE的更新以适应新技术，第二，开源项目不以营利为目的，这就导致了研发资金的不足（这也是很多不错的开源项目存活周期短暂的原因）。第三，众所周知，FireFox是唯一的迭代速度上可以和Chrome不相上下的浏览器。作为一个有些小众的项目，Selenium无法适应如此频繁的项目版本更新和改动。最重要的是，因为在FireFox55以后的新版本从“xpi”格式变更为“web extension” 的机制，不支持原先的技术框架，在新版本的Selenium IDE已经不支持其他形式的拓展类型。

<http://ftp.mozilla.org/pub/firefox/releases/>

读者可以自由选择是倒退版本，牺牲新版本的体验来使用旧版本的Selenium IDE，或花费一定的时间，来了解Katalon Recorder（它的界面和Selenium IDE很相似）。

另外，在Firefox浏览器中，单击 Fx57Menu图标，选择“帮助”→“故障排除信息”。会弹出一个关于读者的Firefox浏览器的具体信息的内容，如图13-5所示（在这里读者可以看到自己的版本，来确定自己是否需要回退版本来追求兼容性）。

图13-5 Firefox的版本信息

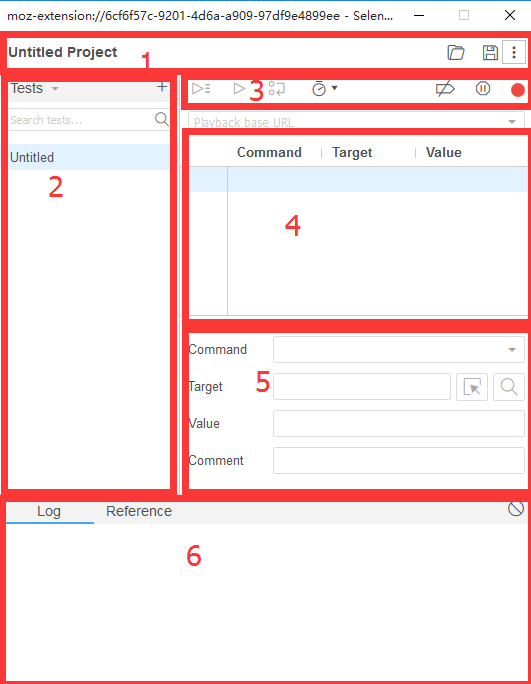
我们来看一下Selenium IDE的界面，这里用红框分别标注出每个部分，如图13-6所示。

图13-6 Selenium IDE的界面

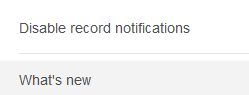
第一部分为最上面的标题栏，双击左上角的Untitled Project后可以修改项目的名字。右边的3个按钮从左到右依次是“打开文件”（用于再次使用已经录制的文件），以及“保存”和“更多”。“更多”这个按钮不是很常用，里面只有很少的内容，如图13-7所示。

图13-7 “更多”按钮下面的选项

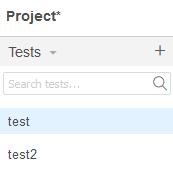
第二部分是项目下面小的分支。总的来说，还是隶属于这个项目的，比如，当在这里创建一个新的文件时，就会出现一个新的子项目（在这里将其命名为test和test2），如图13-8所示。

图13-8 项目分支

第三部分的按钮的作用分别如下。

* 运行项目中的所有测试实例。
* 运行当前实例。
* 运行当前命令。
* 调节测试速度。
* 禁用断点。
* 在异常处理处停止开始/停止录制。

第四部分的内容就相当于编写的代码部分。如果打开百度的页面进行一次录制搜索，读者就会在这里找到id为“su”和“kw”的常见元素，如图13-9所示。

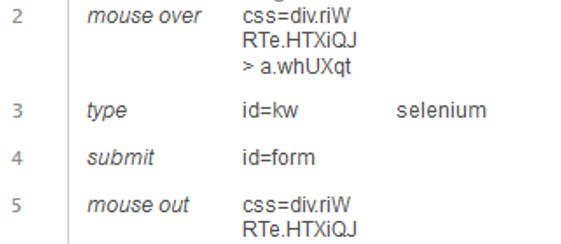


图13-9 脚本部分中的常见元素

第五部分则是对第四部分中单条命令的解释，准确地说是项目下指定实例的指定命令的具体内容。Command参数对应的是读者的操作，Target则是目标元素，Value这个参数是输入的内容，而Comment默认是没有值的，除非读者主动来写。

第六部分共有两项内容。第一项是日志，日志会在读者执行实例的时候输出，格式如下。

Running 'test'

1.open on /... OK

2.mouseOver on css=area[title="单击一下，了解更多"]... OK

3.type on id=kw with value selenium... OK

4.submit on id=form... OK

5.Trying to find css=div.riWRTe.HTXiQJ > a.whUXqt...

而reference的性质更像是Selenium IDE的说明文件，当读者单击指定的具体命令的时候，reference会显示关于命令的详细用法和信息。

mouse out locator

Simulates a user moving the mouse pointer away from the specified element.

arguments:

locator - An element locator.

当具体的内容介绍完了之后，相信读者已经对使用流程有了大概的了解，实际的使用方法倒相当简单。单击红色的start recording按钮，开始录制。进行一遍实际的操作之后，红色的按钮就会变成stop recording，单击stop recording按钮以结束录制。当一个项目中的每个实例都录制完成以后，以side文件的形式保存项目，然后在下次使用的时候，通过Selenium IDE直接打开side文件使用即可。

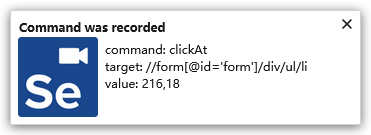
在我们进行录制的时候，会频繁弹出类似图13-10的窗口。请不要担心，出现这个窗口正证明读者正在录制命令。

图13-10 弹出的窗口

接下来，我们录制一个简单的程序。首先，在打开FireFox浏览器后，单击右上角的按钮打开Selenium IDE界面。然后，单击红色的start recording按钮，将Selenium IDE界面最小化，我们进入主页。

接下来，搜索Firefox插件，选择第一个结果，进入页面，并输入Selenium IDE，单击搜索按钮，打开Selenium IDE界面。

接下来，单击停止录制，当我们看到方形的红色按钮变为圆形时，意味着录制完成了。之后，单击第三部分的第二个按钮run current test。

但这时候，读者在运行的时候可能会发现一些问题。我们在百度页面中完成搜索以后，尝试从百度页面跳转到Firefox插件官网，第一次使用的时候会弹出一个询问是否允许baidu.com弹出的页面。这里的弹出其实并不是真的弹出，而是Selenium IDE实现打开页面这一操作的形式被Firefox视为弹出式页面，其对应的命令如图13-11所示。

图13-11 弹出式页面对应的命令

当运行成功后，单击第一部分的第二个按钮，将其保存为side文件的形式，这样就方便我们重复多次使用。

进行测试的时候通过录制脚本，一个脚本完成一个场景（一组完整功能操作），通过回放脚本来进行自动化测试。这是自动化测试目前流行的一种形式。

## 13.3 Katalon Recorder

Katalon Recorder是Selenium IDE的一个强大替代品，可以在任何浏览器中创建和回放实例。在C#、Java、Ruby、Python、Groovy、Robot框架中，捕获的Web元素和操作可以作为自动化测试用例记录、编辑和导出。该扩展可以在不同的场景中使用。简而言之，这是一个支持测试人员执行Web自动化测试或执行重复操作的工具。

**13.3.1** Katalon Recorder的界面

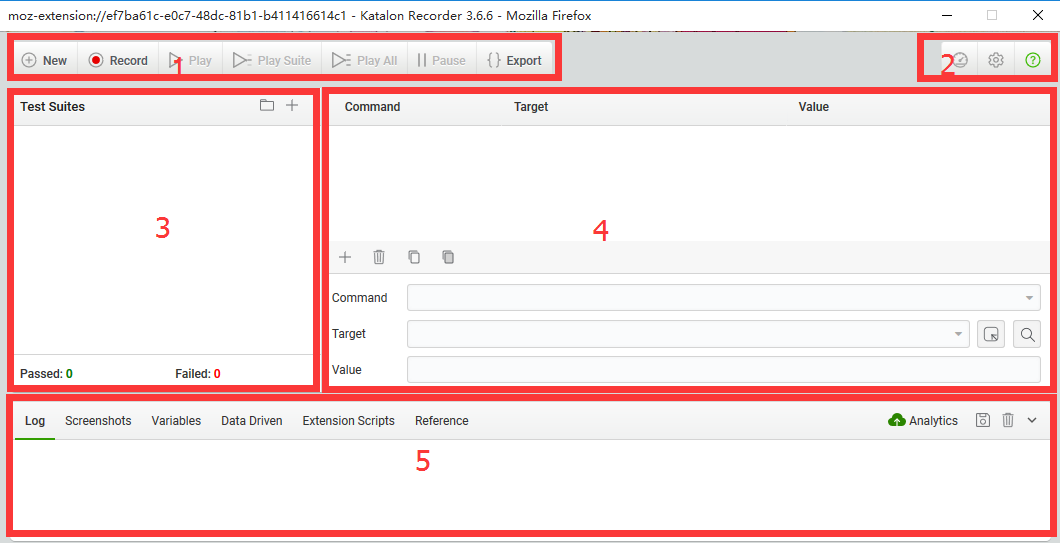
Katalon Recorder号称Selenium IDE for FF55+（Firefox55以后版本的Selenium IDE）确实有一定的缘由，它保存了Selenium IDE早期版本强大的功能。Katalon Recorder的界面（如图13-12所示）和现今版本的Selenium IDE在形式上存在一定的相似之处。

图13-12 Katalon Recorder的界面

事实上，Katalon Recorder是Katalon套件工具中的一个内容（还有Katalon Studio和Katalon Analytics）。Katalon说明文档声称自己是最适合测试团队的自动化测试工具，虽然这有夸大的嫌疑，但不得不说，它们确实都具有这个实力（Katalon有自己的社区和生态）。

同样，先分部分介绍Katalon Recorder的界面。

第一部分是Katalon Recorder的按钮，从左到右依次如下。

* New，创建一个新的实例。
* Record，开始记录操作的当前实例。
* Play，执行当前实例。
* Play Suite，执行当前项目的所有实例。
* Play All，执行所有项目
* Pause，暂停执行实例。
* Export，导出当前套件。

注意，New按钮创建的是一个实例，而不是一个项目。

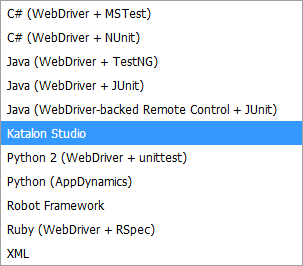
如图13-13所示，Katalon Recorder最强大的功能在于它支持以多种编程语言导出，支持C#、Java、Ruby、Python、Groovy、Robot框架。在这里，建议读者以我们熟悉的Python语言导出。

图13-13 可供导出的语言

第二部分的3个选项的作用分别如下。

* 调节执行速度。
* 端口设置。
* 帮助。

其中调节执行速度和Selenium IDE基本上完全相似，都是一个下拉框，可以滑动地调节滚动轴。

端口设置在Firefox浏览器中暂时无法使用，目前仅仅在Chrome浏览器中可以使用，官方是这样说的——“Allow users to change the default port used by [Katalon Studio](https://www.katalon.com/) to communicate with the active Chrome browser (support for Firefox is coming soon)”。

目前，在Firefox浏览器中我们依然无法使用这一项功能。不过，幸运的是，我们并不是很经常使用到这一功能，默认端口已经完全足够我们使用。

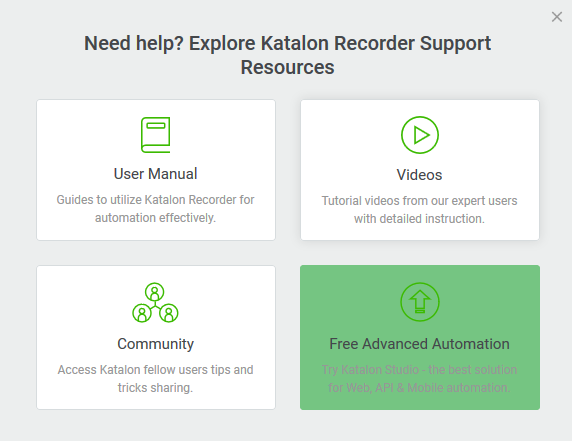
图13-14展示的是“帮助”选项的4部分内容，按照从上到下、从左到右的顺序，依次是User Mannul、Videos、Community和Free Advanced Automation。

图13-14 “帮助”选项的4部分内容

### 实例与步骤

在第三部分会显示测试条件和实例。

一级标题是项目（suite意为套件，称为项目或许更合适）。二级标题的内容是Katalon Recorder的测试实例。一个项目下面可以存在多个测试实例，而一个测试实例对应一个场景的应用，如图13-15所示。

图13-15 测试实例

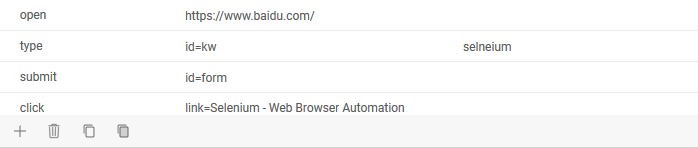
第四部分则是录制步骤。当我们完成一项录制之后，实例执行的具体步骤就会显示在这里，左下角的4个图标分别用于完成添加、删除、复制、粘贴操作，如图13-16。

图13-16 完成添加、删除、复制、粘贴操作的图标

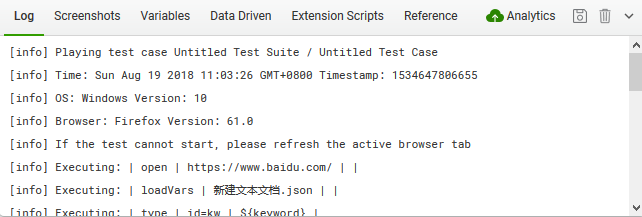
下方是对步骤的具体解释。如图13-17所示，比如，我们在下方找到的这个目标说明它的方式是输入字符，目标是id=kw的元素，输入的值是selenium，这就相当于以下代码。

driver.find\_element\_by\_id(“kw”).send\_keys(“selenium”)

图13-17 步骤的解释

最下方的界面如图13-18所示。其中的选项卡分别如下。

* Log，显示实例执行的日志。
* Screenshots，显示测试实例中的截图。
* Variables，显示当前选择的命令的详细信息。在执行测试时，用户可以查看此选项卡中的命令名、目标和值。
* Data Driven，允许用户上传CSV文件进行数据驱动测试。用户还可以通过编辑名称或删除不需要的数据文件来管理测试数据文件。
* Reference，显示所选命令的详细文档，这个选项卡用来帮助用户确保命令的正确类型和参数数量。
* Analytics，将执行日志上传到Katalon Analytics，以跟踪执行历史、访问测试自动化智能仪表板和报告。

最后两个图标分别用于保存日志和删除日志。

13-18 下方界面

如果使用数据分析功能，读者还需要搭配Katalon Analytics来使用，这需要读者注册一个Katalon账户。不过，不用担心，Katalon的大部分内容都是免费的，包括这个账户在内。

Katalon Recorder提供了一个叫作“captureEntirePageScreenshot”的截图命令。比如，新建一个步骤，然后在Command中设置为“captureEntirePageScreenshot” （如图13-19所示）即可实现截图。获得的截图可以在图13-18所示界面的Screenshots选项卡中查看。

图13-19 截图命令

## 13.4 数据驱动

当我们进行自动化测试的时候，如果数据量特别大，比如，我们尝试一个8位数的密码的所有组合，肯定不能使用重复的录制实现。这时候，我们需要实现数据和程序的分离，在执行的时候，重复调用特定格式的数据，进行模式相同的测试，简单来说，就是循环。

我们使用数据驱动来实现Katalon Recorder的进阶用法。

首先，在桌面创建一个txt文件，输入以下内容，保存，并将txt文件的格式后缀名改为json文件。这时候，我们就会用到上面提到的Data Driven（数据驱动）。

[

{

"keyword": "selenium"

},

{

"keyword": "katalon"

},

{

"keyword": "hello world"

}

]

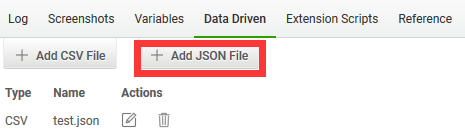
然后，打开Katalon Recorder，在Data Driven选项卡中添加刚才创建的JSON文件，如图13-20所示。接下来，打开百度，搜索一个内容，把步骤录制下来。接着，在循环体的前后分别加入步骤loadVars和endLoadVars。

图13-20 添加JSON文件

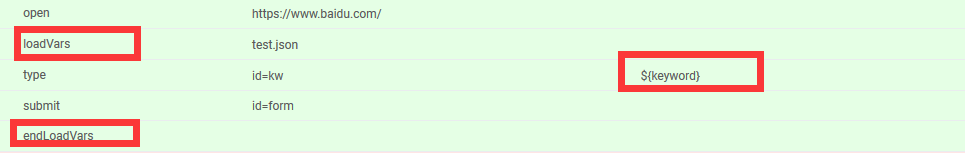
需要更改的3个地方在这里已经用红框标注出来了。loadVars和endLoadVars之间是要重复执行的步骤，在loadVars命令的Target部分指定数据文件的名称，这里使用的是test.json。然后，在type命令的值中添加${keyword}，这是一个定义的变量，如图13-21所示。

图13-21 定义变量

接下来，就可以尝试运行了。如果没有意外，应该可以看到先打开了百度，然后在搜索框依次输入了3个关键词进行搜索。

## 13.5 扩展脚本

从版本3.5.0开始，Katalon Recorder支持扩展脚本（Firefox上Selenium IDE的弃用版本中的AKA user-extensions.js）。此时此功能允许读者将构建的自定义定位器和自己的操作添加到Katalon Recorder中。这里的扩展脚本大多是JavaScript内容，需要读者有一定的Java编程基础。这些并不是我们学习的重点，建议读者在学有余力的情况下对这里的内容进行一些深入探索。涉及到Selenium和Katalon Recorder的底层实现。

**13.5.1 添加扩展脚本**

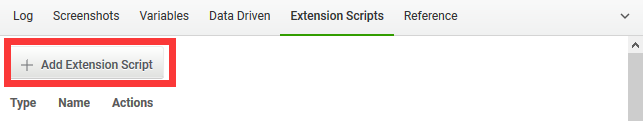
要添加扩展脚本，请单击底部面板上的“Extension Scripts”选项卡，然后单击“Add Extension Script”按钮，如图13-22所示。最好重启一次Katalon Recorder，不要忘记刷新要处理的选项卡，因为新的扩展脚本仅对添加后打开的选项卡有效。

图13-22添加扩展脚本

**13.5.2 定位器构建器**

一个定位器识别一个命令指定的元素，而定位器构造器用来创建这些定位器。在下面这个定位器构建器中，会返回指定元素的CSS选择器路径。

LocatorBuilders.add('custom locator id goes here', function(e) {<font></font>

if (e.id) {<font></font>

return "css=" + e.tagName + '#' + e.id;<font></font>

}<font></font>

return null;<font></font>

});

**13.5.3 定位器构建的自定义顺序**

在扩展脚本中，根据个人爱好，可以使用下面的命令来调整定位器构建的顺序。

LocatorBuilders.\_preferredOrder = ['xpath:position', 'my super locator'];

LocatorBuilders.\_orderChanged();

第一条命令用于设置定位器的默认顺序，第二条命令用于将默认顺序更改为优先顺序。

支持的定位器构建器有以下这些。

* "id"
* "link"
* "name"
* "dom:name"
* "xpath:link"
* "xpath:img"
* "xpath:attributes"
* "xpath:idRelative"
* "xpath:href"
* "dom:index"
* "xpath:position"
* "css"

### 13.5.4 Prototype附加命令

读者使用Selenium的所有“get[name]”和“is[name]”方法都会自动影响下面这些命令的可用性。

* “store[name]”
* “assert[name]”
* “assertNot[name]”
* “verify[name]”
* “verifyNot[name]”
* “waitFor[name]”
* “waitForNot[name]”

例如，如果添加一个“getTextLength()”方法，那么以下命令将自动可用“storeTextLength”“assertTextLength”“assertnoextlength”“verifyTextLength”“verifynoextlength”“waitForTextLength”和“waitfornoextlength”命令。

**13.6 Katalon Recorder Helper工具**

由于JavaScript和浏览器API的限制，Katalon Recorder无法单独执行某些操作。在这些情况下，Katalon Recorder将命令发送到名为Katalon Recorder Helper的CLI工具以完成操作。Katalon Recorder Helper是一个开源项目。它实际上是一个非常轻量级的Java服务器，它可以监听来自Katalon Recorder的命令。它建立在Spring Boot框架之上。

使用的时候，读者需要提前在计算机上安装Java Runtime Environment version 8.x或者以上。

接下来，可以从GitHub网站上的发布页面下载最新版本的“katalon-recorder-helper.jar” 文件。

在下载完成之后，读者从文件所在目录打开cmd（按住Shift键并右击），使用java -jar katalon-recorder-helper.jar就可以打开Katalon Recorder Helper工具了。Katalon Recorder Helper将默认在18910端口侦听命令。

那么Firefox浏览器上的Katalon Recorder本身有什么局限性以至于我们需要添加一个新的工具呢？其实是因为Firefox浏览器API的限制，Katalon Recorder在Firefox浏览器无法上传文件（在Chrome浏览器中不需要Katalon Recorder Helper也可实现上传文件）。

命令“type | locator | filepath”仅仅在Katalon Recorder Helper启动的时候才有效，因为它利用Java成功模拟键盘和鼠标操作来弹出文件上传对话框。

Katalon Recorder Helper并不是仅仅有这一个功能，读者可以使用Katalon Recorder Helper触发系统命令。用户可以将此功能与用户扩展结合使用，以扩展Katalon Recorder的功能。

