

{"Python": "Selenium"}

Pythonと基本情報処理科 菅 良平

2021/07/29日



# Selenium ?

Python/Ruby/Javaなどからブラウザの自動化（テスト開発自動化）を可能にするAPI群などの総称です。

## [Seleniumツール]

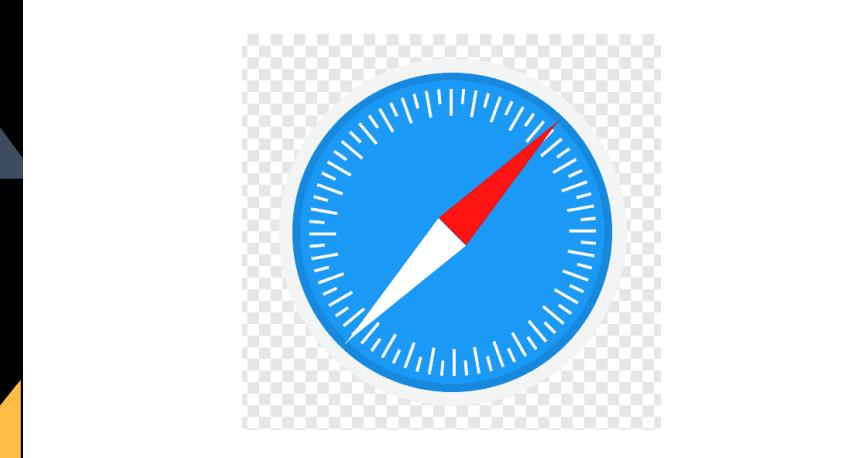
Seleniumは以下のツールが提供されています。

- Selenium Webdriver > Selenium API群(今回、味見)
- Selenium IDE > 統合開発環境
- Selenium RC(Remote Client)
- Selenium Grid

## [主に出来ること]

- 自動的なスクレイピング (+Beautiful Soup)  
HTML要素を探す。
- Web上のページやアプリのブラウザを介した自動テストの実行
- ブラウザの自動操作
- イベントハンドラ (KeyやMouse操作)
- HTML要素の探索

… and more



# Webdriver ?

Webdriverは、オブジェクト指向APIであり、ブラウザを操作できます。

Seleniumで利用できるWebdriverの種類は以下の通りです。

- ✓ Microsoft Edge
- ✓ Chrome - 味見しました :)
- ✓ Firefox
- ✓ Safari
- ✓ Opera

python



# Python (準備編)

venvなどのPython仮想環境で必要なモジュールをインストールすることをお勧めします。

1. まずは仮想環境を作りましょう。

```
> python -m venv [仮想環境名]
```

```
> cd [仮想環境名]
```

2. 仮想環境を有効化します。 (ご使用の環境に応じて、、、使用するコマンドが変わります。)

(For Windows ↓)

```
> .¥Scripts¥activate(仮想環境から戻すには、activateをdeactivateに変更します。)
```

(For MacOS, Linux ↓)

```
> source ./bin/activate(仮想環境から戻すには、activateをdeactivateに変更します。)
```

3. 仮想化完了後、必要なモジュールのインストールを行います。

```
> pip install selenium
```

モジュール確認は、

```
> pip list
```

でおこなえます。

今回、使用したSeleniumのバージョンは、3.141.0です。

```
cmd コマンドプロンプト
(develop) C:¥Users¥HONGO¥Desktop¥PythonWorks¥PySeisaku>pip list
Package           Version
-----
click            8.0.1
colorama         0.4.4
Flask             2.0.1
itsdangerous    2.0.1
Jinja2            3.0.1
MarkupSafe        2.0.1
pip              21.1.3
selenium          3.141.0
setuptools       41.2.0
urllib3           1.26.6
Werkzeug          2.0.1
WARNING: You are using pip version 21.1.3; however, version 21.2.1
         is available.
         You should consider upgrading via the 'c:¥users¥hongo¥desktop¥pythe
         stall --upgrade pip' command.

(develop) C:¥Users¥HONGO¥Desktop¥PythonWorks¥PySeisaku>
```

※ モジュール確認 (コマンドプロンプト)



# WebDriver (準備編)

次にWebDriverをダウンロードします。ここでは、ChromeDriverを使用しています。

- <https://chromedriver.storage.googleapis.com/index.html> (今回はv91を使用しました。)

