



---

# SELENIUM 自动化测试

---



# 目录

<b>第 1 章 Selenium 安装 .....</b>	<b>7</b>
1.1 Selenium 的优点 .....	7
1.2 Selenium Webdriver .....	8
1.3 Selenium 安装 .....	10
1.3.1 windows 用户的详细说明 .....	10
1.3.2 Linux 用户的详细说明 .....	12
<b>第 2 章 页面导航 .....</b>	<b>12</b>
2.1 启动浏览器 .....	12
2.2 打开页面 .....	14
2.3 浏览器基本操作 .....	14
2.4 拖拽窗口 .....	15
2.5 Cookie 的调用 .....	17
2.6 使用多个窗口 .....	18
本章附录 .....	19
<b>第 3 章 元素定位 .....</b>	<b>20</b>
3.1 元素定位是什么 .....	21
3.2 根据 ID 定位 .....	21
3.3 根据 Name 定位 .....	23
3.4 XPath 定位 .....	24
3.5 标签名(Tag Name)定位 .....	26
3.6 Class Name 定位 .....	27
3.7 css 选择器定位 .....	29
3.8 Link Text 定位 .....	30
3.9 Partial Link Text 定位 .....	30
<b>第 4 章 利器——phantomjs .....</b>	<b>31</b>
4.1 PhantomJS 是什么 .....	31
4.2 PhantomJS 下载与安装 .....	32
4.2.1 Windows .....	32
4.2.2 Linux .....	32
4.2.3 检验安装 .....	33

4.3 配置相应的 Webdriver .....	33
4.4 第一个 PhantomJS 小程序 .....	33
4.5 time.sleep() 慢下来 .....	36
4.6 PhantomJS 命令行指令参数 .....	36
4.7 其他问题 .....	37
4.7.1. 中文编码问题 .....	37
4.7.2 不同 frame 间的转换 .....	37
4.7.3 PhantomJS 进程不自动退出 .....	37
<b>第 5 章 实战 part 1——Python 官网.....</b>	<b>38</b>
5.1 Selenium 访问 Python 官网 .....	38
5.2 通过 JavaScript 修改官网标题 .....	38
5.3 在搜索框搜索 .....	39
5.4 获取 latest news 部分 .....	40
5.5 selenium 的等待 .....	41
<b>第 6 章 实战 part2——今日头条 .....</b>	<b>44</b>
6.1 Selenium 访问百度热词 .....	44
6.2 selenium 搜索相关热词 .....	46
6.3 获取第一条结果 .....	47
6.4 元素定位——抓取内容 .....	48
6.5 数据存储 .....	50
<b>第 7 章 数据编解码，处理 .....</b>	<b>53</b>
7.1 读写 CSV 文件 .....	53
7.2 读写 JSON 文件 .....	55
7.3 将字典转化为 XML .....	57
7.4 解析 XML .....	58
7.5 编码 BASE64 .....	60
7.6 词性分析、统计分析 NLTK .....	61
7.7 哈希表 .....	66
7.8 布隆过滤器 .....	67
7.9 关系型数据库 MYSQL .....	69
7.9.1 MySQL 安装 .....	69
7.9.2 MySQL 常用命令 .....	73

7.10 非关系型数据 NOSQL .....	79
7.11 总结.....	79
<b>第8章 实战 part3——猫眼电影.....</b>	<b>80</b>
8.1 网站分析 .....	80
8.2 测试代码 1.0.....	82
8.3 测试代码 2.0.....	84
<b>第9章 实战 part4——淘宝商品.....</b>	<b>87</b>
9.1 准备工作 .....	87
9.2 流程简述 .....	87
9.3 代码解读 .....	90
9.4 可配置项 .....	94
<b>第10章 单元测试.....</b>	<b>95</b>
10.1 为什么要写单元测试 .....	95
10.2 怎样写单元测试 .....	95
10.3 单元测试演示 .....	96
10.4 单元测试样例 .....	97
10.5 Selenium 单元测试.....	100
10.6 美化报告 .....	101
<b>第11章 多线程.....</b>	<b>103</b>
11.1 什么是进程线程 .....	103
11.2 单线程，多线程对比 .....	104
11.3 实际体验 .....	104
11.4 threading 库常用函数.....	107
11.5 锁的概念 .....	107
11.6 主程序是线程还是进程 .....	109
11.7 更多解决方案 .....	109
11.8 多线程+Selenium 的样例.....	110
11.9 GIL .....	111
<b>第12章 发送邮件.....</b>	<b>111</b>
12.1 通讯的选择 .....	111
12.2 获取授权码 .....	112
12.3 发送邮件 .....	113

<b>第 13 章 Selenium IDE.....</b>	<b>115</b>
13.1 IDE 安装.....	116
13.2 Selenium IDE .....	117
13.3 Katalon Recorder .....	121
13.3.2 实例与步骤 .....	123
13.4 数据驱动 .....	124
13.5 扩展脚本 .....	125
13.5.1 添加扩展脚本 .....	125
13.5.2 定位器构建器 .....	125
13.5.3 定位器构建的自定义顺序 .....	126
13.5.4 Prototype 附加命令.....	126
13.6 Katalon Recorder Helper 工具.....	126
<b>第 14 章 Python 拓展.....</b>	<b>127</b>
14.1 2to3 工具.....	128
14.2 测试类型 .....	128
14.3 通配符类型 .....	129
14.4 str 方法.....	129
14.5 异常层次结构 .....	130
14.6 兼容 Python2 和 3.....	132
14.7 兼容性代码 .....	132
14.7.1 导入库 .....	132
14.7.2 输出函数 .....	133
14.7.3 异常捕获 .....	133
14.8 类相关 .....	134
14.9 推导式 .....	134
14.10 经典除法和真除法 .....	135
14.11 __future__ .....	136
14.12 类型转换和关键字 .....	136
<b>第 15 章 GUI，图形化测试.....</b>	<b>137</b>
15.1 tkinter 测试.....	137
15.2 创建界面 .....	138
15.3 按钮 .....	139

15.4 pack 属性.....	139
15.5 继承类.....	140
15.6 Checkbutton 和 Radiobutton .....	141
15.7 Entry 输入框.....	144
15.8 Listbox .....	148
15.9 Text 控件.....	151
15.10 Canvas 控件.....	153
15.11 Menu 控件.....	156
15.12 .....	spinbox 控件
.....	158
15.13 .....	messagebox
.....	158
15.14 .....	filedialog
.....	159
15.15 Message .....	160
15.16 .....	tkinter 界面布局
.....	161
15.17 .....	tkinter 支持的模块
.....	162
15.18 _tkinter 接口.....	162
15.19 tkinter.ttk .....	162
15.20 底层实现流程 .....	163
<b>第 16 章 实战案例 part5 知乎.....</b>	<b>163</b>
16.1 知乎分析 .....	164
16.2 文字部分 .....	166
16.3 图片部分 .....	169
16.4 建议 .....	173
16.4.1 从干净状态开始 .....	174
16.4.2 测试隔离 .....	174
16.4.3 Anaconda .....	174
16.4.4 报告 .....	175
16.4.5 IP 代理.....	175
16.4.6 过滤请求 .....	176

16.5 一些注意事项 .....	176
16.5.1 Captchas .....	176
16.5.2 性能测试 .....	177
16.5.3 六度分割理论与实际使用 .....	177
16.5.4 HTTP 响应代码.....	177
16.6 常见问题 .....	178
16.6.1 元素定位失败 .....	179
16.6.2 Webdriver 调用失败.....	181
<b>第 17 章 实战 part6 微博 .....</b>	<b>181</b>
17.1 微博分析 .....	181
17.2 实现思路 .....	183
17.3 代码改进 .....	189
17.4 结语 .....	191
<b>附录 1 怎样阅读源码 .....</b>	<b>191</b>
<b>附录 2 GIT 分布式计算 .....</b>	<b>194</b>
1 GIT 的背景介绍 .....	194
2 GIT 与其他版本控制系统的区别 .....	194
3 Git 克隆、修改、推送 .....	196
4 Git 分支 .....	198
5 Git 自定义命令 .....	199
6 Git stash 命令.....	199
7 Git 文件对比 .....	200
8 Git 版本日志、回退 .....	202
<b>附录 3 我们都是创客，生来如此 .....</b>	<b>203</b>
<b>附录 4 下一步 .....</b>	<b>205</b>

# 第 1 章 Selenium 安装

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 正是当前最完美工具。它不是屠龙宝刀，它不需要苦练屠龙之技，它是瑞士军刀，十八般武艺样样精通。

Python 如此强大，并且方方面面都做得很好，以致于没人敢说“精通 Python”，这本书只讲了其中关于数据挖掘的很小一部分，详细的讲述了 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”，对此，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，而在大数据的时代，Selenium 则被广泛应用于网络爬虫。需要注意的是，Selenium 项目并不是单独的一个特定软件，而是由多个工具组成，每个具有特定的功能。

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，并且可以完美的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，放心大胆的进行胆码移植。

但请切记不要轻易更换你的 Webdriver 调用的浏览器，Selenium-Webdriver 对浏览器的操作是通过浏览器原生的 api 来实现，这样一来，速度虽然大大提高，稳定性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。

Selenium-Webdriver 支持的部分浏览器和操作平台如下（包括但不限于）：

浏览器：

- ☐ 火狐浏览器
- ☐ 谷歌浏览器
- ☐ 欧朋浏览器
- ☐ Edge 浏览器
- ☐ IE 浏览器

操作系统：

- ☐ 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 方面的自动化工具的话，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：

### 1. 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，这就容易导致一些昂贵的错误，比如说，误删数据库，系统宕机，这样的错误无论是从经济上，还是从声誉上对企业来说都是一种毁灭性的打击。而当把人为因素从这里拿掉，绝大多数学这些人为错误导致的昂贵代价就可以避免。

### 2. 性能优化

自动化技术人员在日常工作中不可避免的需要面临着提高工作效率，传统方式想要解决这种问题是很难的，而当我们应用了自动化技术之后，我们可以通过更加高效的系统能源利用方式来实现更迅速的执行任务，更高效的工作流程，更强大的工作负载。

### 3. 解放生产力

没有一家企业不想要降低成本，想要扩大企业的盈利就需要招收更多的人，而没有更大的盈利规模就难以融资以招收更多的人，这个问题却在自动化技术面前迎刃而解。自动化技术不需要手动工作，这就意味着通过代码的编写、程序的实现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”。

Selenium 作为一个跨语言的项目，很多编程语言都可以使用它，比如说 Java，但在本书中我们采用 Python 来介绍 Selenium 的使用。为什么采用 Python 呢？Python 以其快捷，方便著称，比如说，解决数独问题，C 语言要写 300 到 500 行代码，Java 只需要写 100-200 行代码，而 Python 完全可以在一百行以内的代码实现。再从长远的角度来说，Python 作为目前蓬勃兴起的一门语言，具有美好的前景，在科学与数字计算，网络编程，系统网络运维方面 Python 可以说是引领风尚，而 Selenium 自动化测试与这些方面具有密不可分的关系，基于 Python 的 Selenium 学习，更加有利于读者的未来发展。

## 1.2 Selenium Webdriver

大名鼎鼎的 selenium 在 2004 年由一个叫做 Jason Huggins 的年轻人开发。Jason Huggins 为芝加哥的 ThoughtWorks 公司工作，他的主要职责是为内部的时间和费用系统（Time and Expenses）开发一个基于 java 脚本的运行测试系统（JavaScriptTestRunner），作为一个推崇自动化的技术人员，他意识到相对于对每次改动进行手工测试，他应该用一个程序来代替自己，来让自己的时间得到更有效的利用。

于是他开发了一个可以驱动页面进行交互的 Javascript 库，能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，如果你去翻查字典，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。但这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，TD），但他们失败了，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，所以，在成功开发后，他们就把自己的开发的项目开源，并且起名为 selenium（化学元素硒），来在口头上出出气。

在 2006 年，Google 已是 Selenium 的重度用户，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 有着很大局限性。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，这个项目被命名为 WebDriver。他们需要一款能通过浏览器和操作系统的 native 本地方法来直接和浏览器进行交互的更先进的测试工具，WebDriver 项目的目标就是要解决 Selenium 的痛点。

Selenium 的项目历史上存在着三个重大版本，selenium1.0（如图 1-1 所示），selenium2.0, selenium3.0。项目 1.0 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 selenium1.0 和 selenium2.0 最大区别就在于 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 selenium3.0（如图 1-2 所示），我们来通过图标来看一看 selenium 的历史，也延续了 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 Selenium 的主要项目之外，作为一个分支而存在。



图 1-1 Selenium RC



图 1-2 selenium



图 1-3 Selenium-IDE

细心的读者不难发现，他们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 Selenium 主要分为两个部分——selenium WebDriver 和 Selenium IDE，前者主要是用来创建基于浏览器的回归自动化套件和测试，后者是主要是用来创建快速的 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 1-4 selenium 项目工具分支

Selenium 项目未来将让自己更加的标准化，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 项目未来版本区别及发展趋势如图 1.5 所示。








	Core	Webdriver	W3C Webdriver
1			
2			
3			
4			
5			

图 1-5 Selenium 版本区别及未来方向

## 1.3 Selenium 安装

工欲善其事，必先利其器，接下来，我们从 Selenium 项目的安装及使用说起。

### 1.3.1 windows 用户的详细说明

本书基于 Python 编程语言来讲解 selenium，在这里，我们假定读者使用 Python 编程语言，并了解 Python 的基本语法、配置方法。

当然，如果你想要下载 Python 上的 selenium 库可以从 PyPI 官方库网站上下载，PyPI page for Selenium package 的网页链接为 <https://pypi.org/project/selenium/>，但更好的方法是使用 pip 来下载，你可以像下面这样，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 Python2.X 的版本，那么你可以通过通过手动安装 pip 或者 easy\_install 工具来方便你的安装，在这里，我们同样提供 easy\_install 的安装方法：

```
#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，你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.
>>>
```

没有消息就是最好的消息，当你输入 `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，你不是一定需要安装它，主要用于运行 Selenium IDE 录制的脚本）。

由于 selenium 基于 JavaScript 编写的，所以我们还要安装 Java Runtime Environment (JRE) 1.6 或者更高版本的 JRE。

这里提供官方网页的链接：

<http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>

同样的，笔者可以在 selenium 项目官方网站的一个子页面来下载 Selenium Standalone Server，截止 2018 年 4 月份，Selenium Standalone Server 的最新版本为 3.13.0。

<https://www.seleniumhq.org/download/>

安装完 JRE 后可以通过 win+r 键打开运行窗口，然后输入 cmd，来打开命令行窗 command (CMD)，接着键入以下这条命令：

```
java -jar F:\Python_ADDED\selenium-server-standalone-3.13.0.jar
```

来启动 selenium 服务器，你需要把-jar 参数后面的 selenium-server-standalone 的文件目录替换成你自己文件所在的目录。

如果你刚刚安装完 JRE 后就直接运行这条命令，那么你可能会运行失败，因为端口占用问题，如果是这样的话，那么你可以通过以下这几条命令来解决

```
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，在我们解除了端口占用后，就可以成功调用了。

- ❑ --debug, -debug  
    <Boolean>: 启用 LogLevel.FINE 日志文件，记录调试。  
    默认值: false
- ❑ --version, -version  
    显示版本和退出  
    默认值: false
- ❑ -browserTimeout  
    <整型> 秒数: 允许浏览器会话的秒数  
    在 WebDriver 命令运行时（例如: 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 60，如果未指定，零或负值意味着无限等待。
- ❑ -config  
    <字符串> 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> IP 或主机名: 通常自动确定。最通常在跨国网络连接配置中（例如，使用 VPN 的网络）
- ❑ -jettyThreads, -jettyMaxThreads  
    <整形>: Jetty (Jetty 是一个基于 Java 编写的网页服务器和 Java Servlet 容器) 的最大允许线程数。未指定，零或负值表示将使用 Jetty 默认值 (200)。
- ❑ -log  
    <字符串> 文件名: 用于记录的文件名。如果省略，将输出到 STDOUT
- ❑ -port

<整型>: 服务器将使用的端口号。

❑ -role

<字符串>可选项有[hub], [node], [standalone]。

❑ -timeout, -sessionTimeout

<Integer> in seconds: 在服务器自动关闭上一个 X 中没有任何活动的会话前, 指定服务器超时。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], [node]格式, 还必须设置 cleanUpCycle。

### 1.3.2 Linux 用户的详细说明

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 的话, 作者在这里推荐你再安装一个 3.x 版本 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使用即可:

```
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

笔者之前在第一章第二节提起过 Selenium 项目可以使我们做到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。

接下来, 我们来看一看如何实现这些行为。

### 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 (翻译为 Web 驱动), 那么什么是 webdriver 呢? 假设我们需要找个陪我们一起猎人上山打猎, 那么目前, 我们就已经雇佣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, 从笔者的角度来看, 它就是驱动浏览器运行的一个工具。

读者可以在这个链接查看所有被 selenium 官方承认 (注意, 但并不都是他们开发的) 的第三方驱动:

```
https://www.seleniumhq.org/download/
```

这是谷歌浏览器 webdriver 的官方下载地址:

```
# chrome_webdriver (截止 2018.7, 目前最新版本为 2.40)
```

```
https://sites.google.com/a/chromium.org/chromedriver/
```

可能因为我们伟大的中国国家防火墙 (GFW, Great Firewall), 读者无法访问上面的网址, 读者可以选用国内淘宝网站镜像提供的地址:

```
http://npm.taobao.org/mirrors/chromedriver/
```

```
# firefox_webdriver
```

```
https://github.com/mozilla/geckodriver/releases
```

在本书中笔者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，如果他们的 **webdriver** 有什么区别，笔者也会详细指出。

下载下来并解压之后，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\

然后再配置环境变量，右键单击我的电脑，然后打开属性，双击高级系统设置，打开的界面的第三个选项卡——高级，然后环境变量，修改 **path**，在最后面添加：

请注意，这行内容的前后各有一个分号。

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

环境变量的设置流程如图 2-1 和 2-2 所示：



图 2-1 环境变量 1

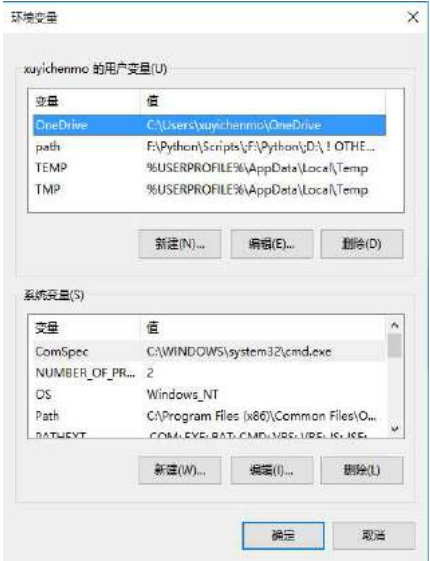


图 2-2 环境变量 2

同样的，我们提供一种更为简单的方法，读者可以将 **chromedriver** 直接放到 **Python** 的安装目录下，读者在安装 **Python** 的时候已经配置过环境变量了，所以这样可以省去很多麻烦。

Linux 系统下，你只需要将把下载好的 **chromedriver** 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，否则就会出现打开浏览器只显示 **data:** 的情况，如图 2-3。

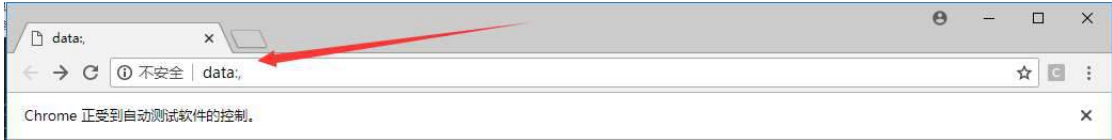


图 2-3 浏览器 data:,

安装与配置就到此结束。剩下的就是使用 **python** 来写代码了。

我们来做第一个代码示范：

#导入 selenium 库

```
from selenium import webdriver
```

#下面一行代码用于启动 Chrome 浏览器

```
driver = webdriver.Chrome()
```

诚如大家所见，以简单方便而著称的 **Selenium**，在导入库后，仅仅一行代码我们便完成了打开浏览器这一操作。



```
driver = webdriver.Chrome()
```

类似的，还有以下这条命令则可以打开火狐浏览器

```
driver = webdriver.Firefox()
```

## 2.2 打开页面

仅仅只能打开浏览器，肯定不能满足我们的需求，我们还需要更进一步的而操作，下面，我们来尝试着打开一个页面。

```
#导入 seleniun 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，看看地址栏的下方，Chrome 浏览器提示我们，正在受到自动测试软件的控制，如图 2-4。



图 2-4 打开百度页面

运行结束后，我们成功的打开了百度页面，类似的，只要将（）中的 URL 地址修改为我们想要打开的页面，那么就可以轻松的打开任意网站的地址（请不要忘记双引号）。

```
driver.get("[url]")
```

代码运行打开的页面默认并不是最大化，有时候，我们并不想要这样，那么可以尝试改变这一默认值，第一个参数值设置的是浏览器的长，第二个参数值设置的是浏览器的宽：

```
#设置打开页面的长和宽
driver.set_window_size(800, 480)
```

还有这另外一条命令，将浏览器窗口最大化。

```
#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，退出：

```
#导入 selenium 库和时间库
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，可能会注意到我们比上一小节多导入了一个时间库，同时，在每个操作前都有一个 `time.sleep()`，这条命令用于将程序停顿一定时间后再执行。之所以这样，并不仅仅是为了让读者看清楚我们的操作，更是为了给浏览器操作网页一定时间。

我们再来看一下在这一小节使用的命令：

```
#前进
driver.back()
#后退
driver.forward()
#刷新
driver.refresh()
#退出
driver.quit()
```

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

## 2.4 拖拽窗口

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轴到底部，然后跳转到下一步页面，为了应对这种情况，但是可以调用滚动轴的相关的命令在 `selenium` 库中是没有的，但我们可以通过使用 `JavaScript` 可以模拟操作，怎么调用 `JavaScript`，为了应对这种情况，`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，这是一条 JavaScript 的代码，用于向下滑动到页面底部的（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)
```

下面，我们来演示一下具体使用方法：

```
#导入 selenium 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()
```

我们来看一下效果，如图 2-5：

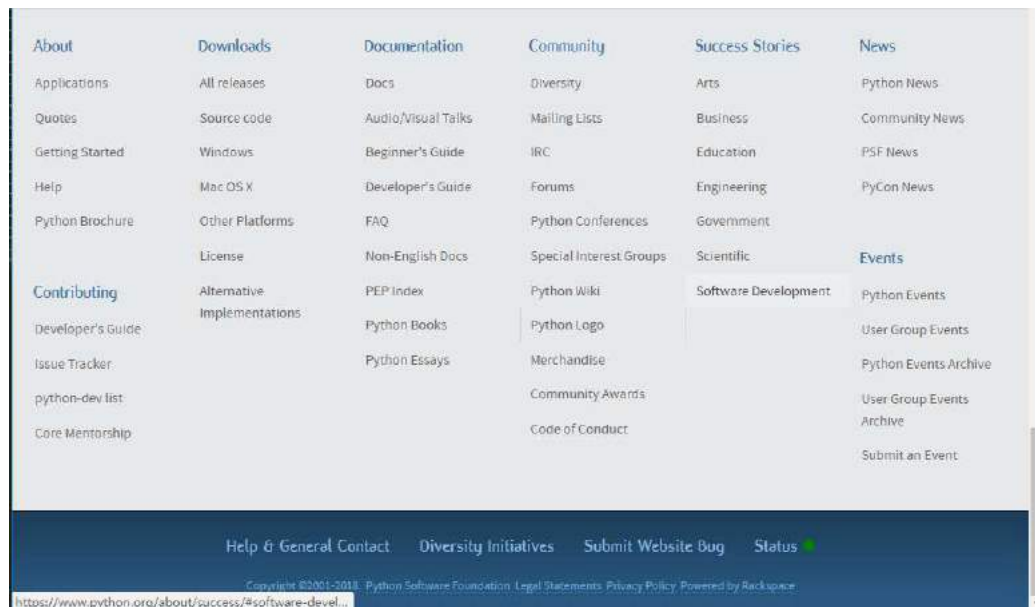


图 2-5 效果展示图

已经成功的跳转到页面的底部了，这时候，我们就可以放心的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
```

```

#我们以今日头条首页为例
#下拉实现方式第二条思路，利用将鼠标移动到最后一个元素位置模拟下拉
driver=webdriver.Chrome()
driver.get("https://www.toutiao.com/")

#保存现在的页面
driver.save_screenshot("save_1.png")

#模拟鼠标操作实现下拉
ac=driver.find_element_by_xpath("//ul[@infinite-scroll-disabled]/li[last()]")
#定位鼠标到指定元素
ActionChains(driver).move_to_element(ac).perform()

#停留两秒也是给加载内容预留 2 秒，然后第二次保存现在的页面
time.sleep(2)#
driver.save_screenshot("save_2.png")

```

读者运行代码，对比两次获取的截图后，就会发现窗口位置确实发生了变化了，并且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 项目为我们提供了成熟的解决方案，这是一些常用的 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”，“域”

我们来做几个例子：

```

#导入elenium、time、ActionChains 库
from selenium import webdriver
import time
from selenium.webdriver import ActionChains

#打开并跳转百度页面
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 中 name 和 value 中添加会话信息
driver.add_cookie({"name":"testname_1234567890"}, {"value":"testvalue_1234567890"})
```

来看一下输入结果：

```
{'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'}
```

返回给我们的 `cookie` 是字典形式的。

来看一看 `add_cookie` 的用法，必须有 `name` 和 `value` 的值，那是直接传参数吗？笔者做了个实验验证一下，并不是这样的。

请读者注意，我们的 `name` 和 `value` 虽然是两个值，但需要放在一个字典中写，不能分开写，否则会报错。报错提示我们需要两个参数，但我们传了三个参数，我们再来检查一下，确实是两个参数，没错啊！那难道是编译器编译错误，也不可能。

这里我们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`，所以才会提示我们给出的参数过多。

而如果我们像这样使用，那么就可以正常使用了

```
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，总是希望多开，到最后，地址栏上可能会有十几个页面。

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

下面是笔者对这一问题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)  
然后我们来做一个示范：  
# coding=utf-8  
  
from selenium import webdriver  
import time  
  
#打开浏览器，窗口最大化  
driver=webdriver.Chrome()  
driver.maximize_window() #  
  
driver.get("http://baidu.com")  
  
#停留两秒后打开搜狗搜索  
time.sleep(2)  
JS1='window.open("https://www.sogou.com");'  
driver.execute_script(JS1)  
#停留两秒后打开有道翻译  
time.sleep(2)  
JS2='window.open("https://fanyi.youdao.com/");'  
driver.execute_script(JS2)  
  
#退出  
#driver.quit()
```

在这个案例中，我们首先打开并且最大化窗口，在停留一定间隔后，分别打开搜狗搜索和有道翻译，退出浏览器。

## 本章附录

历史版本 google 浏览器驱动下载

<http://chromedriver.storage.googleapis.com/index.html>

历史版本 firefox 浏览器驱动下载

<http://ftp.mozilla.org/pub/firefox/releases/>

历史版本 chrome 浏览器下载

[https://google-chrome.en.downloadastro.com/old\\_versions/](https://google-chrome.en.downloadastro.com/old_versions/)

Selenium Webdriver 2.0 包

<http://selenium-release.storage.googleapis.com/index.html?path=2.53/>

Chromedriver 和支持的 Chrome 版本对照表如图 2-6:

chromedriver版本	支持的Chrome版本
v2.40	v66-68
v2.39	v66-68
v2.38	v65-67
v2.37	v64-66
v2.36	v63-65
v2.35	v62-64
v2.34	v61-63
v2.33	v60-62
v2.32	v59-61
v2.31	v58-60
v2.30	v58-60
v2.29	v56-58
v2.28	v55-57
v2.27	v54-56
v2.26	v53-55
v2.25	v53-55
v2.24	v52-54
v2.23	v51-53
v2.22	v49-52
v2.21	v46-50
v2.20	v43-48
v2.19	v43-47
v2.18	v43-46
v2.17	v42-43
v2.13	v42-45
v2.15	v40-43
v2.14	v39-42
v2.13	v38-41
v2.12	v36-40
v2.11	v36-40
v2.10	v33-36
v2.09	v31-34
v2.08	v30-33
v2.07	v30-33
v2.06	v29-32
v2.05	v29-32
v2.04	v29-32

图 2-6 版本对照表

## 第 3 章 元素定位

在计算机中，我们定义，一个页面的最基本组成单元为元素，想要确定一个元素，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，横纵坐标确定平面内唯一的一个点。

在第三章中笔者将会引领读者领略了 Selenium 的八种元素定位方式。通过对本章与前两章的内容的学习，读者可以初步掌握并实现一个略有雏形的自动化程序的能力。

## 3.1 元素定位是什么

在读者开始本章的学习之前,我们需要首先了解一下什么 HTML 元素,简单的来说,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限于:文字、图片、音频、动画、视频。

举个例子,一个 HTML 页面就相当于一个军队,我们要在一个庞大的军队中定位一个人是很困难的,因为你无法记住军队中所有的人,所以,我们需要一种特殊的方法——军牌,来确定一个人所属的部队和身份。而 selenium 给我们提供的方法的作用就和军牌很类似。

Selenium 提供了 8 种定位方式。

- ☐ ID
- ☐ name
- ☐ class name
- ☐ tag name
- ☐ link text
- ☐ partial link text
- ☐ xpath
- ☐ css selector

这几种定位方式很像通过不同的方面来说明一个人,比如说,一个人在家里叫做小明,而他在外面叫做张明明,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 中所对应的方法为:

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

我们在元素定位的过程中,也会遇到很多困难,比如说,前端设计师常常会使用很多方式来隐藏网页中的元素,常见的有这几种:

- ☐ 将 opacity 设为 0
- ☐ 将 visibility 设为 hidden
- ☐ 将 display 设为 none
- ☐ 将 position 设为 absolute 然后将位置设到不可见区域

通常,这些方式也是为了防止网络爬虫而设置的,比如说,网站上存在一个 position 设置为 absolute,位置位于真实用户不可见区域的表单,那么,一旦你提交了这个表单,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 属性的 ID 是很重要的,他们就相当于一个人的身份证。带有特定 ID 值的 HTML 元素可以被 CSS (Cascading Style Sheets, 层叠样式表) 样式选择器和 JavaScript 脚本查找到,

并执行某种特定的功能。

一般为了便于区分在 HTML 页面中我们不会设置 ID 相同的 HTML 元素。并且 HTML 元素 ID 如果重名，也有相当严重的后果，首先，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，如果你用 JavaScript 去获取相应 ID 的元素时，会出错，因为取 ID 只能一个。

下面，我们以百度搜索为例子，来一下如何找到程序的 ID。

这里是百度的搜索框的代码：

```
<input type="text" class="s_ipt" name="wd" id="kw" maxlength="100" autocomplete="off">
```

重点在这里，ID="kw"，好，我们就从这里知道了，百度搜索框的 ID 为 kw，以后，我们就可以找到这个 ID 了。

再来看一下搜索确认按钮的 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"，那么，万事具备了。我们来进行一个实际操作。

```
#导入库
from selenium import webdriver
import time

#打开浏览器和打开百度
driver=webdriver.Chrome()
driver.get("https://www.baidu.com/")

#通过 ID 查找，然后输入，并点击
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 ID 查找百度搜索

代码成功运行，但我们的重点不是只是对代码的重复键入，我们来了解一下原理，

仔细的想一想刚才的代码，首先，我们在进行导入库和打开浏览器等一系列常规操作后，第一次我们查找 ID="kw" 的搜索框，其次通过.send\_key()输入（.send\_key()是一个方法，意思是向搜索框发出特定的字符串，请读者不要忘记在特定字符串的前后要加上双引号）

再次，我们再次查找 ID="su" 的搜索框，然后使用.click()方法进行确认点击（.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）。

可能有好奇的读者想知道，网页的源码一般很长，我们一打开就会感到头大，那么作者是凭什么找到这一段搜索框对应的代码，其实，一般学过后后端开发的人都会对浏览器中的开发者工具（DEVTools, Develop Tools）有所了解，我们在浏览器除了标签页外的任何页面使用 F12 就可以打开开发者工具了，如图 3-2：



图 3-2 开发者工具栏

默认打开后是在 Elements 界面的，但在 Console 页面，我们也可能会在 Console 页面看到这个网站服务的提供商想要给开发者看的一些信息（如图 4.2），比如说，就像上面这个图的百度招聘信息。

然后我们在这里开发者工具栏的左上角可以看到有一个类似于鼠标箭头的图标，我们点击后就可以看到，这里变为蓝色的，而我们的鼠标箭头移动到哪里并点击确认后，在下方就可以看到对应的源码了如图 3-3。