比如，当读者使用下面这一条命令可以用来打开在cmd中执行dir命令，执行结果如图13-23所示。

http://localhost:18910/execute?cmd=dir

图13-23 执行结果

页面返回的内容就是在命令行窗口中执行这条命令会输出的内容。

**第14章 Python扩展**

这一章讨论的内容包括但不限于：把Python2代码转换Python3代码，测试类型，通配符，字符串方法，异常层次结构，类，推导式等。这些内容将会帮助读者更好地处理自己面对的问题，提高编程效率与水平。

## 14.1 2to3工具

众所周知，Python有两个大的版本分支。在这两个版本之间存在着较大的改动，并且Python是不向前兼容的，这就意味着一些基于Python 2编写的优质的库可能由于更新不及时，或者已经不再更新，而无法被使用Python 3 用户使用。

幸好，Python已经提供一个解决方案——2to3.py，并且这个Python文件是安装Python以后自带的文件。

2to3.py存在于Python目录下的一个tool\scripts目录中，例如，作者的计算机上2to3.py所在的目录为F:\Python\Tools\scripts。

首先，在文件管理器中打开相应的目录。然后，在tool\scripts目录下同时按下Shift键盘并右击，选择在此处打开命令行窗口，最后在弹出的命令行窗口中使用如下命令。

python 2to3.py -w [dir]

这条命令指定了需要转换的文件的所在目录，可以同时对该目录下的多个Python 2文件进行批量转换。

另外几个常用的参数分别为-n和-o。-n表示不对被转换的文件进行备份。-o用于指定输出目录，指定修改过后的Python文件的存储目录（当使用-o参数的时候必须搭配-n参数）。

则使用的命令为我们新建一个py2.py内容为：

print "hello world"

这里的输出格式和GIT输出的格式很像。

--- C:\Users\xxx\OneDrive\Documents\SELENIUM\xxx.py (original)

+++

C:\Users\xxx\OneDrive\Documents\SELENIUM\xxxx.py (refactored)

@@ -1 +1 @@

-print "hello world"

+print("hello world")

RefactoringTool: Files that were modified:

都是采用3个减号表示原来的文件，采用3个加号表示新生成的文件，-+表示新旧文件的逐行对比。

再次打开py2.py，Python文件的内容已经修改过了， 生成的新文件的内容为：

print("hello world")

并且在相同下生成了一个新的py2.py.bak文件。生成的py2.py.bak是原先Python 2代码的备份文件，当出现意外的情况，读者可以使用这个备份文件来恢复原来的代码，只需要将文件后缀名的.bak删除即可正常使用。

## 14.2 测试类型

在代码的编写过程中，经常会碰到需要测试和对比类型的情况。以下几种方法可以用来便捷地进行类型测试。

首先，创建一个空的列表用于后面的展示。

l=[]

l=[]定义了一个空的列表，而l也是一个空列表，所以返回值

方法一采用的代码如下。

if type(l) == type([]):

print("L is list")

方法二采用的代码如下。

if type(l) == list:

print("L is list")

list是Python的保留字，可以直接使用。有一点需要说明一下，变量的命名不能和保留字相同，比如，不能把新定义的一个列表命名为list。

方法三采用的代码如下

if isinstance(l,list):

print("L is list")

isinstance()方法用来比较前后两个参数是否是同一类型，更准备地说，验证前一个参数是否是后一个参数（类）的实例。

## 14.3 通配符类型

%这个特殊的字符用于格式化字符串，作用和函数的参数很相似，都用于提供一定程度上的可控性，以%开头的格式化字符都叫作通配符。通配符细分的种类有很多，每个都代表着不同的格式，比如，下面两条函数的输出效果是相同的， %d在这里用于替换Int。

print("我今年%d岁" %12)

print("我今年12岁")

表14-1列出大部分常见的Python通配符。

表14-1 常见的Python通配符

|  |  |  |
| --- | --- | --- |
| 通配符 | 说明 | |
| %d | | 有符号整数(十进制) |
| %s | | 字符串 |
| %c | | 字符及其ASCII码 |
| %u | | 无符号整数(十进制) |
| %o | | 无符号整数(八进制) |
| %x | | 无符号整数(十六进制) |
| %X | | 无符号整数(十六进制大写字符) |
| %e | | 浮点数字(科学计数法) |
| %E | | 浮点数字(科学计数法，用E代替e) |
| %f | | 浮点数字(用小数点符号) |
| %g | | 浮点数字(根据值的大小采用%e或%f) |
| %G | | 浮点数字(类似于%g) |
| %p | | 指针(用十六进制可打印值的内存地址) |
| %n | | 存储输出字符的数量并放进参数列表的下一个变量中 |

## 14.4 str方法

当使用NLTK进行词性分析合对收集到的文字进行统计的时候，必然要处理字符串。表14-2罗列了str类的内置方法。

表14-2 str类的内置方法

|  |  |
| --- | --- |
| 内置方法 | 说明 |
| str.ljust(width) | 获取固定长度，右对齐，如果左边不够，用空格补齐 |
| str.rjust(width) | 获取固定长度，左对齐，如果右边不够，用空格补齐 |
| str.center(width) | 获取固定长度，中间对齐，如果两边不够，用空格补齐 |
| str.zfill(width) | 获取固定长度，右对齐，如果左边不足，用0补齐 |
| str.find('t',start,end) | 查找字符串，可以指定起始及结束位置搜索 |
| str.rfind('t') | 从右边开始查找字符串 |
| str.count('t') | 查找字符串出现的次数 |
| str.replace('old','new') | 替换函数，替换old为new，参数中可以指定maxReplaceTimes |
| str.lstrip([str]) | 从字符串左侧取出指定字符，默认为空格 |
| str.rstrip([str]) | 从字符串右侧取出指定字符，默认为空格 |
| str.strip([str]) | 从字符串中取出指定字符，默认为空格 |
| str.startswith('start') | 表示是否以start开头 |
| str.endswith('end') | 表示是否以end结尾 |
| str.isalnum() | 判断字符串是否全为字符 |
| str.isalpha() | 判断字符串是否全为字母 |
| str.isdigit() | 判断字符串是否全为数字 |
| str.islower() | 判断字符串是否全为小写 |
| str.isupper() | 判断字符串是否全为大写 |
| str.upper() | 全部大写 |
| str.lower() | 全部小写 |
| **str.swapcase()** | **大小写转换** |
| **str.capitalize()** | **句子首字母大写** |
| **str.title()** | **每个单词的首字母都大写** |

有一点需要说明一下。下面两行代码在很大程度上相似，都会返回要查找的字符开始出现的位置，但不同的是使用index查找不到会抛异常，而find则返回－1。

str.index(str, beg=0, end=len(string))

str.find(str, beg=0, end=len(string))

**14.5 异常层次结构**

异常是在程序执行过程中发生并影响程序执行的事件，代表错误的Python对象。通常，当Python无法正确处理程序时，会出现异常。当Python脚本有异常时，程序将终止执行。为了避免这一情况，需要使用try…except语句进行异常的捕获。

捕获异常也有一定规则，这里列出异常的层次结构，方便读者查阅。

BaseException

+-- SystemExit

+-- KeyboardInterrupt

+-- GeneratorExit

+-- Exception

+-- StopIteration

+-- StandardError

| +-- BufferError

| +-- ArithmeticError

| | +-- FloatingPointError

| | +-- OverflowError

| | +-- ZeroDivisionError

| +-- AssertionError

| +-- AttributeError

| +-- EnvironmentError

| | +-- IOError

| | +-- OSError

| | +-- WindowsError (Windows)

| | +-- VMSError (VMS)

| +-- EOFError

| +-- ImportError

| +-- LookupError

| | +-- IndexError

| | +-- KeyError

| +-- MemoryError

| +-- NameError

| | +-- UnboundLocalError

| +-- ReferenceError

| +-- RuntimeError

| | +-- NotImplementedError

| +-- SyntaxError

| | +-- IndentationError

| | +-- TabError

| +-- SystemError

| +-- TypeError

| +-- ValueError

| +-- UnicodeError

| +-- UnicodeDecodeError

| +-- UnicodeEncodeError

| +-- UnicodeTranslateError

+-- Warning

+-- DeprecationWarning

+-- PendingDeprecationWarning

+-- RuntimeWarning

+-- SyntaxWarning

+-- UserWarning

+-- FutureWarning

+-- ImportWarning

+-- UnicodeWarning

+-- BytesWarning

如果不需要根据异常的类型来进行处理，并且可以直接捕获所有的异常，那么可以直接捕获Exception来获取大部分代码在执行过程中出现的错误（基本上这些错误是人为造成的）。

如果要一次性捕获所有的异常，那么直接捕获BaseException即可。

try:

...

except BaseException as e:

...

print(e)

需要注意的是，一般情况下，异常处理都是划分得越细越好。

一般来说，要求尽量不要同时捕获非同一层次结果的异常，尤其是当两个异常含有上下层次的关系的时候。

如果读者处理的是UnicodeError异常，那么无论如何读者也得不到UnicodeDecodeError异常（哪怕读者在except中同时明确指定了UnicodeError异常和UnicodeDecodeError异常），因为会把UnicodeDecodeError异常提交到上级UnicodeError异常。

## 14.6 兼容Python 2和3

一种常见的解决方法是将Python目录下的python.exe改名。比如，将Python 2版本的python.exe更名为python2.exe，这样每次在运行文件的时候，输出Python 2可以执行相应的代码。

但这种方案存在一定的隐患，被修改名字的Python版本对应的pip是无法使用的，每次使用必须先cd切换到对应的Python目录下的Scripts目录中。

不过，还有另一种更好的解决方案——直接指定版本号。

py -2 [python\_file\_name]

py -3 [python\_file\_name]

在Python 3.3以后的版本，Python安装包实际上在系统上安装了一个名为py.exe的启动器。这个启动器默认放在文件夹C:\Windows目录下，并且允许我们指定版本号，甚至pip的版本号。

由于两个包管理器都叫作pip，甚至都存在于环境变量中，因此我们无法通过直接使用pip命令来指定我们想要安装包的Python的版本（默认使用在环境变量中靠前的目录下的pip，系统环境变量大于用户环境变量）。

py -2 -m pip install [lib\_name]

py -3 -m pip install [lib\_name]

可以通过另一种方式指定Python文件的编译器。

#! python2

#! python3

在文件的开头直接指定编译器版本后，可以直接使用py.exe启动器运行Python文件。

py [python\_file\_name]

在声明中，优先使用编译器声明作为第一行，而使用编码声明作为第二行。因为如果没有特殊需要，编码声明仅仅是在Python 2.x版本中是必需的，在Python 3版本中默认支持中文。

所以，Python 2版本的文件声明开头如下。

#! python2

# coding: utf-8

而在Python 3中则只有第一行的编译器声明。

## 14.7 兼容性代码

在实际工作中，我们常常会遇到这样一种情况——在自己计算机上可以正常运行的代码移植到不同操作系统（甚至同一操作系统的不同版本）的不同计算机中之后却无法运行。缺乏兼容性的代码就像是缺了一点绿色点缀的万花丛，即便再鲜艳，也存在着一丝丝难以言明的缺憾。 这就是我们为什么要编写兼容性代码。

## 14.7.1 导入库

虽然在Python 2到Python 3的大版本切换中，很多库的名字都更改了，但具体内容很少变动。我们可以针对Python 2和Python 3写两套导入的库。

try:#Python 2

from UserDict import UserDict

#建议按照Python 3的名字进行导入

from UserDict import DictMixin as MutableMapping

except ImportError:#python3

from collections import UserDict

from collections import MutableMapping

## 14.7.2 输出函数

本节介绍常见的输出函数。Python 3的print()函数可以在Python 2中运行（事实上，在Python 2.6中已经带了print()函数），而Python 2的关键字print无法在Python 3中使用。不管是在Python 3还是Python2中，都建议直接使用print()函数。

但有一点需要注意，Python 3和Python2的print()函数略有不同。

Python3的print()函数如下。

>>> print(1, 2)

1 2

#Python2的print()函数如下。

>>> print(1, 2)

(1, 2)

另外一种解决方案是使用sys库来输出。

import sys

sys.stdout.write("Hello World!\n")

在IDLE交互模式下使用sys.stdout.write()还会返回另外一个缓冲区的数据——字符串长度，而直接执行Python文件则不会出现这个问题。

>>> sys.stdout.write("Hello World!\n")

Hello World!

13

## 14.7.3 异常捕获

在Python 2中，使用如下方式进行异常处理。

try:

...

except ValueError, e:

...

而下面这种方式在Python 2和Python 3中都可以正常使用。

try:

...

except ValueError as e:

...

## 14.8 类相关

先实例化一个类。

test=MyObject()

对于这个类，有表14-3所示的几种方法。

表14-3 类的方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| dir(test) | 获取类的方法列表 |
| hasattr(test,"x") | 测试是否有x属性或方法 即a.x是否已经存在 |
| setattr(test,"y", 19) | 设置属性或方法，等同于a.y = 19 |
| getattr(test,"z", 0) | 获取属性或方法，如果属性不存在，则返回默认值0 |

另外，setattr()方法可以设置一个不能访问到的属性，即只能用getattr()方法获取。比如，设置一个带有空格的属性名，这个属性不能直接访问，但可以通过getattar()方法获取。

setattr(test,"A B", 100)

getattr(test,"A B", 0)

下面这个例子和类的动态绑定有关。动态绑定适用于对已经写好的类新增一个方法或者属性。

>>> class test(object):

pass

>>> case=test()

>>> case.name="name"

>>> def case\_value(self,value):

self.value=value

>>> from types import MethodType

>>> case.set\_value= MethodType(case\_value, case)

>>> case.set\_value(100)

>>> case.name

'name'

>>> case.value

100

在这里例子中，首先从Object继承了一个空的类，然后创建一个实例化对象。对于这个对象，首先直接使用赋值的方法设置其name属性，然后创建了一个case\_value()函数，这个函数的写法和类中函数的写法是相同的。

接下来，导入MethodType()，使用MethodType()方法将刚才定义的case\_value()函数和实例化的类对象绑定到一起。最后，为绑定后的函数赋值。

## 14.9 推导式

假设我们现在有如下列表。

numbers = [1, 2, 3, 4]

如果要让所有元素翻倍，那么按照之前的经验和逻辑来说，可能会选择这种方式。

for i in range(len(numbers)):

numbers [i]=numbers [i]\*2

但事实上，这种写法和方式并不值得推崇，我们有一种更好的实现方式。

numbers =[elem \* 2 for elem in numbers]

这种方法叫作Python的列表推导式。

再看一种略微高级一些的用法。以下代码用于从列中查找可以直接被3整除的数字。

>>> list\_1=[i if i%2==0 else 1 for i in numbers]

>>> list\_1

[1, 2, 1, 4]

可以使用这种方式来实现常见的格式转换。

>>> dic = {"k1":"v1","k2":"v2"}

>>> a = [k+":"+v for k,v in dic.items()]

在这种列表推导式中，从字典中的每一项都取出两个值，然后将其变为列表中的一项。

类似地，Python同样支持字典推导式。

>>> dic = {x: x\*\*2 for x in numbers}

>>> dic

{1: 1, 2: 4, 3: 9, 4: 16}

>>> type(dic)

<class 'dict'>

这样的列表推导式还可以这样实现。

>>> dic = {x: x\*\*2 for x in [1, 2, 3, 4]}

或者这样实现。

>>> dic = {x: x\*\*2 for x in (1, 2, 3, 4)}

既然Python中的列表、字典都有推导式，那么对应地也会有元组推导式吗？其实不然，在Python中圆括号用作生成器的语法。

>>> tup = (x for x in range(5))

>>> print(tup)

<generator object <genexpr> at 0x031217B0>

>>> print(type(tup))

<class 'generator'>

>>> tup = tuple(x for x in range(9))

>>> print(tup)

(0, 1, 2, 3, 4, 5, 6, 7, 8)

>>> print(type(tup))

<class 'tuple'>

虽然无法直接实现Python的元组推导式，但事实上，通过类型转换函数达到的效果是近似的。

## 14.10 经典除法和真除法

在编程语言中，1/2和1.0/2.0的结果是不一样的，前者的结果是0，后者的结果是0.5，前者是整型计算，后者是浮点型计算。

这种说法在其他编程语言上确实没有错误，但在Python中有些不同。在早期的Python 2版本中确实采用这种经典除法，但在后来的Python 3中，这种除法已经逐渐被取代了。

>>> 1/2

0.5

>>> 1.0/2.0

0.5

如你所见，在上面几个例子中（通过Python 3.*x*演示），1/2和1.0/2.0的结果是一样的。

如果仍旧要在Python 3.*x*版本中使用经典除法，那么可以使用以下代码。

>>> 1//2

0

而在Python 2.*x*中使用真除法则可以使用以下代码。

from \_\_fututer\_\_ import division

## 14.11 \_\_future\_\_

因为Python是一种社区驱动的开放源代码和自由开发语言，不受商业公司的控制，所以Python的改进经常非常彻底并且不兼容前面的版本（也就是在当前版本中运行正常的代码，到下一个版本中就可能无法正常运行了）。于是，Python提供了\_\_future\_\_模块以确保顺利过渡到新版本。

\_\_future\_\_模块允许我们在旧版本中试验新版本的一些特性。

可以通过如下方式来单独使用其中的某一个特性。

from \_\_future\_\_ import FeatureName

可用的部分读者可以从源代码中找到，\_\_future\_\_.py位于Python\Lib目录中。

在Python 3.*x*版本，大致有以下内容可以使用。

all\_feature\_names = [

"nested\_scopes",

"generators",

"division",

"absolute\_import",

"with\_statement",

"print\_function",

"unicode\_literals",

"barry\_as\_FLUFL",

"generator\_stop",

"annotations",

]

关于这些内容详细的用法，读者可以直接阅读源代码。

## 14.12 类型转换和关键字

读者在进行数据处理的时候，难免要进行类型的转换和更改。表14-4列出了一些类型转换方法。

表14-4 类型转换方法

|  |  |
| --- | --- |
| 方法  int(*x* [,base]) | 说明  将*x*转换为一个整数 |
| long(*x* [,base] ) | 将*x*转换为一个长整数 |
| float(*x*) | 将*x*转换为一个浮点数 |
| chr(*x*) | 将整数转换为字符 |
| unichr(x) | 将整数转换为Unicode字符 |
| ord(x) | 将字符转换为它的整数值 |
| hex(x) | 将整数转换为一个十六进制字符串 |
| oct(x) | 将整数转换为一个八进制字符串 |
| list(s) | 将序列 s 转换为一个列表 |
| set(s) | 将序列 s转换为可变集合 |
| frozenset(s) | 将序列 s转换为不可变集合 |
| complex(real [,imag]) | 创建一个复数 |
| eval(str) | 用来计算字符串中的有效Python表达式,并返回一个对象 |

Python关键字或者又叫保留字，代表Python使用的一系列字符，变量名不可命名为这些关键字，并且每一个关键字都有特定的含义和方法。

kwlist中列出了Python所有的关键字。

>>> import keyword

>>> dir(keyword)

['\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'iskeyword', 'kwlist', 'main']

>>> keyword.kwlist

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

# 第15章 GUI测试

到目前为止，本书的所有内容都是处于一种与文字交互的状态中。以符号为主的界面虽然看起来更加炫酷，但不太符合用户的使用要求。如果你对于一个开发者来首，要求他能看懂编程语言返回的信息，这么做是完全没有问题，但如果你强迫要求所有使用者都必须看懂这些字符，这就有点不现实。

tkinter库（或者说“Tk接口”）是Tk GUI工具包的标准Python接口。Tk和tkinter都可以在大多数UNIX平台以及Windows系统上使用。(Tk本身不是Python的一部分，它维持在ActiveState，Tk工具最初是为tcl设计的。)

可以认为Tk+interface=tkinter，Python提供的IDLE就是使用tkinter设计出来的。

## 15.1 tkinter测试

在Python 3.2之后的版本中，tkinter库已经内置到Python中了。然而，如果你恰巧使用3.*x*版本的Python同时版本号又在3.2以下，那么需要通过pip来安装tkinter。

pip install tkinter

当要使用tkinter库的时候，可以直接导入该库。

import tkinter

tkinter.\_test()

如果你使用2.*x*版本的Python，那么需要将t换成T。

import Tkinter

Tkinter.\_test()

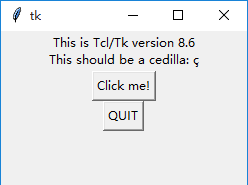
如果一切无误，应该可以打开一个简单的测试界面了，如图15-1所示。

图15-1 测试界面

这个测试窗口还可以通过按Windows+R组合键打开“运行”对话框，然后在“打开”文本框中输人cmd，弹出命令行窗口。在命令行窗口中执行以下命令。

python -m tkinter

该命令的效果也是相同的。

## 15.2 创建界面

GUI并不是由设计师画出来的，而是采用程序生成的。先生成界面，代码如下。

#导入库

import tkinter as tk

#实例化对象

root=tk.Tk()

#设置标题

root.title("selenium 自动化爬虫")

#创建标签

text=tk.Label(root,text="a simple tkinter GUI")

#将文字居中

text.pack()

#程序循环

root.mainloop()

实现的效果如图15-2所示。

图15-2 实现的效果

首先，导入库。为了书写方便，把tkinter库简写成tk，实例化对象实际上就相当于已经创建了界面，但是这个界面还是空白的，需要填充一点内容。

app.tilte()设置并调整窗口的标题，这里采用label标签显示文字，label()控件的第一个参数是实例化对象，而第二个是需要显示的文字，那么text.pack()用来干什么呢？实际上，这个text.pack()用来调节文字的尺寸和位置，后面几节还会用到几个类似的方法。

