ウェブページ情報の自動収集について

常三島技術部門
地域協働グループ　　飯田　仁(IIDA Hitoshi)

１．はじめに

別報にてRPAに関する原稿を執筆したが，RPAの実現の過程でウェブブラウザの操作を直接実施することが可能であることが分かった。今回はウェブブラウザを直接操作し，ウェブページ情報を自動的に収集する方法を紹介する。

２．収集対象について

本学には，学内で共用分析装置を利用する際に使用するウェブ予約システムがある。このシステムは，共用分析装置の有効活用を図る目的で導入され，分析装置利用者が利用装置の予約を実施した後，実際に装置を利用する。また，この予約情報から分析装置利用料金を算出し各研究室に消耗品費等の維持管理費用を請求する運用を実施している。

分析装置利用料金の算出・集計処理は全自動で実施され，適切な研究室毎に分類されたcsvファイルとして取得でき，収集結果との比較が容易なことに加え，ログイン処理等比較的複雑な画面操作が必要なため，このウェブ予約システムを収集対象とした。ただし，地域協働グループが関係する約40台の分析装置に限定した。

３．ウェブページ情報の自動収集技術

ウェブページ情報を自動収集するために，ウェブスクレイピングを利用した。ウェブスクレイピングとは，『ウェブサイトから情報を抽出するコンピュータソフトウェアのこと』と説明されている[1]。簡単には，プログラムによりウェブブラウザに任意の動作をさせることで必要なウェブサイト（ページ）を閲覧し，必要な情報を抽出するものである。今回利用したソフトウェアは

① ウェブブラウザ（Firefox-78.12.0）
② Python3.6.8
③ Selenium3.141.0[2]+WebDriver（Firefox用）

である。

試験環境のOSは，Scientific Linux　7.3を利用した。上記①～③の導入はウェブサーバではあったが，yumコマンドと，pip3コマンドにて簡単に導入できた。上記に加え，WebDriverはFirefox用のものを導入した[3] 。

実際に情報収集対象のウェブ予約システムを利用し，機器の予約状況を確認するには

A) ブラウザでログイン画面に接続する。
B) ユーザーIDとパスワードを入力後，ログインボタンをクリックする。IDとパスワードに問題無ければマイページが表示される。
C) 機器検索/予約/受託サービスをクリック。
D) 対象機器の所属学部をクリック。
E) 機器カテゴリ内から対象機器のカテゴリ名をクリック。
F) 対象機器をクリック。
G) ダイアログボックスが表示されるのでその中の『この機器を予約』ボタンをクリック。
H) 確認する日付をクリック。

以上の操作を行うことで画面下部に予約状況が表示される。図１に機器の利用状況画面例を示す。紙面の関係で画面の不要部分は削除している。

図１　機器利用状況画面例

予約一覧

開始時	終了時	講座	氏名	Mail	内線	連絡
09:00	11:00	系光機...	トモ...	@tokushima-...	8	コメント無し
12:30	14:30	料科学...	ﾋﾛﾑ...	@tokus...	1	コメント無し
15:00	18:00	化学プ...	ﾏﾅﾐ...	@tokushima-...	4	コメント無し
18:00	21:00	系光機...	)...	@tokushima-...	5	コメント無し

図２　利用状況拡大

図１は小さく見づらいので，今回の作業対象となる予約状況の部分を拡大した物を図２に示す。図２の表は開始時・終了時・講座・氏名・Mail・内線・連絡の7項目が表示されており，一部が省略表示されているのが確認できる。図２は個人情報も含まれているため，一部にモザイク処理を施した。

図２の講座名に関する省略部分のソースを図３に示す。図３では画面上に本来ならば『応用化学システム化学プロセスC-9』と全部表示する所，『応用化学システム化学プ…』といった具合に「…」と省略された形で表示される。これ以外に，氏名やメールアドレスも同様で，これは限られた画面サイズ内により多くの情報を表示させるテクニックの一つであり，ウェブ予約システムの不具合ではない。

```
<a href="javascript:void(0);" title="応用化学システム化学プロセス C-9">応用化学システム化学プ...</a>
```