各環境 (Windows, OSX, Linux) に適合するバイナリファイルをダウンロードします。

ダウンロード・解凍後、コマンドラインツール(Dosなど)で環境PATH(コマンドからWebDriverを立ち上げる場所)に配置します。

(echo %path% などで現在利用できるPATH環境変数を確認します。今回はAnacondaが利用できるので、AnacondaのPATHを利用します。)

例) Anacondaがインストールされている場合は、anaconda3¥Library¥bin以下に解凍したWebdriver(**chromedriver.exe**)を配置します。

(Windows環境) [anaconda インストール環境]¥anaconda3¥Library¥bin(内に配置)

charset.dll	2017/11/14 4:55	アプリ
<b>chromedriver.exe</b>	2021/06/08 11:07	アプリ
cilkrts20.dll	2018/08/04 20:18	アプリ

配置後、コマンドプロンプト (またはTerminal) 画面で、以下を実行してみます。

> chromedriver

chromedriverのPATHに問題がなければ、以下のメッセージが表示されます。 (⇒WebDriverを使用する準備が整いました。Ctrl+Cで中断)

```
コマンドプロンプト - chromedriver
(develop) C:\Users\HONGO>chromedriver
Starting ChromeDriver 91.0.4472.101 (af52a90bf87030dd1523486a1cd3ae25c5d78c9b-refs/branch-heads/4472@{#1462}) on port 9515
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
```

## Index of /91.0.4472.19/

Name	Last modified	Size
Parent Directory		
<a href="#">chromedriver_linux64.zip</a>	2021-04-22 20:18:16	5.62MB
<a href="#">chromedriver_mac64.zip</a>	2021-04-22 20:18:18	7.68MB
<a href="#">chromedriver_mac64_m1.zip</a>	2021-04-22 20:18:19	7.05MB
<a href="#">chromedriver_win32.zip</a>	2021-04-22 20:18:21	5.59MB

※ バイナリイメージダウンロードディレクトリ  
(Chrome)



# やってみた（かった）こと

- サンプルWebアプリケーション（Flask + SQLite）を作成して、自動更新・検索などをさせてみる。
  - ① 作成したアプリケーションの新規登録ボタンを自動化し、データの更新などを行う。
  - ② 検索の自動化（ありえない文字列の発見など。）
- 普段ブラウザを利用しているときのような複数動作をさせてみる。

ブラウザを待機させ、ブラウザが自動的にタブに別ページを表示してゆく自動プレゼンテーションプログラムを考えていました。  
(※タブの操作はSelenium4で実装されるため実装不可能でした。新規Windowでは可能。)
- GUI(Tkinter)からSeleniumを呼び出してみる。
- 自動的にスクリーンショットを撮影する
- AIによる自動化 ※ 時間
- UX(User eXperience)の分析を自動化する際にも使える？ (UXへの知見を深めたい)