最后的mainloop()是必不可少的。如果不使用mainloop()，窗口会一闪而过。换句话说，mainloop()用来维持窗口始终可见。

mainloop()一旦检测到事件，就立刻刷新组件；否则，就不断进行循环检查。主循环会一直执行，直到出现某个指定的请求（检查内容包括鼠标、键盘等输入）

## 15.3 按钮

对于一个大的程序而言，和用户进行交互是必不可少的。我们看一下按钮是怎样实现的。

#导入库

import tkinter as tk

#实例化对象

root=tk.Tk()

#设置标题

root.title("selenium 自动化爬虫")

#简单的函数

def hello\_world():

print("hello\_world")

#创建按钮并居中

button=tk.Button(root,text="i'm a button",fg="blue",command=hello\_world)

button.pack()

#程序循环

root.mainloop()

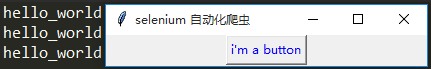
运行上述代码之后，显示的按钮控件如图15-3所示。

图15-3 按钮控件

Tkinter中大部分控件的参数基本上是相同的，比如，我们刚才实现的Lable和Button控件实际上只有一个command参数不同。

## 15.4 pack属性

在pack属性中可以设置位置，设置参数side=tk.LEFT。方位一共有4个，分别是tk.RIGHT、tk.LEFT、tk.TOP和tk.bottom。如果不想让控件直接靠边，还可以设置距离上边框、下边框、左边框、右边框的距离，padx和pady用于设置距离，*x*表示的是距离左边框和右边框的距离，而*y*表示的是距离上边框和下边框的距离。下面给出一段示例代码。

button=tk.Button(root,text="i'm a button",fg="blue",command=hello\_world)

button.pack(side=tk.LEFT,padx=10,pady=10)

pack属性还有另外一种设置控件位置的方式，即使用anchor方式。使用anchor方式NSEW（北、南、东、西的缩写）分别对应上、下、左、右，使用CENTER对应中间方位。注意，需求使用的是大写字母。下面给出一段示例代码。

checkbutton=Checkbutton(root,text=i,variable=variable[-1])

checkbutton.pack(anchor=W)

此外，如果还可以在pack中设置填充，这里填充的是*X*轴和*Y*轴，分别对应横向和纵向，*X*和*Y*同样需要大写，并且fill属性仅仅在部分控件中可以使用。

checkbutton=Radiobutton(root,text=i,variable=variable,value=counter,indicatoron=False)

checkbutton.pack(fill=X)

expand属性表示是否扩展到空白位置，当值为“yes”时，side选项无效。控件显示在该控件的master控件的中心。如果fill选项是“both”，则填充master控件的剩余空白部分。

## 15.5继承类

我们用一种复杂的类继承方法实现在本章最开始的输出hello\_world的窗口和按钮。

#导入库

import tkinter as tk

#继承类

class Application(tk.Frame):

#构造函数

def \_\_init\_\_(self, master=None):

#调用父类

super().\_\_init\_\_(master)

#调整大小和位置

self.pack()

#调用后面的内容

self.create\_widgets()

#创建窗口

def create\_widgets(self):

#say\_hello按钮

self.say\_hello=tk.Button(self,text="click me to say\nHello World",command=self.say\_hello)

self. say\_hello.pack(side="top")

#退出按钮

self.quit=tk.Button(self, text="退出", fg="blue",command=root.destroy)

self.quit.pack(side="bottom")

#构造功能函数

def say\_hello(self):

print("hello\_world")

#实例化对象

root=tk.Tk()

#设置标题

root.title("selenium 自动化爬虫")

#类的继承

app=Application(master=root)

#程序循环

app.mainloop()

继承类实现的效果如图15-4所示。

图15-4 继承类实现的效果

我们先从构造函数\_\_init\_\_看起。super()函数是用于调用父类(超类)的一个方法，self指的是Application对象本身，master指的是从属关系。在tkinter中，如果一个控件从属于另一个控件，比如，在Label在某个Frame中，那么Frame是Label的master，但我们创建的是顶级窗口，它是没有从属关系的，所以默认master=None。

代码super().\_\_init\_\_(master)调用创建的Application类的父类Frame的\_\_init\_\_函数初始化Application类中的Frame类部分。

self.create\_widgets()用于调用后面定义的createWidgets方法。

下面这行代码可以分开写。

self.say\_hello=tk.Button(self,text="click me to say\nHello World",command=self.say\_hi)

分开书写的代码如下。这样写的优点在于格式整齐且对称，比较方便后期代码的维护。

self.say\_hello=tk.Button(self)

self.say\_hello["text"]="Hello World\n(click me)"

self.say\_hello["command"]=self.say\_hi

效果是一模一样的。

self.quit=tk.Button(self, text="退出", fg="blue",command=root.destroy)

我们使用这一行代码来实现“退出”按钮，root.destroy可以用来关闭界面。

## 15.6 复选框和单选按钮

本节讲述代码多选框和单选框的实现。Tkinter中的一些组件（如Button、Label等）可以设置一个textvariable属性，当这个对象的值重新设置的时候，对应字符会自动变成新的值。常见的值有IntVar和DoubleVar。

我们在真正编写代码的时候，获取变量的值是不需要写出来，只需要根据checkbutton状态做出不同的反应。这里特意显示出来只是为了让读者了解这种变量的使用方法和效果。下面给出一段示例代码。

#导入库

from tkinter import \*

#创建窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#选项列表

test\_dic=["test1","test2","test3","test4"]

#变量列表

variable=[]

#循环创建选项

for i in test\_dic

#在变量列表中添加变量

variable.append(IntVar())

#实现选项，变量的值取变量列表中的最后一个

checkbutton=Checkbutton(root,text=i,variable=variable[-1])

#居中

checkbutton.pack()

#显示变量值并居中

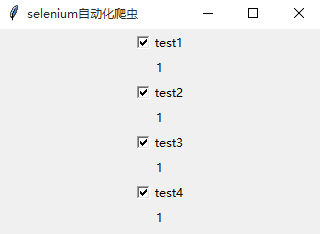
label=Label(root,textvariable=variable[-1])

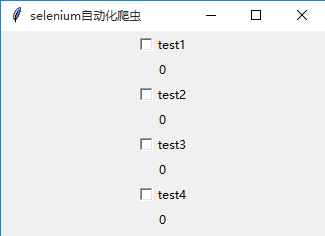
label.pack()

#窗体循环

mainloop()

在这段代码中，首先创建窗体，然后定义选项列表和变量列表。接着使用循环，每一次循环都会向变量列表中添加一个变量。在创建选项的时候，将test\_dic中的元素赋给Checkbutton的文字属性，将变量列表中的最后一个元素赋给变量属性，之后使用label显示变量列表中的最后一个元素，方便我们观察。

单击前后的效果分别如图15-5和图15-6所示。

图15-5 尚未单击 图15-6 单击后

接下来，使用循环来实现单选按钮。

#导入库

from tkinter import \*

#创建窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#定义字典

test\_dic=["test1","test2","test3","test4"]

#定义变量

variable=IntVar()

#计数器

counter=0

#循环创建选项

for i in test\_dic:

#计数器自加

counter+=1

#创建单选框并居中

checkbutton=Radiobutton(root,text=i,variable=variable,value=counter)

checkbutton.pack()

#窗体循环

mainloop()

在代码的开始，按钮常规导入库，创建窗体和标题，定义一个选项字典。这次没有使用变量列表，而使用一个唯一的变量，选择counter计数器来实现每个单选按钮的value属性都不同，最后使用mainloop()实现窗体循环。最终的效果分别如图15-7和图15-8所示。

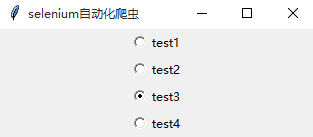
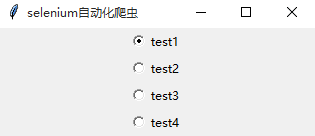


图15-7 单选框1 图15-8 单选框2

使用计数器来实现单选框。因为每个单选框的变量都是variable，只有选中的那一项的value和variable的值相同，所以返回的是1。因为其他选项的value和variable值都不同，所以不会勾选。至此，我们实现了单选框。

如果使用的带圆点的圆形和GUI整体的风格不一致，可以使用这一行代码将圆点改成类似于button类型的单选框，如图15-9所示。

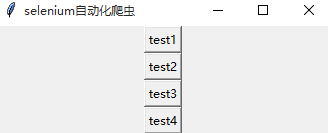
checkbutton=Radiobutton(root,text=i,variable=variable,value=counter,indicatoron=False)

图15-9 风格调换

不同之处在于我们为checkbutton控件添加了indicatoron属性，并将这个属性的值设置为False。

## 15.7 Entry输入框

输入框是用来实现程序和用户进行交互的控件。比如，在登录一个网站的时候，我们需要输入账号和密码，并且有时候密码还会加密显示，如对于密码的每一个字符都用星号代替。

当应用程序或者网站对用户提供个性化服务的时候，为每个用户单独配备一个账户和密码是必不可少的。

通过以下代码，实现一个简单的GUI程序登录框模板。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#创建控件框架

labelframe1= LabelFrame(root,text ="登录框",padx = 5,pady = 5)

labelframe1.pack(padx = 10,pady = 10)

#输入框前面的文字部分

Label1=Label(labelframe1,text="账号")

#使用grid()方式进行布局

Label1.grid(row=0,column=0,padx=5,pady=5)

Label2=Label(labelframe1,text="密码")

Label2.grid(row=1,column=0,padx=5,pady=5)

#账户和密码输入框

entry1=Entry(labelframe1)

#使用grid()方式进行布局

entry1.grid(row=0,column=1,padx=5,pady=5)

entry2=Entry(labelframe1)

entry2.grid(row=1,column=1,padx=5,pady=5)

#窗体循环

mainloop()

前面创建窗体的部分都是一样的。不同的是，这里用到了LabelFrame。这个控件的理解涉及前面讲解的master内容。如果不创建LabelFrame控件，那么接下来通过代码实现的Label标签和Entry标签都从属于顶级窗口；如果加上了LabelFrame控件，就相当于在这些控件和顶级窗口之间又新增了一个中间级别的控件。举一个现实生活中的例子，我们在总经理和业务员之间新增了小组组长，小组组长可以有多位，但总经理只能有一位。

使用的grid()方式进行布局。grid()布局最重要的在于两个参数row和column，分别对应行和列。

Label1.grid(row=0,column=0,padx=5,pady=5)

在这一行代码中，我们将其定位到第0行第0列（Python中的计数值一般都从0开始），因为row=0,column=1的输入框被我们使用grid()布局到Label1标签的右边，所以这两个控件自动呈现左右排列了。

padx和pady参数在前面提到过，分别指的是距离上下页边距和左右页边距的距离。因为登录框并不需要很多，所以这里padx和pady参数均设定为5。实现的简单登录框如图15-10所示。

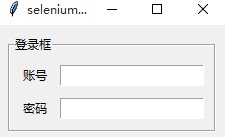


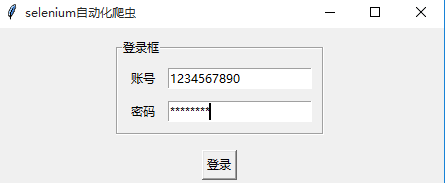
图15-10 简单登录框

但现在仅仅有用于输入的文本框还不够，我们还需要一个按钮来实现登录功能，并且这个按钮最好居中。

接下来，我们还需要对于密码进行加密显示，并且如果这个登录框最好能够实现“记住”账号和密码的功能，也就是说，输入框的默认值。

最终我们要实现的效果应该如图15-11所示。

图15-11 输入的密码加密显示

那么该效果是怎么实现的呢？

首先，使用这一行代码来向entry1（也就是账号对应的输入框）中插入默认字符。

entry1.insert(0,"1234567890")

0表示的是最开始的索引位置。如果要从第二个索引位置开始，可以将0改成1。除此以外，还支持其他设置索引开始编号的位置。

text.insert(1, “[text]”)

text.insert(END, “[text]”)

类似地，Entry控件还有一个删除的方法，在上面的样例代码中并没有使用删除的方法。

这一行代码用于删除第11个索引位置的字符。

text.delete(10)

下面两行代码分别用于删除从第1~4个索引位置的字符和删除所有的字符。

text.delete(0,3)

text.insert(0, END)

然后，在entry2（也就是密码输入控件）中的字符，要用星号来显示。其实，要实现这个功能，只能更改Entry控件的一个属性即可。

entry2=Entry(labelframe1,show="\*")

将Entry控件的show属性设置为星号，这样，当进行输入的时候，会默认使用星号来替代每一个字符。

因为button按钮在LabelFrame框架的外面，从属于顶级窗口，所以第一个参数设置为root。

为了实现按钮的居中，采用pack()方法来进行布局。

button=Button(root,text="登录")

button.pack(padx=5,pady=5)

修改后的代码如下。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#创建控件框架

labelframe1= LabelFrame(root,text ="登录框",padx = 5,pady = 5)

labelframe1.pack(padx = 10,pady = 10)

#输入框前面的文字部分

Label1=Label(labelframe1,text="账号")

#使用grid()方式进行布局

Label1.grid(row=0,column=0,padx=5,pady=5)

Label2=Label(labelframe1,text="密码")

Label2.grid(row=1,column=0,padx=5,pady=5)

#输入框

entry1=Entry(labelframe1)

#使用grid()方式进行布局

entry1.grid(row=0,column=1,padx=5,pady=5)

#设置字符用星号代替

entry2=Entry(labelframe1,show="\*")

entry2.grid(row=1,column=1,padx=5,pady=5)

#在entry1中从第0个索引位置开始插入字符

entry1.insert(0,"1234567890")

#实现按钮

button=Button(root,text="登录")

#使用pack方法进行布局

button.pack(padx=5,pady=5)

#窗体循环

mainloop()

但现在button按钮还是没有任何的功能，我们需要button在进行单击操作以后检验账户和密码是否正确。因此，我们只需要验证用户的输入和我们存储的数据是否匹配即可。如果账户和密码都匹配，返回一个登录成功的状态；否则，返回一个登录失败的状态。

…………

root.title("selenium自动化爬虫")

def check():

ID=entry1.get()

password=entry2.get()

checkID="1234567890"

checkpassword="1234567890"

if ID==checkID and password==checkpassword:

print("you have logged in ")

else:

print("password error")

labelframe1= LabelFrame(root,text ="登录框",padx = 5,pady = 5)

…………..

然后将这个命令对应到button的command属性中，对应的界面如图15-12所示。

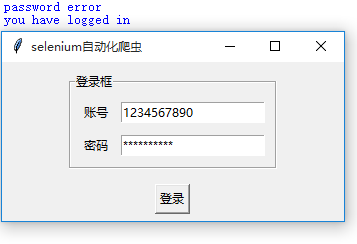
button=Button(root,text="登录",command=check)

图15-12 验证账户和密码的界面

上面两行蓝色的输出分别是读者进行一次错误和一次正确的输入以后输出的内容。

Entry组件提供了完备的解决方法。实际上，如果读者不想看到这个按钮，在输入完成账户和密码后，并且通过Tab键或者Enter键切换完焦点后，要实现自动验证账户和密码的登录，那么只需要添加一个简单的属性即可。(下面的代码因篇幅问题分两行显示，实际上应该在同一行中。)

entry2=Entry

(labelframe1,show="\*",validate="focusout",validatecommand=check,invalidcommand=check2)

validate属性说明了触发validatecommand的条件是focusout，也就是当前控件不再具有焦点了，而当validatecommand属性返回的值是False的时候，会调用check2()函数。

def check():

ID=entry1.get()

password=entry2.get()

checkID="1234567890"

checkpassword="1234567890"

if ID==checkID and password==checkpassword:

print("you have logged in ")

return True

else:

print("password error")

return False

def check2():

print("error")

return True

check2是需要我们自己写的函数。请注意，如果需要使用invalidcommand这个属性，那么前面的check()函数最后还需要返回一个状态码，来告诉控件是否要执行check2()函数，也就是返回True或False。

## 15.8 列表框

列表框是Windows 窗体的一个控件，该控件非常适合用来处理条数比较多的数据，可以显示一个项列表，用户可自主选择列表框中的一项或多项。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#创建列表框

listbox=Listbox(root,selectmode=SINGLE)

listbox.pack()

#创建字典

dic=["test1","test2","test3","test4"]

#循环插入

for i in dic:

listbox.insert(END,i)

#定义删除函数

def delete():

#选出选中的项目

listbox.delete(ACTIVE)

#定义按钮

button=Button(root,text="DELETE",command=delete)

button.pack(padx=5,pady=5)

#窗体循环

mainloop()

列表框控件从属于顶级窗口，第一个参数照旧添加的是root，而对于selectmode选择的是SINGLE模式。Listbox控件有两个方法比较常用，分别是insert()方法和delete()方法。

下面的insert()方法和前面的章节讲解的insert是相同的，用于在已经存在的项目的最后添加一个新的项目。

listbox.insert(END,i)

在上面的示例代码中，listbox.delete(ACTIVE)表示删除当前选中的项目。

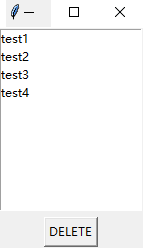
其他的内容都已经讲解过，最终的运行效果如图15-13所示。

图15-13 列表框控件1

列表框的selectmode属性有以下4个可选的内容。

* SINGLE，单选，最多可选一项。
* MULTIPLE，多选，可同时选中多项。
* BROWSE，单选，最多可选一项，和SINGLE在选中方式上存在一定的区别。
* EXTENDED，伪多选，和MULTIPLE在选中方式上存在一定的区别。

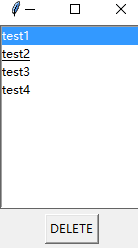
SINGLE和BROWSE两种模式都是单选模式，但不同的是在BROWSE模式中，可以通过拖动改变选项，并且在SINGLE模式中，如果无法使用上下方向键来改变选项，改变的只是游标的位置，例如，图15-14中选项位于test1上，而游标位于test2上。

图15-14 列表框控件2

EXTENDED和MULTIPLE的区别在于，MULTIPLE模式可以直接同时选中多项，但当使用EXTENDED模式的时候，需要按住Ctrl键或Shfit键来进行辅助选择，或者直接拖动鼠标选中多项。

如果不设置，selectmode的默认模式是BROWSE。

当内容条数过多的时候，如此显示就不太方便了，代码窗体需要一个滚动轴。滚动轴的调用实际上还是将滚动轴视为一个控件，然后通过位置布局的不同，让它看起来好像在另一个控件之中。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#创建Frame

labelframe= LabelFrame(root,text='Listbox',padx=5,pady=5)

labelframe.pack(padx=10,pady=10)

#定义scrollbar的属性和窗体，将scrollbar布局到框架中

scrollbar=Scrollbar(labelframe)

scrollbar.pack(side=RIGHT,fill=Y)

#将Y轴控制设置为scrollbar

listbox=Listbox(labelframe,selectmode=SINGLE,yscrollcommand=scrollbar.set)

listbox.pack(side=LEFT,fill=BOTH)

#循环向字典中添加内容，循环插入listbox控件之中

dic=[]

for i in range(0,100):

dic.append("text{}".format(i))

for i in dic:

listbox.insert(END,i)

#负责实现拖动滚动条后窗体内容的改变

scrollbar.config(command=listbox.yview)

#定义删除函数

def delete():

listbox.delete(ACTIVE)

#定义下方的按钮

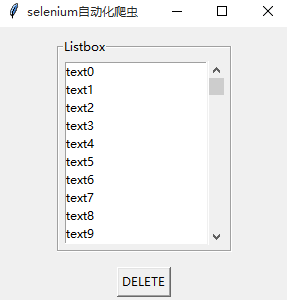
button=Button(root,text="DELETE",command=delete)

button.pack(padx=5,pady=5)

#窗体循环

mainloop()

最终实现的效果如图15-15所示。

图15-15 在列表框中安装的滚动条

在列表框控件上面安装滚动条，读者首先需要设置列表框控件的yscrollbarcommand属性为scrollbar控件的set()方法，然后配置scrollbar.config()的command属性为listbox.yview()方法。

scrollbar=Scrollbar(labelframe)

scrollbar.pack(side=RIGHT,fill=Y)

………

listbox=Listbox(labelframe,selectmode=SINGLE,yscrollcommand=scrollbar.set)

………

scrollbar.config(command=listbox.yview)

这个过程与互相调用很相似，列表框控件需要调用scrollbar控件，而scrollbar控件中的command属性又和listbox相关。

当然，滚动轴不仅在listbox控件中能用，还在一些其他的控件中可以使用。

## 15.9 Text控件

Text控件最初用于显示和处理多行文本，但其丰富的功能导致现在大多被用作文本编辑器。Python附带的IDLE编辑器主体部分就是采用Text控件实现。

这里使用Text控件并在其中插入一张图片。Text控件也可以用于显示图片。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#创建Text控件

text=Text(root,width=36,height=25)

text.pack()

#创建并插入图片

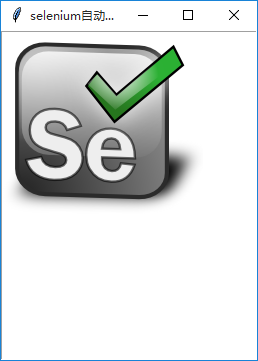
photo=PhotoImage(file="logo.png")

text.image\_create(END,image=photo)

#窗体循环

mainloop()

最终效果如图15-16所示。

图15-16 文本框中的图片

我们只需要定义窗体的宽和高，可以根据需要确定一个适当的宽高比。

PhotoImage()用于实例化一张图片，如果这张图片写入的不是绝对路径，那么必须和代码文件在同一个文件下。然后使用insert方法插入图片。

和Entry控件类似，Text控件也有get()方法。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#定义Text控件

text=Text(root,width=36,height=25)

text.pack()

#插入字符

text.insert(END,"selenium自动化爬虫")

#获取字符

get\_text=text.get("1.0","1.end")

#换行

text.insert(END,"\n")

#插入

text.insert(END,get\_text)

#窗体循环

mainloop()

