

Seleniumテスト完全ガイド - 新人エンジニア向け

対象読者: 新人エンジニア・テスト自動化初心者

目的: Seleniumを使ったWebテストの基礎から実践まで

学習時間: 約30分

目次

1. Seleniumとは何か
2. 環境構築の手順
3. HTML構造の理解
4. 要素取得方法の完全解説
5. 実践的なテストコード
6. ページオブジェクトパターン
7. よくあるエラーと対処法

1. Seleniumとは何か

定義

Selenium（セレニウム） は、Webブラウザを自動操作してUIテストを行うためのツールです。人間がブラウザで行う操作を、プログラムで自動化できます。

なぜSeleniumが必要なのか

【手動テストの場合】

開発者: 「新機能を追加したので、全機能を手動でテストしてください」

↓

テスター: ブラウザを開く

↓

ログイン画面を開く

↓

ユーザー名を入力

↓

パスワードを入力

↓

ログインボタンをクリック

↓

結果を確認

↓

この作業を何度も繰り返す... 😞
(1回のテストに10分、1日に10回テスト = 100分)

【Seleniumテストの場合】

開発者: 「新機能を追加しました」

↓

自動テスト実行 🤖

↓

3分で全テスト完了! ✨

↓

結果をレポート出力

Seleniumの構成要素

あなたが書くテストコード (Python等) |
driver.get("https://example.com") |

↓ 命令

Selenium WebDriver |
(ブラウザ操作のエンジン) |

↓ ブラウザを制御

実際のブラウザ (Chrome等) |
Webページを表示・操作 |

対応ブラウザ

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Safari
- Opera

🔗 2. 環境構築の手順

Python + Seleniumの環境を作る

ステップ1: Pythonのインストール確認

```
bash

python --version
# Python 3.8以上が推奨
```

ステップ2: Seleniumのインストール

```
bash

pip install selenium
```

ステップ3: WebDriverの準備（2つの方法）

方法A: webdriver-managerを使う（推奨・簡単）

```
bash

pip install webdriver-manager
```

コードでの使い方:

```
python

from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service

# 自動でドライバをダウンロード・管理
service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service)
```

方法B: 手動でダウンロード

1. ChromeDriver公式サイトにアクセス
2. 使用中のChromeバージョンを確認（chrome://version/）
3. 対応するドライバをダウンロード
4. PATHの通った場所に配置

3. HTML構造の理解

Webページは「入れ子構造」になっている

Webページは、HTMLというマークアップ言語で書かれており、「親・子・孫」のような階層構造になっています。

実際のHTML例

```
html

<!DOCTYPE html>
<html>
  <head>
    <title>ショッピングサイト</title>
  </head>
  <body>
    <div id="main">
      <h1 id="main-title">ようこそ</h1>

      <div class="content">
        <form>
          <input name="username" type="text" placeholder="ユーザー名">
          <input name="password" type="password" placeholder="パスワード">
          <button class="login-btn">ログイン</button>
        </form>

        <a id="next-link" href="/products">商品を見る</a>
      </div>
    </div>
  </body>
</html>
```

階層構造を図で表現

```
html (ルート)
├── body
│   ├── div (id="main")
│   │   ├── h1 (id="main-title") → 「ようこそ」
│   │   └── div (class="content")
│   │       ├── form
│   │       │   ├── input (name="username")
│   │       │   ├── input (name="password")
│   │       │   └── button (class="login-btn")
│   │       └── a (id="next-link") → 「商品を見る」
```

HTML要素の属性

HTML要素には様々な「属性」があり、これを使って要素を特定します。

```
html
```

```
<input id="email" name="user_email" type="text" class="form-input">
```

この要素の属性:

- **id:** "email" (ページ内で一意)
- **name:** "user_email" (フォーム送信時の名前)
- **type:** "text" (テキスト入力欄)
- **class:** "form-input" (CSSのスタイル用)

```
<a name="section4"> </a>
```

4. 要素取得方法の完全解説

Seleniumでは、HTML要素を「ロケータ (Locator)」という方法で探します。

4-1. By.ID - IDで探す

最も確実な方法。IDはページ内で一意 (1つだけ) のため。

```
python
```

```
from selenium.webdriver.common.by import By
```

```
element = driver.find_element(By.ID, "login-button")
```