などなど。

でしたが・・・、コード掲載が難しかったので、一部をDemo Codeやアプリケーション遷移図を次ページに掲載しました。

作成したコードの一部は、[github](#)で閲覧できます。

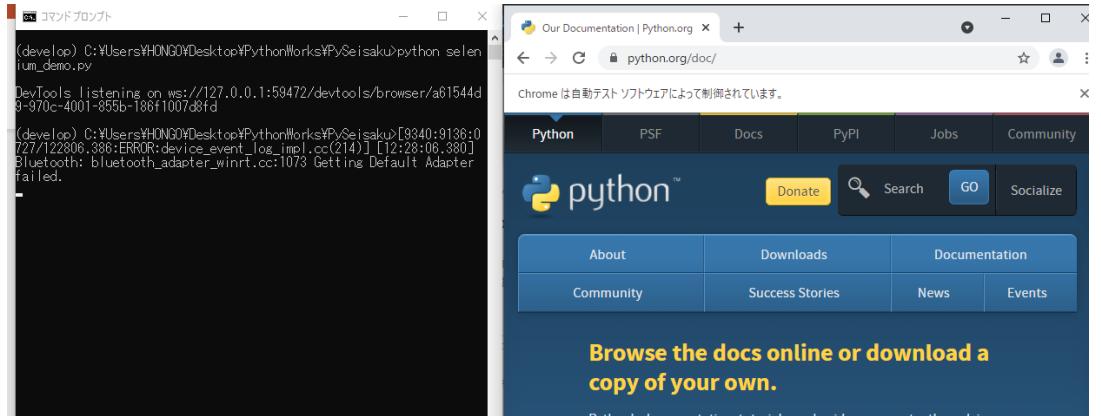


# Demo Code(Chromeを呼び出す)

```
#  
# selenium_demo.py  
# モジュール呼出 - 初めてのSelenium(Chrome編)  
from selenium.webdriver import Chrome, ChromeOptions  
  
# Chromeを呼び出す際のOptionを指定できます  
# 例) --headless : ブラウザを画面表示せずに処理を実行するモード  
# ... and more  
option=ChromeOptions()  
  
# Chromeを呼び出します。  
driver=Chrome()  
driver.get('https://www.python.org/doc/')  
  
# ↓開かれたWindowを閉じます。  
# driver.close()  
# ↓ブラウザを終了します。  
# driver.quit()
```

コマンドプロンプトやterminalで実行してみましょう。

➤ `python selenium_demo.py`



成功するとコマンドからChromeが呼び出され、Pythonのドキュメントページが読み込まれます。



# Demo Code(ScreenShotを撮影する)

```
# selenium_screenshot_demo.py
# モジュール呼出 - 初めてのSelenium(Chrome編)
from selenium.webdriver import Chrome, ChromeOptions

# Chromeを呼び出す際のOptionを指定できます
# 例) --headless : ブラウザを画面表示せずに処理を実行するモード
# ... and more

option=ChromeOptions()
#headless modeで起動。 (ブラウザは立ち上がりませんが、動作しています。)
option.add_argument('--headless')

# Chromeを呼び出します。
driver=Chrome(options=option)
driver.get('https://www.yahoo.co.jp/')

#保存する画像を指定します。()
img='screenshot.png'
driver.save_screenshot(img)

# ↓ブラウザを終了します。

driver.quit()
```