实现效果如图15-17所示。

图15-17 文本框图片

使用get()方法来获取内容。get()方法的两个参数分别指的是开始位置和结束位置，前后加入双引号，如“1.0”，表示的是从第1行的第0个字符开始。

另外，在Text控件中，可以使用insert()方法插入换行符来进行换行。然后又将获取到的内容重新插入Text控件的最后位置。

Text控件的delete()方法的参数也是类似的，使用两个参数分别表示起始位置和结束后位置。示例代码如下。

text.delete("2.0",END)

当读者想要对文字做一些特殊处理，比如，要增大字体，设置背景色、前景色，可以考虑使用tag标签。tag标签需要两行代码来实现功能，第一行用来说明对哪些字符进行处理，第二行用于说明对他它们进行怎样的处理。在下面的代码，文字的前景色设置为蓝色。

text.tag\_add("tag1","1.0","1.end")

text.tag\_config("tag1",background="blue")

这两行代码的位置并不是很重要，可以根据自己的喜好来决定放置在哪里，甚至可以将其进行颠倒。

如果将标签添加到相同范围的文本中，新创建的标签样式将覆盖旧样式。

Mark通常是嵌入在文本组件的文本中的不可见对象。实际上，Mark指定了字符串之间的位置，并随相应字符移动。INSERT和CURRENT是tkinter预先定义的特殊标记，无法删除它们。

如果在Mark之前插入或删除文本，Mark将继续移动。要删除Mark，需要使用mark.unset()方法。即删除Mark两侧的文本不会删除标记本身，除非使用mark.unset()方法，否则Mark标记会在程序运行的时候始终存在。如果Mark两侧的所有文本都被删除了，那么随文本的删除直到移动到1.0的初始位置。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#Text控件

text=Text(root,width=36,height=2)

text.pack()

#插入字符

text.insert(END,"selenium自动化爬虫")

#设置第1行第11个索引位置为mark1

text.mark\_set('mark1', '1.10')

#利用标记插入字符串

text.insert('mark1', 'MARK')

#利用tag标签将使插入的字符变色

text.tag\_config("tag1",background="BLACK",foreground="WHITE")

text.tag\_add("tag1","1.10","1.14")

#窗体循环

mainloop()

在上面的测试代码中，首先创建了一个Text控件，然后在Text控件中插入了一串字符用于测试。将这一串字符的第1行第11个索引位置设置为mark1。接下来，insert()方法调用mark\_set()设置的标记，在这个位置插入一个内容为MARK的字符串。最后，使用tag标签，将背景色设置为黑色，前景色设置为白色。实现的效果如图15-18

图15-18 标记颜色之后的效果

如果要删除掉mark1，那么可以使用下面这一行代码。

text.mark\_unset('mark1')

这时候就无法使用mark1来插入内容了，尝试使用一个不存在的mark来插入内容，那么会返回如下（tkinter前面有一道下划线，作者会在后面提到\_tkinter）。

\_tkinter.TclError: bad text index "mark1"

## 15.10 Canvas控件

Canvas为Tkinter提供了绘图功能。它提供包括线条、圆圈、图像在内的众多图形组件。在本节中，我们使用convas控件来实现一个简单的手写触摸板（如果读者有触摸板，用鼠标来写字、绘画也是可以的）

Canvas控件并没有提供画一个点的功能，因为这个点多大、多小和怎样适配GUI都是无法确定的，但既然无法直接画一个点，那么我们就画一个圆，使用create\_oval()方法创建椭圆（Canvas控件没有提供直接画圆的方法）。

这里会使用到一点中学数学知识，我们知道椭圆有长轴和短轴，椭圆的标准公式为

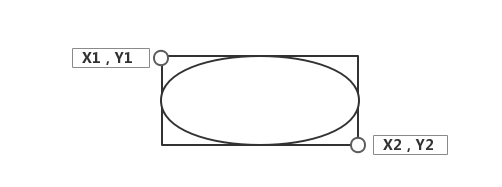
在tkinter中，使用Canvas控件插入一个圆需要两个坐标点。可能有读者会问：“确定一个圆不是需要3个点吗？”不然，Canvas控件不是通过3个点确定圆，而是通过| *X*2-*X*1|和| *Y*2-*Y*1| 分别确定两条轴的长度，通过确定中心坐标，两条轴中较长的作为长轴，较短的作为短轴，当两条轴的长度相等时，即为圆。原理如图15-19所示

图15-19 原理介绍

因此，我们只需要在用户鼠标所在位置旁边构造两个点，然后根据这两个点画出一个圆。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#Text控件

canvas=Canvas(root,width=500,height=300)

canvas.pack()

#根据两个点坐标构造圆

def track(motion):

x1=motion.x-0.1

x2=motion.x+0.1

y1=motion.y-0.1

y2=motion.y+0.1

canvas.create\_oval(x1,y1,x2,y2,fill="black")

#事件绑定

canvas.bind("<B1-Motion>",track)

#窗体循环

mainloop()

这里，首先创建了一个宽度为500，高度为300的画布。然后定义了一个track()函数来使用Canvas.create\_oval()方法创建圆，用黑色填充绘出的圆。

这一行代码用来实现用户鼠标操作和函数的事件绑定。

canvas.bind("<B1-Motion>",track)

Button表示鼠标事件，1表示鼠标左键，2表示鼠标滚轮，3表示鼠标右键，4和5分别表示鼠标向上和向下滚动只在Linux系统中可以使用，因为在Windows和Mac系统中，使用mousewhell表示用户在滚轮上下滚动。

B1表示的是单击鼠标左键，当用户按下鼠标左键并拖动的时候，事件绑定一直处于触发的状态，所以还会不停地执行。

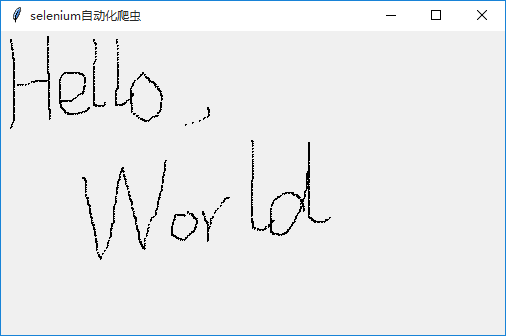
最终实现的画布如下图15-20所示。

图15-20 效果展示

大部分的tkinter控件都可以使用事件绑定，事件绑定的描述具有特定的语法，遵循特定的方式。

<modifier-type-detail>

事件包含在使用双引号包括的尖括号内（部分情况下可以省略尖角括号，例如，匹配一个键盘键），前后的modifier和detail部分都是可选的，而type部分是必须的内容。

type的全部可选值如下。

* Activate，表示控件状态从“未激活”变为“激活”。
* Deactivate，表示控件的状态从“激活”变为“未激活”。
* KeyPress，表示按下键盘按键。
* KeyRelease，表示释放键盘按键
* KeyRelease Visibility，表示释放键盘按键且应用程序至少部分可见。
* ButtonPress，表示按下鼠标按键。
* ButtonRelease，表示释放鼠标按键。
* MouseWheel，表示鼠标滚轮的滚动。
* FocusIn，表示控件获得焦点。
* FocusOut，表示控件失去焦点。
* Map，表示映射控件。
* Unmap，表示取消映射控件。
* Expose，表示窗口或控件部分不再被覆盖。
* Motion，表示鼠标在控件内移动。
* Configure，表示控件的尺寸改变。
* Enter，表示鼠标指针进入控件的时候触发该事件。
* Destroy，表示销毁控件时触发该事件。
* Leave，表示鼠标指针离开控件。

另外KeyPress可以简写为Key，ButtonPress可以简写为Button。

modifier比较常用的值是如下。

* Double，表示后续事件连续两次触发。
* Triple，表示后续事件连续三次触发。
* Alt，表示按下Alt键。
* Any，表示按下任何类型的按键。
* Control，表示按下Ctrl键。
* Lock，表示按下CapsLock键。
* Shift：表示按下Shift键。

detail部分对应键盘上的具体按键，可以直接用按键名代替。

下面举几个例子。

以下代码表示鼠标左键释放之后。

<ButtonRelease-1>

以下代码表示双击鼠标左键。

<Double-Button-1>

以下代码表示连续三次按下H键。

<Triple-Key-H>

## 15.11 Menu控件

现在，我们尝试创建一个简单的界面，要求实现类似于IDLE的功能。也就是说，可以直接看到顶级菜单栏，并且如果移动到菜单栏上面，还会显示出二级菜单。通过以下代码创建菜单栏。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#二级菜单中选项的对应命令

def action():

pass

#顶级菜单

menubar=Menu(root)

#menu1是菜单栏

menu1=Menu(menubar,tearoff=0)

menu1.add\_command(label="新建文件",command=action)

menu1.add\_command(label="打开文件",command=action)

menu1.add\_command(label="打开文件夹",command=action)

menubar.add\_cascade(label="菜单",menu=menu1)

#menu2是编辑栏

menu2=Menu(menubar,tearoff=0)

menu2.add\_command(label="撤销",command=action)

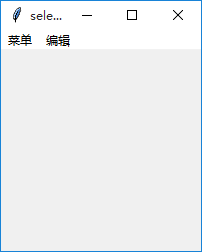
menu2.add\_command(label="重做",command=action)

menubar.add\_cascade(label="编辑",menu=menu2)

#将菜单实例添加到root顶级窗口中

root.config(menu=menubar)

#窗体循环

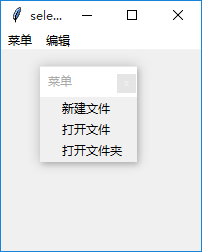
mainloop()

显示的菜单栏如图15-21所示。

图15-21 菜单栏

我们采用嵌套的方式创建菜单。首先，创建的menubar用于实例化一个顶级菜单。然后，在顶级菜单中又实例化了两个menu对象，对应的是顶级菜单中展开的二级菜单的按钮。接下来，在这两个二级菜单中，采用add\_command()方法添加可执行函数的选项。这里简略地用一个action()跳过了命令的实现部分，在实际使用中，根据自己的需要添加相应的内容。弹出的窗口如图15-22所示。

图15-22 菜单栏弹出的窗口

实例化二级菜单的tearoff参数表示这个菜单是否可以独立出来。如果单击最上面的分割线，就会发现这个菜单可以单独拿出来。然而，这个功能一般是不需要的，因此tearoff最好选项设置为0。

除此之外，tkinter还提供了一个OptionMenu，用于实现下拉单选框。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#定义变量

variable=StringVar()

#设置默认值

variable.set("test1")

optionmenu=OptionMenu(root,variable,"test1","test2","test3")

optionmenu.grid(row=0,column=0)

#窗体循环

mainloop()

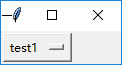
实现的下拉单选框如图15-23所示。

图15-23 下拉单选框

首先，创建一个StringVar。StringVar与IntVar变量的作用类似，不同之处在于StringVar专门针对字符串，而IntVar针对整数。然后，使用set()方法，将StringVar的值设置为test1。对于OptionMenu来说，设置了默认值。如果没有使用set()方法，则没有默认值，直到选中了一个选项之后，选择的菜单上才会出现选择的选项。

OptionMenu()的参数设置并不是很麻烦，第一个参数表示从属于顶级窗口，第二个参数表示设置的变量，第3~5个参数是设置的选项。

## 15.12 spinbox控件

Tk8.4添加了spinbox控件，与之相应的Python版本是Python 3.2。也就是说，spinbox只支持Python 3.2和Tk8.4以上的版本。

spinbox控件和Entry大致上相同。如果不在spinbox控件中加入values参数，spinbox控件几乎可以当作Entry组件来使用，但spinbox控件可以限制用户从一些固定的值中选择一个。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

#选项

option=["test1","test2","test3","test4"]

#创建窗体

spinbox=Spinbox(root,values=option)

spinbox.pack()

#窗体循环

mainloop()

spinbox控件的实现比较容易，唯一需要说明的是传入的values参数可以是一个元组，也可以是一个列表。另外，当用户单击向上和向下的三角形对应的按钮时，还可以使用command参数实现自动调用对应的函数，spinbox控件的效果如图15-24所示。

图15-24 用于限制选择的spinbox控件

当用户单击向上和向下的三角形对应的按钮以后，框中的内容就会从test1到test4进行转换。

## 15.13 messagebox

按照代码的顺序，将这些内容弹出来对话框罗列出来。askokcancel()和askyesno()以及askquestion()在中文状态下弹出的对话框样式是相同的，因此这里只罗列一个。

#导入库

from tkinter.messagebox import \*

askokcancel("title","content")

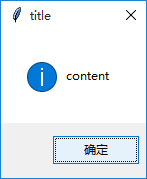
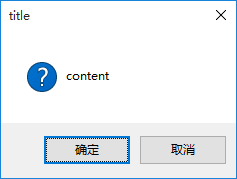
askyesno("title","content")

askquestion("title","content")

askretrycancel("title","content")

showerror("title","content")

showinfo("title","content")

showwarning("title","content")

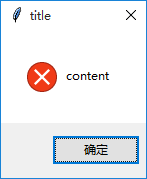
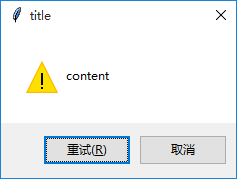
几个弹出的对话框如图15-25（a）~（e）所示。

图15-25 几种弹出的对话框

此外，这些对话框都有3个可选参数。

* icon，用于修改图标，只能指定error、info、question和warning中的一个，不能使用自定义图标。下面这一行确认框将显示错误框的图标。

askokcancel("title","content",icon="error")

* default，用于设置默认的按钮，也就是说,按键会选中的按钮必须是abort、retry、ignore、ok、cancel、no和yes中的一个，不能自定义。下面这一行代码的默认值是取消，按下键键后返回的值是0。

askokcancel("title","content",default="cancel")

* parent，用于将对话框显示在指定的子窗口上。下面这一行代码将弹出的窗口显示在子窗口sub上。

askokcancel("title","content",parent="sub ")

## 15.14 filedialog

filedialog的内容很少，只有askopenfilename()和asksaveasfilename()两个函数。前者用于输出选取的文件的路径，后者用于保存文件。

下面一段代码调用askopenfilename()，该函数输出的是选取的文件的绝对路径。

def choose():

file\_path=askopenfilename()

print(file\_path)

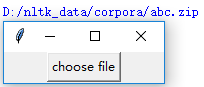
获取到的路径如图15-26所示

图15-26 选取的文件的路径

asksaveasfilename()用来保存文件，并且会弹出一个“另存为”窗口。

两个函数都有一个filetype参数，可以用来设置“保存类型”（如图15-27所示）。向该参数传入的是一个列表或者元组，列表或者数组中的每一个元素是一个二元数组。

file\_path=askopenfilename(filetypes=(('PNG', '.png'), ('JPG', '.jpg'), ('

GIF', '.gif')))

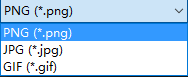


图15-27 文件类型

## 15.15 Message

Message控件和Label控件在大体上是相似的，使用方法和参数也几乎一模一样。然而，Message控件使用起来得心应手，因为Message控件能够自动换行，并调整文本的尺寸使其适应给定的尺寸。下面对比一下Label和Message在输出同一内容时的不同表现。

#导入库

from tkinter import \*

#创建主窗口和标题

root=Tk()

root.title("selenium自动化爬虫")

label=Label(text="in long long long long long long ago")

label.pack()

message=Message(text="in long long long long long long ago")

message.pack()

#窗体循环

mainloop()

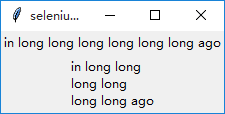
详细区别如图15-28所示。

图15-28 Label与Message控件的详细区别

可以看到，在输出相同文本的条件下，如果没有限定程序的宽和高，Label控件的选择是将顶级窗口撑大，改变其长和宽，而Message控件则会主动换行来适应界面。

那么什么时候使用Label控件？什么时候使用Message控件？什么时候使用Text控件？一般来说，对于这3种控件，需要使用Label控件的情况都可以使用Message控件来代替，如果读者希望使用多种字体来显示文本，那么最好还是搭配使用Text控件和tag标签。

## 15.16 tkinter界面布局

所谓布局，指的是对顶级窗口中每个控件(组件)位置的控制。Tkinter有3个几何布局管理器——pack、grid和place

pack布局，首先将控件添加到顶级窗口中，首先在顶部，然后向下。

pack布局的参数如表15-1所示。

表15-1 pack布局可用的参数

|  |  |  |
| --- | --- | --- |
| 参数 | 说明 | 选项 |
| fill | 设置控件沿水平或垂直方向填充 | X、Y、BOTH 和NONE |
| expand | 表示是否展开，默认不展开。若expand参数为YES时，side选项无效，控件显示在父容器中心的位置；若fill参数是BOTH，则填充父组件的剩余空间 | YES、NO（1、0） |
| side | 设置组件的对齐方式 | LEFT、TOP、RIGHT、BOTTOM |
| ipadx、ipady | 设置*x*方向或*y*方向控件之间的间隙 | 数值，默认0 |
| padx、pady | 设置*x*方向或*y*方向间隙 | 数值，默认0 |
| anchor | anchor选项，当可用空间大于所需大小时，它决定控件在容器中的位置 | N、E、S、W、NW、NE、SW、SE、CENTER |

建议读者使用grid布局。考虑到大多数程序都是矩形，很容易将它们划分为几行和几列组成的网格，然后根据行号和列号将组件放置到网格中。grid布局的参数如表15-2所示。

表15-2 grid布局的参数

|  |  |  |
| --- | --- | --- |
| 参数 | 说明 | 选项 |
| row | 行号 | 数值，从0开始 |
| column | 列号 | 数值，从0开始 |
| sticky | 设置组件在网格中的对齐方式 | N、E、S、W、NW、NE、SW、SE、CENTER |
| rowspan | 控件所跨越的行数 | 数值，默认为1 |
| columnspan | 控件所跨越的列数 | 数值，默认为1 |
| ipadx、ipady、 | 设置*x*方向或*y*方向控件之间的间隙 | 数值，默认0 |
| padx、pady | 设置*x*方向或*y*方向的间隙 | 数值，默认0 |

另外，当使用网格布局时，只需要指定行和列两个参数。

place布局是最灵活的布局，使用坐标来放置控件位置。但是，不建议读者这样使用place布局，因为如果在不同的分辨率下使用这种布局，界面通常是不同的。place布局的参数如表15-3所示。

表15-3 place布局的参数

|  |  |  |
| --- | --- | --- |
| 参数 | 说明 | 选项 |
| anchor | anchor选项，当可用空间大于所需大小时，它决定组件在容器中的位置 | N、E、S、W、NW、NE、SW、SE、CENTER |
| x、y | 左上角的*x*、*y*坐标 | 数值，默认值是0 |
| relx、rely | 组件相对于父容器的*x*、*y*坐标 | 0~1的浮点数 |
| relwidth、relheight | 组件相对于父容器的宽度、高度 | 0~1的浮点数 |
| width、height | 组件的宽度、高度 | 数值 |
| bordermode | 如果设置为INSIDE，组件内部的大小和位置是相对的，不包括边框;如果设置为OUTSIDE，组件的外部大小是相对的，包括边界 | 默认值为INSIDE，可选值有INSIDE和OUTSIDE |

## 15.17 tkinter支持的模块

tkinter强大的功能不仅在于它可以绘制出窗口，还在于它包含大量小控件。以下这些内容都可以在tkinter中使用。

* + tkinter.scrolledtext，表示内置垂直滚动条的文本小部件。
  + tkinter.colorchooser，用于让用户选择颜色的对话框。
  + tkinter.commondialog，此处列出的其他模块中定义的对话框的基类。
  + tkinter.font，用于帮助处理字体的实用程序。
  + tkinter.messagebox，用于实现标准Tk对话框。
  + tkinter.simpledialog，基本对话框和一些其他的便利功能。
  + tkinter.dnd，拖放支持。
  + turtle，用于实现简单的绘图功能。

## 15.18 \_tkinter接口

tkinter提供了Button、Canvas、Label、Text和Frame 等组件。大多数时候，这些组件足够我们编写一个较为完善的GUI程序，但是也有一些额外的模块可用，这些接口位于一个名为\_tkinter的二进制模块中。\_tkinter模块包含Tk的低级接口，开发者一般无法直接使用它，它通常是一个共享库(或DLL)，但在某些情况下可能会与Python解释器静态链接。

## 15.19 tkinter.ttk

tkinter还提供了另一种控件图标——tkinter.ttk。

如果读者是编写一份全新的代码的话

import tkinter as tk

from tkinter import ttk

在一个已经书写完成的GUI上面，要更改控件的整体风格，只需要加入一行代码，整体风格都会改变。

from tkinter import \*

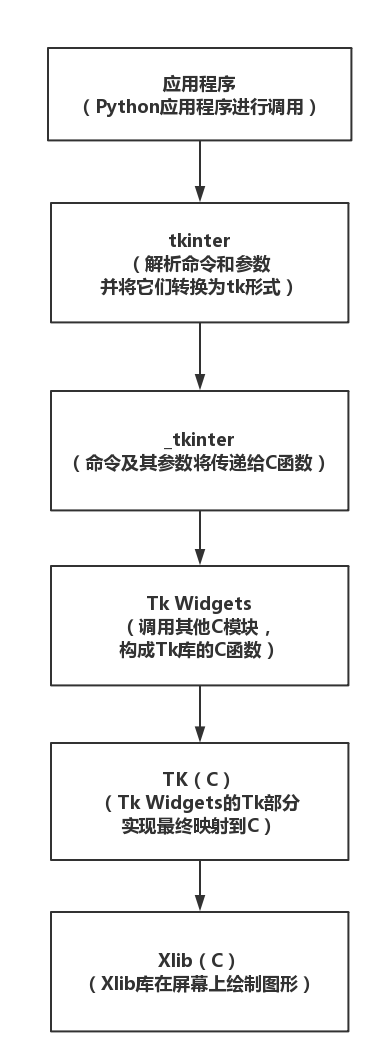
from tkinter.ttk import \*

tkinter.ttk中的大部分控件都和tkinter相同，读者可以直接编写控件，但需要注意的是，tkinter.ttk中的控件是没有indicatoron属性的。