図３　省略部分のソース

今回は，上記A)～H)の操作をPythonにてプログラムし，Firefoxを操作することで，対象のページを表示させ，ウェブ画面の情報を取得，必要な予約状況のみを抽出し，日付・装置ごとにまとめcsvファイルとして出力した。

プログラム例を図４に示す。記述されているユーザーIDやパスワード，URLについては実際とは異なる。図４では最初の2行でユーザーIDとパスワードを変数に代入し，3行目でFirefoxを起動する。4行目でA)の処理を，5~10行目でB)の処理を実施している。5，7，9行目のfind_element_by_[xpathやclass_name等]はウェブページ上の要素を探すための処理で，続く括弧内にそれぞれ目的のキーワードを記述する。このキーワードはウェブブラウザ（Firefox）で，確認したい箇所にマウスカーソルを合わせて右クリックし，表示されるメニューの『調査』をクリックするとブラウザ下部にHTMLソースが表示されるので参考にする（図５）。このキーワード確認作業はサーバではなく個人PCにて実施した。図５のID入力欄の確認結果を図６に示す。図６の結果より図４のプログラム例5行目となる。

```
 1:USERNAME=user001
 2:PASSWORD=password
 3:driver = start_firefox() # Firefoxの起動
 4:driver.get('https://hogehoge.com/')
 5:username_input = driver.find_element_by_xpath('//input[@id="id_id"]') # ユーザーID 入力欄検出
 6:username_input.send_keys(USERNAME) # ユーザーID 入力
 7:password_input = driver.find_element_by_xpath('//input[@id="id_pwd"]') # パスワード入力欄検出
 8:password_input.send_keys(PASSWORD)# パスワード入力
 9:login_button = driver.find_element_by_class_name('login') # ログインボタン検出
10:login_button.click() # ログインボタンクリック
```

図４　プログラム例

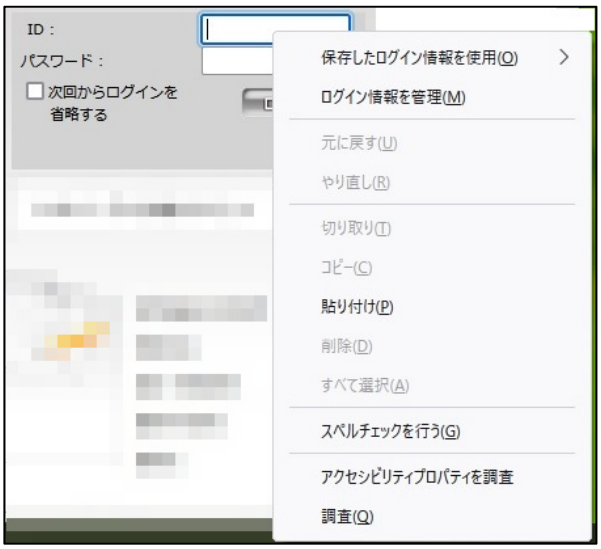

図５　キーワード確認方法

```
<input id="id_id" class=input_text" type="text" name="id" size="12">
```

図６　ID入力欄の調査結果

以上の様にキーワード等を調査しつつ，プログラムの動作確認を実施し，適切にプログラムを作成し，必要情報を自動的にcsvファイルとして収集することができた。

４．動作状況

当初，個人PCを用いて開発を行っていたがイ) ウェブ予約システムの負担を考え，利用者の少ない深夜帯に実行するため，本来停止している個人PCを自動収集の目的で稼働させるのは無駄である。

ロ) 収集した情報をウェブページ上で比較できた方が良い。

以上のような理由から，学内向けウェブサーバを利用することにし，途中からサーバ上で開発を行い，試験稼働にたどり着いた。

毎日0時頃に前日の分析装置利用状況を自動収集するために，crontabにて設定を行った。ログインから最初の装置の情報収集までに約30秒，その後，各装置の情報収集に必要な時間は約3秒と高速であるがサーバへの負担を考慮し17秒の待ち時間を挿入した。1分あたり3台の装置情報が収集できる。