HTML例:

```
html
```

```
<button id="login-button">ログイン</button>
```

図解:

```
ページ全体を検索
```

```
↓
```

```
id="login-button" を持つ要素を探す
```

```
↓
```

```
<button id="login-button">ログイン</button> ← 見つかった！
```

4-2. By.NAME - name属性で探す

フォーム要素（入力欄など）でよく使われる。

```
python  
  
element = driver.find_element(By.NAME, "username")
```

HTML例:

```
html  
  
<input name="username" type="text">
```

図解:

```
フォーム内を検索  
↓  
name="username" を持つ要素を探す  
↓  
<input name="username" type="text"> ← 見つかった！
```

4-3. By.CLASS_NAME - クラス名で探す

スタイルが適用された要素を探す。

```
python  
  
element = driver.find_element(By.CLASS_NAME, "submit-button")
```

HTML例:

```
html  
  
<button class="submit-button">送信</button>
```

注意: 複数の要素が同じクラスを持つ場合、最初の1つが返されます。

4-4. By.TAG_NAME - タグ名で探す

特定のHTMLタグを探す。

```
python
```

```
element = driver.find_element(By.TAG_NAME, "h1")
```

HTML例:

```
html
```

```
<h1>ページタイトル</h1>
```

4-5. By.XPATH - XPathで探す

複雑な階層構造から要素を探す強力な方法。

XPathの基本構文

```
python
```

```
element = driver.find_element(By.XPATH, "//div[@id='main']/div/ul/li[1]/a")
```

XPathの読み方を図解

```
html
```

```
<div id="main">
  <div class="section">
    <ul>
      <li><a href="/item1">商品1</a></li>
      <li><a href="/item2">商品2</a></li>
    </ul>
  </div>
</div>
```

XPath: `//div[@id='main']/div/ul/li[1]/a`

階層を1つずつたどる:

```
//div[@id='main']
```

↓ "id='main'のdivを探す"

```
<div id="main"> ← ここ
```

```
/div
```

↓ "その中のdiv"

```
<div class="section"> ← ここ
```

```
/ul
```

↓ "その中のul"

```
<ul> ← ここ
```

```
/li[1]
```

↓ "最初のli ([1]は1番目の意味) "

```
<li><a href="/item1">商品1</a></li> ← ここ
```

```
/a
```

↓ "その中のa (リンク) "

```
<a href="/item1">商品1</a> ← 最終的にこれが取得される！
```

XPathのよく使う記号

記号	意味	例
<code>//</code>	どこにあってもいいから探す	<code>//div</code>
<code>/</code>	直下の子要素	<code>/ul/li</code>
<code>[@属性='値']</code>	属性で絞り込む	<code>[@id='main']</code>
<code>[数字]</code>	何番目の要素か	<code>[1]</code> <code>[2]</code>
<code>text()</code>	テキスト内容で探す	<code>[text()='ログイン']</code>

4-6. By.CSS_SELECTOR - CSSセレクタで探す

CSSの記法で要素を探す。XPathより短く書ける。

```
python
```

```
element = driver.find_element(By.CSS_SELECTOR, "div#main ul li:first-child a")
```

CSSセレクタの基本

```
html
```



```
<div id="main">
  <div class="content">
    <ul>
      <li><a href="/item1">商品1</a></li>
    </ul>
  </div>
</div>
```

CSS: `div#main div.content ul li:first-child a`

順番に読む:

div#main

↓ "idがmainのdiv"

<div id="main">

div.content

↓ "classがcontentのdiv"

<div class="content">

ul

↓ "その中のul"

li:first-child

↓ "最初のli"

商品1

a

↓ "その中のa"

商品1 ← 取得される！

CSSセレクタのよく使う記法

記法	意味	例
#id名	IDで探す	#login-button
.class名	クラスで探す	.submit-btn
要素名	タグで探す	button, div
親 > 子	直下の子要素	div > span
親子	子孫要素（何階層でも）	div span
:first-child	最初の要素	li:first-child
:last-child	最後の要素	li:last-child
[属性=値]	属性で探す	[type="submit"]