## 15.20 底层实现流程

实际上，tkinter的底层实现并不完全通过Python来实现，tkinter的作用更像是一个中间人，将Python指令转化为TK命令，然后通过TK转化为C语言的指令，再通过Xlib库来实现最终的图像成型。

底层实现流程如图15-32所示。

图15-32 底层实现流程

# 第16章 基于知乎的实战

在本章中，我们将通过一个综合性的案例对前面所学的内容进行融会贯通，并且讨论Selenium项目的进阶用法——结合Selenium与爬虫，从而实现更加强大的功能。

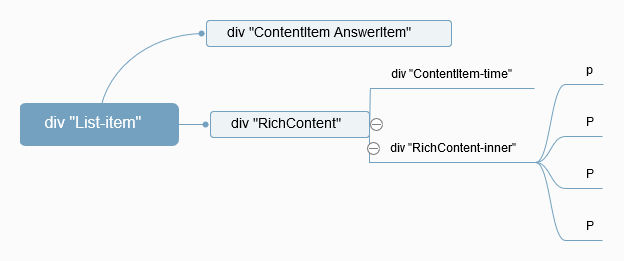
## 16.1 知乎分析

作为目前规模最大、崛起速度最快的问答网站之一，知乎聚集了一大批知识水平高、科学素质强的专业用户。前几年,不断探索的知乎面向公众开放注册权限,这在知识经济的历史上写下了浓墨重彩的一笔。

而在2016年4月的更新更是开放了用户限制，即便不登录，用户也可以浏览一个问题下的所有回答。这刚好适合我们进行内容的爬取。

首先，随意找到一个问题观察一下。PC端的知乎新版页面是动态解析的，这就意味着只有在滑动到页面底部的时候，才会加载全部内容。然而，在源代码中查看时，没有加载的内容都显示为<noscript>标签。

而每一个答案都隐藏在<div class="List-item"></div>中，目录结构大致如图16-1所示。

图16-1 目录结构

与前面爬取今日头条的案例部分类似，我们需要一直模拟键盘向下，来确保当前页面的所有内容已经加载完。可以考虑使用下面这段代码。

def scroll(driver):

driver.execute\_script("""

(function () {

var y = document.body.scrollTop;

var step = 100;

window.scroll(0, y);

function f() {

if (y < document.body.scrollHeight) {

y += step;

window.scroll(0, y);

setTimeout(f, 50);

}

else {

window.scroll(0, y);

document.title += "scroll-done";

}

}

setTimeout(f, 1000);

})();

""")

这段代码和前面使用的代码略微类似，只不过略微高级一些。在学习本书的过程中，只需要了解这段代码，而要进一步探究其中的原理，建议读者学习JavaScript。

而至于正文部分，我们当然可以像前面那样，用找规律的方法来实现。

#正文

/html/body/div[1]/div/main/div/div[2]/div/div/div[2]/div/div/div[2]/div/div[13]/div/div[2]/div[1]/span

/html/body/div[1]/div/main/div/div[2]/div/div/div[2]/div/div/div[2]/div/div[14]/div/div[2]/div[1]/span

/html/body/div[1]/div/main/div/div[2]/div/div/div[2]/div/div/div[2]/div/div[15]/div/div[2]/div[1]/span

#回答者名称

/html/body/div[1]/div/main/div/div[2]/div/div/div[2]/div/div/div[2]/div/div[13]/div/div[1]/div/div/div[1]/span/div/div/a

/html/body/div[1]/div/main/div/div[2]/div/div/div[2]/div/div/div[2]/div/div[14]/div/div[1]/div/div/div[1]/span/div/div/a

规律很容易发现，只是div标签中的数字不同。如果愿意，可以采用上面找规律的方式实现网页抓取，拼接一下字符串即可。

但考虑到知乎的页面是动态加载的，并且所有内容都在同一个页面中，并不像前面的淘宝案例中每页的元素个数固定。我们需要用一些特别的方式来确定上界（不是上确界），比如，设置一个特别大的数，并且使用try…except语句，即便后面没有找到，也不会导致程序退出。

或者提前自己手动下载页面，记录下来答案的最大个数，在写程序的时候直接应用这个数。

但上面两种方法都有一定的局限性，不能完美地应用于知乎的每一个问题。第一种方法对应的反例是可能我们爬取的是一个关注度非常低的问题，只有聊聊几个回答。第二种方法则更不适合爬取任意问题，我们总不可能对于每一个要爬取的页面都提前加载并查看答案的最大个数。因此，我们考虑使用BeautifulSoup 4这一个用于解析页面的库来实现对应功能。

如果读者之前已经对传统的爬虫方式有所涉猎，可以跳过这几个段落，这里介绍本章用到的几个方法。读者没有相应的知识储备也没有关系，新知识的规模和复杂度都是比较小的。

BeautifulSoup 4库的主要内容是一个类，在使用类中的方法之前，需要首先对类进行实例化。

#页面源代码解析

html = driver.page\_source

soup = BeautifulSoup(html,'lxml')

find\_all(name ,attrs ,recursive ,string ,\*\*kwargs )方法用于查找解析后的内容中所有符合条件的标签。示例内容如下。

<title>story</title>

<p class="title"><b>A Long story</b></p>

<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>

<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>

<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>

几个查找语句以及对应的结果如下。

>>> soup.find\_all(id="link2")

>>> [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]

>>> soup.find\_all("a")

>>> [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,

<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,

<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

>>> soup.find\_all("p", "title")

>>> [<p class="title"><b>A Long story</b></p>]

>>> soup.find\_all("title")

>>> [<title>story</title>]

在第一个查找语句中，查找所有包含的id属性为link2的标签，查找结果只有一个符合条件。而在第二个查找语句中，删掉了对于属性的限制，3个a标签都符合条件。在第三个查找语句中，限制属性中必须包含title，因此只有p标签符合条件（title标签中的title是标签名，而不是属性）。在第四个查找语句中，查找title标签。

特别是，下面两种方法是等价的。

#限制只查找第一个标签

>>>soup.find\_all('title', limit=1)

>>> [<title>The Dormouse's story</title>]

>>>soup.find('title')

>>><title>The Dormouse's story</title>

也就是说， find(name,attrs,recursive,string,\*\*kwargs )和finall()方法的用法几乎完全相同，区别在于，find方法只查找第一个标签。

## 16.2 文字部分

我们考虑首先实现文字部分的获取。最开始读者可能想要直接查找包含文字的最下层标签。

<span class="RichText ztext CopyrightRichText-richText" itemprop="text"></span>

但实际上，这层标签类似于find\_element\_css\_selector，因为包含多个class属性，所以用bs4的find()和findAll()方法无论如何也无法提取到这个标签。而itemprop属性又很容易和其他标签查找重复，最终效果并不理想。

在BeautifulSoup 4官网中明确说明了BeautifulSoup 4其实不支持直接查找含多个class属性的div标签——“If you want to search for tags that match two or more CSS classes, you should use a CSS selector”。建议我们使用soup查找CSS选择器的方法查找标签，使用方法如下。

soup.select([css\_selector])

另外一种可行的方式是使用在前面的案例中介绍过的PyQuery库，可是实现像操作jquery一样操作。

在这里我们考虑查找它的上层标签。

#标签

<div class="RichContent-inner"></div>

#查找代码

story.find('div',class\_="RichContent-inner")

获取知乎文字部分的流程如图16-1所示。

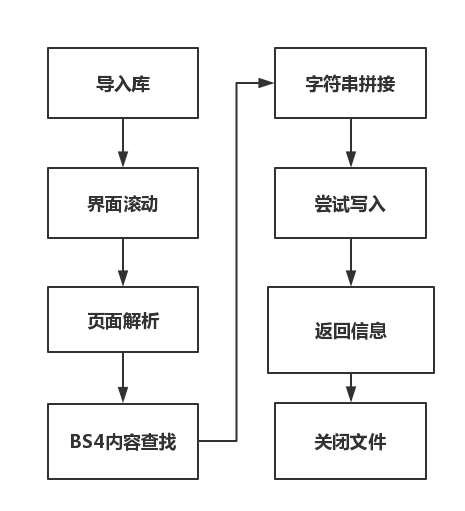


图16-2 获取知乎文字部分的流程

代码如下。

#导入库

from bs4 import BeautifulSoup

from selenium import webdriver

import time

#让界面滚动的函数 --

def scroll(driver):

#功能JS代码

driver.execute\_script("""

(function () {

var y = document.body.scrollTop;

var step = 100;

window.scroll(0, y);

function f() {

if (y < document.body.scrollHeight) {

y += step;

window.scroll(0, y);

setTimeout(f, 50);

}

else {

window.scroll(0, y);

document.title += "scroll-done";

}

}

setTimeout(f, 1000);

})();

""")

#功能函数，用于实现内容查找和存储

def func():

for story in storys :

nameLabel = story.find('meta',itemprop="name")

name=nameLabel["content"]

storyText=story.find('div',class\_="RichContent-inner")

#拼接字符串并写入

try:

string=name+'--------'+storyText.text+'\n'

f.write(string)

print('answer By '+str(name)+' has been finished')

except Exception:

print('Something is wrong while writing to txt')

#程序的入口点

if \_\_name\_\_ == '\_\_main\_\_':

#定义页面，打开浏览器

url='https://www.zhihu.com/question/54732849'

driver=webdriver.Chrome()

driver.get(url)

scroll(driver)

time.sleep(5)

#页面解析

html=driver.page\_source

soup=BeautifulSoup(html,'lxml')

storys=soup.find\_all('div',class\_="List-item")

#存储文件

f=open('tmp.txt','w+')

func()

f.close()

print('That is all')

运行结果如下。

张文--------带宽不行吧 手机和电脑连接的速度达不到内存的速度吧

知乎用户--------其实现在恰恰相反，将手机的主要计算需求转移到云服务器。换言之，手机这种客户端的计算能力、掌握的数据目前实在无法与服务器相比。终端只提供数据。个人理解。

无情来小儿--------别逗了，内存是非常高效的寄存器，而手机存储还不如U盘呢。

airnavy--------用电脑处理手机的问题，类似的东西很早就有人做了。服务器在云端处理，本地只有瘦客户端显示结果，这个就相当于用手机作为瘦客户端。

在功能函数func()中，我们使用了try…except语句。虽然从当前问题来看，这个语句并不是必需的，但其实从实用性和普适性的角度上来说，这个语句保证了程序在单次获取失败的情况下，不至于中断整段代码的运行。

另外，对于下面这两行代码，需要说明一下。

scroll(driver)

time.sleep(5)

之所以在下拉加载函数的后面紧跟着time.sleep方法，是因为网页的加载不可能总能够在瞬间完成，可能一部分的内容和页面显示的内容已经加载结束，但实际上，还有一部分功能代码尚未实现，如JavaScript脚本。又或者页面虽然已经下拉到这个位置，但该位置的内容还未完全加载。

为了安全起见，在面对答案个数更加多的问题时，应该适当将time.sleep()方法暂停的时间延长一些。

通过下面两行代码，我们对页面的内容进行了两次解析。首先通过Selenium获取页面源代码，然后使用BeautifulSoup 4库中的类进行对象实例化。

html=driver.page\_source

soup=BeautifulSoup(html,'lxml')

此外，如果读者想要进一步改进，用于存储大规模数据，还可以考虑结合前面介绍的存储为CSV文件的方式。

import csv

或者采用数据库的方式。

import pymysql

但实际上，考虑到知乎单个问题的存储规模相对较小，使用这些方式对于数据结构以及压缩存储空间的作用并不是很明显，反倒不如直接创建txt文件来存储。

## 16.3 图片部分

和上面爬取文字部分所遇到的困难相似，爬取图中的过程中最大的问题在于如何判断页面是否已经滑动到底部。如果无法通过代码判断，我们只能硬性等待，将以下代码中time.sleep()的参数值设置得略微大一些。

scroll(driver)

time.sleep(5)

titles = soup.find("title").text.replace('\n', '').replace('?', '').split()

刚才获取的文字部分可以全部保存在一个txt文件中，而保存图片则需要专门创建一个全新的文件。

#获取问题的名称

titles = soup.find("title").text.replace('\n', '').replace('?', '').split()

title = titles[0]

#拼接路径

dirpath = os.getcwd() + "\\" + title+ "\\"

#检查是否存在

if not os.path.exists(dirpath):

os.makedirs(dirpath)

首先，使用os.getcwd()方法获取当前路径，并且通过拼接字符串的方式拼接出以爬取的问题命名的文件夹。然后，判断我们想要使用件夹是否存在，如果没有，则创建该文件夹。

可能读者对于这里为何要在find()方法查找的内容中获取第一个标签疑惑。find()确实查找到第一个标签，但在这里，我们获取到的第一个标签中的text却是列表，内容如下所示。

['挪威有哪些震撼人心的自然景色值得一去的？', '-', '知乎']

这便是获取的是titles[0]而不是输出title的原因。另外，于获取获取到的内容，我们使用了两次replace()方法和一次split()，用来替换无意义的内容（split函数不加参数表示为分割空格）。

如果读者使用过知乎，就会发现知乎的问题页面分为两种：一种是推荐回答页面，另一种则是所有回答页面。推荐回答页面从像用户推送的回答的始，仅仅只有寥寥数个答案，并且在页面的底部，存在图16-3所示的跳转按钮。

16-3 跳转按钮

所有回答页面的回答随着用户的下拉动态加载，在些页面里我们只需要一直下拉确保到底即可。

而我们想要获取一个问题下面的所有图片，因此需要从推荐回答页面跳转到所有回答页面。两者的不同从页面源代码中即可看出来。

#page1

https://www.zhihu.com/question/36950102/answer/89597362

#page2

https://www.zhihu.com/question/36950102

我们通过审查可以轻松获取到跳转按钮的CSS选择器路径，而如果读者多尝试几个页面，就会发现下面的CSS选择器路径具有普适性。

div.Card:nth-child(4) > a:nth-child(1)

即便在推荐回答页面进行过下拉操作，一旦跳转到所有回答页面，需要进行一次新的下拉操作，因此这里while True循环。

可以通过下面的代码实现从推荐回答页面到问题页面的跳转。

try:

#首先加载出全部内容，判断是否页面中存在“更多”这一个按钮

while True:

#这里需要注意的是，Selenium 2是不支持类名之中有空格的

try:

self.scroll(driver)

time.sleep(30)

more = driver.find\_element\_by\_css\_selector("button.Button.QuestionMainAction")

actions = ActionChains(driver)

actions.move\_to\_element(more)

actions.click(more)

actions.perform()

#如果我们在页面中找到了“更多”按钮，那么就单击“更多”按钮，然后等两秒

# more.click()

except NoSuchElementException as e:

break

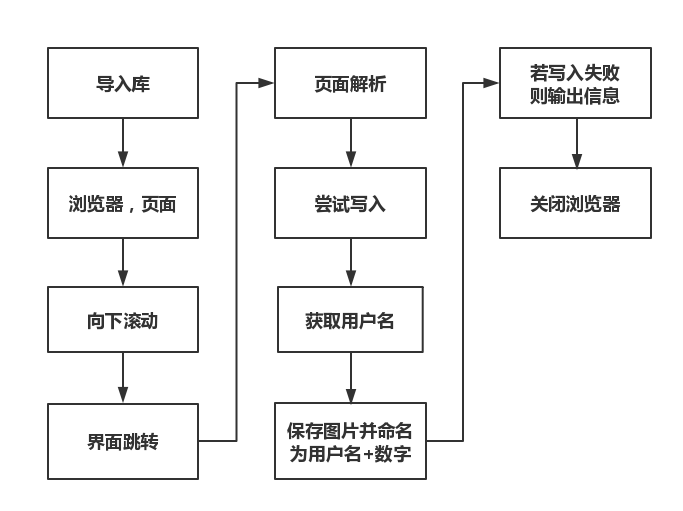
整体思路如图16-4所示。

图16-4 获取图片部分的思路

最终实现代码如下。

#导入库

import requests

from requests.exceptions import MissingSchema

from bs4 import BeautifulSoup

from selenium import webdriver

from selenium.common.exceptions import NoSuchElementException

from selenium.webdriver.common.action\_chains import ActionChains

import os

import time

#定义为类

class func:

#没有需要初始化的变量，跳过

def \_\_init\_\_(self):

pass

#让界面滚动到底部的函数

def scroll(self,driver):

driver.execute\_script("""

(function () {

var y = document.body.scrollTop;

var step = 100;

window.scroll(0, y);

function f() {

if (y < document.body.scrollHeight) {

y += step;

window.scroll(0, y);

setTimeout(f, 50);

}

else {

window.scroll(0, y);

document.title += "scroll-done";

}

}

setTimeout(f, 1000);

})();

""")

#主体功能部分

def beginSpider(self,url):

#打开浏览器，跳转到相应页面

driver = webdriver.Chrome()

driver.get(url)

try:

#首先加载出全部的内容，判断是否页面中存在“更多”这一个按钮

while True:

#这里需要注意的是：Selenium 2 是不支持类名之中有空格的

try:

self.scroll(driver)

#设置较长的等待时间是为了尽量保证页面加载完

#但在这个示例中30s仍然不太够

time.sleep(30)

#这里使用反斜杠（\）将一段代码分为两行

more = driver.find\_element\_by\_css\_selector \

("div.Card:nth-child(4) > a:nth-child(1)")

actions=ActionChains(driver)

actions.move\_to\_element(more)

actions.click(more)

actions.perform()

#当检测不到跳转按钮的时候

except NoSuchElementException as e:

break

#页面解析

soup = BeautifulSoup(driver.page\_source,"html.parser")

# 2. 对soup进行操作，获取出title，和包含内容的列表items

#获取标题

titles = soup.find("title").text.replace('\n', '').replace('?', '').split()

title = titles[0]

print(title)

#如果当前目录下没有title命名的文件夹，则创建一个

dirpath = os.getcwd() + "\\" + title + "\\"

if not os.path.exists(dirpath):

os.makedirs(dirpath)

#将获取到的所有内容另外为items

items = soup.find\_all("div", class\_="List-item")

nimingCount = 0

for item in items:

#分类讨论是否为匿名用户

userName = item.find('img', class\_="Avatar AuthorInfo-avatar").get("alt")

if "匿名" in userName:

userName = "匿名用户\_" + str(nimingCount)

nimingCount += 1

count = 0 #一个用户下有多幅照片的

images = item.find\_all('img',class\_ = "origin\_image zh-lightbox-thumb lazy")

for image in images:

#保存图片

imageSrc = image.get("src")

picName = dirpath + userName + '\_' + str(count) + ".jpg"

count += 1

#写入

try:

imageData = requests.get(imageSrc, stream=True).content

try:

#以二进制方式创建

with open(picName, 'wb') as jpg:

jpg.write(imageData)

except IOError as e:

#写入失败情况下的输出

print(userName + "的一张图片写入错误")

except MissingSchema as e:

#图片获取失败

print(userName + "的一张图片获取失败")

print("地址为：" + imageSrc)

finally:

#关闭浏览器

driver.quit()

#程序入口点

if \_\_name\_\_ == '\_\_main\_\_':

f = func()

f.beginSpider("https://www.zhihu.com/question/36950102/answer/89597362")

如果一切无误，读者在短时间的等待之后应该可以看到文件夹中大量的照片。

下面我们回顾一下刚才的代码，在刚才的代码中，我们编写的时候远远要比文字部分要谨慎得多，讨论了大量失败的情况。

这是因为文字只涉及到对已经存在的页面上的内容的获取和存储。而对于图片，我们的操作却是获取图片的路径，然后根据路径下载图片，这很容易处于网络原因或者服务器的传输问题而导致无法下载中间的某一张图片。

在进行文件的获取和写入的时候，使用到了下面的代码。

requests.get(imageSrc, stream=True).content

request.get()方法用于下载指定网址的内容，而其返回值包括以下几个部分。

#返回headers中的编码解析的结果（文字）

r.text

#返回二进制结果

r.content

#返回JSON格式，可能抛出异常

r.json()

#状态码

r.status\_code

此外需要注意的就是异常返回，MissingSchema是Requests库中涵盖了比较常见的问题，例如，协议不匹配，页面类型不匹配等问题，基本可以避免图片下载过程中各种问题。

from requests.exceptions import MissingSchema

而NoSuchElementException则用来检查当前页面是否仍然为推荐答案页面，不然会一直进行循环。

from selenium.common.exceptions import NoSuchElementException

## 16.4 建议

这里给出使用Selenium的过程中的几条建议，希望能够帮助读者进一步改进自己的程序，使其朝着良好的方向发展。

### 16.4.1 从干净的状态开始

在进行自动化技术实践的时候最好从干净的已知状态开始每个测试。理想情况下，最好为每个测试启动一个新的虚拟机。如果启动新的虚拟机是不切实际的，至少为每个测试启动一个新的WebDriver。这主要是因为浏览器以及Webdriver在使用的过程中会产生一些临时文件，一般情况下虽然不会出问题，但谁知道会因为哪些临时文件就会出问题呢？

举一个很有可能的例子，如果我们通过Webdriver控制浏览器访问一个已经访问过的网站，那么浏览器会优先从内存以及缓存数据中获取网站内容。这时候，即便页面做了更改，我们测试的依然是原先的页面，从而造成不必要的麻烦。

### 16.4.2 测试的隔离

值得一提的是务必需要确保测试彼此隔离。不要尝试共享测试数据。假设每个测试在选择执行之前在数据库中查询一个有效的顺序。如果两个测试选择相同的顺序，可能会出现意想不到的结果。

清除应用程序中可能被另一个测试(如无效的订单记录)更改过的数据将会导致不可以预估的风险。

最好的方法为每个测试创建一个新的WebDriver实例。这有助于确保隔离测试并简化并行化。假设有一个内容管理系统，你可以使用该系统创建一些自定义内容，然后在发布后作为模块显示在你的网站上，并且可能需要一些时间才能在CMS和应用程序之间进行同步。