图 3-3 元素定位器

最后在相应的代码所在位置，我们点击前方的三角形就可以展开代码，右键，选择作为 HTML 编辑（EDIT AS HTML），或者是直接按 F2 可以进入编辑状态。

### 3.3 根据 Name 定位

我们再来试一试另外的方法，通过 Name 定位。绝大部分做过 Web 开发的人都问过一个问问题，元素的 ID 和 Name 的区别在哪里，为什么有了 ID 还要有 Name 呢？其实啊，我们在上面已经解释过一部分了，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 的值却是可以重复的。



类似的，selenium 也有一个 Name 查找的方法，find\_element\_by\_name()。

我们用 find\_element\_by\_name()来实现刚才的功能，再来看以下刚才的代码，搜索框中的代码存在：

```
<input type="text" class="s_ipt" name="wd" id="kw" maxlength="100" autocomplete="off">
```

这个 HTML 标签有个属性——name="wd"，很明显，这个 wd 就是我们所需要查找的。

我们考虑这一行代码来实现相同的功能：

```
driver.find_element_by_name("wd")
```

因为点击按钮所在的 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。

```
#导入库
from selenium import webdriver
import time

#打开浏览器和打开百度
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 查找并点击
driver.find_element_by_id("su").click()

time.sleep(2)
#退出浏览器
driver.quit()
```

同样，这段代码可以正常运行。

## 3.4 XPath 定位

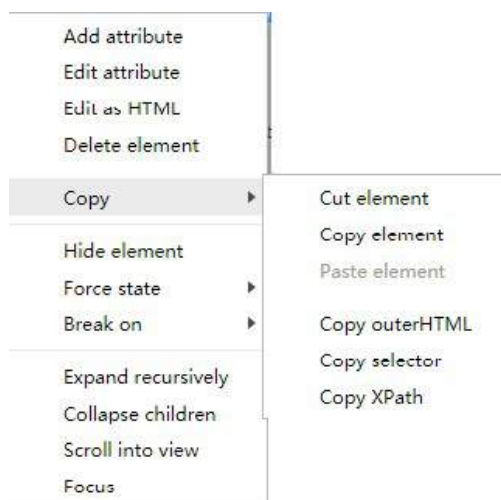
XPath 可谓是多种多样的，即便使用不同的方式书写，你也可以查找到相同的一个元素，比如说，我们来查找百度的搜索框，就有 7、8 种写法：

- ❑ `//*[@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，我们通过一个简单的方式来获取 HTML——浏览器的开发者工具。

首先，我们按照刚才的方式，打开浏览器的开发者工具，然后将其定位到搜索框上，

这时候,我们下方开发者工具应该已经定位到相应的代码上,我们选中这行代码(如果你的开发者工具的代码框被遮住的话,那么你可以通过上下滑动来适当的展开,或者



关掉弹出的 Console 和 What's New), 然后右键点击, 选中 copy, 这时候在弹出的二级选择框中, 点击位于最下方的 Copy XPath, 我们就可以获取到由机器生成的 XPath, 如图 3-4。

图 3-4 XPath 获取

在获取到 XPath 后, 我们来使用一下:

```
#导入库
from selenium import webdriver
import time

#打开浏览器和打开百度
driver=webdriver.Chrome()
driver.get("https://www.baidu.com/")

#通过 name 查找, 然后输入
driver.find_element_by_xpath(r'//*[@id="kw"]').send_keys("selenium")

#通过 id 查找并点击
driver.find_element_by_id("su").click()

time.sleep(2)
#退出浏览器
driver.quit()
```

有一点和刚才不一样, 读者最好将 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")
```

如果我们这样使用的话, 编译器就会告诉我们语法错误:

```
File "C:\Users\xuyichenmo\OneDrive\Documents\SELENIUM\test.py", line 10
    driver.find_element_by_xpath("//*[@id="kw"]').send_keys("selenium")
                                   ^
SyntaxError: invalid syntax
```

SyntaxError: invalid syntax

最好用刚才我们上文的写法

```
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, 也避免看着麻烦。

### 3.5 标签名(Tag Name)定位

标签是 HTML 页面中一个重要的组成部分，比方说，我们正在建房子，那么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，他们有的大小存在差异，有的颜色存在差异，但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子：

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

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，一个标签可以跟着多个属性，但只能有一个标签名。

接下来，我们来学习标签名定位，这种定位方法被称为最不靠谱的定位方法，因为一个 HTML 页面中，可能会有很多的相同的标签。比如说，我们常见的百度搜索，其实它并不是只有一个搜索框（这些搜索框各有各的作用，虽然他们采用了 Input 标签，但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，我们在这里将这些搜索框的 Hidden 属性删掉，如图 3-5 所示：



图 3-5 百度隐藏输入框

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，我们用到这个的机会并不多，很多时候，我们都会采用其他方式。

虽然明知道会失败，但我们还是来进行一次定位：

```
#导入库
from selenium import webdriver
import time

#打开浏览器和打开百度
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，我们来运行一下看看返回：

```
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
网页源代码中对应部分如图 3-6 所示

```

```

<input type="hidden" name="rsv_spt" value="1">
<input type="hidden" name="rsv_iqid" value="0xe99ab9ee00043c4f">
<input type="hidden" name="issp" value="1">
<input type="hidden" name="f" value="8">
<input type="hidden" name="rsv_bp" value="0">
<input type="hidden" name="rsv_idx" value="2">
<input type="hidden" name="ie" value="utf-8">
<input type="hidden" name="rqlang" value="">
<input type="hidden" name="tn" value="baiduhome_pg">
<input type="hidden" name="ch" value="">

```

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 input 标签，我们有理由推测 selenium 找到的是一个隐藏的输入框。

## 3.6 Class Name 定位

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

可以同时给一个 HTML 标签附上多个类，比如说，这样的写法：`<span class="left_menu important">`，同时将 left\_menu 和 important 赋给 span 标签（中间的空格叫做间隔符号，表示的是一个标签有多个 class 的属性名称）。

由于 Selenium 现在的版本已经不再支持复合类名（传输的参数是一个类名，而我们传入的是一个复合的类名），所以像这样的 `class="btn self-btn bg s_btn"` 我们是无法通过 Class Name 来查找的。

那怎么解决呢？

方法一：

只需要删去其他的 Class 类名，只传入一个参数，比如说，

```
driver.find_element_by_class_name("s_btn").click()
```

方法二：

我们可以结合刚才学到的 `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()

```

笔者在这里使用的是 `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，返回一下这个列表：

```
[<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 `selenium` 自身存储定位元素的方式保存，这样子的列表我们是难以直接使用的（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第一个匹配项的索引位置，然后将其赋值给一个变量。接着，我们从列表中找到这个变量值对应的内容，再使用 `click()` 方法。

分别针对方法一和方法二，我们写出两套程序：

#### #方法一

#常规的导入库

```
from selenium import webdriver
import time
```

#打开浏览器和打开百度

```
driver=webdriver.Chrome()
driver.get("https://www.baidu.com/")
```

#class 名查找搜索框

```
driver.find_element_by_class_name("s_ipt").send_keys("selenium")
```

#id 名查找搜索按钮

```
driver.find_element_by_id("su").click()
```

#退出

```
time.sleep(2)
driver.quit()
```

还有另一种方式：

#### #方法二

#常规的导入库

```
from selenium import webdriver
import time
```

#打开浏览器和打开百度

```
driver=webdriver.Chrome()
driver.get("https://www.baidu.com/")
```

#class 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 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 选择器也有很多种书写定位的方式，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]"
- ☐ 笔者在这里为大家讲解几种比较方便的书写方法：
- ☐ [name=wd]表示选择所有 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，你只能选择到一个）
- ☐ 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，再以此类推，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"像这种则是复合写法，首先查找到 class 属性中含有 soutu-btn 的 span 标签，然后进去 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：

```

#常规的导入库
from selenium import webdriver
import time

#打开浏览器和打开百度
driver=webdriver.Chrome()
driver.get("https://www.baidu.com/")

#css 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 Link Text 定位

虽然介绍了这么多定位方法，但在笔者看来，id 和 class 定位最为方便，而下面介绍的这种定位方法最为实用。下面我们换一个目标，不再是搜索框和确认按钮，转到右上角的链接上。

Link text 定位方法定位的是带有超链接的文字。

跟前面几种方法类似，只需要将文字放入传入参数的位置即可：

```
find_element_by_link_text(" [text] ")
```

代码如下：

```
#常规的导入库
from selenium import webdriver
import time

#打开浏览器和打开百度
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 Partial Link Text 定位

部分文字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，当然，如果文本很长，而读者不想要打入全部的字符，那么读者就会用到这个方法。

```
find_element_by_partial_link_text(" [text] ")
```

参数的字符串可以简单到只有一个字。如果读者乐意打上全部的字符也是可以的，集合本身也是集合的子集嘛，如果读者乐意每次都多键入几个字母的话，那么完全可以放弃上面的 Link Text 方法。

与上面的代码类似：

```
#常规的导入库
from selenium import webdriver
import time
```

```
#打开浏览器和打开百度
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

我们通常从 HTML 代码中寻找所需要的数据确实是一个较为普遍适用的思路，但一旦面对十分复杂的页面，使用 JS 渲染，这项工作就变得繁重而富有挑战性，需要我们去一个个分析后台请求。脚本式的操作又会干扰我们的正常操作。

所以，我们需要有一些好用的工具来帮助我们像浏览器一样渲染 JS 处理的页面。Phantomjs 便是这样一个工具，模拟用户手动去摸索 JS 渲染的到的一些结果的隐形浏览器。

### 4.1 PhantomJS 是什么

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，需要搭配 chrome 等第三方的浏览器使用，目前它支持 Chrome 浏览器，Firefox，opera 等主流浏览器。如果你使用这些浏览器，你可以明显的看到一个个页面被打开，然后执行你设定的操作。

但我们不总希望自己在用 selenium 编写的程序的时候什么都不干，不希望程序来占用电脑的操作，这时候，你可能需要的是它——Selenium + PhantomJS。

PhantomJS（官方网站 <http://phantomjs.org/>）便是这样一个工具，协助我们去应对布局，元素多样化的网页。PhantomJS 是一个无界面的，可脚本编程的 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，JSON 等等。

phantomjs 主要应用场景是：

- ☐ Web 测试
- ☐ 自动化页面访问
- ☐ 屏幕捕获



## ❑ 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：完整的网络堆栈，不需要浏览器。PhantomJS 是一个 JavaScript API 的无头 WebKit 脚本。它支持各种网页标准，包括 DOM 处理，CSS 选择器，JSON，Canvas 和 SVG。）

## 4.2 PhantomJS 下载与安装

我们可以从官方网站（<http://phantomjs.org/>）获取 PhantomJS，点击位于页面中央的 DOWNLOAD 按钮会跳转到 <http://phantomjs.org/download> 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 Windows

截止到目前（2018.2.4）PhantomJS 在 windows 上的最新支持版本为 2.1.1，在这里，推荐读者使用最新版本的 PhantomJS，官方文档中有提到说：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。

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

TIPS: 添加到环境变量的方式，在电脑桌面上——右击——我的电脑——选择属性——高级系统设置——高级——环境变量——选择系统变量中的 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，直接点击新建即可。

### 4.2.2 Linux

在 PhantomJS 的官方页面左上角有一个“fork me on GitHub”的超链接，通过此读者可以直接 PhantomJS 位于 GitHub 的仓库，下载已经编译好的二进制文件安装包。截止到目前（2018.2.4）PhantomJS 在 windows 上的最新支持版本为 phantomjs-2-1-3。

- ❑ 读者也可以直接在 download 页面下载对应的“.tar.gz”压缩文件。
- ❑ 下载 phantomjs-2-1-3-linux-x86\_64.tar.bz2。
- ❑ 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”。
- ❑ 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。
- ❑ 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。自己编译所需要的时间长，而且需要挺多的磁盘空间，以笔者为例子，开四个并行的进程进行编译工作，花费了超过 20 分钟的时间，因此由源码编译 PhantomJS 会花费很长的时间。在这里我们推荐读者采用 plan B，直接下载二进

制文件然后安装。待读者熟练了，可以采用这种较为复杂的方式练练手。

### 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，selenium2 开发性的将两者结合，解决了原先难以解决的问题，极大的方便了使用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 语言编程并开发的）。

selenium 是 python 的一个第三方自动化测试库，其是测试库，却也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，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 无头浏览器（headless browser），也称无界面浏览器，不需要界面的同时占用的内存也相对较小，更适用于大规模多进程爬数据（开几十个 Chrome 进程爬数据，那真是内存噩梦！）。

简单容易上手的 selenium 库，是爬动态网页的杀手级武器，但对 PhantomJS 浏览器的支持还不是特别完善。当然，了解存在的问题，并找到对应的解决方法，就能发挥 PhantomJS 的威力了。

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，下面我们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，首先我们创建 PhantomJS 浏览器对象，然后打开 Bing 搜索，首先打印出页面标题，然后在搜索框中输入字符串，截图，点击搜索，截图，点开搜索结果的第一个页面，截图如图 4-1 所示。

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有浏览器页面窗口弹出操作，占用顶层窗口，也不需要获取焦点。

你可以把下面的 Python 代码手打一遍到你的编辑器中（自己写一遍程序远比复制粘贴效果好），读者可以一行一行代码添加删除注释，参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 的实现效果和用其他的 webdriver 支持的浏览器运行效果的区别与相似之处。

图 4-1 是本程序的流程图：



图 4-1 流程图

```
# 导入 webdriver
from selenium import webdriver
import time

# 要想调用键盘按键操作需要引入 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被完全加载，
# 然后才会继续程序，通常测试会在这里选择 time.sleep(2)
driver.get("http://cn.bing.com/")

#打印页面标题
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")

#点击搜索按钮
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")

# 进行当前页面点击第一项
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()

```

这一句需要说明一声：

```
driver = webdriver.PhantomJS()
```

该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，这里应该指定为你 phantomjs 的安装路径，直接指向 bin 文件夹下的 exe 文件，使用如下语句：

```
driver = webdriver.PhantomJS(executable_path="【path】")
```

Selenium 中含有两种退出方法，分别是 close 方法和 quit 方法，区别在于关闭 driver 时使用了 driver.close()，close 方法关闭后并不会清除临时文件中的 webdriver 临时文件。用 driver.quit()，quit 关闭浏览器后，会自动删除临时文件夹。

在这里，我们借用 selenium.webdriver.common.keys 库来输入字符，效果图 4-2 中展开的选择框所示：



图 4-2 测试效果图 1

然后，我们使用 click() 方法，搜索该字符串，效果如图 4-3 所示：



图 4-3 测试效果图 2

接着，仍然使用 `click()` 方法点开了第一个搜索结果，因为并没有重定位到新打开的页面，所以还是显示在搜索结果的页面，但搜索结果已经点击过了，如图 4-4。



图 4-4 测试效果图 3

## 4.5 `time.sleep()`慢下来

在做统计学采集的时候，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，抓取不出来的令人崩溃的问题，可能你向服务器提交自认为已经处理的相当完美的表单却被拒绝，可能你的 IP 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，当你使用一个比普通人正常浏览快得多的频率来访问网址时，很大的概率你会被网站查封。

建议你使用一行代码：

```
time.sleep(2)
```

这样做，有助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安全措施，但为了自己获取数据而不惜拖垮网站，这是一件有伤人和的事情。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。

笔者在这里希望各位读者能够尽量把采集的时间放到夜晚，原因有很多。

- ❑ 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 个小时完全足够你采集到足够的数据。
- ❑ 网站访问的高峰期一般是在白天，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。
- ❑ 没有一项任务是可以在你不停的查看程序运行状况下完成的，想想看，当你从睡梦中醒来，在你的电脑中静静的躺着你所需要的数据，这是一件多么惬意的事情
- ❑ 还有很重要的一个原因：夜间出没的生物包括以下几种——吸血鬼、僵尸、程序猿。程序猿是一种夜习性动物

## 4.6 PhantomJS 命令行指令参数

在本小节笔者简单说明了了 PhantomJS 大部分的命令行执行参数一级使用用法，以方便读者后续查阅使用：

- ❑ `-help` 或 `-h`，这个命令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]`，这个命令用来开关调试模式，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`，这个命令允许你指定 `json` 格式化的配置文件
- ❑ `-cookies-file='[path]'`，使用该参数，你可以指定一个用来存储当前 `cookie` 信息的文本，路径应该是像这样的 `"/path/to/cookies.txt"`
- ❑ `-disk-cache=[true/false]`，指定是否启用硬盘缓存（在桌面服务存储位置，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]`, 你是否忽略 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]`, 确定你是否加载所有内联图像, 默认开 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), [yes/no]也可
- ❑ `-ssl-protocol=[sslv3|sslv2|sslv1.1|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
- ❑ `-local-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

下面挂一漏万的列举几个在 Selenium + PhantomJS 的组合使用过程中可能遇到的问题, 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 python2 使用 selenium 的 `send_keys()` 中输入中文会报错, 其实在中文前面加一个 `u` 变成 unicode 就能搞定了

### 4.7.2 不同 frame 间的转换

可能读者可以在页面源码确实发现某些元素, 但通过 selenium 的元素查找方法 `find_element_by_xpath()`, `find_element_by_id()` 等类似定位函数无法获得该元素对象, 比如说提示 “unable to find element with id 'kw'” 的错误。那么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:

```
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, 如果想要回到默认内容 (默认的 frame), selenium 中有一个函数是专门为这种情况准备的:

```
driver.switch_to_default_content()
```

### 4.7.3 PhantomJS 进程不自动退出

主程序退出后, selenium 不能也无法保证 PhantomJS 也成功退出, 笔者在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免除这个麻烦:

```
driver.quit()
```

## 第5章 实战 part 1——Python 官网

在前面的学习中，读者已经基本了解了 selenium 的大部分与页面进行交互的操作，甚至我们还讨论过关于的网络爬虫的法律与道德的问题，在这里，现在，我们对以上所学进行一个综合的运用，进一步提升读者对 selenium 的理解程序，同时，也是刚刚学过的内容进行一项复习。

光说不练假把式，下面的代码，笔者在这里建议读者在学习的过程中，对着电脑，自己亲自练习一遍，如果只是单纯的看看书，虽然进展比较快，但对于实际编程的效果却并不好。

### 5.1 Selenium 访问 Python 官网

在开始正式的实战面前，我们先热个身，键入几行代码先打下一个基本的模具，为后续编写做一个基础。

这里用到的内容，主要实在第二章介绍的，我们常规的导入库，启动浏览器，然后打开页面，最后退出。

为了易于键入，也为了节省读者的时间，在以后编写的 selenium 程序中，我们将 driver 简写为 dr。

```
#常规的导入库
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，可以说，如果你想要使用 selenium 编写一个程序，那么你可以想都不用想，直接先敲出这几行代码。

### 5.2 通过 JavaScript 修改官网标题

JavaScript 并不是我们学习的重点内容，它的一些关于浏览器的操作读者也可以在搜索引擎中查到。所以笔者在这里给修改页面标题和弹窗的 JavaScript 代码，笔者的想法是通过这两个代码，让读者练习使用 selenium 调用 JavaScript。

```
#修改页面
JS1="document.title='xxxxxx';"

#弹窗
JS2="alert($(document).attr('title'));"
```

在这里，给出我们需要做的内容，希望读者看到这里的时候，不要直接去看下面的代码，先自己想一想应该怎么做，键入几行代码实验一下，整个实现流程如图 5-1 所展示。

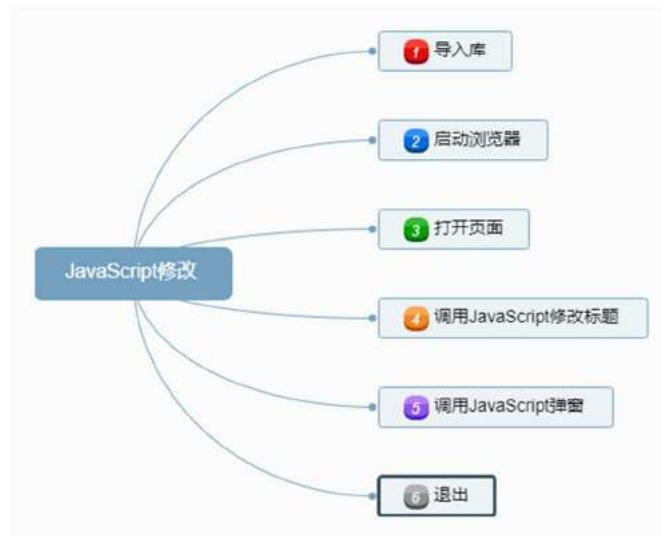


图 5-1 修改标题思路

```

#常规的导入库
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

然后我们来练习一下在第三章学习的元素定位。在 python 官网的搜索框进行一次搜索，首先，我们来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，然后点击。

```

#常规的导入库
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，我们首先通过 `get()` 方法跳转到页面 `www.python.org`，接着，通过执行 JavaScript 修改页面的标题，然后，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，点击第二个元素，实现搜索功能。

## 5.4 获取 latest news 部分

在 5.3 写的代码的基础上进行一定程度修改，我们来进行一个截图，截图获取关于 `pycon` 的最新消息。

Selenium 的截屏功能的调用代码为：

```
dr.save_screenshot("save.png")
```

先来整理一下现在已经做到的功能的思路，如图 5-2：

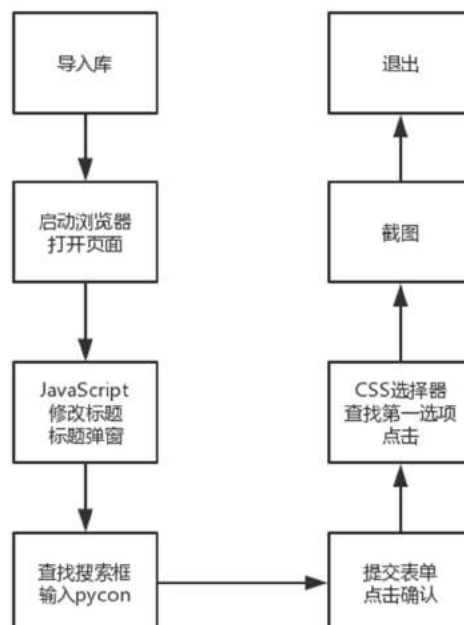


图 5-2 流程整理

```
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
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成就执行下一步出现了错误。

selenium 的等待有很多种方式，除了在上面我们经常用到的 `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，我们来讨论一下这三种等待的区别。

`time.sleep()` 这种等待方式也叫做强制等待，换句话说，就是不管浏览器加载完了与否，我们都要等待特定秒数。做个比喻，笔者和读者要一起出发去公园玩，那么笔者提前到了，但笔者必须等到 9 点再出发，即便读者已经到了，我们两个还是要等到某个时刻出发。

隐式等待则和上一种方式有很大的不同，笔者和读者去公园，笔者 8:50 分到公园门口，那么只等读者 10 分钟，如果到了点读者还没有来，笔者就先自己进去了。如果读者在 8:50 到 9 点的时间内到来了，那么笔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，笔者在公园等待读者，读者因为路上堵车，无法准时到达，于是给笔者打了一个电话，笔者就自己先进去了。

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成，但可能在某些网站上，某个 JavaScript 脚本加载特别慢，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完成才执行下一步。而显式等待只需要判断某个特定的元素出来即可。

我们来对同一个操作用三种方法同时书写一遍：

```

#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成，程序都会等待 2 秒，然后程序继续执行后面的代码，这种方法很适合调试的时候使用，虽然编程的时候很方便，但在运行的时候过于死板，十分影响程序执行速度。

```

#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) # 隐性等待，最长等 30 秒
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，隐性等待设置一次对整个程序运行过程全部起作用，只要设置一次即可，不需要在每次都书写一下。

这是一个显式等待的例子，另外在使用显式等待的时候，读者还需要多导入几个库：

- ☐ selenium.webdriver.support.wait
- ☐ selenium.webdriver.support
- ☐ selenium.webdriver.common.by

```

#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()
```

我们来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() 可选的参数有四个，但一般情况下，我们只会用到其中三个，第一个参数设置的是打开的浏览器，第二个参数设置超时时间，第三个参数设置的是检查频率，比如说，刚刚我们写的是 0.5s，那么就每隔 0.5s 进行一次检查。我们来看一下 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，可供选择的除了 until() 还有一个 until\_not()，这两个选项的使用方法都是差不多的 (method, message=" ")，message 返回的是如果返回结果为 0（即失败）后弹出的信息，不同的是 until 检查的是直到返回值为 True，until\_not 直到返回值为 False。

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

```
#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()
```

可能有细心的读者看到了，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。

我们中间用到了 EC，其实是 ExpectedCondition，EC 的方法其实不只是这一种，还有需要，用法都和上面类似，我们以表格的形式罗列出来，如表 5-1 所示：

表 5-1 ExpectedCondition 方法

alert_is_present	判断页面上是否存在 alert
element_to_be_selected	判断某个元素是否被选中了
element_located_to_be_selected	判断被定位的元素是否被选中
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,上面方法传入 element,这个传入 locator
element_to_be_clickable	判断某个元素中是否可见并且 enable
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	判断某个元素是否被加到了 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, 这个方法也是返回 True 或 False
title_is	判断当前页面的 title 是否等于预期
title_contains	判断当前页面的 title 是否包含预期字符串
text_to_be_present_in_element	判断某个元素中的 text 是否 包含 了预期的字符串
text_to_be_present_in_element_value	判断某个元素中的 value 属性是否包含了预期的字符串
invisibility_of_element_located	判断某个元素中是否不存在于 dom 树或不可见
visibility_of_element_located	判断某个元素是否可见, 传入 locator
visibility_of	同上, 传入 element
注释: visibility 表示可见代表元素非隐藏, 并且元素的宽和高都不等于 0	

## 第 6 章 实战 part2——今日头条

上面的实战部分, 可以说是近似于复习课。接下来, 我们来进行一个稍微复杂一些的 selenium 爬虫——今日头条。

可能有读者就要问了, 像今日头条的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? 其实不然, 在这个大数据的时代, 社会化信息网站虽然可以满足人们对信息的共性需求, 让人们看到, 但仍然存在很大的不足, 它无法针对每个人的个人需求, 进行数据的定制化操作, 在工作中, 你可能经常会碰到如果我们需要收集特定的信息, 而这里信息分布几百个不同的页面上, 我们通过人工来劳动不仅枯燥无味, 而且极其容易出现数据的分类, 整理错误, 眼睁睁的看着就在页面上的数据, 却无比烦躁, 恨不得它们可以自己主动以某种形式呈现。

我们先来确定一下本章的目标, 首先, 我们先通过百度热词来获取当前的实时热点, 然后将百度热词作为关键词在今日头条搜索, 然后保存相关信息。

### 6.1 Selenium 访问百度热词

我们的目标网站为百度搜索风云榜, URL 链接为 (“ <http://top.baidu.com/> ”), 先来打开网站观察一下, 我们的目标在这里——实时热点, 如图 6-1:

① 实时热点		② 七日关注
排名	关键词	搜索指数
1	手指被可乐炸骨折	450358
2	华为首次超越苹果	410689
3	工商局调查拼多多	395317
4	凌晨上班连车祸	301883
5	华夏李君殴打员工 新!	273762
6	重庆再字航被查	268888
7	美军遗骸回国仪式	255165
8	董明珠回应银隆 新!	241893
9	詹姆斯免费学校	228845
10	18岁女孩被烧伤 新!	222246
		完整榜单

图 6-1 实时热点

获取元素，肯定要首先进行元素定位，F12 打开开发者工具，我们对这几条消息进行定位，右键，获取 XPath，然后读者就会惊讶的发现，他们的 XPath 都很类似，不同的只是中间一个叫做 li 的标签，那么就意味着，我们只需要更改这里 li 标签对应的数字，就可以相应的完成对所有 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 中就是用来定义列表项目的，这些热词都在一个<ul>下面，在<ul>列表，所有元素都是采用<li>标签进行并列的，下面所以他们的标签其实也很相像，主要的区别就在<li>标签的序号上，如图 6-2 所示。

```

▼<ul id="hot-list" class="list">
  ▶<li>...</li>
  ▶<li>...</li>
  ▶<li>...</li>
  ▶<li>...</li>
  ▶<li>...</li>
  ▶<li>...</li>
  ▶<li>...</li>
  ▶<li>...</li>
  ▶<li>...</li>
  ▶<li>...</li>
  </ul>

```

图 6-2 <ul>和<li>标签

然后将 XPath 分割为三个部分，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]'
```

可能有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 到 9。

在整合完 XPath 后，我们通过 XPath 查找，然后获取对应的<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：

```

<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&tn=SE_baiduhomet8_jmjb7mjw&rsv_dl=fyb_to

```

```
p&fr=top1000&wd=%BB%AA%CE%AA%CA%D7%B4%CE%B3%AC%D4%BD%C6%B  
B%B9%FB"  
href_top="/detail?b=1&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敲代码了。

```
#常规导入库  
from selenium import webdriver  
  
#打开浏览器和网页  
dr=webdriver.Chrome()  
dr.get('http://top.baidu.com/')  
  
#进行 xpath 整合  
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 selenium 搜索相关热词

刚才我们获取到的关键词是打印出来，我们需要调用这些内容，那么就不能再是打印了，我们考虑将获取到的内容存在字典中。

```
hw_dict=[]  
value=dr.find_element_by_xpath(xpath).get_attribute('title')  
hw_dict.append(value)
```

如果仅仅只是要只获取一条最热新闻，那么这样子写是完全足够了，但我们刚才获取到的新闻一共有 10 条，那么，我们需要将这些代码嵌套进 for 循环中，进行循环存入。

然后，既然我们已经获取到了关键词，那么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 为关键词进行搜索看一看，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
```

我们注意到在 URL 中同样有我们刚才搜索的关键词 selenium，再把关键词换为 webdriver，

页面的 URL 则变成了

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

那么，很明显，我们搜索的内容是通过访问的网址的关键字不同而不同（keyword="xxx" 部分的作用为传递参数）。

那么我们就可以这样进入新的页面了，直接进行刚才获取到的关键词搜索：

```
#常规导入库
from selenium import webdriver
dr=webdriver.Chrome()
dr.get('http://top.baidu.com/')

#建立空列表
hw_dict=[]

#列表添加内容
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)
```

结果如图 6-4：



图 6-4 热词搜索

很好，我们成功的做到了搜索。

## 6.3 获取第一条结果

在搜索结果结束了以后，我们来进行下一步，模拟用户操作，进行第一条新闻的点击跳转。

跟刚才一样，查找元素，我们先看一下第一条结果的能够触发点击操作的最下一层标签（笔者在这里选择的是文字内容），

```
<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
```

然后就很随意了，学到这样，相信读者已经能够轻松的写出查找元素，模拟点击操作的代码。

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？程序运行太快，网页跟不上，经常是程序在查



找第一个元素的时候，网页仍然尚未加载出来，然后程序就会报错，等网页加载出来了，程序也已经退出了。

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，我们使用在第五章学习到的显式等待

显式等待需要虽然只有一行代码，当需要同时导入多个库，这几个库在第五章提到过：

```
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
```

同时，显式等待是一种条件判定执行，所以我们可以和刚刚想要实现的代码功能结合起来，

```
#常规导入库，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=[]
#列表添加内容
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)

#最火关键词搜索
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'

#显式等待调用
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，我们可以安心的进行下一步实战了。

## 6.4 元素定位——抓取内容

然后是下一步，打开页面，我们就要获取页面的内容，再来观察一下 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>
```

发布者，时间所在的标签：

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

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，句柄也是你需要在 Selenium 爬虫中注意到的地方，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，那么你这时候再使用元素查找，毫无疑问，很难成功，因为你查找的是旧的页面。

我们可以使用如下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，你如果查到[-1]，那么就代表你要找到刚刚打开的页面，切换句柄我们会用到这个函数：

```
dr.switch_to_window(all_handles[-1])
```

传入的参数是读者你获取到的所有句柄列表中的一个元素。

由此，因为后面还会重复用到，我们编写一个 `getinfo()` 函数，来获取文章的发布方，发布内容，标题。

```
#常规导入库，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

#我们提前写明了函数
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=[]

#获取最火关键词 TOP10
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)

#获取并搜索最火关键词
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()

#给页面留足时间加载，同时也为了防止被禁 IP
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 数据存储

在第六章第四小节中，我们写的程序已经比较成熟了，但当我们需大规模的数据进行分析的时候，这样子打印出来很明显是不行的，我们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 格式文件我们将会在下一章讲解到，读者可以在学完第七章的 CSV 格式读写后，再回来看一看，会有新的体验。

在开始 6.5 以前，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，所以，我们还需要一个能够模拟下拉的方法，来为我们进行排名靠后的内容的加载。