コマンドプロンプトやterminalで実行してみましょう。

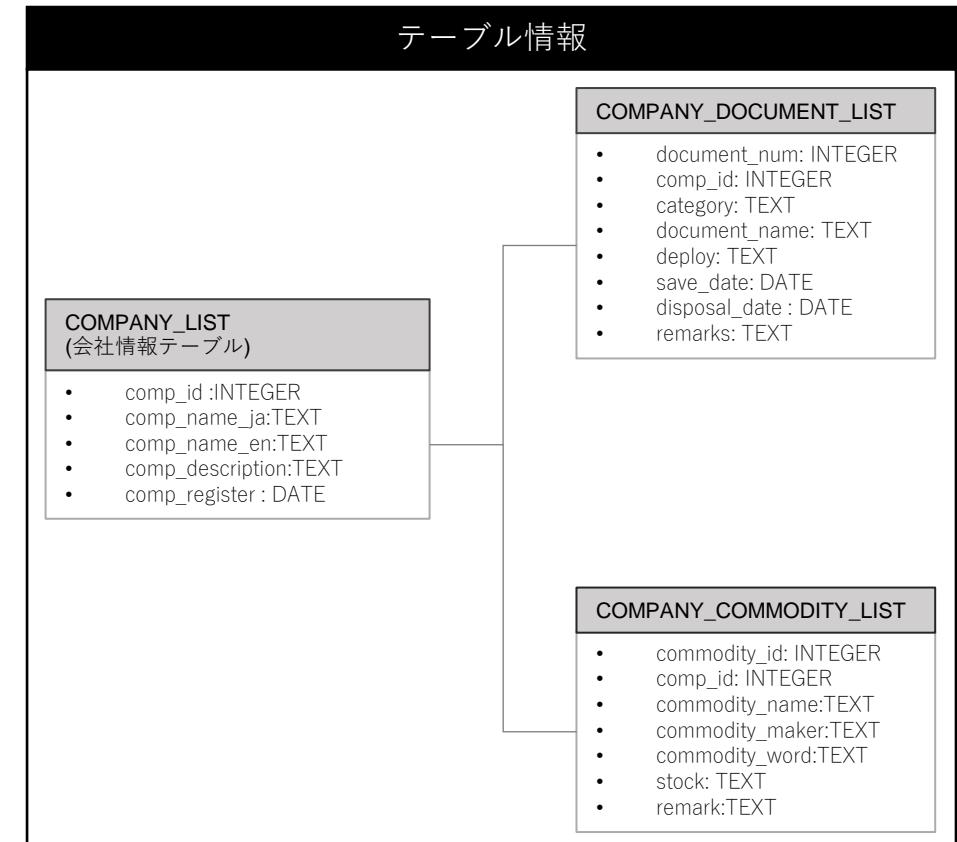
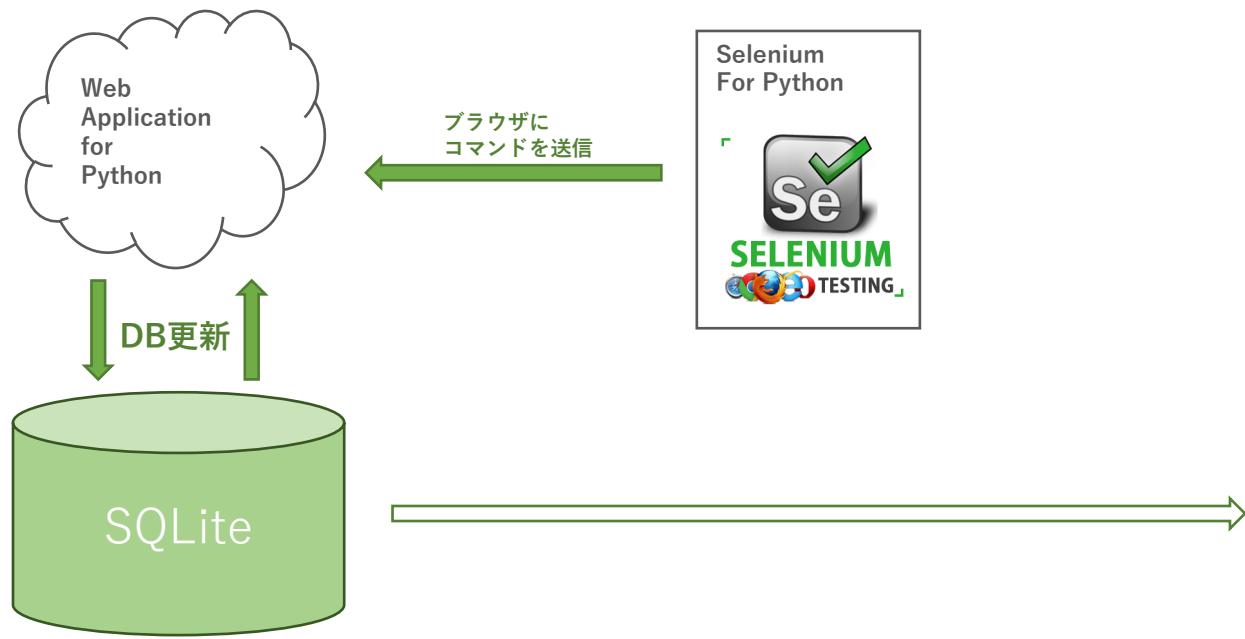
➤ `python selenium_screenshot_demo.py`



成功するとPythonを実行したディレクトリと同じ場所に"screenshot.png"が保存されます。（ここでは、Yahooのスクリーンショットを撮影しています。）



# Sample Web Application遷移図 (自動更新テスト) : DBテーブル情報





# (課題) : 大変だった点

- ブラウザの待ち時間を設定する。（処理スピードが速いので、あっという間にページ遷移が発生してしまう。）  
sleep関数が利用できるが、複数処理の場合、順に処理の実行が行われるためPython内にあるJavaScriptなどが実行されずにWindowが閉じてしまう。  
→ 処理のMulti Thread化を行い、一応、解決。
- ユーザにとってブラウザがどのように操作されるかを予測（もしくは想像）し、Codeを起こすこと。  
→ 整理をすることが難しかった。ブラウザ操作に関するフローを図式にしてみれば・・・・と考えていました。
- 状態遷移を複数化する際、Tab操作を行いたかったが、Selenium次期開発バージョン(ver.4)での実装になっていたため、  
今回の実装はあきらめざるをえなかった。
- TESTアプリケーションを作成するところまでは至らなかった点。（設計・スケジュール管理の大切さを痛感しました。）
- **Hackathon**なスタイルでもやりたかったなと思いました。



## 参考サイト

- [Selenium 日本語ドキュメント — Selenium 日本語ドキュメント \(infoscience.co.jp\)](#)
- [Selenium ブラウザ自動化プロジェクト :: Selenium ドキュメント](#)
- [WebDriver\(W3C.org\)](#)



最後に

有意義な3か月間でした。  
ありがとうございました。