测试模块的错误方法是在一次测试中创建和发布内容，然后在另一个测试中检查模块。这可能会出错，因为在发布后内容可能无法立即用于其他测试。

但读者可以创建在受影响的测试中打开和关闭的存储内容的容器，并使用它来验证模块。

### 16.4.3 Anaconda

Anaconda是一个开源的Python发行版本，在数据以及科学研究领域，Anaconda甚至比Python本身更加流行。

不少人在处理数据处理领域以及自动化方面的都为环境问题时头疼不已，那些提示总是告诉你在安装这个工具时需要安装其他一些来源不明的工具。而包含了conda和 Python的Anaconda可以用于在同一个机器上安装不同版本的软件包及其依赖，并能够在不同的环境之间切换，解决各种依赖关系。这也是么Anaconda更受欢迎的原因。Anaconda另外还包含了160多个科学包及其依赖项。

Anacond包括但不限于以下功能：

* 数据整合、探测、挖掘、预测。
* 对结构化/非结构化数据进行类型转换。
* NumPy+SciPy的底层组合。
* 兼容Winodws系统、Mac OS和Linux系统。

如果可以，强烈建议读者在学习本书后了解Anaconda的使用。

### 16.4.4 报告

编写报告是自动化测试人员的基本功，而良好的报告标准将会让每个人受益，不仅自己的技能和专业,但更重要的是,保存时间为开发人员检查缺陷的复现,并且在公司内部形成的统一的标准可以作为测试过程的操作标准。

通常，常见的缺陷类型可以大致分为3类——接口问题、功能问题和崩溃问题。对于界面问题，应该上传问题截图，用红框标注。对于功能问题，需要上传操作视频(录制视频时，不进行任何无关操作，重现当前问题)。对于崩溃问题，需要上传到视频和崩溃日志。

首先，附件的名称应与缺陷标题一致，不应随意书写，如11111、aaaa或其他冗长的字符名称。其次，在提交缺陷报告时，指出测试操作中使用的移动设备和模型、操作系统版本、测试环境、网络类型、浏览器等。再次，先决条件需要清楚地表明提交Bug发生的条件，如4G网络正常，正常的账户登录。

复制步骤需要通过不同的步骤来描述错误问题，每个步骤都有一个数字序号。对于特定数据的问题，提供特定的测试数据。在步骤之间使用组合链接->，步骤描述中的模块不加引号。同时，对步骤的描述不应该太长，语言应该简洁。

另外，根据需要，可以提供缺陷定位线索或错误日志输出，缩短开发人员重新定位的时间。

### 16.4.5 IP代理

如果你碰到一些特殊情况，无法使用本地的IP地址的释放，那么你可能会需要使用这些代码。

#导入库

from selenium import webdriver

from selenium.webdriver.common.proxy import Proxy

from selenium.webdriver.common.proxy import ProxyType

from selenium.webdriver.common.desired\_capabilities import DesiredCapabilities

#声明代理

proxy = Proxy({

'proxyType': ProxyType.MANUAL,

#代理ip和端口

'httpProxy': 'ip:port'})

#配置对象DesiredCapabilities

dc=DesiredCapabilities.PHANTOMJS.copy()

#把代理ip加入配置对象中

proxy.add\_to\_capabilities(dc)

dr=webdriver.PhantomJS(desired\_capabilities=dc)

在这段代码中，首先创建一个配置对象的副本，然后添加已经提前声明的字典格式的代理，启动浏览器的时候以相应的配置文件启动。

另外一种为自己的爬虫添加保证的方式是修改请求头。

from selenium import webdriver

#设置成中文

options = webdriver.ChromeOptions()

#添加头部

options.add\_argument('lang=zh\_CN.UTF-8')

options.add\_argument('user-agent="Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.103 Safari/537.36"')

#以相应方式启动

driver = webdriver.Chrome(chrome\_options=options)

driver.get("https://httpbin.org/user-agent")

#退出

driver.quit()

### 16.4.6 过滤请求

搭配上了代理的Selenium是一个几乎无所不能的爬虫，但“成也萧何，败也萧何”。Selenium模拟浏览器的行为固然为我们爬取页面内容提供了很大的便利，但也带来了最大的缺点—— 速度慢。因为每次Selenium打开一个页面都要等待页面全部内容加载完，包括加载页面上的图片资源、JavaScript文件和CSS文件。更糟糕的是，对于一些服务器性能特别差的网站，如果你不考虑过滤部分内容，那么爬虫只会一直等待浏览器加载这些毫无用处的资源，直到超时，那么如何过滤不需要的内容呢？方法如下。

* 通过修改hosts文件将服务器性能特别差的网站的地址重定向到本地站点。缺点是需要修改系统文件，不利于应用部署，不能动态添加需要过滤的请求。
* 禁用浏览器的 JavaScript 功能，例如，Chrome可以在打开时设置“--disable-javascript”参数，缺点是可能导致一些借助JavaScript来实现复杂功能的页面无法使用。
* 使用像BrowserMob之类的代理服务器代理拦截不需要的请求。
* 通过Fiddler等代理软件的AutoResponder及类似功能拦截请求。

## 16.5 一些注意事项

了解下面这些注意事项，有利于读者更好地改善自己的程序，避免一些不必要的错误。

### 16.5.1 Captchas

CAPTCHA（Completely Automated Public Turing Test to Tell Computers and Humans Apart，全自动区分计算机和人类的图灵测试）的设计初衷便是阻止计算机的不合理运用。

对于这种防止自动化的方式，目前还没有很完美的解决策略，所以尽量不要尝试。目前可以通过几个主要的策略来绕过验证码检查。

* 在测试环境中禁用CAPTCHA。
* 添加一个钩子以允许测试绕过验证码。
* 通过预先登录等方式跳过CAPTCHA检查环节。

### 16.5.2 性能测试

通常不建议使用Selenium和WebDriver进行性能测试。不是因为它们功能弱小，而是因为它它们对工作没有优化，你不可能得到好的结果。

在用户环境中进行性能测试看起来很理想，很实际上并非如此。一套WebDriver测试要面对许多外部和内部的脆弱性，这超出了我们的控制范围。

通常在以下方面轻微的变化会导致测试结果的变化。我们很难区分网站的性能与外部资源的性能之间的差异，也很难判断在浏览器中使用WebDriver会带来什么性能损失，尤其是在注入脚本的情况下。

* 浏览器的启动速度。
* HTTP服务器的速度。
* 托管JavaScript或CSS的第三方服务器的响应。
* WebDriver实现本身的缺陷检测。

### 16.5.3 六度分割理论及其实际使用

另一个潜在的不安全因素是“设计初衷”——同时进行功能和性能测试。然而，功能测试和性能测试有相反的初衷。要测试功能，测试人员可能需要耐心等待加载，但是这会影响性能测试结果，反之亦然。

要提高网站的性能，需要独立于环境差异分析整体性能，对识别糟糕的代码进行实验，分析单个资源(如CSS或JavaScript)的性能，以便了解需要改进什么。已有一些专门的性能测试工具可以完成这项工作，它们提供报告和分析，甚至可以提出改进建议。

数学领域的猜一个想名为六度分割理论（Six Degrees of Separation）。这个猜想的内容大概如下：你和任何一个陌生人之间所间隔的人不会超过6个，也就是说，最多通过5个中间人你就能够认识任何一个陌生人。

使用WebDriver通过遍历页面上的所有链接进行大规模数据采集的原理便是六度分割理论。如果追求速度，那么它肯定不是最理想的工具。WebDriver需要启动时间。

而与使用WebDriver不同的是，通过执行curl命令或使用像BeautifulSoup和Scrapy这样的库来节省大量时间，因为这些方法不依赖于创建浏览器和导航到页面。仅仅加载需要的部分，就可以节省大量时间，但这可能导致爬虫被服务器所限制。

具体使用哪种方式抑或是结合使用需要读者根据实际情况斟酌。

### 16.5.4 HTTP状态码

Selenium WebDriver是一种完全不同的浏览器自动化方法，它更像一个用户，这是用WebDriver编写测试的方式表现出来的。在自动化功能测试中，检查状态码是判断测试结构的一个特别重要的细节。

浏览器将始终表示HTTP状态码，如404或500错误页面。当遇到这些错误页面之一时，“快速失败”的一个简单方法是在每次加载页面之后检查可靠的页面标题或内容(如标签)。

如果正在使用页面对象模型，可以在类构造函数中包含这个检查，或者在页面加载预期的类似点中包含这个检查。有时候，HTTP状态码可能会出现在浏览器的错误页面中，读者可以使用WebDriver读取这些文字来确定返回的状态并改进调试输出。404页面错误如图16-5所示。

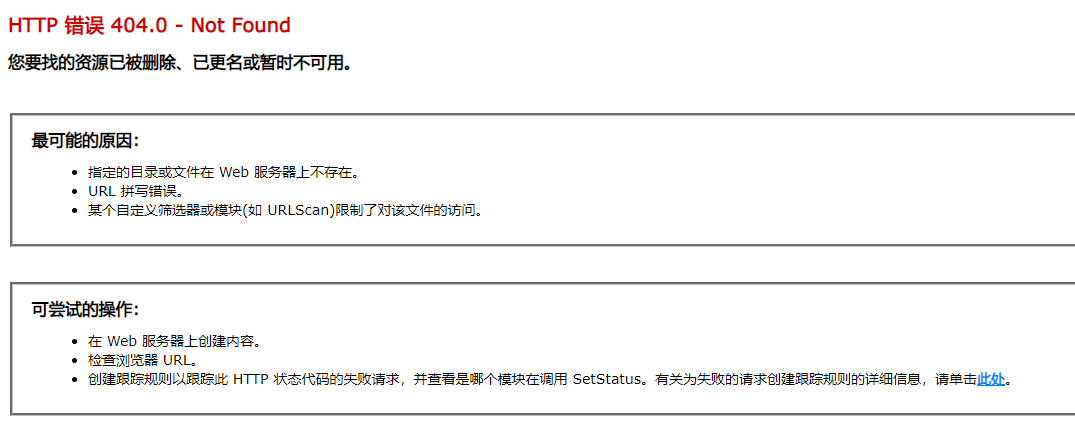


图16-5 页面不存在的404错误

但更常见的情况是，错误页面可能是一张图片或什么都没有，这通常需要我们写try…except语句,并且根据跳转到的页面的内容判断返回的状态码。比如，我们预先记录错误页面的特定信息，通过判断是否存在这些文字来知道是否跳转到了一个不存在的页面，从而了解当前的页面采集是否成功。无法打开页面的404错误如图16-6所示。

图16-6 无法打开页面的404错误

另一种捕获HTTP状态码的高级解决方案是使用代理复制Selenium RC的行为。WebDriver API提供了为浏览器设置代理的功能，有许多代理可以通过编程方式操作发送到Web服务器和接收到的请求的内容。使用代理可以让读者决定如何响应重定向的HTTP状态码。另外，因为并不是每个浏览器都能让WebDriver使用HTTP状态码，所以选择使用代理可以让你有一个适用于所有浏览器的解决方案。

## 16.6 常见问题

这里对于Selenium项目的大部分常见问题进行了汇总，希望有助于读者使用Selenium。

### 16.6.1 元素定位失败

1.出现selenium.common.exceptions.NoSuchElementException

selenium.common.exceptions.NoSuchElementException这种报错是一种常见的错误，其产生的原因可能多种多样，需要读者对自己的代码进行多方面排查。

一种可能的情况是读者想要定位的元素属性已经发生变化。这种错误在一些邮箱类的网站常常见到，属性id会随着每次登录发生变化。理论上，通过id是无法精确定位到该元素的，因此建议通过Xpath相对路径的方法查找元素。

driver.findElement\_by\_id("temp\_82\_82")

此外，可能会无法定位到位于Frame/Iframe（Frame与Iframe两者可以实现的功能几乎完全相同，不过Iframe比Frame具有更多的灵活性）中的元素。为了解决这类问题，我们首先需要理解这些框架的实质，在网页中嵌入一个Frame框架，实际上是嵌入了另一个网页，因此需要首先定位到框架中。

driver=webdriver.Chrome()

driver.get(r'<http://www.126.com/>')

driver.switch\_to\_frame('x-URS-iframe')

#需要首先跳转到iframe框架

tmp=driver.find\_element\_by\_name('email')

tmp.clear()

#如果iframe没有name或id，则可以通过下面的方式定位

#先定位到iframe

ele\_frame= driver.find\_element\_by\_class\_name('APP-editor-iframe')

#再将定位对象传给switch\_to\_frame()方法

driver.switch\_to\_frame(ele\_frame)

最典型的的页面像网易邮箱，如果不切换到Frame中，用以下方式定位登录框都会失败。

* 跳出当前框架，切换到另一框架。

switch\_to.parent\_content()

* 跳出框架，返回最顶层页面。

switch\_to.default\_content()

另一方面，如果程序的中间间隔时间太短，页面还未完全加载完成，而程序已经判定可以进行下一步，就会造成单击速度过快，而页面尚未完全加载出来。

#设置隐式等待

dr = webdriver.Chrome()

dr.implicitly\_wait(30)

#强制暂停

time.sleep()

另外几种小错误如下。

* 元素属性输入错误，如 “password”写成了“psssword”。
* 查找的元素标签不唯一，当通过Class Name来查找元素的时候容易出现这种情况，页面中可能有多个共用的元素。
* 元素为隐藏元素，不可见，却被用户操作。
* 元素虽然可见，但是跳出了页面，显示在加载不到的地方。

2. 元素无法定位的问题

如果读者碰到了一个只读元素无法定位，例如，网页中常见的指定输入限制我们进行更改，这确实相当棘手。

<form action="form\_action.asp" method="get">

Name:<input type="text" name="email" />

Country:<input type="text" name="country" value="China" readonly="readonly" />

<input type="submit" value="Submit" />

</form>

可以考虑使用下面的JavaScript代码删元素的只读属性除，再进行操作。

js\_start="document.getElementById('queryStartTime').removeAttribute('id');

document.getElementById('queryStartTime').removeAttribute('readonly');"

driver.execute\_script(js\_start)

js\_start\_value="document.getElementById('queryStartTime').value='2016-11-05'"

driver.execute\_script(js\_start\_value)

js\_end="document.getElementById('queryEndTime').removeAttribute('id');

document.getElementById('queryEndTime').removeAttribute('readonly');"

driver.execute\_script(js\_end)

js\_end\_value="document.getElementById('queryEndTime').value='2016-11-05'"

driver.execute\_script(js\_end\_value)

3.页面连接失败的问题

如果页面连接失败，可以会报告以下错误。

selenium.common.exceptions.StaleElementReferenceException:

Message: stale element reference: element is not attached to the page document

一般出现该错的原因是由于返回和刷新页面，例如，我们在增加一个内容的时候，如果单击save，通常会返回列表页面，但是列表中没有实时的新数据。如果在列表中查找数据并查找新创建的数据，就会报告错误。

常见的解决方法是添加完成并保存后，首先增加一步刷新页面的操作。

driver.refresh()

然后再定位新创建的数据。

4. 无法直接定位选项的问题

要选中下拉列表中的选项，需要两个步骤。首先定位到下拉菜单，然后对其中的选项进行定位。在下面的代码中，首先定位下拉列表，然后定位其中的选项，效果如图16-7所示。

如果直接定位选项，因为它处于不可见状态，这样的定位是不合理的，很容易被服务器判定为非正常访问。

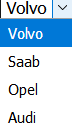


图16-7 定位下拉列表中的选项

实现代码如下。

#定位下拉列表

drop\_down = driver.find\_element\_by\_css\_selector("body > select:nth-child(1) ")

drop\_down.click()

#对下拉列表中的选项进行选择

select=find\_element\_css\_selector ("body > select:nth-child(1) > option:nth-child(2) ")

select.click()

### 16.6.2 Webdriver调用失败

出现以下错误主要是因为hosts文件出了问题。

Exception in thread "main" org.openqa.selenium.UnsupportedCommandException: Bad request

为了修正这个错误，需要在hosts文件中加入以下内容。

127.0.0.1 localhost

但加入的位置还是有要求的，必须放在hosts文件的最开始。如果读者使用一些修改hosts文件的方式来加快联网速度，那么需要在设置后始终将该内容放置到最开始，并且在后面的行中也不能出现同样的 127.0.0.1 localhost（只要有就会出错）。

下面这个错误说明FireFox浏览器并未安装在默认位置（Chrome浏览器强制指定安装位置，一般不会出现该种错误）。

Exception in thread "main" org.openqa.selenium.WebDriverException:

Cannot find firefox binary in PATH. Make sure firefox is installed.

这个错误有两种解决方法。

* 将FireFox的安装路径加入环境变量中
* 在每次启动FireFox浏览器的时候指定启动路径。

出现下面这种问题一般是因为读者在使用的时候，失手关闭了Selenium打开的命令提示行或者打开的浏览器，导致连接关闭。

ConnectionResetError: [WinError 10054] 远程主机强迫关闭了一个现有的连接。

# 第17章 基于新浪微博的实战

在本章中，将采用新浪微博作为案例，爬取微博的部分信息

## 17.1 新浪微博分析

在前面展示的案例中，我们都采用直接从官网入手的方法，但对于微博的爬取，考虑到手机版新浪微博的内容相对于计算机版干扰的内容更加少，无用信息量更小，因此我们考虑采用手机端的新浪微博页面进行内容的获取。

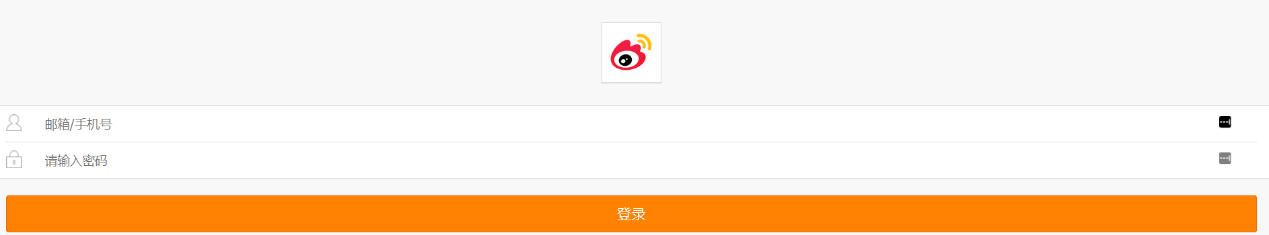
要在计算机上查看手机端的新浪微博页面，可以按F12键，打开审查元素，单击响应式设计模式，即可从拉长变形的页面（如图17-1所示）变换到手机端页面（如图17-2所示）

图17-1 拉长变形的手机端新浪微博网页

图17-2 调整后手机端的新浪微博网页

新浪微博的用户机制比较严格，如果要查看id为xxx的用户，即使获取到了链接仍然无法访问，网页会将跳转到登录页面，要求我们登录后才能查看其他人的微博，因此我们需要首先登录自己的微博账号来获取cookie认证。从网页源代码中可以看到，邮箱/手机号框的id属性为“loginName”，密码输入框的id属性为“loginPassword”，登录按纽的id属性为“loginAction”。

#获取

driver.get('https://passport.weibo.cn/signin/login')

time.sleep(3)

#输入

driver.find\_element\_by\_id("loginName").send\_keys(username)

time.sleep(1)

driver.find\_element\_by\_id("loginPassword").send\_keys(password)

time.sleep(1)

#登录

driver.find\_element\_by\_id("loginAction").click()

time.sleep(3)

而最后的time.sleep(3)是必不可少的，不暂停就会有问题。如果缺失这一行，很容易发生第一个界面还没有完全加载，界面跳转之后登录状态就消失的情况。

这里在每两步之间都间隔了1s，毕竟在一瞬间输入所有内容太不现实了。如果要进一步模拟人类的操作，可以使用以下代码。

String=’12345678’

for i in string:

time.sleep(0.2)

driver.find\_element\_by\_id("loginName").send\_keys(i)

在这一步之后输出cookie并查看一下，但我们不需要记录cookie，因为Selenium会把cookie自动带过去。

通过以下代码，查看输出的cookie。

#查看cookie

cookies = driver.get\_cookies()

cookie\_list = []

#把cookie循环写入cookie\_list中

for dict in cookies:

cookie = dict['name'] + '=' + dict['value']

cookie\_list.append(cookie)

#调整并输出

cookie=';'.join(cookie\_list)

print(cookie)

输出结果如下。

输出cookie键值对信息:

domain login.sina.com.cn

httpOnly False

name login

path /

secure False

value XXX

domain .sina.com.cn

httpOnly False

name ULOGIN\_IMG

path /

secure False

value XXX

平常我们在正常使用新浪微博的时候，如果留心，就会发现大概有以下几部分内容展示在页面中，可以被我们直接获取。

* 用户id，
* 用户昵称，
* 微博数，
* 粉丝数，
* 关注数，
* 微博。

## 17.2 实现思路

我们随手打开一个页面观察一下，需要的信息主要分为用户信息和微博消息这两部分。

我们考虑分开写成两个函数，对于每一个页面直接调用。第一个函数用于获取上面的诸多个人信息，第二个函数用于获取用户的微博消息。

因为我们通过用户ID进行跳转，所以这部分可以当作已知信息，直接输出。接下来，通过Xpath获取保存在页面上的用户的昵称，昵称部分的网页源代码如下。

<html>

<head></head>

<body>

<div class="ut">

<span class="ctt">柳岩

<img src="https://h5.sinaimg.cn/upload/2016/05/26/319/5338.gif" alt="V" />

<a href="http://vip.weibo.cn/?F=W\_tq\_zsbs\_01">

<img src="https://h5.sinaimg.cn/upload/2016/05/26/319/donate\_btn\_s.png" alt="M" /></a>

&nbsp;女/北京 &nbsp;

<a href="/attention/add?uid=1644461042&amp;rl=0&amp;st=18fa71">加关注</a></span>

<br />

<span class="ctt">认证：光线传媒当家女主播 歌手 演员</span>

<br />

<span class="ctt" style="word-break:break-all; width:50px;">顺着天意做事，逆着个性做人… 工作事宜请联络经纪人@张剑斌 1...</span>

<br />

<a href="/im/chat?uid=1644461042&amp;rl=0">私信</a>&nbsp;

<a href="/1644461042/info">资料</a>&nbsp;

<a href="/1644461042/operation?rl=0">操作</a>&nbsp;

<a href="XX">特别关注</a>&nbsp;

<a href="XXX ">送Ta会员</a>

</div>

</body>

</html>

可以看到，我们需要的信息都藏在<div class="ut"></div>标签中，因此可以首先采用CSS选择器“//div[@class='ut']”定位到该位置，然后通过.text()方法获取文字，接着使用.split(' ')方法通过空格进行分割。

分割出来的列表的第一个元素就是我们需要的昵称。

至于微博数、粉丝数和关注数，3项内容都在同一个标签下。

<html>

<head></head>

<body>

<div class="tip2">

<span class="tc">微博[3716]</span>&nbsp;

<a href="/1644461042/follow">关注[666]</a>&nbsp;

<a href="/1644461042/fans">粉丝[32682097]</a>&nbsp;

<a href="/attgroup/opening?uid=1644461042">分组[1]</a>&nbsp;

<a href="/at/weibo?uid=1644461042">@她的</a>

</div>

</body>

类似地，可以通过CSS选择器“//div[@class='tip2']”获取到相应内容。接着，使用正则表达式匹配整数和小数。如果感觉使用正则表达式存在一定的困难，也可以通过.text和.split方法的组合分割出需要的信息。

函数一如下。

def user\_info(user\_id):

string='http://weibo.cn/' + user\_id

driver.get(string)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('用户资料')

#用户ID

string\_id='用户id:' + user\_id

print(string\_id)

#用户昵称

str\_list=driver.find\_element\_by\_xpath("//div[@class='ut']").text.split(' ')

string\_name='昵称:'+str\_list[0]

print(string\_name)

#微博数、粉丝数、关注数

string\_num=driver.find\_element\_by\_xpath("//div[@class='tip2']")

#匹配数字，包含整数和小数

pattern=r"\d+\.?\d\*"

result=re.findall(pattern, string\_num.text)

print(string\_num.text)

print("微博数：" + str(result[0]))

print("关注数：" + str(result[1]))

print("粉丝数：" + str(result[2]))

print('\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

接下来是第二个函数，用于内容部分的获取。我们研究页面的微博消息发现，内容都包含在一个类似的标签中。

<div class="c" id="M\_GDT55dGqR"></div>

但在一次测试性的输出中，可以发现，微博最下方的设置也采用类似的方式。

<div class="c">

设置:

<a href="https://weibo.cn/account/customize/skin?tf=7\_005&amp;st=18fa71">皮肤</a>.