让人愉快的是，Selenium 已经为我们提供了相应的方法，模拟键鼠操作的方法，你可以从 ActionChains 库中调用：

```

from selenium.webdriver.common.action_chains import ActionChains
包含的关于鼠标的操作方法：

```

- 'click(on\_element=None)', 鼠标单击
- 'click\_and\_hold(on\_element=None)', 鼠标单击后不松开
- 'context\_click(on\_element=None)', 右键单击
- 'double\_click(on\_element=None)', 双击
- 'drag\_and\_drop(source, target)', 拖动到某个元素
- 'drag\_and\_drop\_by\_offset(source, xoffset, yoffset)', 拖拽到某个坐标然后松开
- 'release(on\_element=None)', 释放鼠标左键
- 'move\_by\_offset(xoffset, yoffset)', 鼠标移动到某个坐标

- ❑ 'move\_to\_element(to\_element)' 鼠标移动到某元素
- ❑ 'move\_to\_element\_with\_offset(to\_element, xoffset, yoffset)' 移动到距某元素左上方多少坐标距离的位置

包含的关于键盘的方法:

- ❑ 'key\_down(value, element=None)', 按下某个键盘上的键
- ❑ 'key\_up(value, element=None)', 松开某个键
- ❑ 'send\_keys(\*args)', 发送
- ❑ 'perform()', 执行已经添加到生成器的操作

举个例子,如果你没有目标元素,只是单纯的想要在打开某个网页后按下某个键,那么,你可以这样写:

```
ActionChains(dr).key_down(Keys.DOWN).key_up(Keys.DOWN).perform()
```

但如果你还想要向某个表格里写入表单,那么你需要这样写

```
elem=driver.find_elements_by_id()
```

```
ActionChains(dr).key_down(Keys.DOWN,elem).key_up(Keys.DOWN,elem).perform()
```

这种写法叫做链式写法,此外还有另外一种写法被称作分布写法:

```
elem=driver.find_elements_by_id()
```

```
ActionChains(dr).key_down(Keys.DOWN,elem)
```

```
ActionChains(dr).key_up(Keys.DOWN,elem)
```

```
ActionChains(dr).perform()
```

无论哪一种都是可以的,看读者更偏向于哪一种写法。

另外,我们用了这样一行代码

```
for i in range(2000):
```

虽然我们设置 0 到 1999, 但并不意味着需要这么长时间, 这是 Python 计数 2000 需要的时间, 笔者在这里使用和 6.5 不同的代码实现打开页面, 6.5 是打开新页面, 切换句柄来获取内容, 而在这里, 我们在搜索之后, 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, 加载过慢, 或者出现其他意外, 我们使用 try, except 来把万一出现的错误吞回, 避免因为个别报错而让数据采集停止。

```
#常规导入库
```

```
import csv
```

```
import time
```

```
from selenium import webdriver
```

```
from selenium.webdriver.common.action_chains import ActionChains
```

```
from selenium.webdriver.common.keys import Keys
```

```
#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)

#给页面留下足够的时间加载
time.sleep(1)

#刷新出足够的条数，保证加载到底
#这里循环次数尽量大
#经过测试，大概 1600 左右可以到底
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')

#再次留下时间加载，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 来接收报错
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，读者你就可以惬意的泡一杯咖啡，我们来让程序飞一会儿。如果读者不想要一个打开的浏览器窗口的话，那么可以用到我们在第四章讲解到的 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 数据编解码，处理

如果你使用 API 的话，那么很大概率的你会喜欢 API 那种整洁明了的已经处理好的数据，笔者也是，但遗憾的是，在实际工作中，API 的使用总是少数的，大多数时候，都需要我们自己来进行数据的挖掘、整理、分析。

既然是和互联网打交道，那么就难免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的数据有不同的保存方式，比如说有的数据类型是为了节省储存空间，有的则为了使用方便等等，这些数据的编解码问题，往往让人头疼，经常在你意想不到的地方跳出来。

接着，在本章我们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显得很炫酷，但随着数据越来越多，这样子总归是不行的，就好像你可以轻松打开一个包含一篇文章的 txt 文件，但是一部几百万字的小说，你却无法立刻打开它。

### 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，将会学习到 CSV 文件读取和应用，当然，CSV 文件只是一种简单的数据储存方法，还有较为复杂的关系型数据库和非关系型数据库。

CSV (Comma-Separated Values, 逗号分隔值)，顾名思义，就是数据之间采用逗号进行数据分隔，你可以直接在 txt 中编辑，这也意味着它是一个纯文本的文件（只能是数字和文字，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，分别代表着读和写。

假设你有一份 city.csv 文件（读者可以在本书代码托管的网站上下载到），文件的类型像这样：

```
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,Ai 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，由于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，这些都是我们处理数据的大问题，部分可能甚至在 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)
```

#这里通过对元素遍历来逐行输出

```
for i in csv_file:
    print(i)
    print(line)
```

输出如下：

```
['\uffeff 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，那么一定有写入，我们来做一个例子，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"]
```

这是我们的样例代码

```
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)

# 写入的内容都是以列表的形式传入函数
writer.writerow(header)
writer.writerow(file_1)
writer.writerow(file_2)

#关闭
csvFile.close()
```

写入后，文件的内容如下，效果看着不错。

```
name,gender,score
zhang_san,male,100
Li_si,male,80
```

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

截止到第七章之前，我们学习和处理的数据大都是有规则的，简单的并且轻量级，但是，当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，所以，你完全可以考虑用 CSV 做一个轻量级的数据库。

## 7.2 读写 JSON 文件

JSON(JavaScript Object Notation)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，是理想的进行数据交换的格式。

JSON 建构于一种“名称：值”的集合，这和 Python 的字典格式非常类似。其实是因为 JSON 使用了类 C 语言家族习惯（包括 C，C++，Java，Python，Perl 等语言）。如果读者使用过 API，那么你会惊讶的发现，绝大部分的 API 返回的数据都是 JSON 格式的。

其实 JSON 格式和我们下面要讲解的 XML 格式有很大关联，他们的关系更像是老牌强者和后起之秀，本来 XML 一家独大，后来 JSON 半路跳出来，逐渐占了上风。

和 XML 相比，JSON 具有以下特点，所以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，节省宽带
- ❑ C 语言家族习惯，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，你完全不必耗费时间去安装它。

JSON 库是主要含有四个方法，分别是 loads(), dumps(), load(), dump()，其中 load()和 dump()和文件的读写相关。

dumps()方法将 dict 转化成 str 格式，loads()方法将 str 转化成 dict 格式，这是 dumps()，loads()方法的使用案例：

```
#导入库
import json

#这是我们提供的 JSON 格式示例数据
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))
输出如下：
```

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



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

我们可以看到两次输出的内容相比，但最后附加的格式却大不相同，第一个文件编译器告诉我们，<class 'str'>说明它是 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他们是等价的，但在编程语言中，你难以做到对一个字符串做类似于字典的操作。

此外，dump(), load()的作用和 dumps(), 以及 loads()作用相同，只是这两个是把文件的转换和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 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，第一二次都有输出，那么确实是我们刚才的 JSON 数据通过 `dump()` 方法写入了文本。

可能读者在奇怪，笔者为什么要重复多次关闭和打开呢，这不纯粹是在刁难人吗？其实不然，在 `tip1`，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()` 去读取的话，你会什么也读取不到。

在 `tip2`，你不可以同时使用 `f.read()` 和 `json.load(f)`，否则会这样报错：

```
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，但这种可扩展标记语言更加容易让人们理解和掌握，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，它继承了通用标准标记语言的大部分功能，技术实现却要简单的多，但和我们上文提到的 JSON 格式来比，XML 格式的数据在存储空间有效比重上却不太占优势。

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，大标签中含有小标签，凭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 文件转换的可以被人类读取、理解的字符形式的代码。

我们在这个案例中，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，然后对 XML 输出，第一次输出程序返回的是一个内存地址，我们的内容存储在这里，第二次我们转换为字符串输出，然后我们对最外层设置了一个属性，然后再次输出。

```
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)

    #双值循环取出字典中的 key 和 value
    for key, value in dict.items():

        #打印
```

```

        print(key,":",value)

        #相当于为 key 建一个标签
        xml_elem=Element(key)
        #将 key 的值赋给这个标签的值
        xml_elem.text=str(value)

        #添加进 XML 文件
        elem.append(xml_elem)
    return elem

#字典转换
dict_convert = convert("test", dict)
print("-----")
print(dict_convert)

#这是转换为标准可读的 XML
print(tostring(dict_convert))
#设置最外层标签属性
dict_convert.set('id_number','01234567890')
#再次输出
print(tostring(dict_convert))
输出如下:

```

```

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</p
hone_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 16 进制的内存地址，而第二次和第三次输出的区别就在于 id\_number="01234567890"，这是我们设置 id 后修改的内容。

另外有一点需要说明一下，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不同，你需要用到这一个库：

```
from xml.etree import ElementTree as et
```

常规的导入库后，我们来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
{}
```

在这里，我们用到和上面相同的 XML 字符，然后用到了 `formstring()` 方法。

Python 的 XML 库读取文件有两种方法，其中一种就是我们刚才使用的 `fromstring()` 方法，它的作用就和它的字面意思相同，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`，用来获取根节点，`root.attrib` 则可以获取根节点的属性，因为我们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。

下面一个例子还会用到 `.attrib` 方法，多个例子来证明，这样子更有说服力：

```
>>> 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` 返回的不是一个字典，而是一个内存地址），如果你遍历字典，字典中存储的内容我们不能直接通过指定字典中的序数来得到，同样的，你需要 `xml` 的 `.text` 方法，使用这个方法，你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
```

第一个遍历我们很无奈的只获取到了这些元素存储的内存地址，当我们使用 `.text` 方法后，第二遍就输出相当于字典中 `value` 的内容，当然，如果你想组合成类似于字典的 `key:value` 形式，你可以这样用，`i.tag`，`":"`，`i.text`。

我们这里用不同的方式来将 XML 文件转为 `key, value` 这样的键值配对的形式，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。

```
#常规的倒入库
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>'
```

```
#从文本中获取 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 编码 BASE64

Base64 的名字在一定程度上解释它的形式，Base64 基于 64 个可打印字符来表示二进制数据。至于它的 64 怎么来的呢？Base64 编码每 6 个比特为一个单元，而  $2^6$  为 64

Base64 主要应用于对网络上 8 比特字节码传输的编码（例如电子邮件），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是采用 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()。

```
#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)

#打印
print(encoded)
print(decoded)
输出的结果如下
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、人工智慧等多个领域的科学,研究人与计算机之间用自然语言进行有效通信的理论方法,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。

自然语言处理是很大的一门范畴和学问,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,笔者并不是想让读者进行非常高深学问的学习,读者只需要掌握我们用到的可能性较大的方法。

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 Apache2.0 开源协议免费发放。

NLTK 是一个第三方包,你需要通过 pip 或者其他的包管理器更或是源码进行安装,但仅仅是安装 NLTK 库,那么它的大部分方法你都无法使用。NLTK 库中,自带一个函数 download(),会打开一个类似于图 7-1 的界面。

```
import nltk
nltk.download()
```

所以你可以使用这个方法下载它绝大部分的库,但遗憾的是,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,甚至是更长的时间,你在下载的时候还可能会碰到传输失败等等问题。

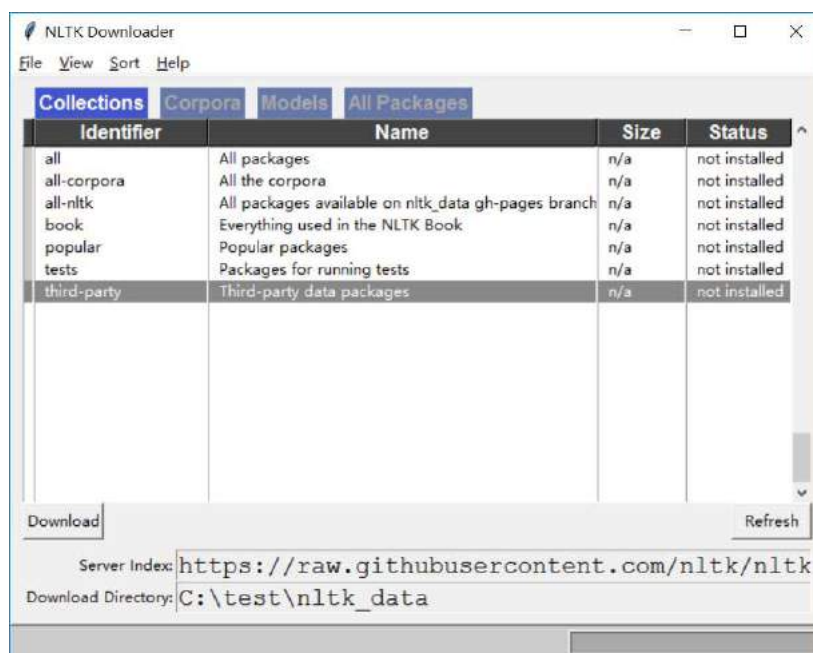


图 7-1 nltk.download()

如果不想重复的打开这个有点单调的页面（作为一个极简主义者，笔者还是觉的使用命令更加炫酷），那么你可以用这样子一条命令

```
nltk.download('包名')
```

下载完成之后，你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\\xuyichenmo\\AppData\\Roaming\\nltk_data'
```

另外部分包是压缩包，需要你解压，解压的时候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 包无法调用 nltk\_data。

当你很自信的进行已经整理好 nltk\_data，并且能够正常使用，那么你可以使用这个命令来确认

```
from nltk.book import *
```

返回的结果大致为这样。

```
*** 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

与其称之为书，笔者觉得称之为数据集更合适，这些 text 从 1-9 每个都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）是一本书叫做 “Moby Dick” 的书，而 text6 则是一部 1975 年的讽刺喜剧。

它们主要是供我们做分析测试使用，在本节的学习中，我们将主要使用 text1 和 text3 来进行自然语言处理。

首先我们来看一个命令：

```
>>> text1.concordance('different')
```

这句话实现的是从这一大串字符串中找寻出包含 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
```

统计标点符号和单词的出现频率，这个命令可以说是在我们进行数据处理的时候相当常用了。

```
>>> 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)
```

我们来分析一下这几行代码，第一行常规的导入要使用的库，第二行我们使用 FreqDist() 方法来格式化 text1，将其排列为一个按照分割后单词（标点）及其使用频率的列表，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])我们用于获取这个字典最前面的十个字符，也即为最常用的单词。

fdist.plot([number])则可以将我们数字的前多少个最高频单词在文本中出现的位置绘制成图（用到这个绘图功能之前，你需要首先安装 NumPy 和 Matplotlib 包）



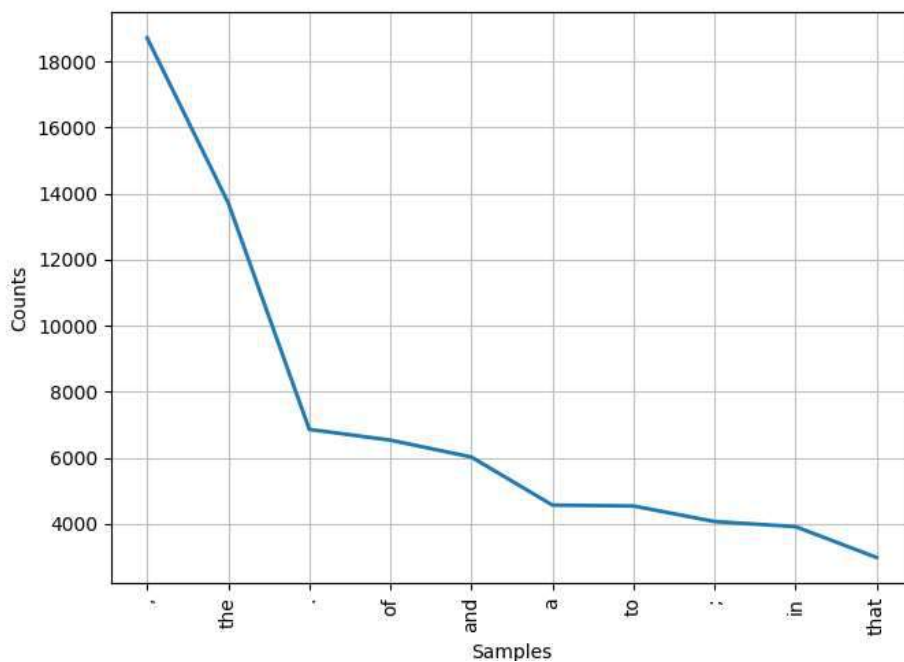
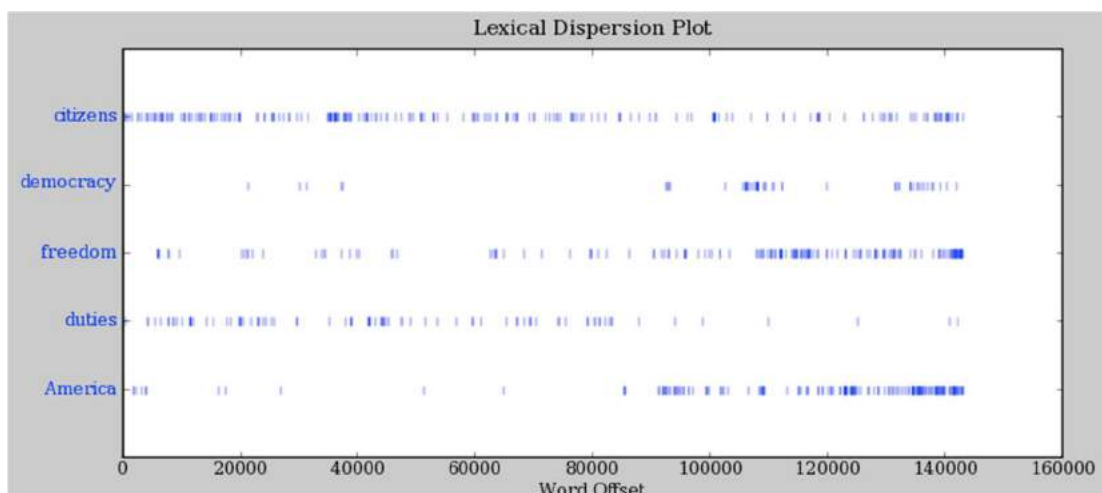


图 7-2 高频统计

另外，如果你使用这个命令，可以查看这些单词在文本中分布位置，x 轴代表全部文本，每一个蓝色的短纵列都代表一次出现。

```
text4.dispersion_plot(['citizens','democracy', 'freedom', 'duties', 'America'])
```

图 7-3 分布图



除了简单的文本统计，NLTK 库还为我们提供了许多的分析方法

```
>>> 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的相同上下文单词，给定的两个参数你必须首先用中括号括起来，然后再用小括号括起来。

还有一个功能很相像的函数：

```
>>> 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，比如说，我们经常使用 `beautiful` 来形容 `girl`，那么这个函数可能就会给我

们返回 handsome（同样用来形容 girl）

NLTK 库还有一个命令用于分句：

```
>>> 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.']
```

可能 1 会疑问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"这种呢，称谓之间的标点符号很明显你是不可以把它单独分割成一个句子的，这时候 NLTK 库的分句用法就会展现出它的强大

```
>>> 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，语义、逻辑上可以保证大体通顺。但稍稍有些遗憾，我们可以从下面的源码中（Python 目录下\Lib\site-packages\nltk中test.py第492行）看到这个功能在新版本的nltk中不再支持，所以如果你想使用的话，那么你可以安装 2.0 版本的 nltk（有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，比如说，我们使用 NLTK 自带的 text1，那么你想要生成和 text1 相同风格的字符，你只需要键入：

```
text1.generate()
```

然后我们介绍几个在进行数据处理的时候常用的方法

- ❑ 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()用于获取含有的元素或者字符的多少，读者可以和其他方法组合起来使用，比如说：

```
#获取不重复单词（标点）个数
>>> len(set(text3))
2789
```

- ❑ 对上面获取到的内容进行一些简单的四则运算，下面这个可以统计每个词出现的频率：

```
>>> print(len(text3)/len(set(text3)))
16.050197203298673
```

- ❑ 还可以这样子使用，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']
```

- 这种使用方法可以用来统计出现次数超过 150 词的单词（标点）（篇幅原因，最后输出结果有删减）。

```
>>> 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']
```

- 该方法会打印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 是一个功能很强大的英语词典，Princeton（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的一种基于认知语言学的英语词典，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())
```

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 哈希表

哈希算法和一般的算法最大的区别在于一般的加密算法是一一映射（都是对特定值进行某种变换，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），因此理论上来说是可逆的，而哈希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)”，每个哈希结果都存在无数个可能的目标文本（参考高中函数一一映射和满射）。故哈希不是一一映射，符合是不可逆的。

哈希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被叫做哈希算法和加密算法，但未必是好的哈希和加密，好的哈希和加密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）。

好的哈希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，如一亿个字符变了一个字符，那么结果应该截然不同。这就是为什么哈希可以用来检测软件的完整性。

哈希算法有着很广泛的应用，加密常用的 MD5，SHA 都是哈希算法。很多网站的密码都是采用哈希值加密的，是当你忘记密码的时候，网站并不是将你忘记的口令发送给你，而是让你重新更改一个密码，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哈希值存储在数据库里，而哈希算法不可逆，所以即使是服务器也不可能通过复原你的密码，而只能重置密码。

哈希算法领域我们常用的是 SHA1（Secure Hash Algorithm，安全哈希算法）和 MD5（Message-Digest Algorithm 5，信息摘要算法 5）加密，我们用 Python 的 hashlib 库实现这两种加密

```
#导入哈希算法库
import hashlib

#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())

#SHA1 加密
sha1 = hashlib.sha1()
sha1.update(b'I'm using sha1 in python')
print(sha1.hexdigest())

输出为：
```

```
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，有一点要注意的是，在哈希算法中空格也是有效的，所以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，同样的，这种加密方式也有分布上传。

那么我们怎么使用哈希呢？如果数据仅仅用作比较验证，不需要还原成明文形式，那么使用哈希算法加密；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，可以用于快速检验一个元素是否在一个集合中，它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哈希算法），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。

在超大型数据面前，可以说，布隆过滤器是必须的，比如说，1T 的数据，当你如果用旧的一个一个比对来查看是否在其中的时候，那么无论你的电脑/服务器配置多高，你都会

至少卡顿几秒。

我们需要一个叫做 `pybloom_live` 的库，可能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，提示你像这样

```
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` 的出现这个问题是因为缺少一个叫做 `bitarray`（这个库基于 `CV++14`）的库，通过在这个网站下载相应的 `whl` 文件 <https://www.lfd.uci.edu/~gohlke/pythonlibs/> 即可。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。

[bitarray-0.8.3-cp37-cp37m-win32.whl](#)  
[bitarray-0.8.3-cp37-cp37m-win\\_amd64.whl](#)

图 7-4 `bitarray` 版本

```
pip install bitarray-0.8.3-cp37-cp37m-win_amd64.whl
```

然后再安装 `pybloom_live` 即可，就可以成功安装了。

我们来基于 `pybloom_live` 实现布隆过滤器。

```
#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  
#存在读取，不存在创建  
is_exist = os.path.exists('test.blm')  
if is_exist:  
    bf = BloomFilter.fromfile(open('test.blm', 'rb'))  
    #没有该文件则创建 bf 对象 最后的时候保存文件  
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，创建/读写文件的方法有两种：

从文件中读取：

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

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。

添加到文件中，

```
bf.add(i)
```

写入到文件中，

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

再来考虑一下刚才写的代码

我们第一次运行插入前十个值，后来查询后二十个值，所以会输出从 11-20 会输出 unwritten，而前十个值输出 written。

如果你再次运行的话，你就会发现里面的 add 全部变为 pass 了

```
pass
.....
pass
pass
```

## 7.9 关系型数据库 MYSQL

接下来是我们的重磅戏，关系型数据库，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 Structured 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 (关系型数据库管理系统, Relational Database Management System) 中检索数据的一门语言。

### 7.9.1 MySQL 安装

MySQL 也属于 RDBMS 的其中一员。

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。有尽可能小的冗余度, 避免了重复输入信息。甚至它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），你可以直接在其中创建、添加表，虽然那样可能更方便一些，但笔者还是觉得直接使用命令行更方便我们理解数据库。

接下来，我们以 MySQL 为例子，来介绍关系型数据库，读者可以在这个网站下载到 Windows 系统下的 MySQL 数据库。

<https://dev.mysql.com/downloads/windows/installer/8.0.html>

下面我们说明几个需要注意的点，其他情况下读者选择默认值就可以了，建议读者选择约莫 300M 的离线安装包，15M 的在线安装包到安装过程中还需要读者联网下载，这一过程需要的时间远远超过读者直接在官方网站上下载的时间。

要点一，安装方式

笔者建议读者在第二步选择 **server only**（我们暂时只需要做测试使用），这样你可以避免下载一系列和数据库相关的软件，如图 7-5。

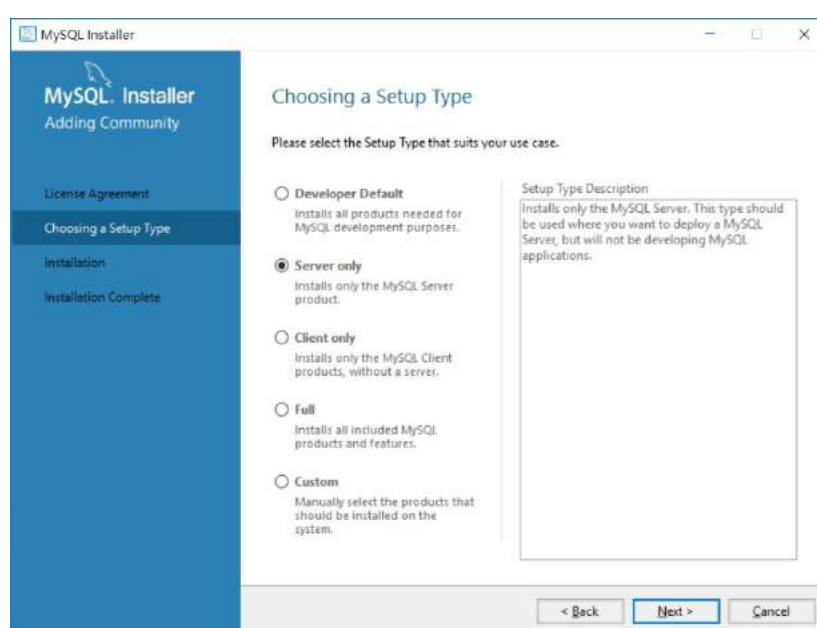


图 7-5 选择安装类型

要点二，类型和网络配置如图 7-6，这一步默认即可

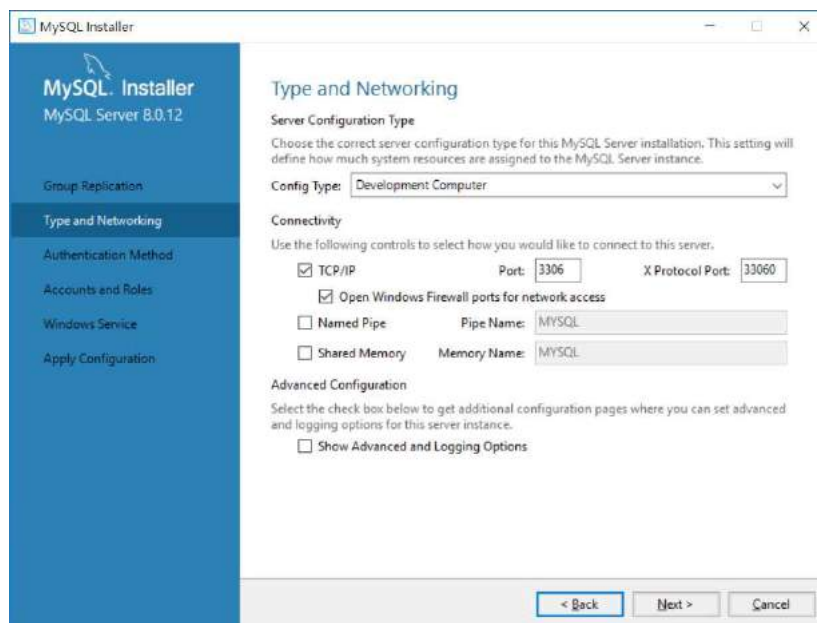
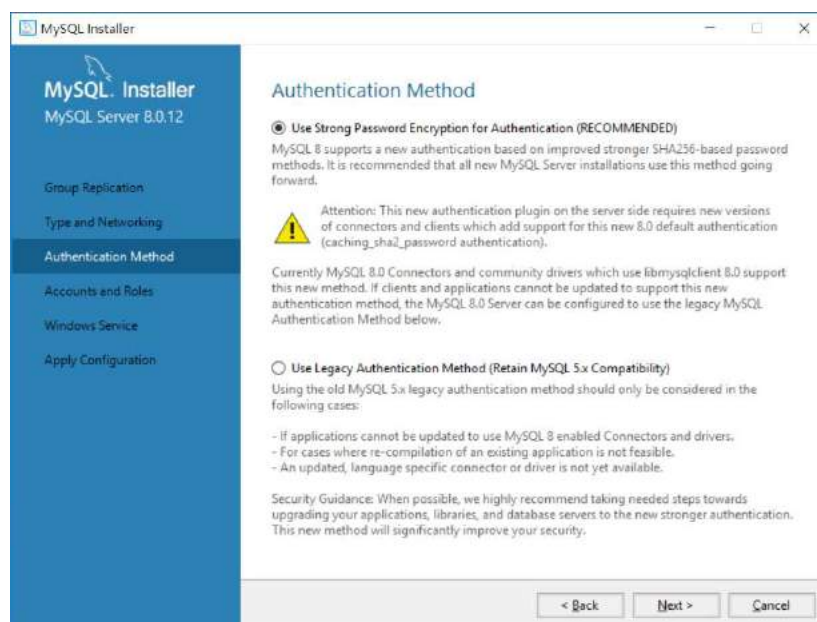


图 7-6 类型和网络配置

要点三，这一步是让我们选择使用认证方式（如图 7-7），使用强加密验证或是传统的方式，读者点选哪一步均可，但第一个存在一定会的安装失败风险，可能会存在不支持的情况。

图 7-7 选择认证方式



这一步配置（如图 7-8）第一个勾选项询问是否把 MySQL 作为一个系统服务来运行，第二个勾选项询问我们是否开机启动。第三个选择项，询问我们 MySQL 作为系统服务的运行身份，A 选项是标准定制账户，B 选项是定制用户。



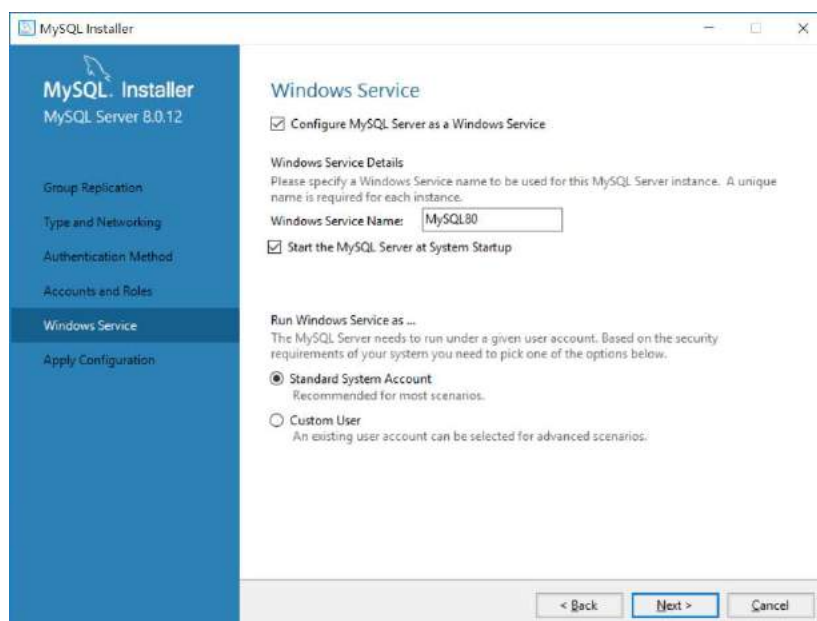


图 7-8 是否作为系统服务

建议这三个选项第一个开启，第二个关闭，第三个作为标准系统账户使用。

这是最后的配置步骤（如图 7-9），在这一步的执行 **Execute**，然后静静的等待所有的事项完成后，MySQL 的安装步骤就完成了。

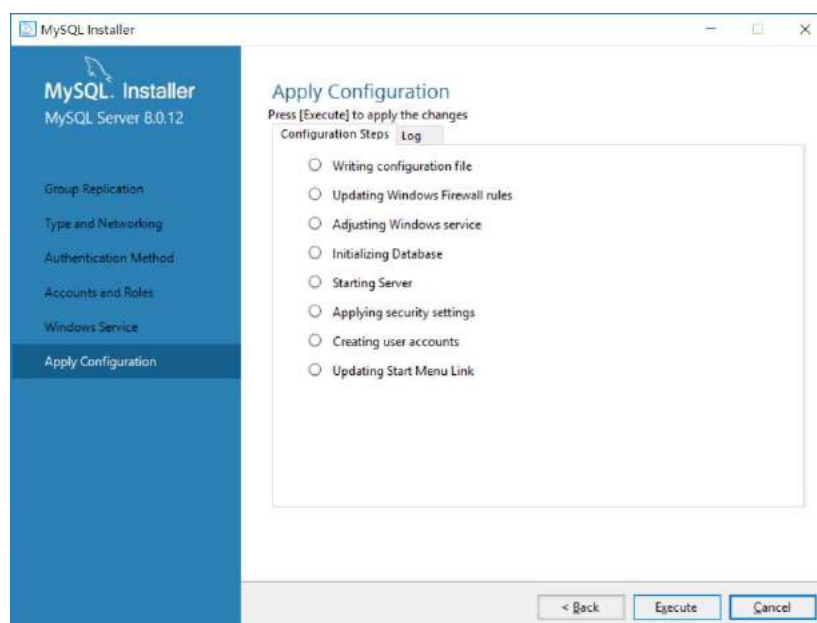


图 7-9 配置完成

在安装完成了以后，读者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也可以。