５．収集結果

2021年10月から1ヶ月間限定で自動収集を実施した所，問題無く装置利用状況を収集できた。しかし，時々図４に示す3行目のFirefoxの起動に失敗することがある。失敗するとcrontabがエラーのメールを送信するためその都度，サーバにリモートログインし手動にて再実行していた。この問題に対し，crontabの設定も0時のみでなく，2時・4時・6時を追加し，1日4回実行に変更した所，0時台に失敗しても2時台に確実に処理が終了するようになった。本格稼働を目指す場合，原因の究明と対策が必要である。

６．応用例

今回ウェブブラウザを利用して自動収集を実施したが，ウェブブラウザを使用する業務等は多くあると思う。可能性という意味で自動化の一例を示したい。

ある大手通販サイトでは，休祝日に購入処理を実施する事で一律数%の割引を受けることが可能になる。通販サイトのプライベートブランド商品ではさらに割引率が大きくなり，法人（組織）としての経費削減効果は大きいと考える。

個人で利用する場合は，平日に必要な商品を買い物かごに追加しておき，休日に購入処理を実施すれば良いが，法人にて実施する場合には，たとえ5分の作業やリモート（自宅からの遠隔）作業，スマートフォンで可能な作業であっても，休祝日業務となり手当等の労働条件に関して特別な処理が必要になるが，今回のウェブスクレイピングにより購入処理を自動化すれば大きな経費削減につながると思う。具体的には

a) ウェブブラウザを起動する。
b) 通販サイトに接続する。
c) カレンダー画面を表示させ，キャンペーンコードを取得する。
d) ログインする。
e) 買い物かごの内容を確認する。
f) 購入処理へ進む
g) キャンペーンコードを入力し割引適用処理を行う。
h) 出荷日が遅くなる商品があれば，その旨

の通知が表示されるので確認する。

i) 注文内容の確定を行う。

j) ウェブブラウザを終了する。

という流れの処理である。h)については出荷日に変更が無い場合は表示されない。

実際上記a)～j)の処理を筆者の個人ID・パスワードにて実施した所，問題無く自動操作ができた。こちらは自宅PCで実施したが，実行時にマウスカーソルが勝手に動くのには不思議な感じがした。

もちろん上記の処理をRPAで実現することも可能であるが，商用RPAは導入コストが高いため，これだけの自動化に導入することは本末転倒となる。また，RPAによりウェブブラウザを操作するよりも，直接ウェブブラウザを操作するウェブスクレイピングの方が処理速度で有利になると考える。

上記例の様に，利用方法の可能性は広がるが，通販サイトの利用規約にロボット禁止と明確に表示がある場合や，著作権情報の収集に利用する場合は，法律に触れる危険性があるので注意が必要であることを付け加えておく。

７．まとめ

今回は技術紹介の意味合いが強かったので短期間の試験運用であるが，複雑な処理も可能であることが分かった。ウェブスクレイピングやRPAをはじめとする自動化処理やAIの活用は，PCを使用する作業の一大変革をもたらすと実感した。

８．最後に

今回はディスプレイの無いサーバを利用した。ディスプレイを使用しない場合は，『ヘッドレスモード』によりプログラムを実行させる必要がある。この処理を実現するプログラム例[4]を図７に示す。図７は関数として記述した。実際の使用では図４の3行目の様に利用する。

参考文献

[1] https://ja.wikipedia.org/wiki/ウェブスクレイピング

[2] https://ja.wikipedia.org/wiki/Selenium_(ソフトウェア)

[3] https://centos.perlzemi.com/blog/20210105090837.html

[4] https://blog.n-hassy.info/2021/05/selenium-raspi-headless/


```
def start_firefox():
    Options = webdriver.FirefoxOptions()
    Options.headless = True
    Options.add_argument('--no-sandbox')
    Options.add_argument('--disable-gpu')
    Options.add_argument('--ignore-certificate-errors')
    Options.add_argument('--allow-running-insecure-content')
    Options.add_argument('--disable-web-security')
    Options.add_argument('--disable-desktop-notifications')
    Options.add_argument("--disable-extensions")
    Options.add_argument('--lang=ja')
    Options.add_argument('--blink-settings=imagesEnabled=false')
    driver = webdriver.Firefox(options=Options)
    return driver
```

図７　ヘッドレスモードのプログラム例