<a href="https://weibo.cn/account/customize/pic?tf=7\_006&amp;st=18fa71">图片</a>.

<a href="https://weibo.cn/account/customize/pagesize?tf=7\_007&amp;st=18fa71">条数</a>.

<a href="https://weibo.cn/account/privacy/?tf=7\_008&amp;st=18fa71">隐私</a>

</div>

但这一部分并不是我们想要的信息，所以我们需要在这里需要对所有获取到的内容进行判断，通过以下代码查看文字中是否有“设置”这一部分（如图17-3所示）的问题。如果有，则跳过；如果没有，则当作微博消息进行爬取。

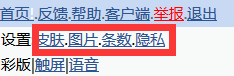


图17-3查看文字中是否有“设置”

if "设置:皮肤.图片.条数.隐私" not in content:

在下面的代码中，在写入的中间用几个变量分别定义我们写入的条数，不仅是为了方便记录条数，还是为了保证最后输出的数据的有序性。

page\_num=1

#当前页的第几条微博内容

curr\_msg=1

#全部微博中的第几条微博

curr\_msg\_in\_all = 0

通过Xpath查找标签，获取其中的文字，确定总页数。

<div class="pa" id="pagelist">1/372页</div>

另外，对于函数的主体，使用try…except语句进行异常处理并抛出异常，增加程序的稳定性。

函数二如下。

def user\_message(user\_id):

page\_list\_1 = driver.find\_element\_by\_xpath("//div[@class='pa']")

print(page\_list\_1.text)

page\_list\_2 = re.findall(r"\d+\d\*",page\_list\_1.text)

#总共有多少页微博

total\_page = page\_list\_2[1]

print("页数："+total\_page)

#页数

page\_num=1

#当前页的第几条微博

curr\_msg=1

#全部微博中的第几条

curr\_msg\_in\_all = 0

content\_path="//div[@class='c'][{0}]"

while(page\_num <= int(total\_page)):

try:

contentUrl = "http://weibo.cn/" + user\_id + "?page=" + str(page\_num)

driver.get(contentUrl)

content = driver.find\_element\_by\_xpath(content\_path.format(curr\_msg)).text

#print("\n" + content) #微博内容，包含原创和转发

if "设置:皮肤.图片.条数.隐私" not in content:

curr\_msg += 1

curr\_msg\_in\_all += 1

with open("content.txt", "a", encoding = "gb18030") as file:

file.write(str(curr\_msg\_in\_all) + '\r\n' + content + '\r\n\r\n')

else:

page\_num += 1

curr\_msg = 1

time.sleep(20)

except exception as e:

print("curr\_msg\_in\_all:" + curr\_msg\_in\_all)

print(e)

finally:

pass

print("finish!")

采用编码GB18030则是因为从新浪微博获取到的大部分是汉字。如果读者愿意，也可以采用UTF-8。

中间在对content\_path变量处理的时候，采用了字符串格式化.format()方法，连接不同类型的变量或内容。

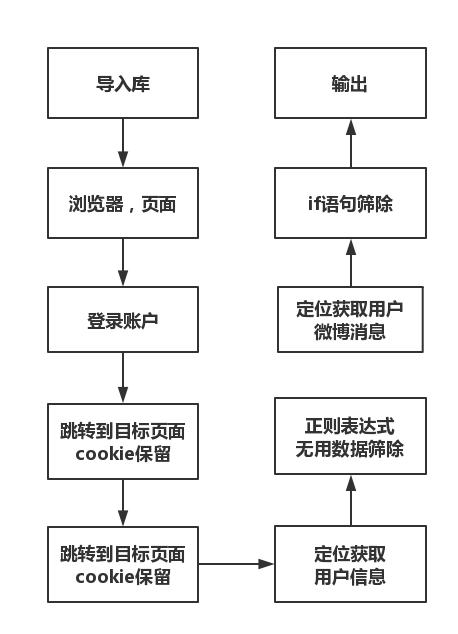
整体思路如图17-4所示。

图17-4 整体思路

最终代码如下。

#导入库

from selenium import webdriver

import time

import re

#打开浏览器

driver = webdriver.Chrome()

#登录账户

def log\_in(username, password):

#进入页面并暂停

driver.get('https://passport.weibo.cn/signin/login')

time.sleep(3)

#输入账户和密码，单击“登录”

driver.find\_element\_by\_id("loginName").send\_keys(username)

driver.find\_element\_by\_id("loginPassword").send\_keys(password)

driver.find\_element\_by\_id("loginAction").click()

#查看cookie

cookies = driver.get\_cookies()

cookie\_list = []

#把cookie循环写入到cookie\_list

for dict in cookies:

cookie=dict['name'] + '=' + dict['value']

cookie\_list.append(cookie)

cookie=';'.join(cookie\_list)

print(cookie)

#获取用户资料

def user\_info(user\_id):

string='http://weibo.cn/' + user\_id

driver.get(string)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('用户资料')

#用户ID

string\_id='用户id:' + user\_id

print(string\_id)

#用户昵称

str\_list=driver.find\_element\_by\_xpath("//div[@class='ut']").text.split(' ')

string\_name='昵称:'+str\_list[0]

print(string\_name)

#微博条数、粉丝数、关注数

string\_num=driver.find\_element\_by\_xpath("//div[@class='tip2']")

pattern=r"\d+\.?\d\*" #匹配数字，包含整数和小数

result=re.findall(pattern, string\_num.text)

print(string\_num.text)

print("微博数：" + str(result[0]))

print("关注数：" + str(result[1]))

print("粉丝数：" + str(result[2]))

print('\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

#获取微博消息

def user\_message(user\_id):

#查看总页数

page\_list\_1 = driver.find\_element\_by\_xpath("//div[@class='pa']")

print(page\_list\_1.text)

page\_list\_2 = re.findall(r"\d+\d\*",page\_list\_1.text)

#总共有多少页微博

total\_page = page\_list\_2[1]

print("页数："+total\_page)

#第几页

page\_num=1

#当前页的第几条微博内容

curr\_msg=1

#全部微博中的第几条微博

curr\_msg\_in\_all = 0

contentPath="//div[@class='c'][{0}]"

while(page\_num <= int(total\_page)):

try:

conten\_url = "http://weibo.cn/" + user\_id + "?page=" + str(page\_num)

driver.get(conten\_url)

content = driver.find\_element\_by\_xpath(contentPath.format(curr\_msg)).text

#print("\n" + content) #微博内容，包含原创和转发

if "设置:皮肤.图片.条数.隐私" not in content:

curr\_msg += 1

curr\_msg\_in\_all += 1

temp=str(curr\_msg\_in\_all)+content

print(temp)

else:

page\_num += 1

curr\_msg = 1

time.sleep(20)

except exception as e:

print("curr\_msg\_in\_all:" + curr\_msg\_in\_all)

print(e)

finally:

pass

print("finish!")

if \_\_name\_\_ == '\_\_main\_\_':

#输入微博账号

username = '15037168088'

#输入密码

password = 'qscrgn123'

#登录

log\_in(username, password)

time.sleep(3)

#部分域名

uid='guangxianliuyan'

#获取用户基本信息

user\_info(uid)

#获取微博内容

user\_message(uid)

## 17.3 代码改进

现在的代码只能够输出，并不能够存储，由于新浪微博用户发送的微博条数可能会达到4位数乃至5位数，因此我们考虑使用为每个用户单独创建一个txt文件以进行存储。

def user\_info(user\_id):

string='http://weibo.cn/' + user\_id

·····

with open("user\_info.txt", "w", encoding = "gb18030") as file:

file.write("用户ID：" + user\_id + '\r\n')

file.write("昵称：" + nickname + '\r\n')

file.write("微博数：" + str(result[0]) + '\r\n')

file.write("关注数：" + str(result[1]) + '\r\n')

file.write("粉丝数：" + str(result[2]) + '\r\n')

在另一个函数中的代码如下。

def user\_message(user\_id):

·····

while(page\_num <= int(total\_page)):

try:

if "设置:皮肤.图片.条数.隐私" not in content:

·····

with open("content.txt", "a", encoding = "gb18030") as file:

temp=str(curr\_msg\_in\_all) + '\r\n' + content + '\r\n\r\n'

file.write(temp)

else:

·····

except exception as e:

·····

finally:

·····

print("finish!")

此外还需要制订一些避免被反爬虫的措施，例如，让程序在每两页跳转之间暂停25s——time.sleep(25)，设置一个暂停的时间更短但还在正常人类阅读速度范围内的页面加载频率，这样可以降低读者的爬虫被封禁(如图17-6所示)的可能性。

有一些保护良好的站点可能会阻止快速提交表单或快速与站点交互。即使没有这些安全措施，让一个爬虫从网站上高度下载大量信息简直就是让它去自杀。

import random

temp=random.randint(0,9)\*0.1

time.sleep(3+temp)

合理控制速度是每一个编写爬虫的自动化人员都不应该违反的规则。虽然多线程程序可以快速加载页面——在一个线程中处理数据，在另一个线程中加载页面——但对于编写良好的爬虫程序来说，这是一个可怕的策略。过度使用别人的服务器资源是非法的，更糟糕的是，它会让一个小网站崩溃。没有人会想让自己的网站被别人的爬虫拖崩溃，因此请尽量控制自己的收集速度。

此外，还有一些比较经典的方法。

* 更换用户的IP地址。
* 同时采集多个页面，随机访问。
* 经常更换UserAgent。
* 了解网站屏蔽规则。
* 处理cookie文件。

最后，如果有条件，可以多申请几个公网IP地址，以及尝试使用ADSL，申请多条线路，编写程序进行断线重拨，远程重置硬件，动态追踪IP地址。采用分布式爬虫，一台中央机器可以控制多台计算机的页面抓取，至于页面的去重可以考虑布隆过滤器。

被封禁IP地址可能造成图17-5所示的后果

如果可能，可以考虑与网站的管理员协商，购买一部分数据。如果你所需要的数据量比较大，并且相应网站的管理员要价合理，适当购买一部分数据是完全可以考虑的的，不然可能服务器消耗的功率会非常庞大。



图17-5 拒绝访问

此外，还有一些内容是我们需要改进的，我们现在输出的文件仍然有许多问题。可以考虑使用下面的方式改进代码，删除输出的文本中无用的内容。

fread=open('weibocontent.txt')

fwrite=open('weibocontent2.txt', 'w')

try:

for line in fread:

text = line.strip()

if("" == text):

continue

if( 5 >= len(text)): #一般来说都是四位数

continue

if(-1 != text.find("转发理由")):

continue

if(-1 != text.find("原图")):

continue

string\_1 = 0

if("发布了" in text or "转发了" in text):

if(-1 != text.find("：")):

string\_1 = text.find("：") + 1

elif(-1 != text.find(":")):

string\_1 = text.find(":") + 1

string\_2 = len(text)

if(-1 != text.find("http")):

string\_2 = text.find("http")

elif(-1 != text.find("全文")):

string\_2 = text.find("全文")

elif (-1 != text.find("赞")):

string\_2 = text.find("赞")

elif(-1 != text.find("[组图共")):

string\_2 = text.find("[组图共")

content = text[string\_1 : string\_2]

fwrite.write(content + '\r\n')

finally:

fread.close()

fwrite.close()

在这里，我们过滤掉了“转发理由” “原图”等信息，在每条判断语句中都加入“-1 !=”用于判断文本中是否存在这些文字，如果不存在，则返回－1。

## 17.4 结语

到了这里，本书的内容也进入了尾声，但我们的前进之路不会停止。正如梭罗在《瓦尔登湖》中写道：“使我们视而不见的光亮，对于我们就是黑暗。当我们清醒时，曙光才会破晓。来日方长，太阳只是颗启明星。”不要轻言自己会怎样怎样，因为我们会不停向着更美好的方向前进，即使那条路布满荆棘和乱石，即使它会给我们留下不可磨灭的伤痛，我们也不会退缩。

前行勇往直前，前进永无止境！哥伦布的航海日志的最后一句话是“我们继续前进”。看似平凡的前进包含着无比的信心和毅力。当某一天你蓦然回首遥望走过的路时，你会为自己骄傲，更会问心无愧！那么，从现在开始，也让我们一起携手前进吧！

# 附录A 如何阅读源代码

作者经常思考如何读一份陌生的代码，最近有一些想法。现有的源代码都是你的资源，但是这些良莠不齐，需要取其精华去其糟粕。

读代码通常要能回答下面两个问题。

* 要解决什么问题？
* 解决方案是如何实现的？

大到整个项目，小到一个模块、函数，看的时候，都要带着这两个问题去看。看完了，这两个问题能答上来，才有效。代码不仅要读，还要理和试。

具体可以这样做。

第一，理解需求。代码的作用是什么？如果可能，首先，做一个背景调查，看官网介绍、维基百科，了解主要功能，应用于的项目，以及同类的框架。其次，写demo，调用API，理解接口的设计，使用框架至少参照入门示例做个小 demo。然后，调试，单步跟踪，理解各模块的联系和调用层次。最后，理解其中的哲理、性能和功能的取舍。你要看的不只是语法上的技巧，更重要的是设计上的思路和原理。判断读没读懂最简单的标准是，假如给充足的时间，有没有信心写出一个差不多的东西来。

第二，了解代码的生成机制。自己先思考，如果你来做，该怎么做，再看源代码。比如，“输出日志的时候是异步的吗？如果我需要把日志输出到另外一台机器上，我该怎么做？”带着这些问题去思考解决方案，盲目地看所谓的源代码，没有任何意义。最适合看源代码的情况就是，你对完成一个功能感兴趣，所以你想要了解源代码。

另外，读代码的时候要多读，才能读出感觉来。比如，你看某个函数抛了一个 up\_call\_info，那凭直觉你就应该猜到这应该是个自定义结构体，用来当作 message 传递给上层，没准与之对应的还有一个 down\_call\_info。

在阅读代码的时候，下列方法有助于你更好地理解代码。

* 看git历史。
* 拖进source tree，查看提交代码的历史更改，从第一个commit开始读，而不是最后一个。
* 一定要运行代码。
* 想办法运行。不运行的代码，并不能看懂。
* 借助 IDE 的代码结构分析工具。
* 在IDE里阅读代码，在IDE里可以跳转，查看定义，比在网页上看效率高得多。比如， Visual Studio中就可以自动生成代码结构图及调用关系图， Notepad++中可以自动折叠代码结构。
* 找到合适的代码，如果直接看Nginx的源代码应该看不懂，如果看HTTP 服务器的几十行源代码，却是可以理解的。最重要的一点是理解代码的历史，理解代码的发展过程，从小到大的过程，理解功能为什么一步一步添加。
* 留意代码指纹和联系方式。代码不是凭空变出来的，写代码的人会在博客和书上留下痕迹，比如，留下思路、论文和联系方式，其实解决很多疑惑的办法是直接联系作者。
* 调试代码。
* 对作者而言，我一般先对源代码进行搜索，找到函数入口，读懂入口函数，然后一层一层向下递归，展开单步调试，调试打印变量。写Python代码，写Objective-C代码很重要的一点是调试好。如果入口函数是个大的 if-else 结构，那么每一个分支就对应一套处理逻辑，分支里一般都是封装好的函数调用，然后你调用那些函数就可以了，函数调用里再调用的结构体、数组都可以一步一步跟下去。如果入口函数只是一个事件循环，对于事件队列进行分发，交给后续的事件处理程序进行 处理，就可以从事件队列的初始化、事件进入队列、处理程序如何注册等一直阅读下去。尽可能编译、调试，能调试的代码，几乎没有看不懂的。
* 画UML图。UML图不是没用的图，它们会告诉你如何分析代码。UML图包括动态图、静态图、类图、部署图、用例图。网络上存在相关的工具，这些工具可以分析代码的流程图和类调用关系。
* 仿写代码。仿写代码耗费的时间比较长，但是照着写一遍，是有益处的。
* 买书。知识比钱知值钱，时间比钱值钱。书可以节约很多时间。如果是十分知名的开源项目，在网上一般可以搜到“XXX源代码结构分析”之类的书或者文章，参照类似的资料就可以，即使文章并不是很完整，也可以起到比较好的梳理作用。

阅读代码是非常非常枯燥的一件事，尤其是阅读编码不符合自己习惯的代码。最笨但最有效果的方法是，抄一个。新手想要通过阅读代码来提升，建议参与GitHub上正在活跃的小型开源项目。找到一个小型的几千行左右的开源项目，然后试着思考能不能添加什么功能，然后就可以动工了。Git的log和项目自身的维基百科是两大助手，如果拿到PR是非常有成就感的事。一开始添加一些小功能并不需要把全部代码都阅读完，在写的过程中会逐渐了解整个项目的结构和细节实现。

个人认为，兴趣推动远远比“今天我要把这个东西啃掉这样的目标推动”要坚持得久。

开始之前，读者最后对版本号管理有一些基本的认识。在作者阅读前端库、Python后台库的过程中，都是以造轮子为目的展开的。所以在最开始的时候，需要一个可以工作并且拥有期望功能的版本，早期版本往往用最直接的方法实现最核心的功能，理解起来效率最高，后期的异常处理、重构等反而增加了阅读难度，错失重点。读了半天，可能看的是个 corner case。而在我们理解了核心功能后，我们就可以向后查看大、中版本的更新内容了。看早期版本。

因此，这里有几个简单的结论。

* 阅读最早的有核心代码的版本往往有助于你更快地了解一段源代码。
* 1.0版本的Release往往是最有效、最直接的实现方式。
* 如果有必要，往后更改较大的版本也需要了解一下。

这里就要涉及《GNU 风格的版本号管理策略》。

* 在项目初版本中，版本号可以为 0.1 或 0.1.0, 也可以为 1.0 或 1.0.0，建议选择那个主版本号为 0 的版本。
* 当项目在进行了局部修改或修正了Bug时，主版本号和子版本号不变，修正版本号加1。
* 当项目在原有的基础上增加了部分功能时，主版本号不变，子版本号加 1，修正版本号复位为 0，因而可以忽略掉。
* 如果项目进行了重大修改或局部修正累积较多，而导致项目整体发生全局变化，主版本号加 1。

另外，编译版本号一般是编译器在编译过程中自动生成的，我们只定义其格式，并不进行人为控制。

内核版本指的是在Linus领导下的开发小组开发出的系统内核的版本号。第一个数字叫主版本号，第二个数字叫次版本号，第三个数字叫修订版本号。一般来说，次版本号还有特定的意义。序号的第二位为偶数的版本表明这是一个可以使用的稳定版本，如2.0.35；而序号的第二位为奇数的版本一般新加入了一些东西，是不一定很稳定的测试版本，如2.1.88。这样稳定版本来源于上一个测试版的升级版本号，而一个稳定版本发展到完全成熟后就不再发展。

此外，对于质量过关的开源项目，源代码存放的路径、源代码文件的命名、源代码中变量的命名都会符合特定的规约。可以根据规约梳理代码结构。比如， lib 一般用来存放所用到的各种库文件，比如对某种协议栈的实现，utils 一般用来存放utility 之类的东西，比如什么把某个 id 映射到 MAC 地址，自己实现的某种 Hash 算法的小函数啊之类的。

不管横向分层还是纵向分块，代码都是分模块的，有的是 core, 有的是util，parser 之类的。了解一个项目的文件命名规则，有利于你更好地读懂一段代码。下面以Python的软件库结构为例介绍一般目录的作用。