```
mysql -uroot -p
```

在 MySQL 中，Windows 系统下，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，数据库和表名是区分大小写。

## 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，比如说，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，哪怕你把命令写成小写也是没有关系的。

### 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，我们创建了一个名为 **test** 的数据库。

### 使用数据库：

```
mysql> USE test;  
Database changed
```

在调用某个数据库的内容之前，你需要先使用 **USE** 命令来切换到这个数据库

### 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紧接着被从系统中移除，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，MySQL 中，命令以分号结尾，当你命令结束却没有使用分号可能会让你多行式输入命令：

```
mysql> DROP DATABASE test  
-> ;  
Query OK, 0 rows affected (0.02 sec)
```

### 查看数据库：

```
mysql> SHOW DATABASES;
```

这句命令用来查看当前已经有的数据库，效果如图 7-10 所示，前四个数据库都是 MySQL 默认存在的，如果你不了解的话，最好不要对他们做任何更改。如果读者想要和笔者的输出内容相同，那么读者需要再一次重复创建 **test** 数据库的过程。

图 7-10 查看数据库

```
SHOW DATABASES:  
+-----+  
|Tables_in_test|  
+-----+  
|user_info     |  
+-----+
```

### 创建表：

表在数据库中用单词 **TABLE** 来表示

```
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，我们创建了一个有三个参数的表，第一个参数设置为不超过 10 个字符的字符串，第二个 **user\_id**，**pro\_id** 都设置为整形，即数字。

然后我们向其中加入信息：

```
INSERT INTO user_info VALUES("li_si",0,123456);
```

每一个参数都和刚才设置的参数多少相互对应。这里建议读者多插入几条数据，不仅是为了练习，更是为了方便我们后面的步骤的进行。

### 查看表：

```
mysql> SHOW TABLES;
```

### 查看数据:

```
SELECT * FROM user_info
```

效果如图 7-11 所示:

```
+-----+-----+-----+
|log_in  |id   |password|
+-----+-----+-----+
|li_si   |0    |123456  |
+-----+-----+-----+
1 rows in set (0.00sec)
```

图 7-11 查询结果 1

这个命令有很多种用法,可以和其他的命令互相结合,比如说,读者想要实现条件化查询,比如说这几种查询方法:

#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 语句之后可以跟随任意我们设置的参数,然后对其进行比较,大于小于等于,符合条件的数据会被返回。

### 表中数据更新:

```
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;
```

### 删除行:

```
DELETE FROM user_info WHERE id=1;
```

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

另外如果读者你想要删除数据库中的所有数据,比如说,游戏中进行一个删档测试,那么你会用到这个数据

```
DELETE FROM user_info;
```

### 删除表:

如果读者想要在删除所有数据的同时删除掉这个表,那么你可以直接使用删除表和其中的所有数据。

```
DROP TABLE user_info;
```

在对 SQL 语句有着基本的理解之后,我们来实现 Python 和 MySQL 的交互,我们需要借助 pymysql 这个第三方库,你可以使用 pip 来安装:

```
pip install pymysql
```

或者你可以在网络下载这个库的源码,然后进入目录后,执行这条命令。

```
python setup.py install
```

下面笔者通过代码来展示怎样使用 pymysql 来进行 MySQL 和 Python 的交互。

示例：

```
#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
#因为游标现在相当在输出所有数据的下一行
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：
```

```
Database version : 8.0.12
*****=====*****
('li_si', 0, 123456), ('zhang_san', 1, 12345678))
('zhang_san', 1, 12345678)
('li_si', 0, 123456)
```

pymysql.connect()这个语句参数比较多，我们来讲解一下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'

```
cur=conn.cursor()
```

效果如图 7-12 所示:

```
+-----+-----+-----+
|log_in  |id   |password|
+-----+-----+-----+
|li_si   |0    |123456  |
|zhang_san|1    |12345678|
+-----+-----+-----+
2 rows in set (0.00sec)|
```

图 7-12 查询结果 2

首先我们使用这条命令来连接并创建了一个游标或者一个类似于游标的对象, 笔者将游标理解成类似于这个东西。

```
cur.execute()
```

execute()函数可以用来执行 SQL 语句

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

scroll()用来移动我们创建的游标, 我们使用这条命令默认是相对模式的, 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, 如果你想要使用绝对模式, 那么读者可以指定需要 mode, 当使用绝对模式的时候, 光标在相当于从第一行的数据输出开始, 大致在这个位置 (如图 7-13 加重标记):

```
+-----+-----+-----+
|log_in  |id   |password|
+-----+-----+-----+
|li_si   |0    |123456  |
|zhang_san|1    |12345678|
+-----+-----+-----+
2 rows in set (0.00sec)
```

图 7-13 光标位置

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, 那么是游标向下移动, 负数表示向上移动。

获取输出的有两个命令, 分别是 fetchone()和 fetchall()这两个命令, 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, 接着我们创建一个游标, 然后执行 SQL 命令, 获取数据, 对所有数据进行查询并输出, 因为游标的参数为绝对值的时候, 默认是在数据输出的第一行, 所以我们向下移动一行获取到的是第二条张三的数据。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, 很重要。所以 MySQL 的执行机制跟 GIT 很像, 需要我们在完成操作后再使用 commit()提交修改。

整个流程如图 7-14 所示:

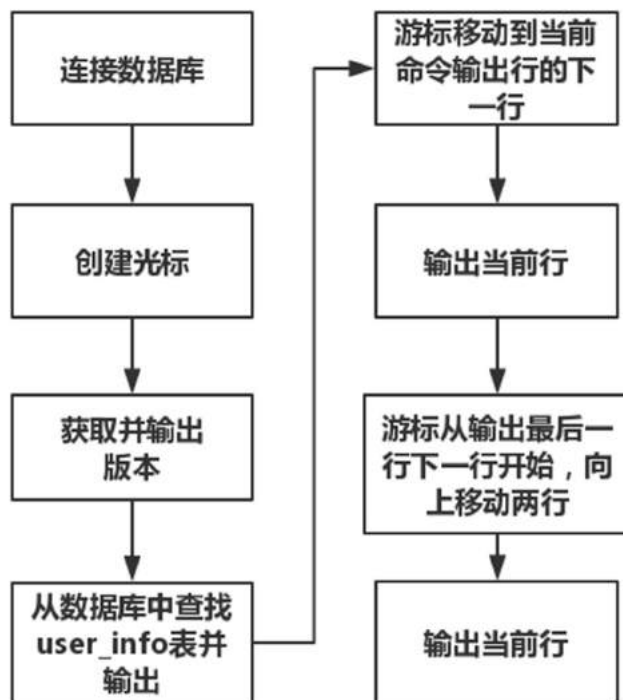


图 7-14 流程图

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，笔者这样子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
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
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
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))
*****=====*****

```

我们第一次执行的 SQL 语句（查询语句不算在内）向其中添加了李六（其实正确的顺序应该是张三李四王五赵六）的数据，然后对数据进行输出。第二条 SQL 语句，我们删除了王五的数据。然后第三条语句，我们修改了李四的 log\_in 参数，li\_si 被替换成了 im changed。

同样，pymysql 库还有一个可以让你返回的操作——rollback()命令，这条命令在你没有最终 commit 之前，一切操作都是可以回退的。

```

>>> 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))

```

笔者在 Python 的 IDLE 交互式环境下演示了 `rollback()` 操作，初次执行查询语句，表中只有两条数据，接着我们向其中插入一条数据，再次查询，表中已经变为三条数据，我们执行 `rollback()` 后，最后一次查询，返回的数据跟刚开始的数据一模一样。

## 7.10 非关系型数据 NOSQL

常见的非关系型数据库（NoSQL，Not Only SQL，不仅仅是 SQL.）：

- ☐ MongoDB
- ☐ CouchDB
- ☐ Redis
- ☐ Big Table

非关系型数据库其实本意是不仅仅 SQL，笔者认为，翻译为超关系型数据库或者其他的更合适，但事实上，非关系型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，可以想象成表中的主键和值的对应关系，而且不需要经过 SQL 层的解析，所以性能非常高。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。

还有一点商业上的区别，大部分的关系型数据都是商业运行的，这就意味着如果你使用关系型数据库，你需要为之付出一定的费用。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。

如果你在小型化创业公司运行，非关系型数据是一个不错的选择。

刚才已经有了关系型数据的基础，非关系型数据库使用的查询语言大多和 SQL 相似，但非惯性数据的各个种类又略有不同，我们就不再以单一一个为例子详细说明。

## 7.11 总结

我们再来聊一聊编码问题。



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，读者就会发现，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，正相反，笔者认为，它们关系紧密，不可分割。

一个幽灵，编码问题的幽灵，在编程路上游荡。

一旦踏上了编程的道路，如果不把编码问题搞清楚，那么你随时有可能踩到编码的地雷，总有一天它会在你意想不到的地方为你挖一个大坑。

编码问题可能占据了你在编程中的大部分时间。

我们上面讲的编码更多是偏向于不同数据格式的解析，加解密，存储，而网页的编码和我们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，这两个编码进行转换，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，举个例子，笔者在刚刚学习爬虫的时候，碰到 ASCII，ANSI，GB2312，GBK，UTF-8，UTF-16，ISO 8859-1 这些编码的时候，几乎被它们搞得要崩溃，明明数据就在那里，但你就是无法转码存储。

只有充分发挥不屈不挠的精神，和编码问题死磕到底，这样子，读者才有可能走出这个雷区。

接下来，对读者能够读到这里道贺，截止本章节，读者已经学习到了本书的大部分内容。但看完书是一码事，真正会用又是另一码事，不经过实践，掌握的再好也只是花把势。

想学的技能，把理论看上千百遍都未必能熟练运用，而要经过真实的实践；想开发的项目，在心中想一百遍都未必能有个完美的策划，而要真正动手去做。

在接下来的内容中，我们将会对以上我们所学的内容进行统一的整合，通过两个项目的实践，来让读者对以上所学的内容有着更加深刻的理解。

## 第 8 章 实战 part3——猫眼电影

在这一章节中，笔者通过一个假设的项目需求，帮助读者详细讲解从了解客户需求到网站分析、处理的全套流程，并在分析中逐步引入代码的说明与实现。

本章节的学习将会帮助读者有效提高自己的业务能力。

### 8.1 网站分析

第三个实战项目，我们以猫眼为例子，猫眼电影是国内有名的一个电影评论网站，性质和评分机制和豆瓣很相似，如果读者你是一个资深的影迷，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，客户是一位自媒体的运营人员，自媒体推广内容和电影相关，需要我们写一个程序来获取当前评价最高的电影榜单为自己公众号写文章提供参考。

作为乙方的我们，首先要选择一个获取数据的对象，然后，我们将枪口，哦，不，是目标选择为猫眼电影。

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）。虽然从法理上来说，一个网站对公众开放接入权限，允许人类以一种方式获取数据，却不允许人类以另一种方式获取数据，这并不合理。

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 8-2 所示), 那么我们就可以放心的进行采集了, 但读者最好在夜间进行数据采集, 夜间人流量少, 会减少对网站的正常的运行的影响。

```
User-agent: *  
Disallow:
```

图 8-1 robots.txt 内容

```
maoyan.com/robots.txt
```

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, 但作为专业人员的我们, 要对客户的需要进行扩展。首先我们来分析一个电影榜单, 如图 8-3:



图 8-3 参考样例

这里我们几个要获取的要素, 霸王别姬, 这是电影名, 主演, 这也是我们要获取的内容, 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, 然后就是电影的评分, 客户肯定是优先参考评分, 但有了这些要素还远远不够, 我们还需要一个关于电影的内容的介绍。

```
<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, 先来看一下他们 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, 变化都是 dd[] 中的数字, 一个页面上有 10 条数据, 又因为数字是从 1 开始的, 而我们平常在 Python 中的 range() 函数是从 0 开始记的, 我们考虑使用 range(1,11), 即计数从 1 到 10。

我们的内容都在 title 标签中, 我们使用 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 .text 方法, 通过元素查找找到元素位置, 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写复杂, 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, 都是在一个属性为 score 为 p 标签下。

再来看一下主演和评分部分的 XPath

主演 XPath:

```
//*[@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 XPath:

```
//*[@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 拼接的方法来应对，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 都增加了 10，刚好对应页面的数据个数。但第一个页面却又有所不同，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，所以没有显示。

实践出真知，直接访问检验看是不是这样，好的，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 测试代码 1.0

首先我们导入库，导入的库是本书主要使用的 Selenium 项目和 Time 模块。使用 Time 模块而不是使用显式等待，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，我们从整体来看，TOP100 榜单，每个页面 10 个数据，所以我们需要构造 10 个页面链接。当然，如果读者想的话，每个页面访问一遍把 URL 复制下来也可以。

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

然后再次使用 FOR 循环，对列表中的每个页面，都进行通过 XPath 获取标题，分数，演员，然后通过点开页面，获取电影的内容介绍（流程如图 8-4 所示意）。

```
#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("演员: %s" %actor)
    print("分数: %s" %score)
    print(introduce)
    dr.get(i)

#关闭
dr.close()

```

可以看到输出类型如下所示:

电影标题: 罗马假日

演员: 主演: 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。

很好，符合我们的预期，我们再想回顾一下本程序的流程，图 8-2 为本程序的流程：



图 8-4 流程图

## 8.3 测试代码 2.0

既然能够输出了，我们再来对刚才的代码进行一次修改，将其写入到 CSV 文件中。

为什么采用 CSV，而不使用数据库呢？第一，我们的数据比较少，没有大型化存储需求。其二，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，检验，这一套流程走下来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。数据库是很强大，但应用在这种场景就像是我们拿着一把重型加特林去扫蚊子，有些大材小用了。

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
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，对每个页面中的链接再次进行获取操作
    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
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()
```

程序返回的输出应该是这样的：

```
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
```

当程序运行完成了，我们直接以 EXCEL 表格的形式打开，最终文件中的数据如图 8-5 所示：

霸王别姬	主演：张国荣,张丰毅,巩俐	9.6	影片借一出《霸王别姬》的京戏，牵扯出三个人之间一段随时代风云变幻的爱恨情仇。段小楼（张丰毅 饰）与程蝶衣（张国荣 饰）是一对从小一起长大的师兄弟，两人一个演生，一个饰旦，一向配合天衣无缝，尤其一出《霸王别姬》，更是誉满京城，为此，两人约定合演一辈子《霸王别姬》。
肖申克的救赎	主演：蒂姆·罗宾斯,摩根·弗里曼,鲍勃·冈顿	9.5	20世纪40年代末，小有成就的青年银行家安迪（蒂姆·罗宾斯 饰）因涉嫌杀害妻子及她的情人而锒铛入狱。在这座名为肖申克的监狱内，希望似乎虚无缥缈，终身监禁的惩罚无疑注定了安迪接下来灰暗绝望的人生。未过多久，安迪尝试接近囚犯中颇有声誉的瑞德（摩根·弗里曼 饰），请求对方帮自己搞来小锤子。以此为契机，二人逐渐熟稔。

图 8-5 数据集

说实话，当你写好程序，出去小憩一下，等你回来发现所有的数据都静静的躺在那里，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。

这里，笔者解释下为什么要使用 UTF-8-SIG 编码，UTF-8-SIG 编码即 UTF-8 with BOM。

但我们如果将 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 的时候是通过读取文件头上的 BOM（Byte Order Mark，字节顺序标记）（BOM 是一种文件头部协议，存储在文件头部，标识文件编码）来识别编码的，并且按照 UNICODE 编码来识别没有 BOM 的文件。

而我们生成的 CSV 却没有 BOM，Excel 自动按照 UNICODE 编码读取，文件的编码却不是 UNICODE，编码问题就会出现。

就像笔者在上一章节所说的，作为一个开发人员，编码问题，读者必须彻彻底底的搞懂它。甚至有时候要把自己原先认为理解对的地方要推翻重来。这是我们必须面对的。

## 第9章 实战 part4——淘宝商品

在这一小节中，笔者以淘宝商品为例子，阐述如何应对复杂网站的页面数据采集，帮助诸位追赶较为新颖的实现方式，解决读者困难以下手而犹豫不决，无法直接应用的难题。

笔者利用 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做 pyquery 的库，这么库依赖于 lxml，而 lxml 安装依赖于 VC++14（由错误可见，我们 pip 安装的时候，在建立 lxml.etree 扩展的时候失败了，提示我们需要安装 VC++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
```

所以读者直接使用 pip 有很大的可能会报错，我们建议读者从上次向读者推荐的网站下载 wheel 文件，然后通过 pip 安装 wheel 文件，来避免这一系列麻烦。

<https://www.lfd.uci.edu/~gohlke/pythonlibs/>

但是此次淘宝爬虫有一个地方可能需要读者更改一下，就是我们使用 Chrome 浏览器进行测试，但在爬取过程开启一个 Chrome 浏览器确实不太方便，所以笔者在这里可以对接我们在第四章学习的 PhantomJS 无头浏览器，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。如果笔者以前写过其他爬虫的话，那么一定使用过这个库，没有使用过的读者可以去了解一下这个库是干什么的，但不必详细去学习，我们仅仅会用到其中的一个方法。

### 9.2 流程简述

首先我们来看一下淘宝的链接组成，

比如说，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](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，淘宝返回的链接是这样的

[https://s.taobao.com/search?q=usb&imgfile=&js=1&stats\\_click=search\\_radio\\_all%3A1&initiative\\_id=staobaoz\\_20180805&ie=utf8](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，而 imgfile 参数和图片搜索相关，而 initiative\_id 参数中包含了访问的时间，ie 参数说明了使用的编码，诸如此类等等。

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，但不排除部分时候有些内容下载特别慢，影响我们爬虫的速度，而我们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出来，而不是必须要等到所有的内容都加载完全。



结合笔者在本书的前面讲述的显式等待和隐式等待，我们需要的内容在这里面（如图 9-1）：



图 9-1 淘宝分析

所以，我们只需要判断这个内容所属的标签完全加载出来就可以了，当然，是所有商品的内容。

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成：

```
.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 获取详细信息的话，需要对商品价格，商店，商家等等信息重复写多次 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，笔者不采用 find\_elements\_by\_css\_selector()

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。

导入库并将其简写为 PQ:

```
from pyquery import PyQuery as PQ
```

解析页面源码：

```
html=dr.page_source
pq_file=PQ(html)
```

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()
```

然后分别选择下级中 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

```

```

类似的，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()]
```

内容的部分结束了，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，而对指定页数的页面进行采集需要我们进行页面的跳转，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。

代码流程图如图 9-2：

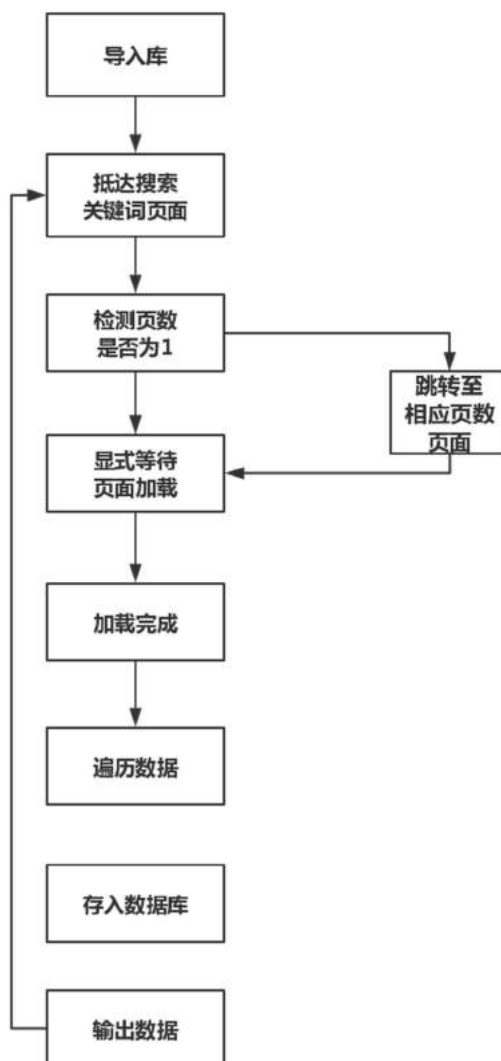


图 9-2 流程图

## 9.3 代码解读

在理清了思路和实现流程之后，我们就可以尝试着实现淘宝抓取的功能，最终我们实现的功能代码如下：

```
#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')
    #防止某个页面失败导致爬虫停止
    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)
    #点击
    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，特殊字符（包括汉字）的字符串会被替换成 `%xx` 的格式，仅仅只有字母，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，这个函数含有一个可选参数来指定不格式化的内容。

当 Python 编译器编译一个源码文件的时候，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码文件，而不是被当做库导入），会赋值 `__main__` 给 `__name__`，但如果是被当做库导入的，那么会赋值这个源码文件的名字给 `__name__`。

我们定义了一个 `main()` 函数，然后只有在主动运行的时候，运行 `main()` 函数。

当我们运行程序的时候，会返回类似这样的结果：

```

正在爬取第 1 页
[/g-search1.alicdn.com/img/bao/uploaded/i4/imgextra/i1/13022581/TB2s3yWkbZnBKNjSZFrXXaR
LFXa_!!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/TB1mX4wmAomBKNjSZFqXXXtqVX
a_!!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']

```

内容可以重新打印，第一步已经完成了，我们考虑将数据存储到数据库，毕竟，这次的数据已经算是有些规模了，另外我们的目的是对前面所学的内容进行统一的整合，实践的知识范围总是越多越好。

这次多一个 `pymysql` 库，首先我们连接数据库，注意，读者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（笔者命名为 info），表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 或是商品标题。

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，既是为了避免数据的重复，也是为了检验代码的可重复使用性。

笔者建立的 info 表有 6 个字段，和下面的代码相对应，第一个 URL 字段的 VARCHAR() 中的数字要设置的特别大一些。

```
['/g-search1.alicdn.com/img/bao/uploaded/i4/imgextra/i3/14414497/xxx.jpg',  
'¥79.00', '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 方法将其插入到数据(在这里，由于代码略微有些长，所以笔者采用的空格+“\”+换行的格式，表示我们当前这一行代码将延续到下一行，两行代码将被拼接并视为一行代码：

```
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，定义成函数  
def mysql_save(product):  
    #tip 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):
    #输出正在爬取页面
    print('正在爬取第',page,'页')
    #防止某个页面失败导致爬虫停止
    try:
        #跳转关键词链接
        url='https://s.taobao.com/search?q='+quote(keyword)
        dr.get(url)

        #检测页面是否大于 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)
            #点击
            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 tip1 的位置，笔者使用 `{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，后的 `%s` 跟的需要格式化进入的内容。

而 `format()` 的基本语法是使用 `{}` 来代替 `%` 这样冗杂的语法（因为字符串，数字等等对应百分号后跟不同的字母，需要读者单独记忆），我们在 `{}` 中加入的数字表示顺序，依然是从 0 开始，当然，读者如果喜欢的话，这样子按照默认顺序也可。

```
"{} {}".format("hello","selenium")
```

输出的结果和上面相同：

正在爬取第 1 页

```
[//g-search1.alicdn.com/img/bao/uploaded/i4/imgextra/i3/14414497/TB2moiOfVooBKNjSZPhXXc2CXXa_!!0-saturn_solar.jpg', '¥n79.00', '496 人付款', '绿联 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', '华趣数码专营店', '上海']
```

```
[//g-search3.alicdn.com/img/bao/uploaded/i4/i4/2451941673/TB2RSMpv_IYBeNjSszcXXbwhFXa_!!2451941673-0-item_pic.jpg', '¥n10.00', '146676 人付款', 'type\n- \n\n 数据线 t 三星 s8 华为 v8v9 荣耀 V10p10p9 充电器 4\n\n 原装 6mix2s 小米 5x 手机 5s 乐视 1s2 正品快充 tp\n\n 魅族 mx6p20pro', '塔菲克旗舰店', '上海']
```

```
[//g-search3.alicdn.com/img/bao/uploaded/i4/i2/713464357/TB20uQla6bguuRkHFrdXXb.LFXa_!!713464357-0-item_pic.jpg', '¥n7.90', '13968 人付款', '绿联\ntype\n- \n\n 数据线手机通用小米五 5x/5s/6x/8/max2mix2s 华为 p9nova2s 荣耀 v9 三星 s8 魅族 mx6 乐视 2 高速快充充电器线', '绿联数码旗舰店', '广东 深圳']
```

.....

最终获取到的部分数据如图 9-3：

image	price	deal	title	shop	location
//g-search1.alicdn.com/img/bao/uploaded/i4/im...	¥ 10.80	1330人付款	Type-c数据线快充6小米8se/5/5note3乐视手机...	能适旗舰店	广东 深圳
//g-search3.alicdn.com/img/bao/uploaded/i4/i4/...	¥ 10.00	146676人付款	type - c 数据线三星s8华为v8v9荣耀V10p10p...	塔菲克旗舰店	上海
//g-search3.alicdn.com/img/bao/uploaded/i4/i2/...	¥ 7.90	13968人付款	绿联 type - c 数据线手机通用小米五 5x/5s/6x...	绿联数码旗舰店	广东 深圳
//g-search3.alicdn.com/img/bao/uploaded/i4/i4/...	¥ 29.00	23279人付款	绿联usb分线器一拖四转接头3.0高速 type - c...	绿联数码旗舰店	广东 深圳
//g-search2.alicdn.com/img/bao/uploaded/i4/i3/...	¥ 299.00	2135人付款	绿联 Type - C 扩展坞拓展usb转接头HDMI...	绿联数码旗舰店	广东 深圳
//g-search2.alicdn.com/img/bao/uploaded/i4/i4/...	¥ 25.00	52246人付款	一拖三数据线三合一充电器手机快充多头...	cafele旗舰店	广东 深圳
//g-search1.alicdn.com/img/bao/uploaded/i4/i1/...	¥ 27.00	28321人付款	第一卫小米6耳机8转接头 type - c 数据线mix...	第一卫旗舰店	广东 深圳
//g-search1.alicdn.com/img/bao/uploaded/i4/i4/...	¥ 32.00	13321人付款	倍思苹果数据线一拖三iPhone耳机三合一 ty...	baseus倍思超图卫专卖店	江苏 南京
//g-search3.alicdn.com/img/bao/uploaded/i4/i2/...	¥ 220.00	2072人付款	倍思苹果电脑转换器 type - c 扩展坞转接头...	baseus倍思超图卫专卖店	江苏 南京
//g-search3.alicdn.com/img/bao/uploaded/i4/i1/...	¥ 28.00	23360人付款	苹果流光数据线Phonex发光安卓iPhone6s...	四眼数码专营店	广东 深圳
//g-search2.alicdn.com/img/bao/uploaded/i4/i2/...	¥ 85.00	3193人付款	Type - C 扩展坞拓展usb转接头hub华为Ma c 8...	炜炜wei2018	广东 深圳
//g-search3.alicdn.com/img/bao/uploaded/i4/i2/...	¥ 85.00	2494人付款	派尔胜 Type - C 扩展坞拓展usb转接头hub苹...	派尔胜数码配件旗舰店	广东 深圳
//g-search2.alicdn.com/img/bao/uploaded/i4/i2/...	¥ 19.90	13774人付款	华为原装数据线 Type - C 快充Mate10 9 P20 P...	鑫楚天数码专营店	广东 深圳
//g-search3.alicdn.com/img/bao/uploaded/i4/i2/...	¥ 99.00	2839人付款	紫科 Type - C 扩展坞拓展usb转接头hub苹果...	紫科旗舰店	广东 深圳

图 9-3 数据集

## 9.4 可配置项

当读者已经能够熟练掌握 MySQL 的命令之后就不在推荐读者使用命令行了，因 MySQL 的命令行仍有部分不足，比如说，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），笔者推荐读者使用它们，比如说，笔者正在使用的 MySQL WORKBENCH，这是一个 MySQL 官方推荐的图形化工具，读者同样可以在官网上下载到此工具。

# 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。

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，但系统的复杂程度是无限的。随着更大的挑战的出现，当系统的复杂性超过了你的“内存”时，你必须依靠其他工具来帮助你减少问题。

假设读者在一家 IT 公司工作，然后使用了 30000 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 个左右的嵌套、互相调用，这时候你的自我感觉相当良好。并且读者的代码风格很好，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，你无法保证自己知道这个 bug 一定出现在哪个部分，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的体现在代码的维护上，而不是在代码编写的时候。

有人可能会想，我们打开程序界面，一个一个按钮点过去，不就知道了功能是否实现了吗？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，我们可能对照着检查表逐个排查，加上程序运行的时候，中间的切换动画，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，一个模块一个模块的进行检查，它们本身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仅节约了时间和成本，而且降低了错误率。

## 10.2 怎样写单元测试

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，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。

比如说，我们对一个图书馆书库管理系统进行测试，那么要求存入的书的名字不能为空，另一方面，我们还需要测试用户的借书卡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，使用的时候导入到待测试的代码使用。

继承自 `unittest.TestCase` 的类的函数的书写有一定的规范

- ❑ 使用 `setup()` 函数开始单元测试，`tearDown()` 函数来结束单元测试
- ❑ 所有以 `test_` 开头的函数都会被当做单元测试执行，其余的函数都会被忽略（除 `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掉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** 存在。

在单元测试中，我们常用的断言有以下几种：

```
assertEqual(self,[assert 1],[assert 2])
assertTrue([bool])
assertRaises([exception])
```

## 10.3 单元测试演示

`unittest` 是 Python 的标准库，所有未经修改过了 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，仅仅是为了实现演示的效果：

```
#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，然后定义了初始化和结束的函数（没有实际内容，

仅仅是一个 `print()` 函数)，接下来写了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，如果想要实现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和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__'` 是为了实现程序只有在自我运行的时候才会直接开始测试，如果是其他文件导入的话，则导入的只有一个单元测试类。

再来定义一个断言错误的代码：

```
def test_wrong_minus(self):
    result=2-1
    self.assertEqual(2,result)
```

断言错误会格式化的告诉我们进行了多少个测试，多少个测试错误。并且很清楚的说明错误原因，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）。

```
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 也叫做关键字参数，而关键字参数允许我们传入小到 0 个，大到无穷个的含参数名的参数，也就是说，使用\*\*kw 传入的参数会被当做一个字典来处理，参数名会被当做关键字，而参数值会被当做关键字的值。

可能读者看到类中的第二个函数感到有些困惑：

```
def __getattr__(self, key):
```

\_\_getattr\_\_ 其实是 python 中的一个内置函数，可以用来实现返回未知属性，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，当类的实例属性尝试赋值时，就会调用这个方法，比如说：

```

>>> 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（除了一开始字典中的元素和值）都会被添加到一个叫做\_\_dict\_\_的属性中，并且这个属性原有并不为空，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是否可以正常使用：

isinstance([object],[class])也是一个内置函数，如果前一个参数（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，否则为 False，因为我们得 instance 是 book 的一个实例，而 book 类继承自 dict，所以 instance 也可以算是 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，笔者将刚才实现的 `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（是的，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
```

笔者在所有函数的外层定义了一个 `instance` 变量，用 `global` 声明其是全局变量，在这个类中的所有函数都可以使用。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。

我们一共测试了五个部分，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被正确解析，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。

接着，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，我们检验是否可以正常抛出错误，这种行为更多的是检查是否可以人为（恶意用户）的主动生成 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()` 执行的步骤和它是几乎相同的，首先在找出所有以 `test_` 开头的函数，然后将其添加到 `TestSuite` 中，然后使用 `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可能比生产代码本身更加冗余，这也是为什么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着清晰的了解，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解决对代码的回忆，那么其实单元测试并不是必须的。

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采用分

离化处理，以减少改掉 1 个 bug，却出现 3 个 bug 这种让人痛苦的情况的出现的可能性

## 10.5 Selenium 单元测试

在网页测试的领域，传统的爬虫技术（urllib，BS4，requests）传统一般难以做到对 Ajax 和 JavaScript 的处理，但实际上，绝大部分前沿的网站都会采用这些技术框架来达到或是安全性，或是提高体验等诸多目的。

幸运的是，我们有 Selenium，其实，Selenium 项目的本意就是用来进行网络测试，只是后来在网络爬虫中却反而被应用的越来越多。

Selenium 项目测试方法和内置库 unittest 语法还是有些不太一样的，它不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 单元测试更多的是和 Python 本身的断言结合起来，这种断言不需要括号，但要求采用空格分割，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信息提示：

```
assert "test" in driver.title
```

它的单元测试更像是一张闯关的性质，如果不满足某个条件就不会再向下执行，而不是像 unittest 会将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。

```
from selenium import webdriver

dr=webdriver.Chrome()
dr.get("https://en.wikipedia.org/wiki/Python")

assert "Python" in dr.title

dr.close()
```

运行这段代码的时候如果没有意外，读者应该是可以打开一个维基百科的 Python 界面，紧接着会关闭这个界面。

如果我们将“Python”换成一个其他字符，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，比如说，获取某个元素的文本，点击某个按钮，就是对页面的一种单元测试，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，比如说，获取到的文本内容为空，那么我们断言

```
assert string is not None
```

然后对中间出现的错误进行统一的异常处理，使用 try，except 方法来避免因为某一步的出错而整个测试代码的失败。

```
dic= {'Michael': 95, 'Bob': 75, 'Tracy': 85}
try:
    d=dic['test']
except KeyError as e:
    print(e)
```

在这个样子代码中，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，但当我们执行的时候，虽然出现了错误，却可以正常的执行完成。

## 10.6 美化报告

`HTMLTestRunner` 是一个基于 `unittest` 的第三方库，由 Wai Yip Tung 开发。

为什么要使用它呢？在命令行输出测试结果的 `unittest`，输出的内容也许开发者可以看懂，但这样的报告是无法直接发给相关人员使用，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，但安装的时候命令却是

```
pip install html-testRunner
```

安装出来的包是 `HtmlTestRunner`，我们需要的是 `HTMLTestRunner`，虽然看似只有大小写的区别，但我们是无法使用这个库。

因此读者需要到 Wai Yip Tung 的个人网站下载（截止笔者编写本书的时候，最新版本为 0.8.2，并且是针对 Python2 开发的）

<http://tungwaiyip.info/software/HTMLTestRunner.html>

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 2.x 版本的 Python，应该保存完文件后立刻就可以导入库了，如果使用的是 Python3.x，则还需要做一定的修改：

```
No module named 'StringIO'
```

这些修改主要是由于 `StringIO` 和 `io` 库的不同（python3 中将其取代为 `io.StringIO`），以及 Python2 和 Python3 的编码问题造成的。

笔者用箭头标记出来如何修改：

- ❑ 第 94 行，`import StringIO` → `import io`
- ❑ 第 539 行，`self.outputBuffer = StringIO.StringIO()` → `self.outputBuffer = io.StringIO()`
- ❑ 第 642 行，`if not rmap.has_key(cls):` → `if not cls in rmap:`
- ❑ 第 766 行，`uo = o.decode('latin-1')` → `uo = e`
- ❑ 第 775 行，`ue = e.decode('latin-1')` → `ue = e`
- ❑ 第 631 行，`print >> sys.stderr, '\nTime Elapsed: %s' % (self.stopTime-self.startTime)` → `print(sys.stderr, '\nTime Elapsed: %s' % (self.stopTime-self.startTime))`

在修改过后，如果没有意外的话，应该可以正常导入而不出错：

```
import HTMLTestRunner
```

接下来，我们进行代码的编写：

```
#导入库
```

```
import HTMLTestRunner
```

```
import unittest
```

```
#测试文件所在的独立目录
```

```
directory=r"[dir]"
```

```
#遍寻测试文件
```

```
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 文件位置
    filename="E:\\result.html"
    #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()

#只有在主动运行的时候才执行
if __name__ == '__main__':
    main()

```

首先我们导入 `unittest` 和 `HTMLTestRunner` 库，接着，我们定义一个目录（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，比如说我们使用 `*_unittest.py` 表示所有以 `_unittest` 作为文件名结尾的 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：

```

if __name__ == '__main__':
    main()

```

定义了只有在主动运行的时候才执行 `main()` 函数，如果运行无误的话，应当会在我们定义了目录下产生一个 HTML 文件，这个 HTML 文件可以直接使用浏览打开，网页内容如图 10-1:

## 测试报告

Start Time: 2018-09-01 19:17:26

Duration: 0:00:00.062504

Status: Pass 5

实例执行情况：

Show Summary Failed All

Test Group/ Test case	Count	Pass	Fail	Error	View
bookunittest.book_test	5	5	0	0	<a href="#">Detail</a>
test_attr_add			pass		
test_attrerror			pass		
test_init			pass		
test_key			pass		
test_keyerror			pass		
Total	5	5	0	0	

图 10-1 unittest 报告

## 第 11 章 多线程

在最早程序执行路线中，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，必须前面的执行好，后面的才会执行。而且不能充分发挥 CPU 多核的优点，我们都知道，多核多线程的实际能力才是代表着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，否则的话，即便读者的电脑的实际配置很高，但实际运行体验却依旧很糟。

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和系统资源，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是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，那么小绳子越多，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运行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，比如说，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，我们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上使用多线程，而哪些环境则完全没有必要。

- ❑ 程序响应速度，多线程可以同时处理并行任务，在这一点上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，线程的消耗还是小于进程，所以，在程序响应速度上，多线程具有一定的优点。
- ❑ 内存占用，如果仅仅是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，在内存占用上，单线程具有优点
- ❑ 程序出错可能性，在这一点上，多线程任务由于线程过多会导致控制较为复杂，最终反而会导致许多原本不会发生的 **bug**，比如说，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。在程序出错可能性上，单线程占据优势面。
- ❑ 运行效率，这很像我们在小学做过的烤面包 10 分钟，刷牙 5 分钟，洗脸 5 分钟，求最优解的问题，单线程处理的情况就相当于我们傻傻的等待面包烤完之后再再进行刷牙洗脸，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，在这一点上，多线程完败单线程。

## 11.3 实际体验

我们首先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 46:39 到 46:45，一共花费了 6 秒钟的时间。

然后我们再来写第二个版本的多线程的程序：

```
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()))
```

第二次输出：

```
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
```

虽然一秒钟不到就运行完了程序，但读者可能会感到奇怪，为什么输出只有三行，变得这么少了，应该是出了什么问题

不然，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毕，而子线程还未执行完毕，程序就已经退出了。

我们做一个简单的四则运算，从 51:41 到 51:44，程序一共运行了 3 秒，比上面的 6 秒效率提高了 50%。

读者实际在运行的时候，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，却被挤到一行了，这时因为我们使用的 `time.sleep()` 并不是严格的 1 秒钟，它存在毫秒级别上的误差，当程序在运行的时候，很有可能上一个线程的 `sleep` 的时间长一些，而下一个线程 `sleep` 的时间略微短一些，这就导致打印时两者内容被杂乱地一起写入缓冲区，所以打印出来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()`，开始线程的活动

这个方法每个线程最多只能调用一次，它安排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` 错误示警。

❑ `run()`，表示线程活动的方法。

您可以在子类中重写此方法。标准 `run()` 方法调用传递给对象构造函数的可调用对象作为目标参数(如果有的话)，分别使用 `args` 和 `kwargs` 参数中的顺序参数和关键字参数。

❑ `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` 参数的值的话，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()`

顾名思义，用来获取 `name` 值

❑ `setName()`

用来改变 `name` 的值

❑ `ident`

此线程的“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()`

返回线程是否存在，这个方法在 `run()` 方法开始之前返回 `True`，直到 `run()` 方法终止之后。

❑ `daemon`

一个布尔值，指示该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为不是守护线程。

笔者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 库常用函数

`threading` 库提供非常多的方法，但对大部分人来说，完全弄清楚是不现实的。这是那些“专家”才会去做的事情。笔者在这里汇总了 `threading` 比较库常用的函数，掌握了这些，相信读者便可以应付绝大部分需求了。具体内容如表 11-1:

表格 11-1 `threading` 库常用函数

<code>threading.active_count()</code>	返回 <code>Thread</code> 当前活动的对象数,返回的计数等于 <code>enumerate()</code> 返回的列表的长度。
<code>threading.current_thread()</code>	返回当前 <code>Thread</code> 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。
<code>threading.get_ident()</code>	返回当前线程的”线程标识符”，线程标识符是一个非零整数。它的值没有直接意义。当线程退出并创建另一个线程时，可以回收线程标识符。
<code>threading.enumerate()</code>	以列表形式返回当前活动的所有对象
<code>threading.main_thread()</code>	返回主线程，一般情况下，这个主线程都是启动 <code>Python</code> 解释器的线程
<code>threading.TIMEOUT_MAX</code>	最大超时时间
<code>threading.stack_size([number])</code>	返回创建一个线程所需要的最小堆栈的大小，如果指定了一个参数，那么会将这个参数的值设置为最小堆栈的大小
<code>threading.setprofile([func])</code>	对从 <code>threading</code> 库启动的所有线程配置文件
<code>threading.settrace([func])</code>	对从 <code>threading</code> 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，比如说，我们在前面学习的 `SMTP`，当我们发送一封邮件的时候，如果程序想要接收另一封邮件，不可以，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之一：进程同步和数据同步，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()
Lock.release()
```

第一个 `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，那么会直接扣上一把锁，但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。

第二个 `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，如果读者尝试去释放一个释放状态的对象，程序会返回一个 `RuntimeError` 来示警。

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，终点处有一瓶水，当口令一吹响，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，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递给别人。

`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，但如果 `blocking` 的值设为 0，如果想要加锁一个加锁对象，那么会直接返回 `False`。

我们分别对应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
        # 类实例化
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，然后重载 `run()` 方法，`number` 计数自加，然后停止 1 秒钟，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，多线程化执行

```
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
```

然后再来看当我们使用锁的时候的数据

```
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，那么可以说，这个 threading 的作用很类似于我们的程序入口点，它究竟是一个线程还是一个进程呢？实际上，在现有的操作系统中，我们的这个程序对外来说，它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着这一个线程，而这个线程对于它的子线程来说它又是一个进程。

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过度纠结。

## 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，thread 库，threading 库，queue 库和 multiprocessing 库。

thread 库在 Python3.x 中更名为\_thread，\_thread 和 threading 都涉及到线程创建和管理，\_thread 涉及到的内容，更多涉及到底层，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（比如说，没有守护线程的概念），在同等需求的情况下，笔者推荐读者使用 threading 库。简单的来说，当需求对能力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 库的效果比单纯的 threading 会好上一些，使用 multiprocessing 可以实现真正的多核，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 多线程+Selenium 的样例

实际上，在维护一个大型爬虫的时候，我们基本实现思路一般是获取 URL，采用笔者在第七章第八节讲到的布隆过滤器判断是否已经爬取过，如果是，则直接跳过。如果不是，则对该 URL 进行爬取并且记录存储。

我们假设 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 的功能，另一个实现对于页面中需要信息的获取。

通常在实际使用中，对于大型的爬虫，我们都会采用将代码分开写在不同的 Python 文件当中。这样不仅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

```
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 Interpreter 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减少。

锁住 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,要做到多线程安全都会比单使用 GIL 一个锁要难的多,有这么多第三方的扩展也在依赖 GIL。对 Python 社区来说,这不异于自断手脚。

再回到本章开始说的小学算术题,我们使用 threading 实现的多线程更像是一个人高效率的完成多项任务,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,反而大脑会供应不了。

# 第 12 章 发送邮件

本章讨论程序反馈最低廉且合理的方式,通过实现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 通讯的选择

当我们进行自动化测试程序无人值守的时候,我们需要一种通讯方式来告诉我们程序运行到哪里,但笔者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,甚至如果读者选用第三方的服务器的话,成本可以降到零,接着,我们需要考虑人的需求的问题,作为一个正常人,读者你肯定不想在睡觉睡到一半的时候突然来一个预先设定好的数字电话叫你回去修复 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，那么可以直接由发送机器送达接收机器，实际情况却没有这么顺利，在一封电子邮件送达的过程中，我们需要多台电脑作为中转机器（MTA）来负责邮件的路由，队列处理，直到最终的送达，如果你打开一封邮件的头文件，你就会惊讶的发现，一封邮件的送达之后，其实已经被打上了很多道 MTA 的印记。

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，被称为现在电子邮件的基础。恰巧，Python 支持 SMTP，我们可以直接使用 SMTP 来实现电子邮件的发送。

笔者在这里使用 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为严格的限制，在正式进行代码实现的过程之前，我们还需要打开 QQ 的 POP3/SMTP 协议，读者登录自己的 QQ 邮箱，然后打开设置页面，跳转到这一栏，笔者第一行因为已经开启过了，所以显示的是已开启|关闭，读者显示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，如图 12-1。



图 12-1 开发 POP3/SMTP 服务

POP（Post Office Protocol）协议用来实现下载邮件，最新版本是第三版，所以一般称之为 POP3

在开启协议之后，我们还需要获取一个授权码，来供我们的程序使用，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。

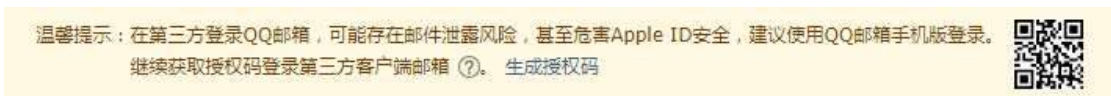


图 12-2 生成授权码

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 和 RFC 1869 查看有关简单邮件传输协议和 SMTP 服务扩展的内容。

`smtplib.SMTP(host="", port=0, local_hostname=None, [timeout, ]source_address=None)`

我们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 参数指定停止操作，如尝试连接 10 秒后无反应（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 指的是如果读者指定了这个参数，那么读者在使用 HELO/EHLO command 进行初始化的时候，这个参数的值就会被当做本地主机的 FQDN (Fully Qualified Domain Name, 全限定域名，FQDN 使用 “.” 将主机名和域名连接起来，比如读者的主机名为 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 @，这也就是我们邮箱地址的由来)。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 常用方法如表 12-1 所示：

表 12-1 SMTP 常用方法

<code>sendmail([from],[to],[msg])</code>	将邮件[msg]（用列表或者元组来表示）从[from]发送到[to]
<code>login([user],[password])</code>	使用用户和授权码来登录服务器
<code>ehlo()</code>	初始化 SMTP 或 ESMTP 会话

helo()	同上
starttls(keyfile=None,certfile=None)	启用 TLS 模式，默认 keyfile 和 certfile 为空如果给定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：

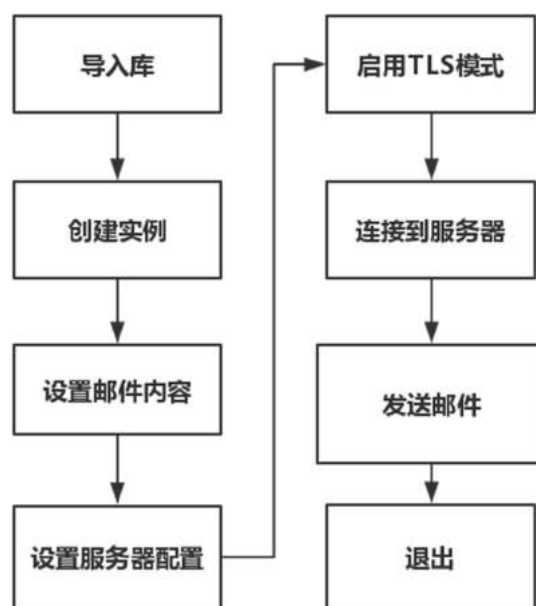


图 12-5 发送邮件流程图

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。smtpplib 模块用于建立关于服务器连接的信息。

在这里用到了 Python 的两个包来发邮件: smtpplib 和 email, 我们来看一下程序:

```
#coding:utf-8
import smtpplib,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")

```

如果出现类似于这样的错误，那么证明读者在获取授权码的某个步骤出错了。

```
smtplib.SMTPAuthenticationError.....
```

如果一切无误的话，读者应该可以看到一个我们的程序输出 **done** 并且受到邮件。

下面这条命令是一个复合命令，首先，我们使用 `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），然后我们使用 `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（如图 12-6）。



图 12-6 邮件发送成功

## 第 13 章 Selenium IDE

Selenium IDE 和 Katalon Recorder 是两个简单且方便的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。本章将通过对比学习这两个录制工具为读者提供一条功能化的测试之道，以快捷生成自动化脚本。

### 13.1 IDE 安装

Selenium IDE 是一个官方的插件，有 Chrome 和 FireFox 两个版本，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），笔者在这里提供一个 FireFox 历史版本下载的网站（此为 FireFox）官方网站。

在这里，我们转战浏览器，不再主要使用 Chrome 浏览器，我们以 FireFox 浏览器作为例子，不仅仅是因为 Selenium IDE 支持 FireFox，更重要的是，我们将会讲解并使用一个比官方版本插件更好的第三方插件（笔者个人看来）。


方法一：

从 selenium 的网站下载扩展组件文件，和之前的几个内容的页面位置相似，同样实在 selenium 官方网站的下载页面上，如图 13-1。



图 13-1 IDE 下载

方法二：

点击菜单按钮 ，然后在弹出来的界面，选择附加组件（如图 13-2），或者读者直接使用快捷键 Ctrl+Shift+A 也是可以的（这个快捷键和 QQ 的截图快捷键是相同的，可能会发生按键冲突的问题）。

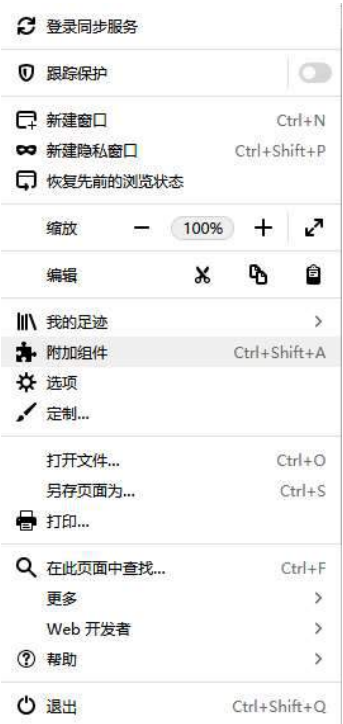


图 13-2 设置——选项

我们在右上角的输入框中输入 Selenium，然后点击搜索，图 13-3 即为我们需要的插件。

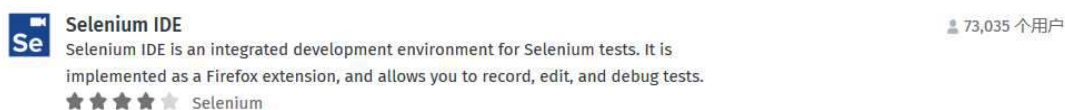


图 13-3 搜索结果

搜索结果显示的排名第一的内容就是我们需要的插件，单击，点开即可（下一步如图 13-4 所示）。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。

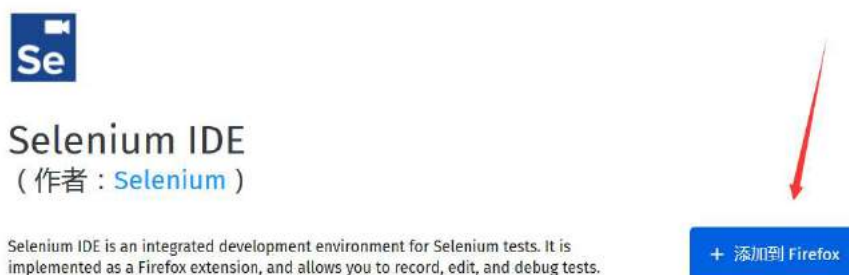


图 13-4 插件页面

然后选择添加到 Firefox 选项，之后，只需要静静的等待插件安装完成即可，这可能会花费十几秒钟的时间。

## 13.2 Selenium IDE


当两个内容都安装完成以后，我们来从难到易，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，从 Firefox55 之后的版本的，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）。

原因大概有这么几个，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没精力推动 Selenium IDE 更新适应新技术，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，总所周知，Firefox 是唯一的迭代速度上可以和 Chrome 比肩的浏览器。Selenium 作为一个有些小众的项目，并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”格式拓展变更为“web extension”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，或是花费一定的时间，来了解 Katalon Recorder（它的界面和 Selenium IDE 很相似）。

另外，点击菜单按钮 ，点击帮助，然后选择故障排除信息。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（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）：



名称	Firefox
版本	61.0.1
版本 ID	20180704003137
更新历史	显示更新历史
更新通道	release-cck-mainWinStub-mozillaonline
用户代理	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
操作系统	Windows_NT 10.0
配置文件夹	打开文件夹 C:\Users\yuyichenmo\AppData\Roaming\Mozilla\Firefox\Profiles\bcoxtj2s.default
已启用的插件	about:plugins
构建配置	about:buildconfig
内存使用	about:memory
性能	about:performance
已注册的 Service Worker	about:serviceworkers
多线程窗口	1/1 (默认启用)
网页内容处理进程	4/4
企业策略	非活跃
Google 密钥	存在
Mozilla 位置服务密钥	存在
安全模式	false
配置文件	about:profiles

图 13-5 版本信息

我们来看一下 selenium ide 的界面，笔者在这里用红框来分别标注出每个部分，如图 13-6。

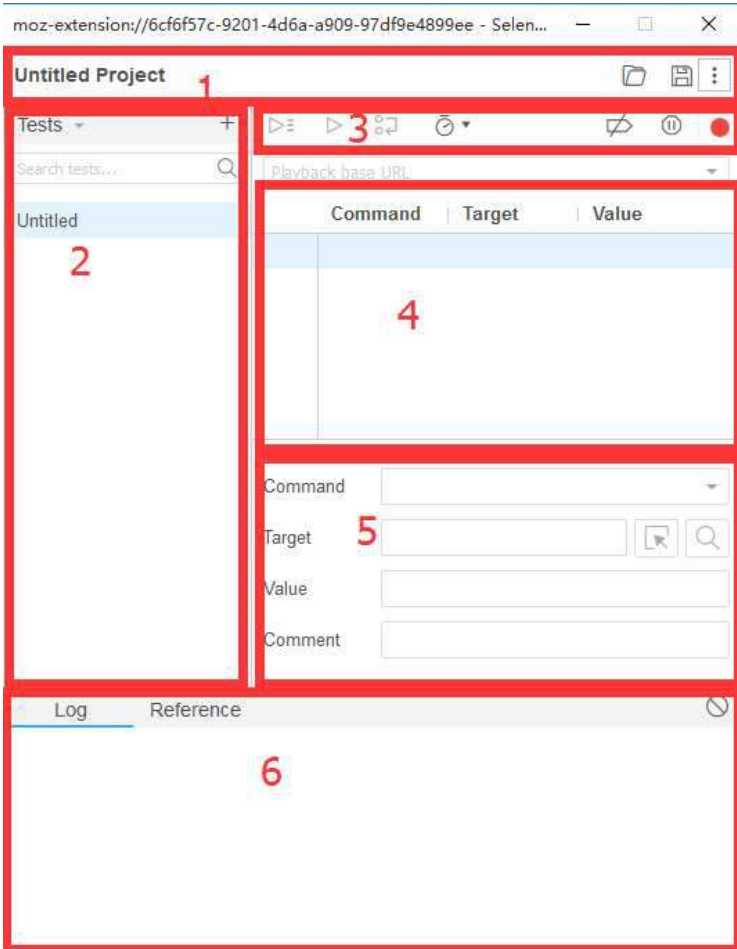


图 13-6 界面介绍

第一部分为最上面的标题栏，左上角的 **Untitled Project** 双击后可以修改项目的名字。右

边的三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），保存，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：

图 13-7 更多按钮

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，比如说，当笔者在这里创建一个新的文件就会出现一个新的子项目（读者在这里将其命名为 `test` 和 `test2`），如图 13-8 所示：

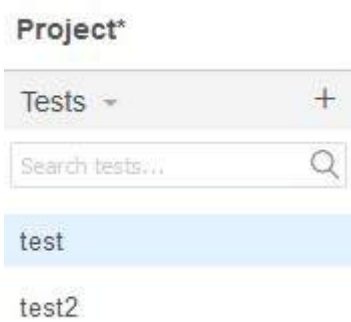


图 13-8 项目分支

第三部分的按钮分别是：

- ☐ 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我们写的代码部分的内容，如果我们打开百度的页面进行一次录制搜索，读者就会在这里找到熟悉的 `id` 为“`su`”和“`kw`”的元素，如图 13-9：

2	<i>mouse over</i>	css=div.riW RTe.HTXiQJ > a.whUXqt	
3	<i>type</i>	id=kw	selenium
4	<i>submit</i>	id=form	
5	<i>mouse out</i>	css=div.riW RTe.HTXiQJ	

图 13-9 脚本部分

第五部分则是对第四部分的单条命令的解释，准确的说是项目下的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` 这个参数是当读者输入内容的时候，`value` 的值就是键入的内容，而 `comment` 默认是没有的，除非读者主动来写。

第六部分共有两个内容，第一个是日志，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="点击一下，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.rjWRTe.HTXiQJ > a.whUXqt...
```

而 `reference` 的性质更像是 Selenium IDE 的说明文件，当读者点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，实际的使用方法倒是相当简单，单击 `start recording` 的红色按钮，开始录制，然后进行一遍实际的操作之后，这时红色按钮就会变成 `stop recording`，单击以结束录制，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，可以通过 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。

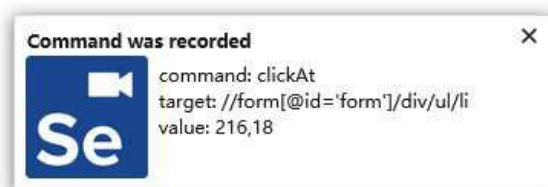



图 13-10 弹出窗口

接下来我们来录制一个简单的程序，再打开 Firefox 浏览器后，点击右上角的  按钮打开 Selenium IDE 的界面，然后点击红色的 `start recording` 按钮，这时，将 Selenium IDE 的界面最小化，我们进入主页，然后搜索火狐插件，选择第一个结果，进入页面，然后输入 Selenium IDE，点击搜索，打开 Selenium IDE 的页面，最后，点击停止录制，当我们看到方形的红色按钮变为圆形，意味着我们录制完成了，然后点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，我们在百度页面完成搜索以后，尝试从百度页面跳转到火狐插件官网，第一次使用的时候会弹出一个是否允许 `baidu.com` 弹出页面，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 认为是弹出式页面，如图 13-11。

3	<code>type</code>	<code>id=search-key</code>	火狐插件
4	<code>submit</code>	<code>id=search-form</code>	
5	<code>click at</code>	<code>linkText=Firefox 附加组件</code>	77.9
6	<code>mouse over</code>	<code>linkText=Firefox 附加组件</code>	

图 13-11 弹出页面对应命令

当运行成功后，我们点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），通过对脚本的回放来进行自动化测试。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 界面介绍

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 早期版本的功能的强大,它的界面和现今版本的 Selenium IDE 有一定形式上的相似,如图 13-12。

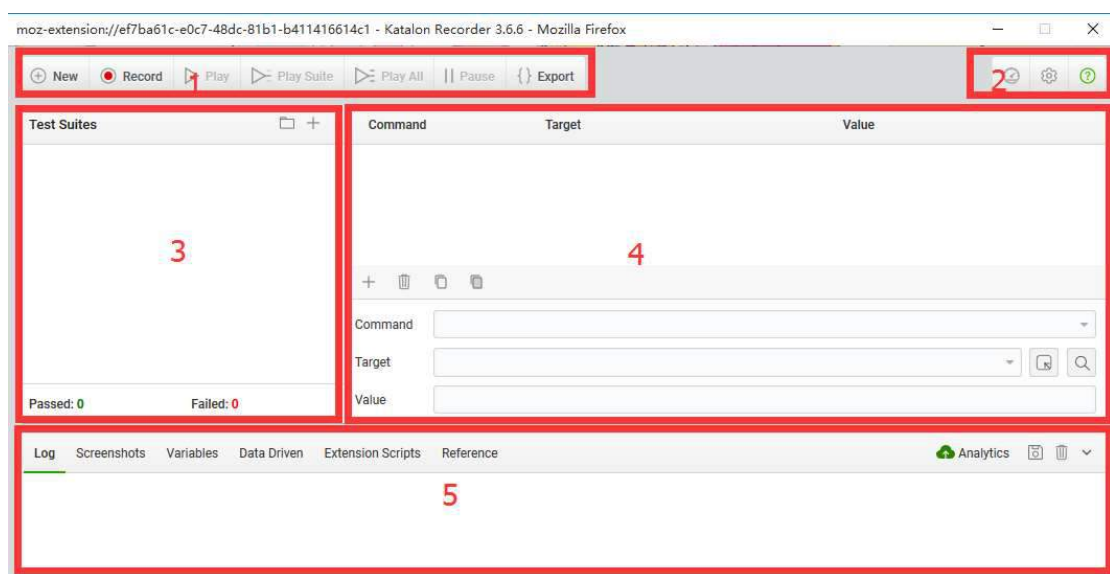


图 13-12 界面介绍

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 说明文档称自己为测试团队最好的自动化测试工具,虽然有一定夸大的嫌疑,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)。

同样的,我们先来分部分介绍 Katalon Recorder 的界面。

第一部分是 Katalon Recorder 的按钮,从左到右依次是

- ☐ 创建,创建一个新的实例
- ☐ 记录/停止,开始记录操作到当前实例
- ☐ 执行,执行当前实例
- ☐ 执行,执行当前项目的所有实例
- ☐ 暂停,暂停执行实例
- ☐ 导出,导出当前套件

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，推荐读者以我们熟悉的 Python 语言形式导出

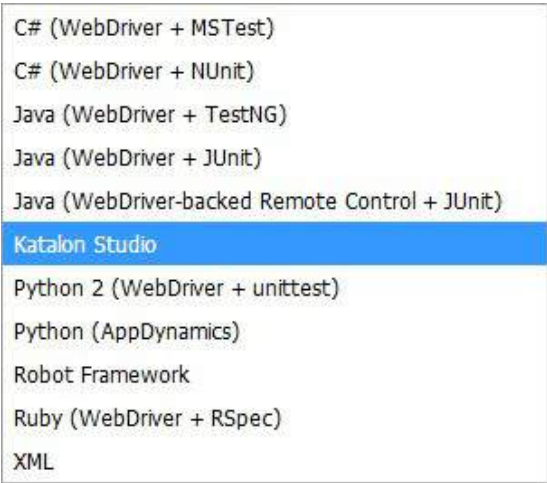


图 13-13 可供导出的语言

然后第二部分三个按钮分别是：

- ☐ 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，可以滑动的调节滚动轴。

端口设置在火狐浏览器中暂时是无法使用，目前仅仅针对 Chrome 浏览器可以使用，官方是这样说的——“Allow users to change the default port used by [Katalon Studio](#) to communicate with the active Chrome browser (support for Firefox is coming soon)”。

目前，在火狐浏览器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 展示的是帮助部分四个内容，按照从上到下，从左到右的顺序，依次是用户手册，生态社区，教程视频，更多扩展自动化工具。

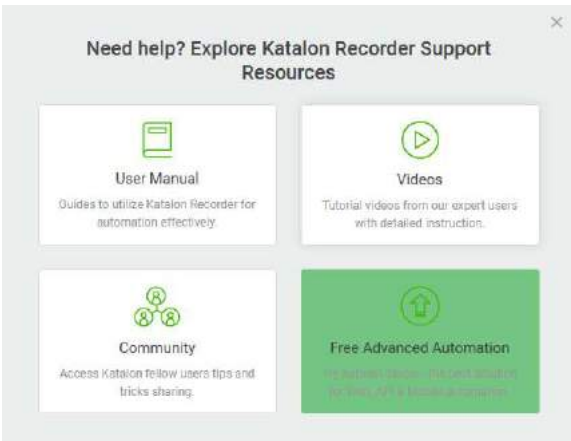


图 13-14 帮助部分

### 13.3.2 实例与步骤

在第三部分会显示我们的测试条件和实例：

一级标题是项目（suite 意为套件，称之为项目或许更合适），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。



图 13-15 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，左下角的四个按钮分别是添加步骤，删除步骤，复制步骤，粘贴步骤，如图 13-16。