表A-1 Python的软件库结构

|  |  |
| --- | --- |
| 目录名称 | 用途 |
| Bin | 存放项目的一些可执行文件，当然，可以命名为script等 |
| Doc | 存放说明文档 |
| DLLs | 存放程序运行的动态运行库等文件 |
| Lib | 存放Python的标准库 |
| Lib\site-packages | 存放第三方库。 |
| Libs | 存放Python的内置库、语法 |
| include | 存放编译器的C语言头文件源代码 |
| Tools | 存放Python的工具和示例代码 |
| tcl | 存放Python默认内置的GUI工具tkinter |
| LICENSE.txt | 存放授权许可 |
| NEWS.txt | 存放更新的日志，记录版本更改 |

# 

# 附录B Git分布式计算

## B.1 Git 的背景介绍

作为一个伟大的开端，Git的诞生有着不平凡的背景，其中伴随着创造与破坏，争吵与辩论。众所周知，Linux系统的开发并不是由单个人或者团队完成的，而是由来自全世界各地的开发者协作共同完成的，因此就需要有一个分布式计算平台，用于收集并记录各个开发者提交的变动。

2002年以前，Linux项目的开发者把源代码文件通过diff的方式发给Linus（Linux之父），然后由Linus本人通过手工合并这种最原始的方式来迭代代码。2002年，Linux内核项目启用Bitkeeper作为DVCS（Distributed Version Control System，分布式版本控制系统）。不得不承认，作为一个早期的解决方案，DVCS还是相当完备的。然而，好景不长，2005年，开发Linux内核的社区与开发Bitkeeper的商业公司之间的关系破裂，生气的Bitkeeper公司取消了对Linux社区的软件免费授权。当然，Linux之父Linus也生气了，于是他花了两周时间写出了现有git的原型，1个月之后，Linux操作系统源代码的版本管理系统已经转为Git。

## B.2 Git 与其他版本控制系统的区别

在这里你还要明白一个问题，Git与GitHub并不是同一个东西，它们之间的关系更像是steam和《侠盗猎车手》的关系，一个是平台，一个是内容。Git是一款免费、开源的分布式版本控制系统，而GitHub是用Git做版本控制的代码托管平台。

那么分布式版本控制系统与其他版本控制系统的区别在于哪里呢？区别如下。

* 完整性

Git在进行底层建构时，就将“存储前计算校验和（数据通信领域重要的验证信息，用于保证传输完整性）”作为一项基本哲学，这意味着很难对Git进行文件或目录修改而让Git不知情。同时，这种方式也保证了文件传输的完整性，使得在传输过程中几乎不可能丢失或损坏文件。

* 存储方式

Git和其他版本控制系统的差别还在于怎样对待数据。Git在提交更改的时候，将全部文件制作成一张快照，纵览一遍所有文件的校验和并与已有的校验和进行比较。如果校验和不相等，那么会保存一个快照并建立指向快照的索引；如果校验和相等，那么会建立一个指向上次保存的文件的快照的索引，因为Git关心的是文件的数据在整体上是否变化。而其他文件系统会在最初提交项目的时候建立一个基本文件，并记录文件内容的具体差异，建立一个随着时间变化而变化的文件变更列表，记录并关心的是文件内容的具体差异。

* 离线操作

集中式版本控制系统的内容都存储在中央服务器，在需要使用的时候必须先从服务器获得推送。而如果当天断网，就无法使用。更糟糕的是，这种方式对网络传输速度提出了极高的要求，例如，有一个300MB的文件，10个人同时需要修改，你的网络传输速度为5Mbit/s，那么你可能要等到10min以后才能开始工作。而Git分布式版本管理系统没有主库的概念，每个复制后的项目都是一个仓库，没有网络时依然可以历史查看，比较变更，提交修改，等到有网络时一次性上传。

需要注意的是，Git虽然可以告诉你变化的内容（例如，你在上一个版本删除了readme文件），但是如果你使用的是图片和视频这些二进制文件，版本控制系统就无法获取文件具体的变化，它只能告诉你文件由1MB增加到了2MB，但具体修改了什么，版本管理系统无法获知。另外Microsoft Word也是二进制格式，这意味着Git同样无法获取具体内容的变化，因此建议读者以纯文本方式编写文件。提到文本编写，还需要说明的是，如果你使用的是Windows系统，那么请不要用记事本编辑任何附带有代码的问题，记事本会在文档开头加入0xEFBBBF(UTF-8)。默认编码设置为UTF-8，支持所有语言。

在了解Git的基本命令之前，你需要理解git的工作区、暂存区、本地仓库、远程仓库的区别。

* 工作区：在复制一个项目到本地或初始化一个文件夹后，项目所在的目录文件夹就是工作区。
* 暂存区：在对工作区做出了更改后，使用git add命令就会将工作区的更改（包括添加、删除、修改）添加到暂存区中
* 本地仓库：变更过的内容就会从暂存区提到本地仓库（使用git commit –m [修改描述]），本地仓库是你推送到远程仓库前的最后一道门。但请不要误会，本地仓库和远程仓库是同级的，比如，它们通过复制、推送互相交换内容。区别只在于文件内容略有差异，比如，在你从远程仓库复制和下载项目之后，又增加了一个ReadMe.txt，这时你的本地仓库中的项目就是远程仓库项目的一个新版本。
* 远程仓库：在多人协作的开发过程中，将主文件存到某个人的计算机上并且以这个人的进度为基准是不现实的，比如，这个人的项目文件损坏了、计算机坏了，或者停电了、网络传输速度不够等因素都会影响项目开发进程，这时就需要通过云端的一个远程仓库来存储项目。我们从远程仓库复制文件，在本地的开发公工作完成后，再推送到远程仓库（git push origin [分支名]推送到特定分支上）。

那么为什么要分为git add和git commit两步呢？通过add可以多次添加，但通过commit提交一次就会推送所有已经add到暂存区的修改。

下面通过一个多次使用Git命令的例子来展示Git命令的用法。Git常用命令如表B-1所示。

表B-1 Git常用命令

|  |  |
| --- | --- |
| 命令 | 说明 |
| $ git init | 在当前目录新建一个Git代码库 |
| $ git init [项目名称] | 新建一个目录，将其初始化为Git代码库 |
| $ git status | 查看当前 Git 仓库的状态（在Git仓库的目录中使用） |
| $ git config --list | 显示当前的Git配置 |
| $ git config -e | 编辑Git配置文件 |
| $ git clone [链接] | 下载一个项目和它的整个代码历史 |
| $ git push | 推送更改到远程仓库 |
| $ git push orgin [分支名] | 推送到分支上 |
| $ git branch | 查看分支，当前分支会显示为绿色，并在前面加星号 |
| $ git branch [分支名] | 创建分支 |
| $ git branch -d [分支名] | 删除分支 |
| git checkout [分支名] | 切换分支 |
| $ git checkout -b [分支名] | 创建并切换到某个分支 |
| $ git log --pretty=oneline | 显示从最近到最远的提交日志 |
| $ git reset --hard head^^ | head^表示倒退一个版本，head ^^倒退两个倒退，如果版本多，可以用HEAD～10 |
| $ git checkout -- [文件名] | 撤销工作区刚才做的更改 |
| # git reset HEAD [文件名] | 撤销暂存区刚才的修改 |
| $ git checkout -- [文件名] | 恢复文件，用版本库中文件替换工作区的版本 |
| $ git config --global alias.[别名] [命令] | 例如git config --global alias.br branch，输入git br和git brach效果相同 |
| $ git stash | 在不修改的情况下，暂时封存现有分支 |
| $ git stash list | 列出已经封存的工作 |
| $ git stash pop | 恢复已经封存的工作，并删除stash中的文件 |
| $ git branch -D feature-vulcan | 强行删除分支 |

**注意：**当你想不起来某一个Git命令的具体用法时，你可以加一个-h参数，例如 git config –h，来获取帮助。

## B.3 Git中的复制、修改和推送

首先，新建一个文件夹。假设你使用的是Windows系统，你可以在任意一个目录（包括桌面）右击，选择Git Bash Here来打开Git的控制台。然后使用下面这条命令来新建一个文件夹并进入当前目录（当然，如果不习惯，也可以直接在文件夹中右击，来新建文件夹）。

$ mkdir test\_dir

$ cd test\_dir/

然后，可以使用这条命令来复制一个仓库。这里选取了作者新建的一个仓库，建议读者也自己建立自己关于Github的远程仓库，然后替换仓库的地址。否则，虽然复制后可以在本地修改文件，但无法推送，因为读者的SSH公钥不在这个仓库的账户列表中。

$ git clone https://github.com/xuyichenmo/test\_\_repo

Cloning into 'test\_\_repo'...

remote: Counting objects: 5, done.

remote: Compressing objects: 100% (4/4), done.

remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (5/5), done.

另外，有一点需要说明，在国内，GitHub网站在部分地区速度较快，在部分地区速度则很慢，在你熟悉Git的基本操作后，你可以使用国内的一些代码托管平台。

在安装Git的时候默认安装了Vim（这是一个常见于Linux的编辑器，在Git Bash中即进行编辑，而不需要另外打开一个程序，Vim的学习难度较高，但对于程序员来说好处也非常明显，可以让你摆脱频繁切换键鼠的烦恼）并且以Vim作为默认编辑器，因此可以使用Vim来编辑文件。以下命令vi即使用Vim来编辑，如果文件不存在，那么创建该文件。

$ vi file.txt

使用Vim的时候，需要按键盘上的i键进入编辑模式。当编辑完了，按Esc键退出编辑模式，然后同时按住“Shift+;”键进入命令模式，输入wq保存并退出。

git status这条命令用于查看当前的状态。

$ git status

On branch master

Your branch is up to date with 'origin/master'.

Untracked files:

(use "git add <file>..." to include in what will be committed)

file.txt

nothing added to commit but untracked files present (use "git add" to track)

在运行结果中我们可以看到返回结果中的提示。我们刚才新建的file.txt处于untracked的状态，这是因为我们仅仅在工作区更改了文件，并未将文件移动暂存区中。

还需要使用git add命令来将文件修改从工作区移到暂存区中。

$ git add file.txt

然后使用git commit命令来将修改从暂存区推送到本地仓库中，-m参数是message的缩写，后面跟的是需要为这里commit添加的注释。

$ git commit -m "we are sending file.txt"

[master 2d1d634] we are sending file.txt

1 file changed, 1 insertion(+)

create mode 100644 file.txt

本附录开篇解释过add和commit的区别。读者如果觉得add、commit太麻烦，可以使用git commit –am这个命令，作用和使用两条命令单通，唯一的缺陷是这个命令只对已经加入版本控制系统的文件有效。也就是说，如果你新建了一个文件，那么使用这条命令是无效的。

当使用git commit命令填写注释的错误的时候，可以使用git commit –amend来修改最后一次注释，只需要在编辑框中修改位于最上方的注释内容。

最后，使用以下命令来推送来远程仓库。因为你是第一次推送，所以在你输入并执行这个命令后Git Bash会要求你添加自己的账户和密码。

$ git push

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Delta compression using up to 2 threads.

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 304 bytes | 304.00 KiB/s, done.

Total 3 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To https://github.com/xuyichenmo/test\_\_repo

b680f38..2d1d634 master -> master

## B.4 Git 分支

Linux系统是由来自全世界各地的开发者共同开发完成的，而Git是由Linus编写的用来帮助Linux进行版本控制的工具，因此Git主要面向的情景是多人协作开发，但每个人开发后就有了一个新的版本，很难确保多人开发的内容是完美并且互相兼容的，为了方便开发人员把自己的版本直接推送到发布版本，因此需要增设一个新的概念——分支。

在下面的代码中，使用Python的#符号来对每一个命令进行解释和说明。

#查看现有的分支，并在现在所在的分支的分支名后加\*

$ git branch

\* master

#新建一个叫作br2的分支

$ git branch br2

#checkout用于切换分支，现在已经切换到了br2分支

$ git checkout br2

Switched to branch 'br2'

#查看现有分支，这里已经多了br2分支，并且br2分支前有\*符号，这说明已经切换到了br2分支

$ git branch

\* br2

master

# git checkout –b命令的作用相当于以上两条命令的作用，创建并切换到br3分支

$ git checkout -b br3

Switched to a new branch 'br3'

#尝试推送br3，但br3分支目前没有任何更改且是一个空分支，自然无法推送，因此需要新建一个文件

$ vi file3.txt

$ git add file3.txt

$ git commit -m "we are committing file3.txt"

[br3 a5181a4] we are committing file3.txt

1 file changed, 1 insertion(+)

create mode 100644 file3.txt

#尝试推送br3，分支推送命令是git push origin [分支名]

$ git push origin br3

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Delta compression using up to 2 threads.

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 303 bytes | 303.00 KiB/s, done.

Total 3 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To https://github.com/xuyichenmo/test\_\_repo

\* [new branch] br3 -> br3

一个最基本的项目要有一个用于发布的稳定版本，一个用于开发的测试版本，一个修复漏洞的版本，大型的项目甚至还要细分更多版本。在刚开始的时候一部分人为了有一个清晰的版本建构，以及规范开发和方便后续维护，采用几个特定的通俗易懂的单词作为分支名，紧接着采用这种命名方式的人越来越多，也就形成了分支命名的默认共识。久而久之，也就形成了一种常见的分支命名方法。

一个标准的项目一般会有以下几个分支。

* Master，可用的稳定版本。
* Develop，用来开发新版本，用来存放、开发即将发布到Release分支的几个最新版本。
* Release，这个分支并不是存放正式发布的产品，而是待发布分支，用来做和发布有关的事情，比如，编写发布文档、打包测试。
* Feature，开发项目的新功能。
* Feature—fix，功能修复分支，用来修复生产事故。
* Hotfix，紧急修复Bug分支，一般使用Issue [ID序号]来命名。

## B.5 Git 自定义命令

如果读者觉得Git自身的命令太难用，输入速度太慢，那么读者可以自定义一个替换命令，使用方法是--global alias.[别名] [原命令]。比如，在下面这个命令中使br命令等同于branch命令，以后输入git br即可。

$ git config --global alias.br branch

使用这条命令来看一下效果。

$ git br

br2

\* br3

master

git branch –D [分支名]这条命令用于删除分支，但使用这条命令有一个要点。读者在使用的时候必须先切换到其他分支，才能删除切换前的分支，你无法位于一个分支的同时删除一个分支，就像你无法提起自己一样。

在下面的例子中，刚开始，我们处于br3分支，尝试删除，出现的错误消息告诉我们无法删除。接着，我们切换到br2分支，再删除br3分支，就会成功。

$ git branch -D br3

error: Cannot delete branch 'br3' checked out at 'E:/Git/test\_dir/test\_\_repo'

$ git checkout br2

Switched to branch 'br2'

$ git branch -D br3

Deleted branch br3 (was a5181a4).

## B.6 Git stash命令

可能会有这样的情况，读者正在开发一个项目的一个测试版本，但是这时发布版本出现一个紧急的Bug，测试版本才做到一半又无法推送，而我们又不想舍弃已经做好的内容。为了应对这种情况，Git设置了专门的命令——git stash，一个用于存储现有的更改的命令。stash单词可以翻译为存储，但相比于存储，我更喜欢称之为封存，我觉得封存这个词汇用来说明git中这个命令的作用更加恰当一些。

下面给出一段示例代码。首先，用vi命令来新建一个文件。然后，通过git add添加到版本管理系统中。接下来，我们第一次查看状态，git bash提示我们有一个文件已经添加但是没有提交。接着，我们使用git stash命令暂时封存已经做出的更改（封存的是暂存区的更改，如果没有提交到暂存区或者已经转到本地仓库，那么命令无效）。最后，第二次使用git status查看状态，git bash告诉我们没有文件可以提交。

$ vi test2.txt

$ git add test2.txt

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: test2.txt

$ git stash

Saved working directory and index state WIP on master: 3a36337 sh

$ git status

On branch master

nothing to commit, working tree clean

下面介绍如何查看封存和释放封存。git stash list用于查看做出的封存，而git stash pop是释放已经的封存。事实上，在执行pop这个命令的时候，git bash就已经提示我们释放了test2.txt。使用git status来确认一下， t确实已经释放了test2.tx，状态和刚才的一样——尚未提交。

$ git stash list

stash@{0}: WIP on master: 3a36337 sh

$ git stash pop

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: test2.txt

Dropped refs/stash@{0} (7731390a1c83f6a77ea9bf70bf26ba189c16db2b)

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: test2.txt

## B.7 Git 文件的对比

既然Git是版本管理系统，就少不了多个版本的内容进行对比。git diff [file\_name]是专门用来进行对比文件差异的，这里选择一个新建的文件夹来展示git diff的用法

下面仍然使用Python的注释符号#来进行解释。

#新建了一个文件夹，并初始化它，创建了一个test.txt，内容为空。然后将其添加到版本控制系统并提交

$ git init

Initialized empty Git repository in C:/Users/Administrator/Desktop/新建文件夹/.git/

$ vi test.txt

$ git add test.txt

$ git commit -m "we are committing test.txt"

[master (root-commit) f4f644a] we are committing test.txt

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 test.txt

#更改test.txt，在里面加入了一行“this is a test file”,cat是Linux系统的命令，用于显示文件的内容，git bash中同样有这个命令

$ vi test.txt

$ cat test.txt

this is a test file

下面使用git diff命令来进行第一次对比。 ---a/test.txt显示旧版本的内容，+++b/test.txt显示新版本的内容，例如，“+this is a test file”表示新增了这一行内容。不加任何参数默认对比工作区和暂存区。

$ git diff test.txt

**diff --git a/test.txt b/test.txt**

**index e69de29..493021b 100644**

**--- a/test.txt**

**+++ b/test.txt**

@@ -0,0 +1 @@

+this is a test file

--cached参数表示对比暂存区和最新的本地仓库。因为我们刚才已经添加了第一次编写的没有任何内容的test.txt并提交，在第一次对比中暂存区和最新本地仓库的内容相同，所以没有返回任何结果。而在添加后暂存区的内容就已经更改，因此这里第二次对比结果和工作区、暂存区对比结果相同。

$ git diff --cached test.txt

$ git add test.txt

$ git diff --cached test.txt

**diff --git a/test.txt b/test.txt**

**index e69de29..493021b 100644**

**--- a/test.txt**

**+++ b/test.txt**

@@ -0,0 +1 @@

+this is a test file

第三次使用git diff head命令进行对比。这次对比工作区和最新的本地仓库，当我们将已经添加的test.txt文件提交后，在第二次对比中同样没有任何返回结果，因为这是工作区和最新版本仓库的内容相同。

$ git diff head test.txt

**diff --git a/test.txt b/test.txt**

**index e69de29..493021b 100644**

**--- a/test.txt**

**+++ b/test.txt**

@@ -0,0 +1 @@

+this is a test file

$ git commit -m "Submit the change for the second time"

[master 63e1b77] Submit the change for the second time

1 file changed, 1 insertion(+)

$ git diff head test.txt

总之，git diff常见的3种对比模式如下。

* git diff [filename]，对比工作区和暂存区。
* git diff --cached [filename]，对比暂存区和最新一版的本地仓库。
* git diff head [filename]，对比工作区和最新一版的本地仓库。

## B.8 Git 版本日志、回退

很多人对于版本回退这个词很恐惧，总是担心在公司的计算机万一错删稳定发布版本或者错删数据库，就会铸成大错，这种心态是不好的。同时，为了避免出现刚才那样的错误，强烈不建议使用公司的项目来练习。

首先，按照常规创建一个文件夹，并初始化。然后。新建一个test.txt，第一次添加，并提交。

$ git init

Initialized empty Git repository in C:/Users/Administrator/Desktop/新建文件夹/.git/

$ vi test.txt

$ git add test.txt

$ git commit -m "the first commit"

[master (root-commit) 540cfdd] the first commit

1 file changed, 1 insertion(+)

create mode 100644 test.txt

现在，再次修改test.txt并添加提交。

$ vi test.txt

$ git add test.txt

$ git commit -m "the first commit"

[master c3f52a3] the first commit

1 file changed, 2 insertions(+), 1 deletion(-)

使用git log命令可以查看版本日志，来确定最近几个修改的版本。当然，如果你觉得这种方式不够简洁，那么你可以采用git log --pretty=oneline来在一行输出每个版本。

$ git log

commit c3f52a37af1a0950b11b1a6b87744b5342d7454f (HEAD -> master)

Author: xxxx <xuyichenmo@gmail.com>

Date: Sun Jul 8 12:51:49 2018 +0800

the first commit

commit 540cfdd92c15c32d438017ffa7831f904a2040a2

Author: xxxx <xuyichenmo@gmail.com>

Date: Sun Jul 8 12:51:27 2018 +0800

the first commit

$ git log --pretty=oneline

c3f52a37af1a0950b11b1a6b87744b5342d7454f (HEAD -> master) the first commit

540cfdd92c15c32d438017ffa7831f904a2040a2 the first commit

git reset –hard HEAD^用于版本回退，一个^表示回退一个版本，^^表示回退两个版本，依次类推。当要倒退的版本太多的时候，可以采用HEAD~*n*这种方式。

$ git reset --hard HEAD^

HEAD is now at 540cfdd the first commit

git reset存在3种回退模式，分别是sof、mixed和hard。无论采用哪种模式，本地仓库版本都会回退，但区别在于工作区和暂存区。下面简单介绍这3种模式。

* soft模式，工作区和暂存区都保留。
* mixed模式，工作区保留更改，暂存区清空。
* hard模式，工作区和暂存区都清空。

另外，读者在前面和刚才的看到的由字母与数字组成的长字符串（如“540cfdd9…….”）是Git内部的版本号，Git内部并不采用1~4这样的整数，而是基于安全散列算法生成一个十六进制的字符串。

1. isibility表示可见性，代表元素没有隐藏，并且元素的宽和高都不等于0。 [↑](#footnote-ref-1)