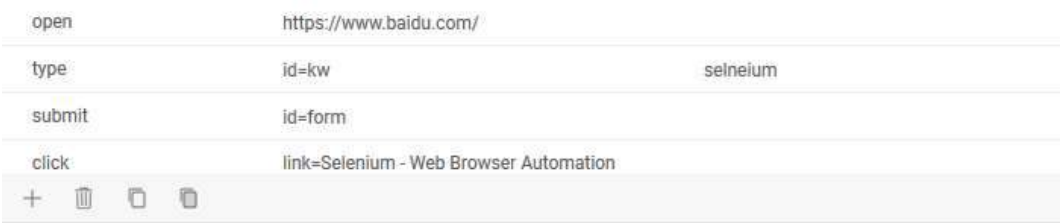


图 13-16 具体步骤

下方是对步骤的具体解释如图 13-17，比如说，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，键入的值是 selenium，就相当于：

```
driver.find_element_by_id("kw").send_keys("selenium")
```



图 13-17 步骤解释

最下方的界面如图 13-18 所示，内容分别为：

- ☐ 日志，实例执行的日志
- ☐ 截图，显示了测试实例中的截图
- ☐ 变量选项卡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 选项卡允许用户上传 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 选项卡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。
- ☐ 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。
- ☐ 保存日志

## ❑ 删除日志



13-18 下方界面

如果使用数据分析功能的话，读者还需要搭配 Katalon Analytics 来使用，这需要读者注册一个 Katalon 的账户，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 提供了一个叫做“captureEntirePageScreenshot”的截图命令，比如这样，读者新建一个步骤，然后在命令设置为“captureEntirePageScreenshot”即可实现截图，获得的截图可以在下方的第二个选项卡查看，如图 13-19：



图 13-19 截图命令

## 13.4 数据驱动

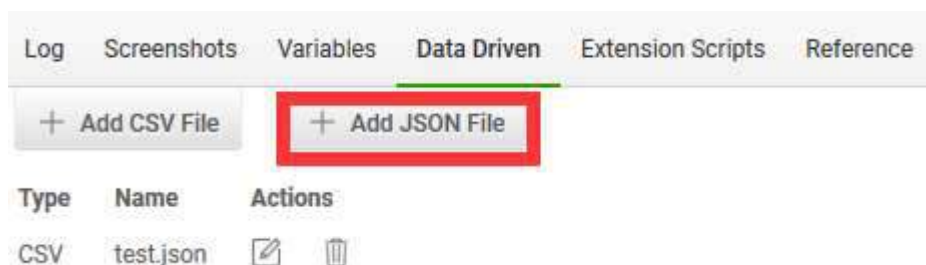
当我们进行自动化测试的时候，如果数据量特别大，比如说，我们尝试一个8位数的密码的所有组合，肯定不能使用重复的录制实现。这时候，我们需要实现数据和程序的分离，在执行的时候，重复调用特定格式的数据，进行模式相同的测试，简单的来说，就是循环。

我们使用数据驱动来实现Katalon Recorder的进阶用法：

首先我们在桌面创建一个txt文件，输入以下内容，保存，将txt文件的格式后缀名改为json文件，这时候，我们会用到上面提到的Data Driven（数据驱动），



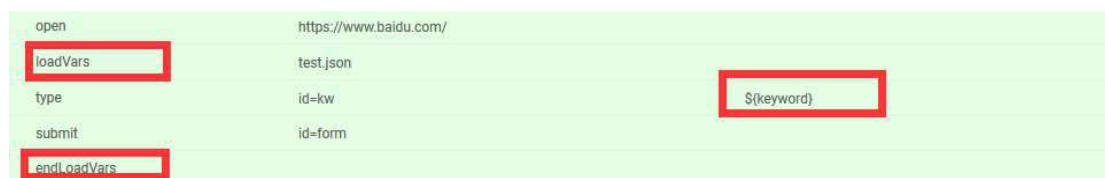
我们打开 Katalon Recorder，然后在 Data Driven 中添加，我们刚才创建的 json 文件，然



后打开百度，搜索一个内容，把步骤录制下来，接下来，我们在循环体的先后分别加入步骤 loadVars 和 endLoadVars，如图 13-20：

图 13-20 添加 JSON 文件

需要更改的三个地方笔者在这里已经用红框标注出来了，loadVars 和 endLoadVars 之间是我们重复执行的步骤，在 loadVars 命令的 Target 部分指定数据文件的名称，我们使用的是 test.json，然后，我们在 type 命令的值中添加 \${keyword}，这时一个定义的变量，如图 13-21。



open	https://www.baidu.com/
loadVars	test.json
type	id=kw <span style="border: 1px solid red; padding: 2px;">\${keyword}</span>
submit	id=form
endLoadVars	

图 13-21 定义变量

好了，接下来读者就可以运行试试看了。如果没有意外，应该可以看到先打开了百度，然后在搜索框逐次的输入了 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。这里的扩展脚本大多是 JavaScript 内容，需要读者对于 Java 有一定的基础，这些并不是我们学习的重点，只推荐读者在学有余力的情况下可以对这里的内容进行一些深入的探索。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“添加扩展脚本”按钮。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，如图 13-22：

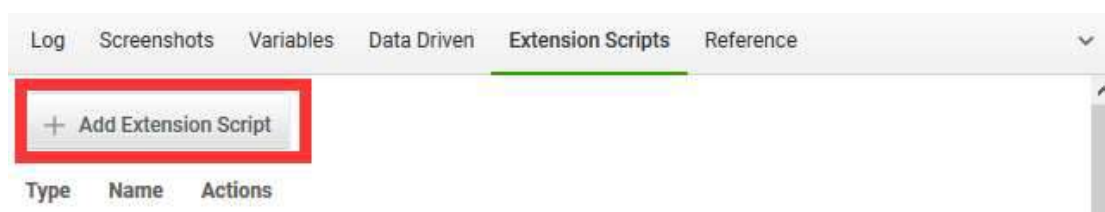


图 13-22 添加扩展脚本

### 13.5.2 定位器构建器

一个定位器意味着识别一个命令指定的元素，而定位器构造器是用来创建这些定位器，在这面这个定位器构建器，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，可以使用下面的命令来调整定位器构建的顺序到读者想要的样子

```
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

当读者使用 Selenium 的所有 “get[name]” 和 “is[name]” 方法都会自动影响这些命令的可用性

- ☐ “store[name]”
- ☐ “assert[name]”
- ☐ “assertNot[name]”
- ☐ “verify[name]”
- ☐ “verifyNot[name]”
- ☐ “waitFor[name]”
- ☐ “waitForNot[name]”

例如，如果您添加一个 “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，读者需要提前在电脑上安装 Java Runtime Environment version 8.x 或者以上。

接下来，读者可以从发布页面下载最新版本的“katalon-recorder-helper.jar”文件

<https://github.com/katalon-studio/katalon-recorder-helper/releases>

在下载完成之后，读者从文件所在目录打开 cmd（shift+右键），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 本身是有什么局限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 即可实现上传文件）。

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 成功来实现模拟键鼠操作来触发文件上传对话框。

Katalon Recorder Helper 并不是仅仅只有这一点功能，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。

比如说，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
JSON 原始数据 头
保存 复制
command: "dir"
output:
0: " 驱动器 C 中的卷是 系统"
1: " 卷的序列号是 7464-B46A"
2: ""
3: " C:\Users\  \Downloads 的目录"
4: ""
5: "2018/08/19  12:14    <DIR>          ."
6: "2018/08/19  12:14    <DIR>          .."
7: "2018/08/19  12:14          16,134,550 katalon-recorder-helper.jar"
8: "                      1 个文件      16,134,550 字节"
9: "                      2 个目录 39,657,947,136 可用字节"
```

图 13-23 执行结果

页面返回的内容就是在 cmd 中执行这条命令会输出的内容。

## 第 14 章 Python 拓展

这一章节讨论内容包括但不限于：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，推导式等等扩展性内容。这些内容将会帮助读者更好的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 Python2 编写的优质的库可能由于更新的不及时，或者已经不再更新，而无法被使用 Python3 的用户使用。

幸好，Python 已经为我提供一个解决方案——2to3.py，并且这个 Python 文件是 Python 安装以后自带的文件。

2to3.py 存在于 Python 目录下的一个 `tool\scripts` 目录中，例如笔者的 2to3.py 的目录为：

```
F:\Python\Tools\scripts
```

首先在文件管理器中打开相应的目录，其次在 `tool\scripts` 目录下同时按下 `shift` 和鼠标右键，选择在此处打开命令窗口，最后在弹出的命令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 Python2 文件进行批量转换。

另外几个常用的参数分别为 `-n` 表示不对被转化的文件进行备份，`-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:
```

都是采用三个减号表示原来的文件，采用三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 文件的内容已经被修改过了，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` 是对原先 Python2 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，存在以下几种方法可以用来便捷的进行类型测试：

首先我们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

```
if type(l) == type([]):
```

```
    print("L is list")
```

#方法二

```
if type(l) == list:
    print("L is list")
```

list 是 Python 的保留字，可以直接被拿来使用，有一点需要说明一下，变量的命名不能和保留字相同，比如说，我们不能把新定义的一个列表命名为 list。

```
#方法三
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，更准备的说，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，都是用于提供一定程度上的可控性，以%开头的格式化字符都叫做通配符。通配符细分的种类有很多，每个都代表着不同的格式，比如说，下面两条函数的输出效果是相同的， %d 在这里用于替换 Int 整形。

```
print("我今年%d 岁" %12)
print("我今年 12 岁")
```

笔者在表 14-1 中罗列出大部分常见的通配符，供读者查阅。

表 14-1 通配符

Python 通配符	
%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(用十六进制打印值的内存地址)
%n	存储输出字符的数量放进参数列表的下一个变量中

### 14.4 str 方法

使用 NLTK 进行词性分析以及对收集到的文字进行统计归纳的时候，必然少不了对字符串的处理，这里罗列了 str 类的内置方法。

表 14-2 str 方法

str.ljust(width)	获取固定长度，右对齐，左边不够用空格补齐
------------------	----------------------

<code>str.rjust(width)</code>	获取固定长度，左对齐，右边不够用空格补齐
<code>str.center(width)</code>	获取固定长度，中间对齐，两边不够用空格补齐
<code>str.zfill(width)</code>	获取固定长度，右对齐，左边不足用 0 补齐
<code>str.find('t',start,end)</code>	查找字符串，可以指定起始及结束位置搜索
<code>str.rfind('t')</code>	从右边开始查找字符串
<code>str.count('t')</code>	查找字符串出现的次数
<code>str.replace('old','new')</code>	替换函数，替换 old 为 new，参数中可以指定 maxReplaceTimes
<code>str.lstrip([str])</code>	从字符串左侧取出指定字符，默认为空格
<code>str.rstrip([str])</code>	从字符串右侧取出指定字符，默认为空格
<code>str.strip([str])</code>	从字符串中取出指定字符，默认为空格
<code>str.startswith('start')</code>	是否以 start 开头
<code>str.endswith('end')</code>	是否以 end 结尾
<code>str.isalnum()</code>	判断字符串是否全为字符
<code>str.isalpha()</code>	判断字符串是否全为字母
<code>str.isdigit()</code>	判断字符串是否全为数字
<code>str.islower()</code>	判断字符串是否全为小写
<code>str.isupper()</code>	判断字符串是否全为大写
<code>str.upper()</code>	全部大写
<code>str.lower()</code>	全部小写
<code>str.swapcase()</code>	大小写转换
<code>str.capitalize()</code>	句子首字母大写
<code>str.title()</code>	每个单词的首字母都大写

有一点需要说明一下：

```
str.index(str, beg=0, end=len(string))
```

```
str.find(str, beg=0, end=len(string))
```

两者的使用在很大程度上相似，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。

## 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，代表着错误的 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，我们需要使用 `try..except` 语句进行异常的捕获。

捕获异常也有一定规则，在这里笔者罗列出异常的层次结构，方便读者查表对照使用：

```
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，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（基本上这些错误来源于人为）。

如果想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，异常处理都是划分的越加明细越好。

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** 和 **UnicodeDecodeError**），因为

UnicodeDecodeError 异常会被递交到上级 UnicodeError 异常。

## 14.6 兼容 Python2 和 3

一种常见的解决方法是将 Python 目录下的 python.exe 改名，比如说，将 Python2 版本的 python.exe 更名为 python2.exe，这样每次在运行文件的时候，输出 python2 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 到对应的 Python 目录的 Scripts 目录中。

不过，我们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 Python3.3 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，它默认放在文件夹 C:\Windows 目录下，这个启动器允许我们指定版本号，甚至是 pip 的版本号。

由于两个包管理器都叫做 pip，甚至是都存在于环境变量中，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]
```

还有另一种方式指定 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，优先使用声明编译器作为第一行，而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 Python2.x 的版本中是必须的，在 Python3 的版本中默认支持中文。

所以，Python2 版本的文件声明开头如下：

```
#!/python2
# coding: utf-8
```

而在 Python3 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——在自己电脑上可以正常运行的代码移植到不同的操作系统（甚至同一操作系统的不同版本），不同的电脑中却无法运行。缺乏兼容性的代码就像是是缺了那么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 Python2 到 Python3 的大版本切换中，很多库的名字都被更改掉了，但具体内容

却很少存在变动，我们可以针对 Python2 和 Python 写两套导入的库。

```
try:#python2
from UserDict import UserDict
#建议按照 python3 的名字进行 import
from UserDict import DictMixin as MutableMapping

except ImportError:#python3
from collections import UserDict
from collections import MutableMapping
```

## 14.7.2 输出函数

然后是常见的输出函数，Python3 的 `print()` 函数可以在 Python2 中运行（事实上在 python2.6 中已经带了 `print()` 函数），而 Python2 的关键字 `print` 则无法在 python3 中使用。推荐读者不管是在 Python3 还是 Python2 中直接使用 `print()` 函数。

但有一点需要注意，Python3 和 Python2 的 `print()` 函数略有不同：

#python3

```
>>> print(1, 2)
1 2
```

#python2

```
>>> print(1, 2)
(1, 2)
```

还有另外一种解决方案，使用 `sys` 库来实现输出：

```
import sys
```

```
sys.stdout.write("Hello World!\n")
```

`sys.stdout.write()` 在 IDLE 交互模式下使用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 Python2 中，我们使用如下方式进行异常处理：

```
try:
...
except ValueError, e:
...
```

而下面这种方式在 Python2 和 Python3 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()
```

那么对于这个类有如表 14-3 所示的几种方法：

表 14-3 类方法

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，比如说，我们设置一个带有空格的属性名，这个属性不能被直接访问，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，动态绑定使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，我们先从 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。

然后导入 MethodType()，使用 MethodType()方法将我们刚才定义的 case\_value()函数和实例化的类对象绑定到一起，然后为绑定后的函数赋值。

# 14.9 推导式

假设我们现在有一个列表是这样的

```
numbers = [1, 2, 3, 4]
```

如果想让所有元素翻倍，那么按照我们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是很值得被推崇，我们有一种更好的实现方式：

```
numbers =[elem * 2 for elem in numbers]
```

这种方法叫做 Python 的列表推导式。

再看一种略微高级一些的用法，我们从列表中查找可以直接被 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]
```

我们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，我们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。

类似的，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]}
```

或者是：

```
>>> 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，那么对应的也会有元组推导式吗？其实不然，在 Python 中圆括号被用作生成器的语法。

```
>>> 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'>
```

是的，虽然无法直接实现 Python 的元组推导式，但事实上，通过类型转换函数我们达到的效果也是近似的。

## 14.10 经典除法和真除法

可能读者在其他任何地方上可能听到过这样的教学，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。

这种说法如果用在其他的编程语言上确实没有错误，但在 Python 中却有些不同，在早期的 Python2 版本中确实是采用这种经典除法，但在后来的 Python3 中，这种除法已经逐渐被取代了。

```
>>> 1/2
0.5
>>> 1.0/2.0
0.5
```



诚如读者所见到的，我们在进行上面几个例子的时候（Python3.x 演示），1/2 和 1.0/2.0 的效果是一样的。

如果读者仍旧想要在 Python3.x 的版本中使用经典除法，那么可以

```
>>> 1//2
0
```

而在 Python2.x 中使用真除法则可以使用：

```
from __future__ import division
```

## 14.11 \_\_future\_\_

鉴于 Python 是一个社区驱动的开放源码和自由开发语言，不受商业公司的控制，所以 Python 的改进经常是非常彻底并且不兼容前面的版本（也就是在当前版本运行正常的代码，到下一个版本运行就可能不正常了）。因此，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。

如果可以通过如下的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 位于这一目录：

```
Python\Lib
```

在 3.x 的版本，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，不可避免的要进行类型的转换和更改，表 14-4 中的内容，或许可以帮到读者：

表 14-4 类型转换方法

int(x [,base])	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 转换到一个浮点数
chr(x)	将一个整数转换为一个字符

<code>unichr(x)</code>	将一个整数转换为 <b>Unicode</b> 字符
<code>ord(x)</code>	将一个字符转换为它的整数值
<code>hex(x)</code>	将一个整数转换为一个十六进制字符串
<code>oct(x)</code>	将一个整数转换为一个八进制字符串
<code>list(s)</code>	将序列 <code>s</code> 转换为一个列表
<code>set(s)</code>	转换为可变集合
<code>frozenset(s)</code>	转换为不可变集合
<code>complex(real [,imag])</code>	创建一个复数
<code>eval(str)</code>	用来计算在字符串中的有效 <b>Python</b> 表达式,并返回一个对象

**Python** 关键字或者又叫保留字，代表着一系列已经被 **Python** 使用的字符，变量名不可命名为这些关键字，并且每一个关键字都有着特定的含义和方法。

`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，图形化测试

到目前为止，我们的所有内容都是处于一种文字交互的状态，

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，简直是反人类的操作。

**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。）

`Tk+interface=tkinter`

**Python** 提供的 **IDLE** 就是使用 **tkinter** 设置出来的

### 15.1 tkinter 测试

在 **Python3.2** 之后的版本，**tkinter** 库已经被内置到 **python** 当中了

但如果读者比较巧合的使用 3.x 版本的 Python 同是又在版本号又在 3.2 以下的话，那么读者需要通过 pip 来安装 tkinter

```
pip install tkinter
```

当读者想要使用的时候，可以直接导入 tkinter 库。

```
import tkinter
tkinter._test()
```

如果读者使用的是 2.x 的 Python，那么你需要将 t 换成 T:

```
import Tkinter
Tkinter._test()
```

如果一切无误的话，读者应该可以籍此打开一个简单的测试界面了，如图 15-1 所示。

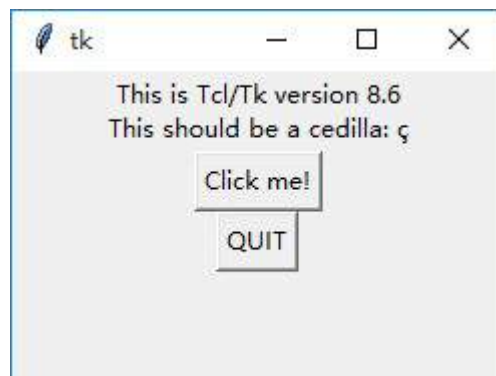


图 15-1 测试

这个测试窗口还可以通过另一种方式打开，win+r 键打开运行，然后输入 cmd，在命令行中执行命令

```
python -m tkinter
```

效果也是相同的。

## 15.2 创建界面

我们的 GUI 界面并不是像设计师那样画出来的，而是采用程序生成的，先来生成界面，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 Label 标签
text=tk.Label(root,text="a simple tkinter GUI")
#将文字居中
text.pack()

#程序循环
root.mainloop()
```

效果如图 15-2:



图 15-2 实现效果

我们首先导入库，为了书写方便，把 `tkinter` 库简写成 `tk`，实例化的对象实际上就已经相当于创建完成了界面，但是这个界面还只是一个空架子，需要我们填充一点内容上去。

`app.title()` 设置并调整窗口的标题，笔者采用 `label` 标签显示文字，`label()` 控件的第一个参数是我们的实例化对象，而第二个是需要显示的文字，那么 `text.pack()` 用来干嘛呢？实际上，这个 `pack` 用来调节文字的尺寸和位置，在下面的小节还会用到几个类似的方法。

最后的 `mainloop()` 是必不可少的，如果我们不使用这个的话，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，否则的话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，和用户进行交互式必不可少的，我们来看一下按钮的是怎样实现的。

```
#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()
```

结果如图 15-3 所示：



图 15-3 按钮控件

`tkinter` 大部分控件的参数都基本上是相同的，比如说，我们刚才实现的 `Lable` 和 `Button` 控件实际上只有一个 `command` 参数的不同。

## 15.4 pack 属性

`pack()` 中可以设置位置，设置参数 `side=tk.LEFT`，方位一共有四个，分别是 `tk.RIGHT`, `tk.TOP`, `tk.bottom`，如果读者不想要让控件直接靠边，还可以设置距离上下，左右边框的距离，`padx`,

`pady` 用于设置距离，`x` 表示的是距离左右边框的距离，而 `y` 表示的是距离上下边框的距离。

```
button=tk.Button(root,text="i'm a button",fg="blue",command=hello_world)
button.pack(side=tk.LEFT,padx=10,pady=10)
```

`pack` 还有另外一种设置控件位置的方式，可以使用 `anchor` 方式，使用 `anchor` 方式 `NSEW`（北，南，东，西的缩写）分别对应上下左右以及 `CENTER` 对应中间方位，注意，读者需求使用的是大写字母。

```
checkboxbutton=Checkbutton(root,text=i,variable=variable[-1])
checkboxbutton.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。

```
checkboxbutton=Radiobutton(root,text=i,variable=variable,value=counter,indicatoron=False)
checkboxbutton.pack(fill=X)
```

`expand` 属性表示的是我们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的继承的方法来实现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()
```

效果如图 15-4 所示:



图 15-4 继承实现效果

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, 比如说, 在 `Label` 在某个 `Frame` 中, 这时候 `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, 所以我们默认 `master=None`

```
super().__init__(master)
```

这一行代码调用我们创建的类 `Application` 的父类 `Frame` 的 `__init__` 函数初始化 `Application` 类中的 `Frame` 类部分。

```
self.create_widgets()
```

此代码用于调用后面定义的 `createWidgets` 方法

```
self.say_hello=tk.Button(self,text="click me to say\nHello World",command=self.say_hi)
```

这行代码读者也可以这样分开写, 分开书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 Checkbutton 和 Radiobutton

下面我们来讲一讲代码多选框和单选框的实现。Tkinter 中一些组件, 如 `Button`, `Label` 等可以设置一个 `textvariable` 属性, 这些字符会随着当这个对象的值被重新设置的时候自动变成新的值。

常见的值有 `IntVar`, `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 `checkboxbutton` 状态的不同做出不同的反应，这里，笔者特意显示出来只是为了让大家了解这种变量的使用方法和效果。

```
#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，变量值取变量列表中的最后一个
    checkbox=Checkbox(root,text=i,variable=variable[-1])
    #居中
    checkbox.pack()

    #显示变量值并居中
    label=Label(root,textvariable=variable[-1])
    label.pack()

#窗体循环
mainloop()
```

在这个代码中，我们首先创建窗体，然后定义了选项列表和变量列表，然后使用循环，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，我们将 `test_dic` 中的元素赋值给 `Checkbox` 的文字属性，将变量列表中的最后一个元素赋值给变量属性，然后使用 `label()` 显示变量列表中的最后一个元素，方便我们观察。

点击前后的效果分别如图 15-5 和 15-6 所示：

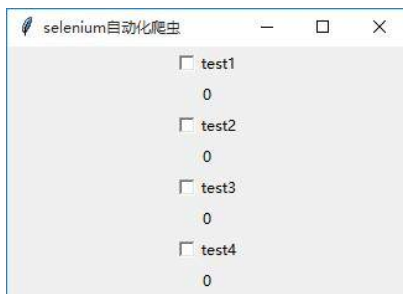


图 15-5 尚未点击



图 15-6 点击

接下来，我们使用循环来实现单选框：

```
#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，我们常规的导入库，创建窗体和标题，定义了一个选项字典，这次我们没有使用变量列表，而是使用一个唯一的变量，我们选择 **counter** 计数器来实现每个单选的值属性都不同，最后使用 **mainloop()** 实现窗体循环。最终效果分别如图 15-7 和 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** 的值是相同的，所以返回的是 1，其他的选项 **value** 和 **variable** 的值都是不同，所以不会勾选，籍此，我们实现了单选框。

如果使用圆形的圆点设计和读者设计的 GUI 整体的风格不符合的话，读者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:

```

checkbutton=Radiobutton(root,text=i,variable=variable,value=counter,indicatoron=False)

```

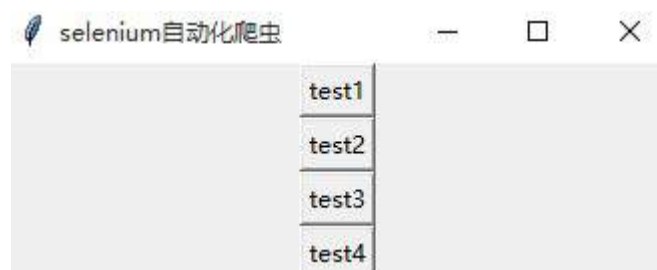


图 15-9 风格调换

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为 **False**。



## 15.7 Entry 输入框

输入框是我们用来实现程序 and 用户进行的交互的控件，比如说，在我们登录一个网站的时候，我们需要键入账号和密码，并且有时候密码还会加密显示，比如说，对于密码的每一个位置都用星号代替。

当应用程序或者网站对用户 provide 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不可避免的。

借此，我们来实现一个简单的 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()
```

前面的创建窗体部分都是一样的，不同的是，在这里，笔者用到了 `LabelFrame`，这个控件的理解涉及到我们在前面讲解的 `master` 内容，笔者如果不创建 `LabelFrame` 控件，那么接下来通过代码实现的 `Label` 标签和 `Entry` 标签都是从属于顶级窗口，而如果加上了 `LabelFrame` 控件，那么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，举一个显示生活中的例子，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。

我们使用的 `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，例如说：

```
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 中的计算一般都是从 0 开始），因为还有 `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` 参数我们在前面提到过，指的是距离上下，左右边距的距离。因为登录框并不需要很多，所以笔者设定各为 5。实现效果如图 15-10 所示：

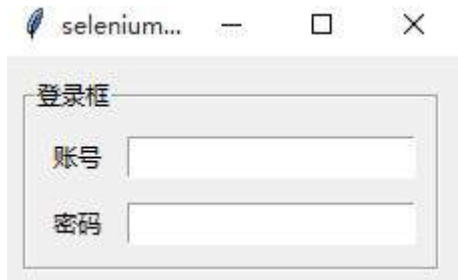


图 15-10 简单登录框

但现在仅仅有一个基本的输入的地方还是不够，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。

然后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：



图 15-11 输入加密显示

那么是怎么实现的呢？

首先，我们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，如果想要从第二个索引位置开始的话可以将 0 改成 1，除此以外，还支持其他方式设置开始的位置：

```
text.insert(1, "[text]")
```

```
text.insert(END, "[text]")
```

类似的，`Entry` 控件还有一个删除的方法，我们在上面的样例代码中并没有使用。

这一行代码用于删除第十一个索引位置的字符，

```
text.delete(10)
```

下面两行代码分别用于删除从第一个到第四个索引位置的字符，和删除所有的字符

```
text.delete(0,3)
```

```
text.insert(0, END)
```

然后是在 `entry2`，也就是密码输入控件中的字符，我们要用星号来显示，其实，要实现这个功能，我们只能更改 `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，当我们进行输入的时候，会默认使用星号来替代每一个字符。

`button` 按钮因为实在 `LabelFrame` 框架的外面，从属于顶级窗口，所以第一个参数我们设置为 `root`。

为了实现按钮的居中，我们采用 `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")
#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` 实现在进行点击操作以后检验我们的账户和密码是否正确

那么思路就有了，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，返回一个登陆成功状态，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，界面如图 15-12:
button=Button(root,text="登录",command=check)

```

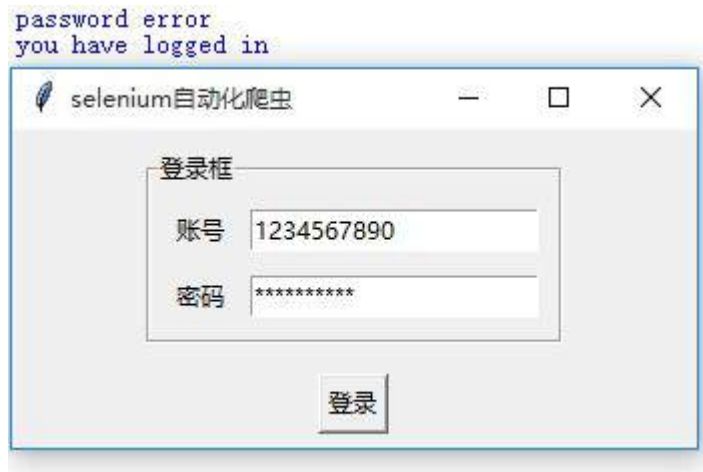


图 15-12 验证账户和密码

效果如图所示，上面的两行蓝色的输出分别是读者进行，一次错误，一次正确的输出以后输出的内容。

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的话，想要实现在输入完成账户和密码后，通过 TAB 或者 ENTER 键切换完焦点后自动验证登录，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 这个属性的话，那么前面我们的 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 Listbox

Listbox 是 Windows 窗体的一个控件，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 Listbox 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 listbox
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()
```

Listbox 控件从属于顶级窗口，第一个参数我们照旧添加的是 root，而 selectmode 我们选择的是 SINGLE 模式，Listbox 控件有两个方法比较常用，分别是 insert() 插入方法，和 delete() 删除方法。

```
listbox.insert(END,i)
```

则和前面的章节讲解的 insert 是相同的，在已经存在的项目的最后添加一个新的项目。笔者在上面的样例代码中写的：

```
listbox.delete(ACTIVE)
```

表示删除的是当前被选中的项目。

其他的内容都是已经讲解过的，最终运行的效果展示如图 15-13：

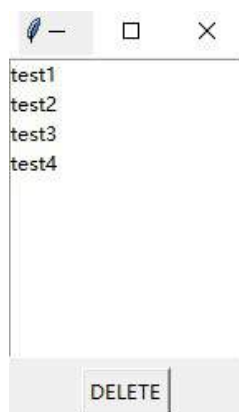


图 15-13 listbox 控件 1

Listbox 组件的 `selectmode` 属性有四个可选的内容，分别是：

- ☐ `SINGLE`，单选，最多一个项目可选
- ☐ `MULTIPLE`，多选，可同时选中多个项目
- ☐ `BROWSE`，单选，最多一个项目可选，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，但不同的是在 `BROWSE` 模式中，读者可以实现拖动改变选项，并且在 `SINGLE` 模式中，如果读者无法使用上下键来改变选项，改变的只是游标的位置，例如下图 15-14 中选项位于 `test1`，而游标位于 `test2`。

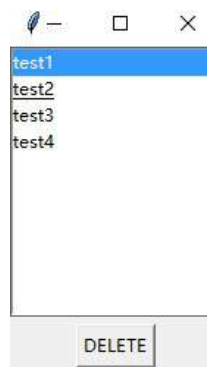


图 15-14 listbox 控件 2

`EXTENDED` 和 `MULTIPLE` 的区别在于，`MULTIPLE` 模式可以直接同时选中多个项目，但当读者使用 `EXTENDED` 模式的时候，需要按住 `Ctrl` 或是 `Shfit` 来进行辅助选择，或者直接拖动鼠标选中多个项目。

如果不写的话，`selectmode` 的默认模式是 `BROWSE`。

当我们的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看做一个控件，然后通过位置布局的不同，让其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 布局到框架之中
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，循环插入到 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()

```

最终实现的效果如下图 15-15:



图 15-15 listbox 安装滚动条

在 listbox 控件上面安装滚动条，读者首先需要设置 listbox 的控件的 yscrollcommand 属性为 scrollbar 控件的 set() 方法，然后另外配置 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)
```

这个过程跟互相调用很是相似，listbox 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，滚动轴不是在 listbox 控件中能用的，还有一些其他的控件同样可以使用。

## 15.9 Text 控件

Text 控件最初的目的是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 编程语言附带的 IDLE 编辑器主体部分就是采用 Text 控件实现

笔者在这里使用 Text 控件并在其中插入一张图片，是的，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：

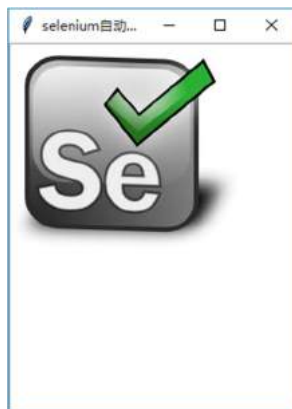


图 15-16 文本框图片

我们只需要定义窗体的宽和高，具体的设计比例，读者可以根据需要在网络上查找一个适当的宽和高。

PhotoImage()用于实例化一张图片，这张图片如果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

笔者使用 `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为止, 前后加入双引号, 比如“1.0”, 表示的是从第一行的第 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。然后笔者又将获取到的内容重新插入到 `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最后位置, 如下:

```
text.delete("2.0",END)
```

当读者想要对文字做一些特殊处理, 比如说, 字体增大, 设置背景色, 前景色的时候, 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们进行怎样的处理。在下面的代码, 笔者将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, 放置在哪里读者可以根据自己的喜好来决定, 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文本中的不可见对象。实际上, 标记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, 则无法删除它们。

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** 周围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, 如果周围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")

#设置第一行第十一个索引位置为 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，笔者首先创建了一个 `Text` 控件，然后在 `Text` 控件中插入了一串字符用于测试，将这一串字符的第一行第十一个索引位置设置为 `mark1`，然后 `insert()` 方法调用 `mark_set()` 做的标记，在这个位置插入一个内容为 `MARK` 的字符串，然后使用 `tag` 标签，将背景色设置为黑色，前景色设置为白色。实现效果如图 15-18



图 15-18 标记颜色

如果读者像删除掉这个 `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（`tkinter` 前面有一道下划线，笔者会在后面说道 `_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。在这个小节，我们使用 `canvas` 控件来实现一个简单的手写触摸板（如果读者有触摸板的话，当然，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，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的知识，我们知道椭圆有长轴和短轴，椭圆的标准公式为：

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \quad (a, b > 0)$$

在 tkinter 中，使用 Canvas 控件插入一个圆需要两个坐标点，可能会有读者有疑问，确定一个圆不是三个点吗？不然，Canvas 控件不是通过三点确定圆，而是  $|X2-X1|$  和  $|Y2-Y1|$  分别确定两条轴的长度，然后  $(\frac{X1+X2}{2}, \frac{Y1+Y2}{2})$  确定中点坐标，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

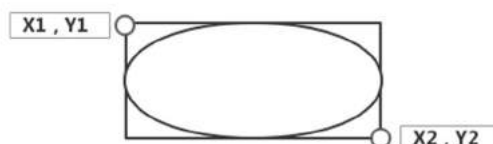


图 15-19 原理介绍

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()

#两点坐标构造圆
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()
```

笔者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，将绘出的圆用黑色填充。

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 系统上可以使用，因为 Windows 和 Mac，使用 mousewheel 表示用户是在滚轮上下滚动。

所以 B1 表示的是鼠标左键点击，当用户按下鼠标左键并拖拽的时候，事件绑定一直处于被触发的状态，所以函数不停的执行。

最终实现的画板如下图 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 的全部可选值包括：

- ☐ **Activate, Deactivate**  
当控件状态从“未激活”变为“激活”、控件的状态从“激活”变为“未激活”
- ☐ **KeyPress, KeyRelease, KeyRelease Visibility**  
用户按下键盘按键、释放键盘按键、释放键盘按键且应用程序至少部分可见
- ☐ **ButtonPress, ButtonRelease, MouseWheel**  
按下鼠标按键、释放鼠标按键、鼠标滚轮滚动
- ☐ **FocusIn, FocusOut**  
控件获得焦点、控件失去焦点
- ☐ **Map, Unmap**  
控件被映射，控件被取消映射
- ☐ **Expose**，窗口或控件部分不再被覆盖
- ☐ **Motion**，鼠标在控件内移动
- ☐ **Configure**，控件的尺寸改变
- ☐ **Enter**，鼠标指针进入控件的时候触发该事件
- ☐ **Destroy**，控件被销毁时触发该事件
- ☐ **Leave**，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 比较常用的取值是：

- ☐ **Double**，后续事件被连续两次触发
- ☐ **Triple**，后续事件被连续三次触发
- ☐ **Alt**，按下 Alt 按键
- ☐ **Any**，按下任何类型的按键
- ☐ **Control**，按下 Ctrl 按键
- ☐ **Lock**，大写字母锁定键打开
- ☐ **Shift**：按下 Shift 按键

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

**<ButtonRelease-1>**

鼠标左键释放之后

<Double-Button-1>

双击左键

<Triple-Key-H>

连续三次点击 H 键

## 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，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，

```
#导入库
from tkinter import *

#创建主窗口和标题
root=Tk()
root.title("selenium 自动化爬虫")

#二级菜单中的选项的对应命令
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:



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` 是实例化一个顶级菜单，然后在顶级菜单中又再次实例化了两个 `menu` 对象，对应的是顶级菜单中的展开二级菜单的按钮，然后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，笔者在这里简略的用一个 `action()` 跳过了命令的实现部分，在实际使用中，读者根据自己的需要添加相应的内容即可。弹出窗口如图 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单独独立出来，如果读者点击最上面的分割线，就会发现这个菜单被单独拿出来。但这个功能我们一般是不需要的，所以 `tearoff` 选项设置为 0 为宜。

除此之外，tkinter 还提供了一个 `OptionMenu`，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()

实现效果如图 15-23:
```

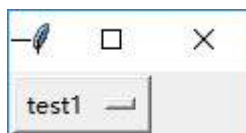


图 15-23 下拉单选框

首先我们创建了一个 `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` 针对的整形，然后使用 `set()` 方法，将 `StringVar` 的值设为 `test1`，对于下面的 `OptionMenu` 来说，我们就相当于设置了默认值，如果没有使用 `set()` 方法，则没有默认值，直到读者选中了一个选项之后，选择菜单上才会出现读者选择的选项。

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，第三四个参数是我们设置的选项

## 15.12 spinbox 控件

Tk8.4 添加了 spinbox 控件，与之相应的 Python 版本是 Python3.2,也就是说，Spinbox 只支持 Python3.2 和 Tk8.4 以上的版本。

Spinbox 控件和 Entry 大致上相同，如果读者不在其中加入 values 参数的话，几乎可以当做 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，另外还可以使用 command 参数实现用户点击调节箭头的时候自动调用对应的函数，效果如图 15-24 所示：

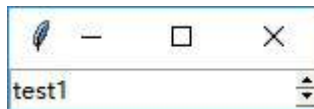


图 15-24 限制选择

当用户点击上下按钮以后，框中的内容就会从 test1 到 test4 进行转换。

## 15.13 messagebox

笔者按照代码的顺序，将这些内容弹出来对话框罗列出来，askokcancel()和 askyesno()，askquestion()在中文状态下弹出的对话框样式是相同的，因此只罗列一个。

```
#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")
```

几个弹出框口如图 15-25 所示:

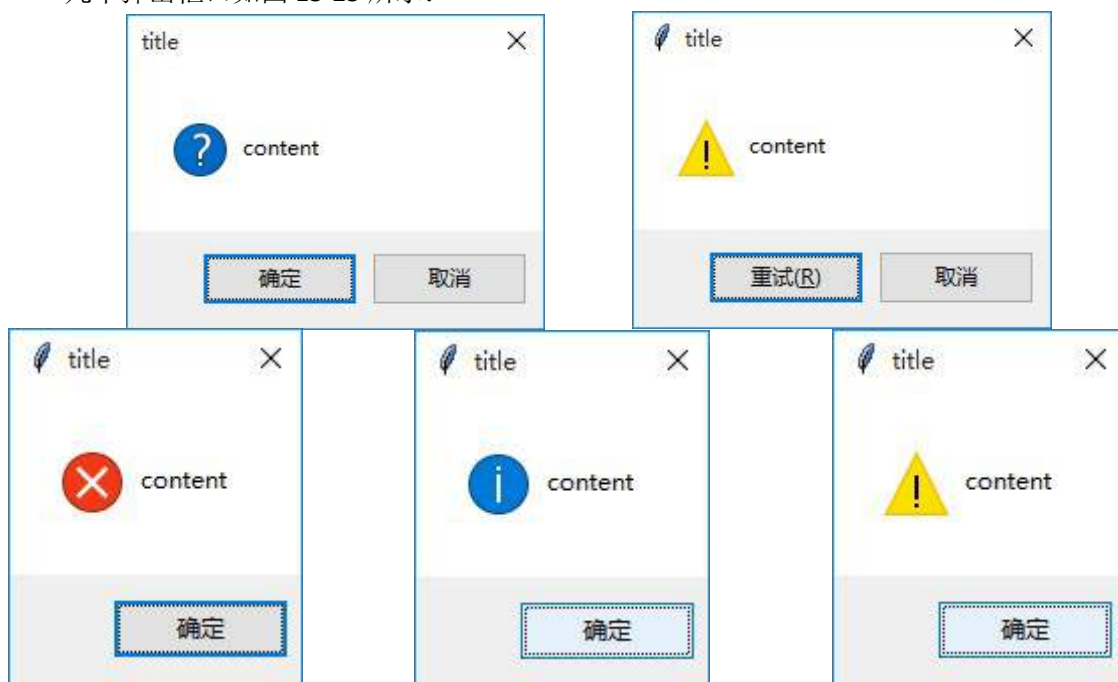


图 15.25 几种弹出窗口

此外, 这些对话框都有三个可选参数:

`icon`, 修改图标, 只能指定 `error`, `info`, `question`, `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`, 设置默认的按钮, 也就是说回车会选中的按钮, 必须是 `abort`, `retry`, `ignore`, `ok`, `cancel`, `no`, `yes` 中的一个, 不能自定义。

下面这一行代码的默认值是取消, 按下回车键后返回的值是 0:

```
askokcancel("title","content",default="cancel")
```

`parent`, 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, 只有两个命令, 前者用于选取文件的路径, 后者用于保存文件, 分别是 `askopenfilename()` 和 `asksaveasfilename()`。

笔者简单写了一个代码, 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

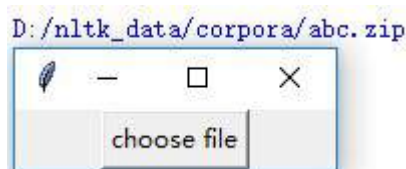




图 15-26 选择文件路径

```
asksaveasfilename()
```

用来保存文件，会弹出一个另存为的窗口：

两个函数都有一个 `filetype` 参数，可以用来设置保存类型（如上图所示）的可选项，该参数是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杨，但 `message` 控件使用起来更加舒服和得心应手，因为其能够自动换行，并调整文本的尺寸使其适应给定的尺寸，我们来对比一下 `Label` 和 `Message` 在输出同一内容时候的不同表现

```
#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 ago")
label.pack()

message=Message(text="in long long long long long ago")
message.pack()

#窗体循环
mainloop()
```

详细区别如图 15-28 所示

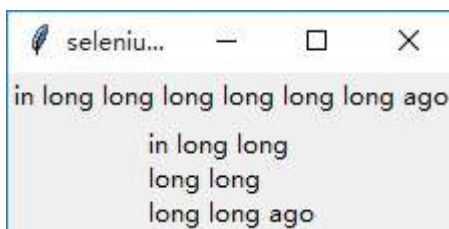


图 15-28 Label 与 Message 控件

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`，什么时候使用 `Message`，什么时候使用 `Text`？这三种控件一般来说，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，那么最好还是使用 `Text` 控件搭配 `tag` 标签使用。

## 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` 有三个几何布局管理器：`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` 布局有以下这些参数可用：

表 15-1 pack 布局

参数	说明	选项
fill	设置控件向水平或垂直方向填充	X、Y、BOTH 和 NONE
expand	是否展开，默认不展开。当值为 YES 时，side 选项无效，控件显示在父容器中心位置；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

`grid()`布局，笔者十分推荐读者使用这种网格布局。考虑到大多数程序都是矩形，很容易将它们划分为几个行和列组成的网格，然后根据行和列号将组件放置到网格中。

表 15-2 grid 布局方法

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

另外当读者使用网格布局时，只需指定两个参数行和列。

`Place` 布局是最灵活的布局，使用坐标来放置控件位置。但是，笔者不建议这样读者使用 `place` 布局，因为如果使用这种布局 GUI 在不同的分辨率下，界面通常是不同的。

表 15-3 place 布局参数

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 0
relx、rely	组件相对于父容器的 x、y 坐标	0~1 之间浮点数
relwidth、relheight	组件相对于父容器的宽度、高度	0~1 之间浮点数
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; 如果为 OUTSIDE, 组件的外部大小是相对的, 包括边界	默认值为 INSIDE, 可选 INSIDE,OUTSIDE

## 15.17 tkinter 支持的模块

tkinter 的功能强大不仅仅在于它可以绘制出窗口, 更在于它包含的小控件的丰富性, 以下这些内容都可以在 tkinter 中使用:

- ❑ tkinter.scrolledtext, 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, 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, 实现简单的绘图功能

## 15.18 \_tkinter 接口

tkinter 提供了 Button, Canvas, Label, Text, Frame 等一系列组件, 大多数时候, 这些组件足够我们编写一个较为完善的 GUI 程序了, 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。这个模块包含 Tk 的低级接口, 开发者一般是无法直接使用它的, 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, 读者如果想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` 的底层实现并不是完全通过 `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

## 第 16 章 实战案例 part5 知乎

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回顾和整合，并且讨论 `Selenium` 项目的进阶用法——将 `Selenium` 与传统的解析页面方式的爬虫进行结合，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,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,这在知识经济的历史上写下的浓墨重彩的一笔。

而在不久前(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。

首先随意点开一个问题观察一下,我们发现,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。而没有被加载到的,在源代码中查看都显示为<noscript>标签。

而每一个答案都藏在<div class="List-item"></div>中,目录结构大致如下图 16-1:

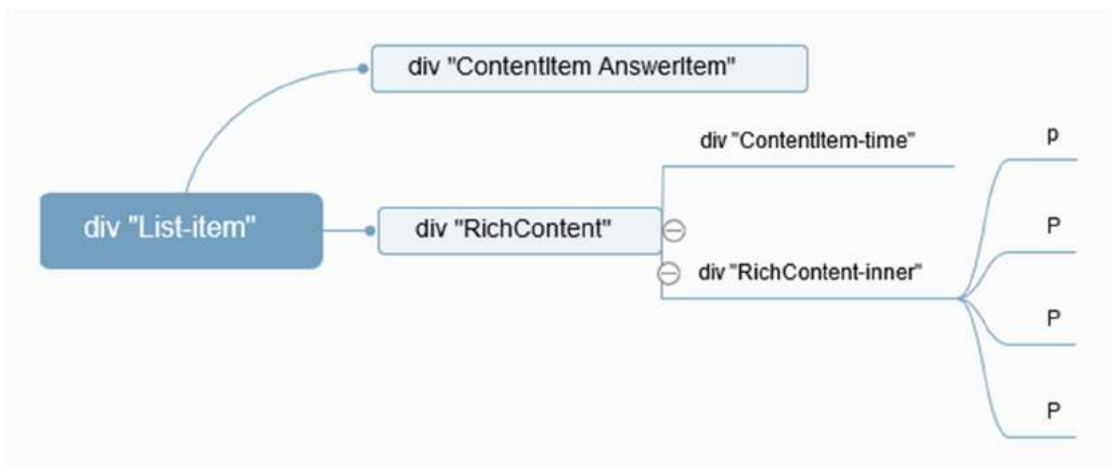


图 16-1 目录结构

跟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);
        })();
    """)
```

这段代码和我们前面使用的代码略微类似，只不过略微高级一些，笔者对读者的建议是，在本书的学习过程中，只需要了解，而至于更深一步的探究其中的原理，读者可以自主学习 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，都只是 `div` 标签中的数字不同。如果读者愿意，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，并且所有内容都是在同一个页面中，并不像我们前面在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），比如说，设置一个特别大的数，然后使用 `try, except` 语句，即便后面没有找到，也不会致使程序退出。

或者提前自己手动下载一下页面，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，不能完美的应用到知乎的每一个问题，第一种方法对应的反例是可能我们爬取的是一个弱关注的问题，只有聊聊几个回答，第二种方法则更不适合爬取任一问题，我们总不可能对于每一个要爬取的页面都提前加载看一下最大回答个数把。

因此，我们考虑使用 `BeautifulSoup4` 这一个用于解析页面的库来实现功能。

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，笔者将会向读者介绍本章节用到的几个方法，没有相应储备也没有关系，新知识的规模和复杂度都是比较小的。

`BeautifulSoup4` 这一库的主要内容是一个类，在使用类中的方法之前，我们需要首先对类进行实例化

```
#页面源码解析
html = driver.page_source
soup = BeautifulSoup(html, 'lxml')
find_all(name, attrs, recursive, string, **kwargs), 这一方法用于查找解析后的内容所有符合条件的标签，比如说，我们的样例内容为：
```

```
<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的：

```
>>> 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>]

```

在第一个语句中，我们查找所有包含 `id` 属性为 `link2` 的标签，查找结果只有一个符合，而在第二个查找当中，我们删掉了对于属性的限制，三个 `a` 标签都符合，第三个查找当中，我们限制属性中必须含有 `title`，因此只有 `p` 标签符合（`title` 标签中的 `title` 是标签名，而不是属性），第四个查找，我们查找 `title` 标签。

特别的，下面两种方法是等价的：

```

#限制只查找第一个
>>> 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，另一种查找 `find(name, attrs, recursive, string, **kwargs)` 和 `findall()` 方法用法几乎完全相同，区别在于，`find` 方法只查找第一个。

## 16.2 文字部分

我们考虑首先实现关于文字部分的获取。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，对于这层标签，类似于 `find_element_css_selector`，因为含有多个 `class` 属性，用 `bs4` 的 `find()`，`findAll()` 方法，读者是无论如何也无法提取到这个标签的。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 `BeautifulSoup4` 官网中明确说明了 `BeautifulSoup4` 其实是不支持直接查找含有多个 `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，使用方法如下：

```

soup.select([css_selector])

```

另外一种可行的方式是使用笔者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')

```

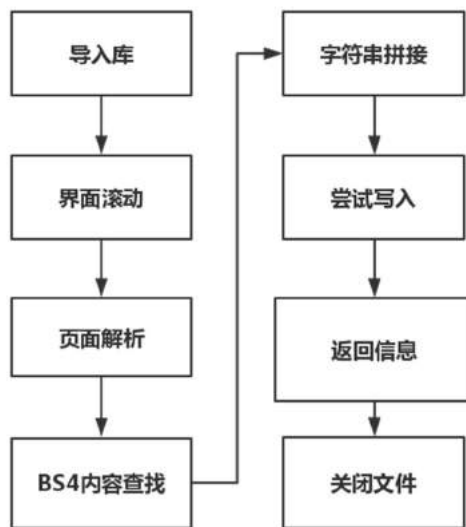


图 16-2 知乎文字部分思维导图

代码如下所示：

```

#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，实现内容查找，存储
def func():

    for story in stories :
        nameLabel = story.find('meta',itemprop="name")
  
```



```

name=nameLabel["content"]
storyText=story.find('div',class_="RichContent-inner")

#字符串拼接写入
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,'xml')
    storys=soup.find_all('div',class_="List-item")

    #文件存储
    f=open('tmp.txt','w+')
    func()
    f.close()
    print('That is all')

```

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、掌握的数据目前无法实在与服务器相比。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，而手机储存还不如 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，这个就相当于手机做瘦客户端。

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，这个语句并不是必须的，但其实从实用性和普适性的角度上来说，这个语句保证了我们的程序在面对单个获取失败的情况的时候，不至于打断整个代码的运行。

另外对于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，例如：JS 脚本。又或者我们的页面虽然已经下拉到这个位置，但该位置的内容还未加载完全。

为了安全起见，在面对回答数更加多的问题时，读者应该适当将 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 获取页面源码，然后使用 BeautifulSoup4 库中类进行对象实例化。

```
html=driver.page_source
soup=BeautifulSoup(html,'xml')
```

此外，如果读者想要进一步改进，用于存储大规模数据，还可以考虑结合我们前面介绍的存储为 CSV 文件的方式。

```
import csv
```

或者是采用数据库的方式：

```
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 文件存储。

## 16.3 图片部分

和上面爬取文字部分的所遇到的困难相似，最大的问题在于我们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，将下面代码中的 `time.sleep()` 参数设置的略微大一些。

```
scroll(driver)
time.sleep(5)
titles = soup.find("title").text.replace("\n", "").replace('?', "").split()
```

不同于刚才获取的文字部分可以全部保存在一个 txt 文件中，图片我们需要创建一个全新的文件用于专门保存相应的图片：

```
#获取问题名称
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)
```

我们首先使用 `os.getcwd()` 方法获取当前路径，并且通过拼接字符串的方式拼接处我们以我们爬取的问题命名的文件夹，接着判断我们想要的文件夹是否存在，如果没有，则创建该文件夹。

可能读者对于笔者为何要在 `find()` 方法查找的内容中获取第一个有些疑惑，`find()` 确实是查找到的第一个，但在这里，我们获取到的第一个标签中的 `text` 确实一个列表，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']
```

这便是我们为何获取的是 `titles[0]`，而不是输出 `title` 的原因。另外对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开始，仅仅只有寥寥数个答案，并且在页面的底部，存在如图 16-3 所示的跳转按钮。

[查看全部 47 个回答](#)

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,在这里页面我们只需要一直下拉确保到底即可。

而我们想要的是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)
```

即便虽然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仍需要进行新一次的下拉操作,所以笔者采用 `while true` 循环。

因此,我们可以通过下面的代码实现从问题的推荐回答到问题页面的跳转:

```
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: selenium2 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()
            # more.click() # 如果我们在页面中找到了更多,那么就点击更多,然后等两秒
        except NoSuchElementException as e:
            break
```

总体思路如图 16-4 所示:

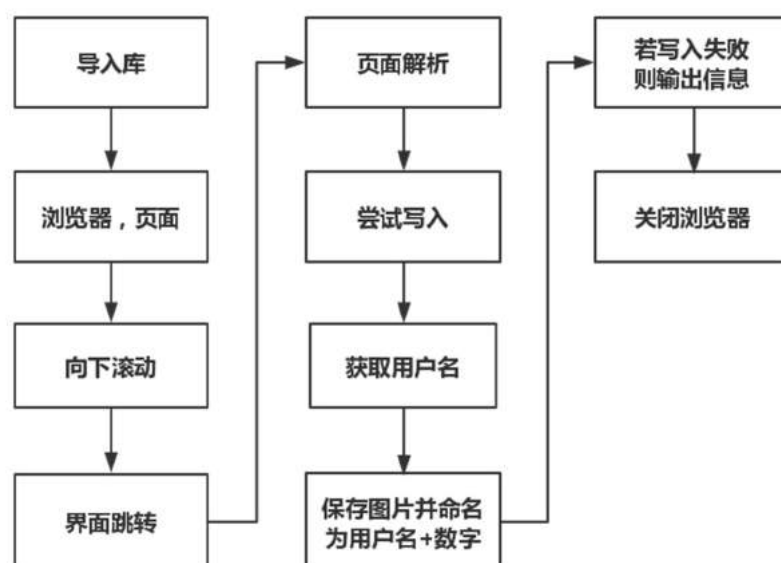


图 16-4 图片部分思维导图

最终实现代码:

```

#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：selenium2 是不支持 类名之中 有空格的
                try:

```

```

self.scroll(driver)
#设置较长的等待是尽量保证页面加载完全
#但在这个样例中 30 秒仍然不太够
time.sleep(30)

#这里使用\反斜杠，将一段代码分为两行来写
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存为 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 # 一个用户下有多个照片的
    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:

```

```

        #二进制方式创建
        with open(picName, 'wb') as jpg:
            jpg.write(imageData)
    except IOError as e:
        #讨论写入失败的情况输出
        print(userName + "的一张图片写入错误")
    except MissingSchema as e:
        #讨论图片获取失败
        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")

```

如果一切无误的话，读者在短时间的等待之后应该可以看到在文件夹中大量的照片。

下面我们来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的多，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到图片的路径，然后根据路径下载图片，这很容易因为网络原因或者服务器的传输的缘故而导致下载中间某一张图片的时候失败。

在进行文件的获取和写入的时候，我们使用到了下面的代码：

```

requests.get(imageSrc, stream=True).content
request.get()这一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我们的异常返回，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 这一返回则是我们用来检查当前页面是否仍然为推荐答案页面，不然的话会一直进行循环。

```

from selenium.common.exceptions import NoSuchElementException

```

## 16.4 建议

笔者给出几条 Selenium 使用过程中的建议，希望能够帮助读者进一步改进自己的程序，使其朝着更加良好的方向、积极的方面发展。

## 16.4.1 从干净状态开始

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，为每个测试启动一个新的虚拟机是再好不过了。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** 在使用的时候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，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，您可能会遇到意想不到的行为。

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。这有助于确保测试隔离并简化并行化。假设有一个内容管理系统，您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您的网站上，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。

但读者可以创建可以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 **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。

不少人在涉及到数据处理领域以及自动化方面的都为环境问题头疼不已，那些提示总是告诉你在安装这个工具需要安全其他一堆不明所以的工具。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，解决好了各种依赖关系。这也是为什么它更受欢迎的原因。**Anaconda** 另外还包含了 160 多个科学包及其依赖项。

以及包括但不限于以下功能：

- ❑ 数据整合、探测、挖掘、预测
- ❑ 对结构化/非结构化数据类型转换
- ❑ **NumPy+SciPy** 的底层组合
- ❑ **Win/MacOS/Linux** 兼容

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

编写报告是一个自动化人员的基本功,而一个良好的报告标准将会让每个读者终生受益,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三类:接口问题、功能问题和崩溃问题。界面问题应该上传问题截图,用红框标注。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);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,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,如果有需要可以提供缺陷定位线索或错误日志输出,减少开发人员重新定位的时间。

## 16.4.5 IP 代理

当读者碰到一些特殊情况,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
proxy.add_to_capabilities(dc)
dr=webdriver.PhantomJS(desired_capabilities=dc)
```

我们首先创建一个配置对象的副本,接着把已经提前声明的字典格式的代理加入进去,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无比的便利,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成,包括页面上的图片资源,JS 和 CSS 文件加载,更头痛的是,如果有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到浏览器加载的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?

- ❑ 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使用 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,有利于读者更好的改善自己的程序,避免陷入一些不必要的错误。

### 16.5.1 Captchas

Captchas (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)的设计初衷便是为了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!目前有两个主要的策略来绕过验证码检查:

- ❑ 在测试环境中禁用 Captchas。
- ❑ 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 Captchas 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。不是因为它无能，而是因为它对工作没有优化，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。

- ❑ 浏览器启动速度
- ❑ HTTP 服务器的速度、
- ❑ 托管 JavaScript 或 CSS 的第三方服务器的响应
- ❑ WebDriver 实现本身的检测缺陷。

通常在这些点上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。

## 16.5.3 六度分割理论与实际使用

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，但是这会影响力性能测试结果，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践，分析单个资源(如 CSS 或 JavaScript)的性能，以便了解需要改进什么。已有一些专门针对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。

在数学一个数学领域的猜想，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六个，也就是说，最多通过五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 需要时间启动。

而与使用 WebDriver 不同的是，通过执行 curl 命令或使用像 BeautifulSoup 结合 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，可以节省大量时间，但这可能导致读者的爬虫被服务器所限制。

具体使用哪种方式亦或是结合使用需要读者根据实际情况斟酌。

## 16.5.4 HTTP 响应代码

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，检查状态代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 状态代码，例如 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点的页面标题或内容(例如标签)。

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 代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状态并改进调试输出，如图 16-5 所示。

**HTTP 错误 404.0 - Not Found**

您要找的资源已被删除、已更名或暂时不可用。

**最可能的原因:**

- 指定的目录或文件在 Web 服务器上不存在。
- URL 拼写错误。
- 某个自定义筛选器或模块(如 URLScan)限制了对该文件的访问。

**可尝试的操作:**

- 在 Web 服务器上创建内容。
- 检查浏览器 URL。
- 创建跟踪规则以跟踪此 HTTP 状态代码的失败请求，并查看是哪个模块在调用 SetStatus。有关为失败请求创建跟踪规则的详细信息，请单击[此处](#)。

图 16-5 404 页面 1

但更常见的情况是，错误页面可能是一张图片或是什么都没有，这通常需要我们写 try, except 语句根据跳转到的页面的内容判断返回的状态码，比如说，我们预先记录错误错误页面特定信息。通过判断是否存在这些文字来知道是否被跳转到了一个不存在的页面，了解当前对当前页面采集是否成功，如图 16-6 所示。



图 16-6 404 页面 2

另一种捕获 HTTP 状态代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的方式操作发送到 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响应代码。另外，并不是每个浏览器都能让 WebDriver 使用响应代码，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

笔者在这里对于 Selenium 项目的大部分常见问题进行了汇总，希望能够对读者接下来

使用 Selenium 的过程有帮助。

## 16.6.1 元素定位失败

问题 1:

```
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 会随着每次登陆发生变化。理论上，通过 id 是无法精确定位到该元素的，所以推荐读者通过 Xpath 相对路径的方法进行查找

```
driver.findElement_by_id("temp_82_82")
```

此外，可能会发生读者无法定位到位于 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需要首先理解这些框架的实质，网页嵌入一个 Frame 框架，实际上是嵌入了另一个网页，所以我们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')

#需先跳转到 iframe 框架
tmp=driver.find_element_by_name('email')
tmp.clear()

#如果 iframe 没有 name 或 id 的话，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)
```

最典型的页面像网易邮箱，如果不切换到 Frame 中的话，用各种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()
```

在另一方面，如果读者设计的程序中间间隔时间太短，页面还未完全加载完成，而程序已经判定可以进行下一步，这就会造成点击速度过快，而页面尚未完全加载出来。

```
#设置隐性等待
dr = webdriver.Chrome()
dr.implicitly_wait(30)
```

```
#强制暂停
```

```
time.sleep()
```

另外几种小错误包含：

- ☐ 元素属性键入错误，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，页面中可能有多个元素共用。

- ❑ 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。

问题 2:

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：

```
<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>
```

限制我们进行更改，确实相当棘手。可以考虑使用下面的 JS 代码将元素的只读属性删除，再做操作：

```
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)
```

问题 3:

```
selenium.common.exceptions.StaleElementReferenceException:
Message: stale element reference: element is not attached to the page document
```

页面没有连接成功，一般出现该报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。如果读者在列表中查找数据并查找新创建的数据，那么就会报告错误。

常见的解决方法是添加完成保存后，先增加一步刷新页面的操作：

```
driver.refresh()
```

然后再去定位新创建的数据。

问题四:

如果想要选中下拉菜单中的选项，需要两个步骤，首先先定位到下拉菜单，其次对其中的选项进行定位。在下面的代码中，我们首先定位了菜单，再定位其中的选项，效果如图 16-7 所示。

如果直接定位选项的话，因为其处于不可见状态，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。

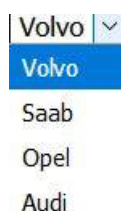


图 16-7 下拉框

```
#定位下拉菜单
```

```
drop_down = driver.find_element_by_css_selector("body > select:nth-child(1) ")
drop_down.click()
```

```
# 再对下拉菜单中的选项进行选择
```

```
select=find_element_css_selector ("body > select:nth-child(1) > option:nth-child(2) ")
select.click()
```

## 16.6.2 Webdriver 调用失败

Exception in thread "main" org.openqa.selenium.UnsupportedCommandException: Bad request  
出现该报错主要是因为 hosts 文件出了问题，需要在 hosts 文件中加入：

```
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，如果读者使用一些采用修改 hosts 文件的方式来加快联网速度的话，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。

Exception in thread "main" org.openqa.selenium.WebDriverException:  
Cannot find firefox binary in PATH. Make sure firefox is installed.

这个错误说明 Firefox 浏览器并未被安装在默认位置（Chrome 浏览器强制指定安装位置，一般不会出现该种错误），这时候有两种解决方法。

第一种，我们将 Firefox 的安装路径加入环境变量中

第二种，在每次启动 Firefox 浏览器的时候指定启动路径。

ConnectionResetError: [WinError 10054] 远程主机强迫关闭了一个现有的连接。

出现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。

# 第 17 章 实战 part6 微博

在本例子中，笔者将采用微博作为案例，爬取微博的部分信息

## 17.1 微博分析

在我们一直进行的案例展示中，我们都是采用直接从官网入手的方法，但在对于微博的爬取中，考虑到手机版微博的内容相对于电脑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微博页面进行内容的获取。

<https://passport.weibo.cn/signin/login>

如要在电脑上查看手机端的样式，读者可以通过按下 F12，打开审查元素，点击响应式设计模式，即可从拉长变形的页面变换（如图 17-1 所示）到手机端页面（如图 17-2 所示）



图 17-1 拉长变形的手机端微博网页



图 17-2 调整过后手机端微博网页

微博的用户机制比较严格，如果想要查看 id 为 xxx 的用户，即使获取到了链接仍然进不去，网页会将我们跳转到登录页面，要求我们进行登陆后才能查看其他人的微博。

所以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”，登录按钮的 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)
```

而最后一个暂停 `time.sleep(3)` 是必不可少的，不停会有问题。如果缺失很容易发生第一个界面还没有完全加载，执行界面跳转之后登陆状态就消失的情况。

笔者在每两步之间都间隔了一秒，毕竟在一瞬间输入完成所有内容太不现实了。如果想要进一步的模拟人类的操作，可以使用：

```
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 查看一下，但我们不需要记录 cookie，因为 Selenium 会把 cookie 自动带过去：

我们查看一下输出的 cookie，代码：

```
#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)
```

```
#调整，输出
cookie=';'.join(cookie_list)
print(cookie)
```

输出结果：

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大致就分为用户信息和微博消息这两个部分。

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。

由于我们是通过用户 ID 进行跳转，所以这部分可以当做已知信息，直接输出，接下来，通过 Xpath 获取到保存在页面上的用户的昵称，昵称部分网页源码如下：

```
<html>
<head></head>
<body>
<div class="ut">
<span class="ctt">柳岩

<a href="http://vip.weibo.cn/?F=W_tq_zsbs_01">
</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&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, 因此我们可以首先采用 CSS 选择器 “//div[@class='ut']” 定位到该位置, 其次通过.text()方法获取文字, 再次使用.split(' ')方法, 进行空格分割。

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, 关注数, 三个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>

```

类似的, 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, 如果感觉正则表达式存在一定的困难, 也可以通过.text 和.split 方法组合分割出需要的信息。

函数一:

```

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)

    # 3.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，内容部分的获取，我们研究页面的微博消息发现，内容都是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，笔者发现，微博最下方设置也采用类似的方式来显示：

```
<div class="c">
  设置:
  <a href="https://weibo.cn/account/customize/skin?tf=7_005&st=18fa71">皮肤</a>.
  <a href="https://weibo.cn/account/customize/pic?tf=7_006&st=18fa71">图片</a>.
  <a href="https://weibo.cn/account/customize/pagesize?tf=7_007&st=18fa71">条数</a>.
  <a href="https://weibo.cn/account/privacy/?tf=7_008&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一个判断，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，则当做微博消息爬取下来。

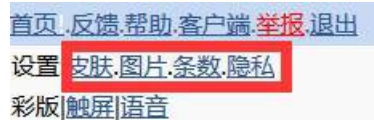


图 17-3 文字判断

```
if "设置:皮肤.图片.条数.隐私" not in content:
```

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，更是为了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，增加程序的稳定性。

函数二：

```
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
    #当前页第几条微博
    curr_msg=1
    #全部微博中第几条
```

```

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 则是因为微博获取到的大部分是汉字,如果读者愿意的话,采用 UTF-8 也可以。

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:

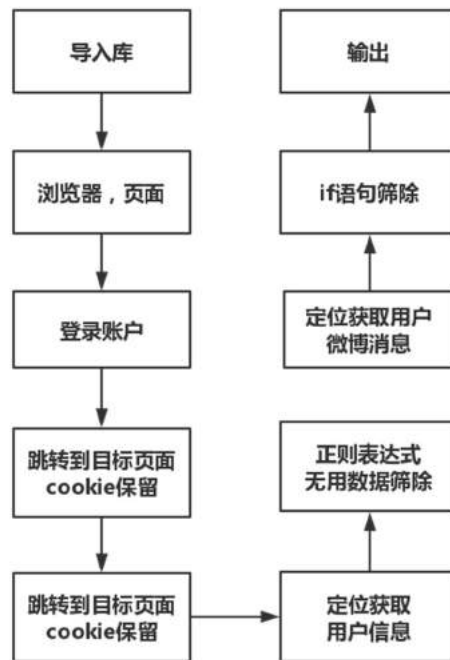


图 17-4 实现思路

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)

    #输入账户密码，点击登录
    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，而并不能够存储，由于微博的用户发送的微博条数可能会达到 4 位数乃至 5 位数，我们考虑使用为每个用户单独创建一个 txt 文件存储。

```
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：

```
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!")
```

此外还需要制定一些避免被反爬虫的措施，例如让程序在每两页跳转之间 `time.sleep(25)`，设置一个略微快一些，但还在正常人类阅读速度范围内的页面加载频率。这样可以减小读者的爬虫被封禁(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，所以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 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
- ☐ Cookie 文件的处理

最后，如果有条件，可以多申请几个公网 IP，以及尝试使用 ADSL，申请多条线路，编写程序进行断线重拨，远程硬件重置，动态 IP 追踪。采用分布式爬虫，一台中央机器控制多台电脑进行页面的抓取，至于页面的去重可以考虑布隆过滤器。

被封禁 IP 可能造成如图 17-5 所示的后果

如果可能的话，可以考虑跟网站的管理员协商，出钱购买一部分数据，当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的时候，适当出一些小钱是完全可以考虑的，不然可能服务器工作的电费就让你得不偿失。

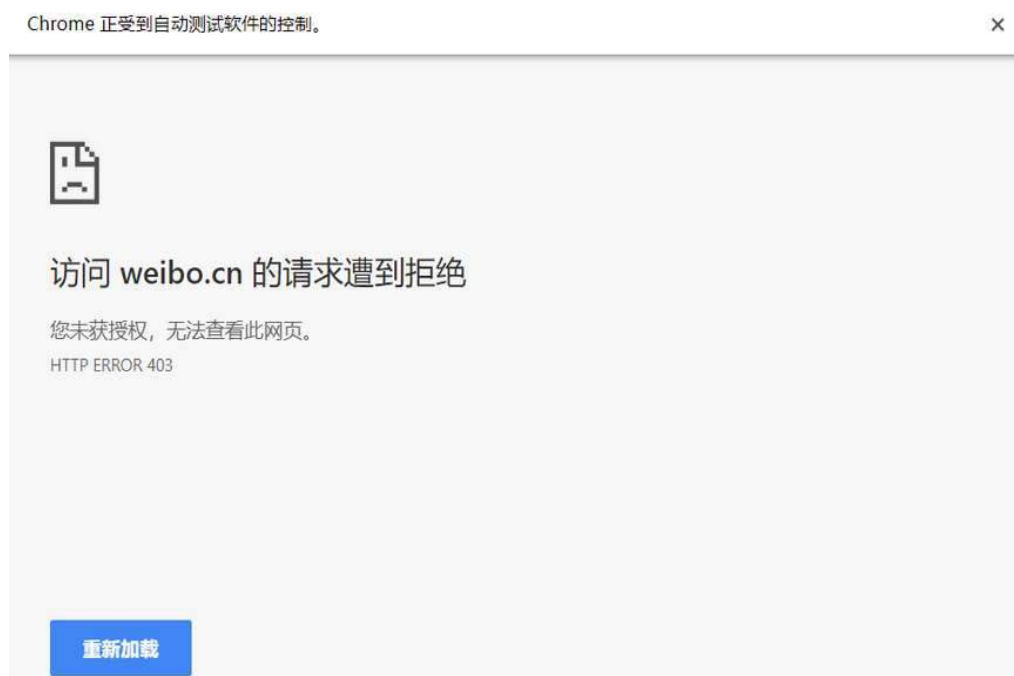


图 17-5 拒绝访问

此外，还有一些内容使我们需要改进的，我们现在的输出文件仍然有许多问题，可以考虑使用下面的方式改进代码，删除掉输出文本中无用的内容：

```
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”等等信息，在每条判断语句中都加入“-1 !=”是在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，但我们的前进之路却不会停止，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”。不要轻言自己怎样怎样，因为我们会不停的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！

让我们一起见证！

## 附录 1 怎样阅读源码

经常思考如何读一份陌生的代码，最近有一些想法。天底下的源码都是你的老师，但是这些老师有好有坏，千万别东一锤子西一锤子的去瞎看。

读代码通常要能回答两个问题，

- ☐ 要解决什么问题？
- ☐ 如何实现的？



大到整个项目，小到一个模块，函数，看的时候，都要抱着这两个问题去看。看完了，这两个问题能答上来，才是有效。代码不仅是读，还要理和试。

具体可以这样做：

首先理解需求，这玩意到底干嘛的，如果可能的话做一个背景调查，看官网介绍，维基百科，了解主要功能，被应用于哪些项目，同类框架还有哪些？其次写 demo，调用 api，理解接口的设计，使用框架，至少 follow "Get Started" 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。最后理解作者设计的哲学，性能，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。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。

想了解他的代码生成机制，自己先去思考，如果是你做，该怎么做，再去看源码。比如说“打日志的时候是异步的吗？如果我需要把日志打到另外一台机器上，我该怎么做？”带着这些问题去思考解决方案，没有目标的去看所谓的源码，没有任何的意义。最适合看源码的就是，你对完成一个功能感兴趣，所以你想要了解他。

另外读代码的时候也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`，那凭直觉就应该猜到这玩意应该是个自定义结构体，用来当作 `message` 传递给上层的，没准与之对应的还有一个 `down_call_info`。

阅读代码的时候，下列方法有助于你更好的理解他：

- ☐ 看 git 历史
- ☐ 拖进 source tree，查看提交代码的历史更改，从第一个 commit 读起来，而不是最后一个。
- ☐ 一定要跑起来
- ☐ 想办法运行。不运行起来的代码，并不能看懂。
- ☐ 借助 IDE 的代码结构分析工具
- ☐ 在 IDE 里读，IDE 里可以方便跳转，查看定义，比起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。
- ☐ 找合适的代码
- ☐ 找到合适的代码。如果直接看 nginx 应该看不懂，如果看一个几十行的 http server，却是可以看懂的，是可以让人理解的。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的添加。
- ☐ 代码指纹，联系作者
- ☐ 代码指纹，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，留下思路，留下的论文。联系方式，其实解决很多疑惑的办法是直接联系作者。
- ☐ 调试
- ☐ 对我自己而言，我一般是先对源代码进行搜索，找到函数入口，先把入口函数读懂，然后一层一层向下递归展开单步调试，调试打印变量。写 python，写 Objective-C 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，然后你去跟那些函数调用就好了，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 Event Loop，对于事件队列进行分发，交给后续的 Event Handler 处理，然后就可以从事件队列的初始化、事件进入队列、Handler 如何注册等等跟下去。尽可能编译调试，能调试的代码，几乎没有看不懂的。
- ☐ 画 UML
- ☐ UML 图不只是没用的图，他们是告诉你代码如何分析的方面，画 UML 类图。动态

图，静态图，类图，部署图，用例图。网络上存在相关的工具，工具可以分析代码生成流程图和类调用关系图。

- ❑ 仿写
- ❑ 这个耗费时间比较长。但是照着写一遍，一步一步是最有益处的。
- ❑ 买书
- ❑ 知识比钱值钱，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，网上一般可以搜到“xxx 源码结构分析”之类的书或者文章，跟着那个看就可以了，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。最笨的但也最有效果的法子，抄一个。新手想要通过阅读代码来提升的话，比较建议的是参与 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 Wiki 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到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”要坚持的久。

开始之前，我们希望大家对版本号管理有一些基本的认识。在我阅读的前端库、Python 后台库的过程中，我们都是以造轮子为目的展开的。所以在最开始的时候，我需要一个可以工作，并且拥有我想要的功能的版本，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。而在我们理解了基本的核心功能后，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，我们有几个简单结论：

- ❑ 阅读最早的有核心代码的版本往往有助于你更快的了解一段源码
- ❑ 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，往后较大更改的 Release 也需要了解一下
- ❑ 这里就要扯到《GNU 风格的版本号管理策略》：
- ❑ 项目初版本时，版本号可以为 0.1 或 0.1.0，也可以为 1.0 或 1.0.0，如果你为人很低调，我想你会选择那个主版本号为 0 的方式。
- ❑ 当项目在进行了局部修改或 bug 修正时，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，因而可以被忽略掉。
- ❑ 当项目在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时，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 Linux 领导下的开发小组开发出的系统内核的版本号。第一数字叫主版本号，第二个叫次版本号，第三个叫修订版本号。一般说来次版本号还有特定的意义，以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有一些新的东西加入，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，质量过关的开源项目，源码存放的路径、源码文件命名、源码中变量的命名都会符合特定的规约，根据规约去梳理代码结构。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，比如 `utils` 一般用来存放一些 `utility` 之类的东西，比如什么把某个 `id` 映射到 `MAC` 地址啊，自己实现的某种 `Hash` 算法的小函数啊之类的。

不管你是横向分层，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，有利于你更好的读懂一段代码，下面我们以 `python` 的软件库结构，示例一般目录的作用：

表附录 1-1 软件库结构

目录名称	用途
Bin	存放项目的一些可执行文件，当然你可以起名 <code>script/</code> 之类的也行。
Doc	存放说明文档。
DLLs	存放程序运行的动态运行库等文件。
Lib	<code>python</code> 的标准库。
Lib\site-packages	存放第三方库。
Libs	<code>python</code> 的内置库、语法。
include	编译器的 C 语言头文件源码。
Tools	<code>python</code> 的工具，存放些示例代码。
tcl	这文件夹里包含了 <code>python</code> 默认内置的 GUI 工具 <code>tkinter</code> 。
LICENSE.txt	对你使用的授权许可。
NEWS.txt	更新日志，记录版本更改。

# 附录 2 GIT 分布式计算

## 1 GIT 的背景介绍

作为一个伟大的开端，Git 的诞生有着不平凡背景，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，来自全世界各地的开发者协作共同完成，就需要有一个分布式计算平台，可以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，DVCS 作为一个早期的解决方案，还是相当完备的，但好景不长，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，一个月之后，Linux 操作系统源码的版本管理系统已经转为 `Git`。

## 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，他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相较于其他版本控制系统区别在于哪里呢？

## ❑ 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 做出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，让传输过程中丢失或损坏文件几乎不可能发生

## ❑ 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个快照。纵览一遍所有文件的校验和并和已有的做出对照，若不相等，那么会并保存一个快照并建立指向快照的索引，而如果校验和相等，那么会建立一个指向上次保存的文件的快照的索引，Git 关心的是文件的数据的整体是否变化。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，更要命的是，这种方式对网络传输速度提出了极高的要求，例如有一个 300M 的文件，10 个人同时需要修改，你的网络传输速度为 5M/s，那么你可能要等到 10 分钟以后才能开始干活。而 Git 分布式版本管理系统没有主库的概念，每个克隆后的项目都是一个仓库，不联网依然可以进行查看历史，比较变更，提交修改，等到有网络时一次性传上去。

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），但是如果你使用的是图片，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 1M 增加到了 2M，但具体修改了什么，版本管理系统无法获知，另外 Microsoft 出版的 Word 也是二进制格式，这意味着 git 同样无法获取具体内容变化，所以在建议读者以纯文本方式编写文件。说到文本编写，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。

接着，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：**在克隆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：**在你对工作区做出了更改后，然后使用 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。但请你不要误会，本地仓库和远程仓库是同级的存在，比如说它们通过克隆、推送互相交换内容。区别只是在于文件内容略有差异，比如说你在从远程仓库克隆下载下来项目，然后你在增加了一个 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：**我们在多人协作开发过程中，将主文件存到某个人的电脑上以这个人的进度为基准是不现实的，比如说，这个人项目文件损坏了、电脑坏了、停电了、网络传输速度不够等等因素都会影响我们的项目开发进程，这时候我们需要一个在云端的远程仓库来存储项目。我们从远程仓库克隆下来文件，在本地开发完成后，在推送到远程仓库（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？add 可以多次添加，但 commit 一次就会推送所有已经 add 到暂存区的修改。

下面我们使用一个尽可能多的使用 **git** 的命令的例子来展示 **git** 命令的用法:

表附录 2-1 Git 常用命令列表

\$ 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 origin [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^倒退一个版本，^^就是两个，如果版本多的话可以用 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
当你想不起来 git 的某一个命令具体用法时，你可以加一个-h 参数，例如 git config -h	

### 3 Git 克隆、修改、推送

我们首先建立一个新的文件夹，我假设你使用的是 windows 系统，你可以在任意一个文件夹目录包括桌面，右键，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（当然，你如果不习惯的话，也可以直接在文件夹中右键，新建文件夹）

```
$ mkdir test_dir  
$ cd test_dir/
```

然后我们可以使用这条命令来克隆一个仓库，笔者在这里选取了一个自己新建的仓库，建议读者也自己建立一个 github 的自己远程仓库，然后替换掉仓库的地址（[https://github.com/xuyichenmo/test\\_repo](https://github.com/xuyichenmo/test_repo)），否则你虽然克隆后可以在本地修改文件，但无法推送上去，因为读者的 SSH Key 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，网站 [Github](#) 在国内因为伟大的中国国家防火墙 GFW，Great Firewall of China 的因为，在部分地区速度较快，部分地区速度则很慢，在你将 Git 的基本操作使用熟练后，你可以使用国内的一些代码托管平台，笔者就不做推荐了。

在你安装 Git 的时候默认是安装了 Vim（这是一个常见于 linux 的编辑器，在 Git Bash 中即展开编辑，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，可以让你脱离频繁切换键鼠的烦恼）作为默认编辑器的，所以你可以使用 Vim 来编辑文件，命令 `vi` 即使用 Vim 来编辑，如果文件不存在，那么创建该文件。使用 Vim 的时候，你需要按键盘上的 `i` 键进入编辑模式，当你编辑完了，按 `Esc` 键退出编辑模式，然后同时按住 `Shift` 和 `;` 进入命令模式，输入 `wq` 保存并退出。

```
$ vi file.txt
```

`git status` 这条命令用于查看当前的状态，使用后我们可以看到返回的结果中提示，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。

```
$ 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)
```

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，后面跟的是我们需要为这里 `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
```

在文章的开头我们解释过 `add` 和 `commit` 的区别，读者如果觉得 `add`、`commit` 两步太麻烦的话，可以使用 `git commit -am` 这个命令，作用和使用两条命令通，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。

当你使用 `git commit` 命令填写注释的错误的时候，那么你可以使用 `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即可。

最后，使用这个命令来推送来远程仓库，读者是第一次推送，所以在你键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
```

## 4 Git 分支

我们知道，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，所以 Git 的主要面向情景是多人协作开发，但每个人开发后就是一个新的版本，很难确保多人开发的内容是完美并且互相兼容的，让他直接推送到发布版本上，所以我们需要增设一个新的概念——分支。

在下面一段代码中，我们使用 python 的#符号来为每一个命令做解释、说明。

#查看现有的分支，并在现在所在的分支的分支名加\*。

```
$ git branch
```

```
* master
```

#我们新建一个叫做 br2 的分支。

```
$ git branch br2
```

#checkout 用于切换分支，我们现在已经切换到了 br2 分支。

```
$ git checkout br2
```

```
Switched to branch 'br2'
```

#查看现有分支，这里我们看到，已经多了 br2 分支，并且 br2 分支前有\*符号说明我们已经切换到了 br2 分支。

```
$ git branch
```

```
* br2
```

```
master
```

# git checkout -b 这一个命令的作用相当于以上两个，创建并切换到 br3 分支

```
$ git checkout -b br3
```

```
Switched to a new branch 'br3'
```

#我们将要尝试推送 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，我们需要新建一个文件

```
$ 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：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稳定版本来发布，一个测试版本来开发，一个修复漏洞，大型的项目甚至还要细分更多。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月，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，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。

## 5 Git 自定义命令

如果读者 git 自身的命令太难用，键入速度太慢，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]，比方说在下面这个命令中我们将 br 命令等同于 branch 命令，那么以后使用输入 git br 即可：

```
$ git config --global alias.br branch
```

我们使用这么命令来看一下效果。

```
$ 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要说明，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，尝试着删除，回馈告诉我们无法删除，接着我们切换到 br2 分支，这时候再删除 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).
```

## 6 Git stash 命令

可能会有这样的情况发生，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 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，一个用于储藏现有的更改的命令。Stash 单词翻译为储藏，但相比于储藏，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。

我们先用 vi 命令来新建一个文件，然后 git add 添加到版本管理系统中，接着第一次查看状态，git bash 提示我们有一个文件已经添加但是没有提交，然后我们使用 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），最后我们第二次使用 `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
```

下面我们来学习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，`test2.txt` 确实已经被释放出来了，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
```

## 7 Git 文件对比

`git` 既然是版本管理系统，那么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，笔者在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 `#` 来做解释说明

#我们新建了一个文件夹，并将它初始化，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
```

#然后我们对 `test.txt` 做出了更改，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
```

我们使用 `git diff` 命令来第一次对比，显示`---a/test.txt` 显示的旧版本的内容，`+++b/test.txt` 显示的新版本的内容，例如“`+this is a test file`”表示我们新增了这一行内容。不加任何参数即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本地仓库，因为我们刚才已经将第一次编写的没有任何内容的 `test.txt` 添加并提交，第一次对比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。而在添加后暂存的内容就已经更改所以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，这是对比工作区和最新本地仓库，当我们将已经添加的 `test.txt` 文件提交后，第二次对比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
```

下面我们来总结一下，`git diff` 常见三种对比模式

- ☐ 对比工作区和暂存区  
`git diff [filename]`
- ☐ 对比暂存区和最新一版的本地仓库  
`git diff --cached [filename]`
- ☐ 对比工作区和最新一版的本地仓库  
`git diff head [filename]`

## 8 Git 版本日志、回退

虽然很多人对于版本回退这个词很恐惧，总是担心在公司的电脑万一错删稳定发布版本或者错删数据库，就会铸成大错，但是这样对于版本学习的心态是不好的，同时为了避免出现刚才那样的错误，下面这段学习我们强烈不建议你使用公司的项目来练习。

我们常规的创建一个文件夹，初始化，然后新建一个 `test.txt`，第一次添加，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
```

现在我们进入再次修改 `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` 命令可以查看我们的版本日志，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` 来输出每个版本一行。

```
$ 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 `^` 表示倒退一个版本，`^^` 表示倒退两个版本，以此类推，当你要倒退的版本太多的时候，你可以采用 `HEAD~n` 这种方式。

```
$ git reset --hard HEAD^
HEAD is now at 540cfdd the first commit
```

`git reset` 存在三种回退模式，分别是 `soft`, `mixed`, `hard`，无论是哪种模式，本地仓库版本一定倒退，但区别在于工作区和暂存区下面我们简单辨析一下这三种模式。

- ☐ `soft` 模式，工作区，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，暂存区清空

另外读者在前面和刚才的看到的由字母和数字组成的长字符串像“540cfdd9.....”是 `git` 内部的版本号，`git` 内部中并不采用 1,2,3,4 这样的数字，而是基于安全哈希算法生成的一个十六进制字符串。

## 附录3 我们都是创客，生来如此

我不知道该怎样开口，不知道该用怎样的文字，把自己的心情完整记下来，让他们如挂在风墙边的蓝色风铃般静静地挂在那里，静谧而美好。我想和你谈一谈梦想这个遥远而又熟悉的词汇。

不久之前，一位学姐写信向我诉苦，她说，和闺蜜一同去比赛，因为种种原因，她没有拿到奖项，而闺蜜不仅拿到了奖，而且是全校第一，每次回到宿舍，周围的同学总是在说闺蜜拿到了这个奖就可以怎样怎样。她感觉闺蜜夺走了属于她的光芒，因为以前，闺蜜的成绩远远不如她。在发自内心的替她感到的同时，又有一丝丝嫉妒。

良久沉吟，这或许是一种缺失，一种害怕不被人重视，害怕被世界遗忘的内心缺憾。

学姐也曾是一个骄傲的人，只不过现实和理想的巨大落差给了她重重一击，自己的现实和他人的情况对比又是一记重锤砸下，渐渐磨灭了她的骄傲。在社会的大磨盘下她从一个开始有棱有角的正方形，被打磨、抛光成一个光滑的球，又在一次又一次的刺激中逐渐变得麻木。

每个人都或多或少会有这种感觉，此刻，我想把我们探讨的结论转赠给你，愿致你以宁静。

这种缺失实际是孤独，内向，敏感，细腻导致了你的孤独。但不要因此而悲伤，正是孤独让你变得出众，而不是合群。阅读和思考是你获得能量最高效的方式，你总有无数的灵感，想着别人无法理解的事情。你在社会中就像个异类，就像世人难以读懂梵高，达芬奇的作品一样，你以为自己缺失，其实你是拥有太多。就像美剧的超能英雄一样。人和动物的根本区别就是人是在不断的思考中沉淀优势，明确航向，变成一个崭新的自我，你真正的缺失是对自己的不了解以及别人对你的不理解，弥补缺失的唯一方法就是阅读，探寻自己的内心，寻找自己真正想做的事情。才华横溢、有所作为的人，都是会享受和利用孤独的人，他们在孤独的时候积蓄能量。不要试图让别人理解你，正如你不能让盲人只凭触觉感受就做出绘画。任何一个害怕独处而在餐桌上、在夜总会、在酒吧里寻找一种存在感的人，都会在人群中被淹没。

不是你的错，既然无法适应这个世界，为什么不尝试去做一些事情来做出事实呢？就像创立特斯拉的硅谷钢铁侠埃隆·马斯克一样，凭借送入太空的特斯拉电动车，让外界看到他做白日梦的功力，把未来科技之美带给了世界。

疯狂的人有两种，一种是真疯，一种是真疯，前者强调在一个真字，后者强调在一个疯字，只有疯狂到认为自己能够改变世界的人才，才能够真正改变这个世界。

1776年，造就蒸汽革命瓦特的灵感源于他的异想天开，

1879年，造就万家灯火的爱迪生的灵感源于他的天方夜谭

1886年，造就“钢铁长城”卡尔本茨的灵感源于他的荒诞无稽

1903年，造就银鹰展翅莱特兄弟的灵感源于他的痴人说梦

大胆一些，不要害怕做“白日梦”，不要怀有渺小的梦想，他们无法打动人心，最重要的是，无法打动你的心。

无知不是错，错的是贪图安逸，放下你自己的骄傲，承认自己的不足吧！最怕碌碌无为一生的你，还安慰自己平凡可贵。

少年的少不在于骨龄，而在于心，心若鸿鹄比天高，纵是年岁已高，也尚可作为；若是年纪轻轻便已心老，了无生趣，对生命毫无激情，纵是年轻，又有何用，少年，心若在，梦就在。

高晓松说周长是憧憬和怀念的天平，当它倾斜得颓然倒下的时候，那些逝去的日光的夜

晚该用怎样的声音去抚慰？你的状态如何，决定你的未来，你现在的位置就是你过去的心态之中的体现，你如果过去奋发向上，你现在不言自明。如果你是异性，会不会爱上现在的自己？

一个人可以非常清贫，困顿低微，但是不可以没有梦想，只要梦想一天。那一天就可以改变自己的处境。威尔逊用一句话总结：“我们因梦想而伟大，所有的成功者都是大梦想家，在冬夜的火堆旁，在阴天的雨雾中，梦想着未来，有些人让梦想悄然绝灭，有些人的悉心培育、维护，直到它安然渡过困境，迎来光明，希望，而光明和希望总是降临在那些真心相信梦想一定会成真的人身上”。无梦想何茫，每伏于我之心，使我之心常不平，至此梦为实。就像埋在地下的种子一样，它们必须发芽，长出地面去追寻太阳的痕迹。

梦想就像镜中月，看起来很近，但你若是想要去触摸，你必须付出难以想象的努力，远方的美好。站在金字塔顶端俯视可看到众生，站在金字塔的底端，充斥在你眼前的只是砖石。高度，同样代表着你的眼界，若你周围都是生命的事事烦琐，你的心中也不会有天地的浩荡。是的，海纳百川，在于它的海纳百川，但你看不到那一切，又怎会收容包容一切？

人有梦，因梦不同。梦想得久，则成，我们难道不是在前人的梦想之中吗？春有春之暖，夏有夏之热，秋有秋之获，冬有冬之寒，得受生活的乐趣。

奋发是生命的常态，生命的岁月若不像流星燃烧，激情四射。又有何意义？不要让你就像过去百万年逝去的岁月一样，在历史的长河中逐渐归于尘埃，汲汲无名度过一生，你愿意吗？买菜洗衣做饭，朝九晚六，一眼可以看到底的生活是一种深渊。当你凝视深渊的时候，深渊也在凝视着你。

鹿晗在《勋章》中唱到：

“可我会像奥德修斯一样，  
朝着心中的方向，  
哪怕众神会在彼岸阻挡，  
当我需要独自站在，  
远方的沙场，  
武器就是我紧握的梦想，  
而我受过的伤，  
都是我的勋章。”

你不能把这个世界维持在你所不喜欢的样子。努力和遗憾，哪个更痛苦？坚持的人都是自己的英雄。在一次又一次的打击中“重整山河待后生”，将惨不忍睹的失败化作动力，在和血吞牙中变得越来越刚强。请不要放弃，不能放弃，你的每一次失败都将化作脚下的基石，为你的下一步奠定坚实的基础，请坚信你的梦想，哪怕代价是在现实面前，撞得头破血流，在他人嘲讽中心被纠的生疼。顾里说：“我不知道什么是年少轻狂，我只知道胜者为王。

一个人要实现自己的梦想，最重要的是具备以下两个条件，勇气和行动。

你是你一生的主角，不能轻易放弃；承载了太多期许的目光，怎能轻易放弃？

那就让我们提笔为刀，奋战到底。

不相信世界就是这样，明知道：

有的时候必须低头，

有的人必将失去，

有的东西命中注定不能长久，

仍然要说在第一千个选择之外，还有第 1001 扇窗等着我打开

然后有光透进来！

## 附录 4 下一步

我们工作的目标是提高工作效率，将采用人力运营维护的成本降低到最低，来更加充分有效的运用劳动力这一珍贵资源。可事实上，我们在现实生活中都在与这一宗旨相违背。

在编程界中，我们称呼这一过程为重复造轮子（明知车轮设计成圆形是最完美的，但你一定要寻找另一个形状的轮子），知道已经有现成的轮子，可是受于不开源的限制，许许多多的人都要做相同的事情，这种感觉是很痛苦的，既然大家都在做相同的事情，为什么不能由一个人开源出来，大家一起使用、完善呢？即便因为商业竞争的原因，核心部分的代码无法公开，那么无关紧要的内容也可以让大家众享啊！这样子对整个行业以及人类的进步来说，都是十分有益的。你不需要另起炉灶，只需要稍作调整。



图附录 4-1

许多公司和个人还是处于保守的观念中，自己创造的东西即便再微小也应该商业化使用，怎么能白给别人？其实不然，很多项目都是因为开源才活起来的，TensorFlow 的火爆并不纯粹是因为它的功能强大，如果没有办法从源码级别上来研究它，又有几个人能真正并且恰当的使用它呢？如果没有 Linux，现在互联网的格局简直无法想象？如果安卓当初没有选择开源，现在只能手机恐怕是苹果一家独大吧？

项目并不因为开源而没有市场，正相反，项目因开源而存活，开源意味着你选择向世界共享自己的成果，选择和全世界的程序员们一起来一场头脑风暴，你的项目不再仅仅局限于你的团队的几个人，会有众多的用户来体验，来帮你完善项目的不足，很有可能，当犯了一个错误的时候，你会惊讶的发现这个错误别人已经走过了。一些软件存活周期不长，正是因为他们采用了闭源，本来就不太完美的项目，参与的开发者少，那么它就更加不可能活下去了。

举一个典型的例子，你如果看 Spring（一个开源的 Java / Java EE 全功能栈的应用程序框架）最早的代码，和现在相比，会真心的感到差很多，最初不太友好的代码，正是因为开源，在不断的讨论和提交建议中演化成高质量的代码。

开源是一种文化，代码在其中交织，旋转，跳跃，升华。

即便从商业的角度来看，中国的市场并不适用于收费模式的商业模式，和绝大部分普通用户并不会花钱购买付钱软件一样，绝大部分的开发者也不喜欢使用闭源的项目。没有开源的项目，只会停留在一种孤芳自赏的状态，群体的力量远大于个人。在互联网的时代，个人（小团队）的英雄主义早已不在适用。

当你热衷于开源技术,同时愿意公开自己写的代码来为开源文化奉献一部分自己的力量  
的时候,你会对自己所致力于的伟大的事业抱有荣誉感。通过开源项目实现自己的个人价值,  
只要你愿意,没有人能阻挡你。就读者个人而言,参与开源项目能获得提升,这种提升更像  
是对自己个人的一种投资,这和老板逼迫你来实现某个功能是不一样的,真正的参与其中,  
你才能体会到其中的美妙。

遵循拿来主义的同时,也别忘了奉献,你会从中收获到欣赏和感谢,当你遇到让人欣赏  
的代码或者碰到 bug 的时候,不要忘记反馈给对应的开源项目,或者将自己的感想记录下来,  
分享给世界,哪怕它只是几句和朋友聊天的闲言碎语,让一些事情自然的发生,更多的去参  
与和感受,就像万物生产那样。你的参与,让世界更加美好。

这是我写作本书的初衷,也是我最终的答案——大脑是用来思考的,重复的工作交给机  
器吧!