XPath vs CSSセレクタ 比較表

目的	XPath	CSSセレクタ
IDで探す	//div[@id='main']	div#main
クラスで探す	//div[@class='content']	div.content
直下の子	/ul/li	> ul > li
最初の要素	/li[1]	li:first-child
テキストで探す	//button[text()='送信']	不可（XPathを使う）

💡 5. 実践的なテストコード

5-1. 基本のログインテスト

python

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

```
# ブラウザを起動
```

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

```
try:
```

```
    # 1. ページを開く
```

```
    driver.get("https://example.com/login")
```

```
    print("ページを開きました")
```

```
    # 2. 要素が表示されるまで待機 (最大10秒)
```

```
    wait = WebDriverWait(driver, 10)
```

```
    # 3. ユーザー名を入力
```

```
    username_field = wait.until(
        EC.presence_of_element_located((By.ID, "username"))
    )
```

```
    username_field.send_keys("test_user")
```

```
    print("ユーザー名を入力しました")
```

```
    # 4. パスワードを入力
```

```
    password_field = driver.find_element(By.ID, "password")
```

```
    password_field.send_keys("test_password")
```

```
    print("パスワードを入力しました")
```

```
    # 5. ログインボタンをクリック
```

```
    login_button = driver.find_element(By.ID, "login-button")
```

```
    login_button.click()
```

```
    print("ログインボタンをクリックしました")
```

```
    # 6. ログイン成功を確認
```

```
    success_message = wait.until(
        EC.presence_of_element_located((By.CLASS_NAME, "success"))
    )
```

```
    # 7. メッセージの内容を確認
```

```
    assert "ログインしました" in success_message.text
```

```
    print("✅ テスト成功!")
```

```
except Exception as e:
```

```
    print(f"❌ テスト失敗: {e}")
```

```
finally:
```

```
# 必ずブラウザを閉じる
driver.quit()
```

5-2. 要素の操作メソッド一覧

```
python

# テキストを入力
element.send_keys("入力する文字列")

# ボタンをクリック
element.click()

# 要素のテキストを取得
text = element.text

# 属性値を取得
value = element.get_attribute("value")
href = element.get_attribute("href")

# 要素が有効かチェック
is_enabled = element.is_enabled()

# 要素が表示されているかチェック
is_displayed = element.is_displayed()

# 要素が選択されているかチェック (チェックボックス等)
is_selected = element.is_selected()

# 入力欄をクリア
element.clear()
```

5-3. iframeの扱い方

iframeは「ページの中の別のページ」です。iframe内の要素を操作するには、特別な手順が必要です。

```
python
```

```
# iframe内に切り替え
iframe = driver.find_element(By.TAG_NAME, "iframe")
driver.switch_to.frame(iframe)

# iframe内の要素を操作
element_inside = driver.find_element(By.ID, "some-id")
element_inside.click()

# 元のページに戻る
driver.switch_to.default_content()
```

図解:

メインページ
└─── iframe (別のページ)
 └─── 操作したい要素 ← ここにアクセスするには切り替えが必要

5-4. 複数の要素を取得する

```
python

# すべてのリンクを取得
links = driver.find_elements(By.TAG_NAME, "a")

for link in links:
    print(f"リンク: {link.text} → {link.get_attribute('href')}")

# すべての商品名を取得
products = driver.find_elements(By.CLASS_NAME, "product-name")
for i, product in enumerate(products, 1):
    print(f"商品{i}: {product.text}")
```


6. ページオブジェクトパターン

なぜページオブジェクトパターンを使うのか？

悪い例（コードの重複）

```
python
```

```
# テスト1
```

```
def test_login_success():  
    driver.find_element(By.ID, "username").send_keys("user1")  
    driver.find_element(By.ID, "password").send_keys("pass1")  
    driver.find_element(By.ID, "login-button").click()  
    # テスト内容...
```

```
# テスト2
```

```
def test_login_failure():  
    driver.find_element(By.ID, "username").send_keys("user2")  
    driver.find_element(By.ID, "password").send_keys("wrong")  
    driver.find_element(By.ID, "login-button").click()  
    # テスト内容...
```

問題点:

- 同じコードを何度も書いている
- HTMLが変わったら全部修正が必要
- 読みにくい

良い例（ページオブジェクトパターン）

```
python
```

```
# ページオブジェクト
```

```
class LoginPage:
    def __init__(self, driver):
        self.driver = driver
        self.wait = WebDriverWait(driver, 10)

    def enter_username(self, username):
        field = self.wait.until(
            EC.presence_of_element_located((By.ID, "username"))
        )
        field.send_keys(username)
        return self

    def enter_password(self, password):
        field = self.driver.find_element(By.ID, "password")
        field.send_keys(password)
        return self

    def click_login(self):
        button = self.driver.find_element(By.ID, "login-button")
        button.click()
        return HomePage(self.driver)
```

```
# テスト
```

```
def test_login_success():
    login_page = LoginPage(driver)
    login_page.driver.get("https://example.com/login")

    home_page = (login_page
        .enter_username("user1")
        .enter_password("pass1")
        .click_login())

    assert "ホーム" in home_page.driver.title
```

ページコンポーネントパターン

複数の製品がある一覧ページの場合：

```
python
```

```
# 製品1つを表すコンポーネント
```

```
class Product:
```

```
    def __init__(self, root_element):
```

```
        self.root = root_element
```

```
    def get_name(self):
```

```
        return self.root.find_element(
            By.CLASS_NAME, "product-name"
        ).text
```

```
    def get_price(self):
```

```
        price_text = self.root.find_element(
            By.CLASS_NAME, "product-price"
        ).text
        return float(price_text.replace("$", "").replace(",", ""))
```

```
    def add_to_cart(self):
```

```
        button = self.root.find_element(
            By.CLASS_NAME, "add-to-cart-button"
        )
        button.click()
```

```
# 製品一覧ページ
```

```
class ProductsPage:
```

```
    def __init__(self, driver):
```

```
        self.driver = driver
```

```
    def get_products(self):
```

```
        product_elements = self.driver.find_elements(
            By.CLASS_NAME, "product-item"
        )
        return [Product(element) for element in product_elements]
```

```
    def get_product_by_name(self, name):
```

```
        products = self.get_products()
        for product in products:
            if product.get_name() == name:
                return product
        return None
```

```
# テスト
```

```
def test_product_prices():
```

```
    products_page = ProductsPage(driver)
    products_page.driver.get("https://example.com/products")

    backpack = products_page.get_product_by_name("Backpack")
```



```
assert backpack.get_price() == 29.99

print("✅ Backpackの価格は正しいです")
```

図解:



⚠️ 7. よくあるエラーと対処法

エラー1: NoSuchElementException

エラーメッセージ:

```
selenium.common.exceptions.NoSuchElementException:
Message: no such element: Unable to locate element
```

原因: 要素が見つからない

対処法:

```
python
```

```
# 悪い例
element = driver.find_element(By.ID, "button")

# 良い例：待機を追加
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

wait = WebDriverWait(driver, 10)
element = wait.until(
    EC.presence_of_element_located((By.ID, "button"))
)
```

エラー2: ElementNotInteractableException

原因: 要素がクリックできない状態

対処法:

```
python

# クリック可能になるまで待つ
element = wait.until(
    EC.element_to_be_clickable((By.ID, "button"))
)
element.click()
```

エラー3: StaleElementReferenceException

原因: ページ更新後に古い要素を参照している

対処法:

```
python

def safe_click(driver, locator, max_attempts=3):
    for attempt in range(max_attempts):
        try:
            element = driver.find_element(*locator)
            element.click()
            return
        except StaleElementReferenceException:
            if attempt == max_attempts - 1:
                raise
            continue
```

8. テスト観点チェックリスト

画面・操作系

- ☐ 画面遷移が正しいか
- ☐ ボタンの活性/非活性制御が正しいか
- ☐ 入力項目の初期値が正しいか
- ☐ エラーメッセージが表示されるか
- ☐ ローディング表示が出るか

データ取得・表示系

- ☐ サーバリクエストが1回だけか
- ☐ 取得結果が設計通りか
- ☐ 0件の場合もエラーにならないか
- ☐ 表示項目の編集（日付、金額など）が正しいか

DB更新系

- ☐ 登録・更新・削除が正しく行われるか
 - ☐ トランザクションが正しく動作するか
 - ☐ エラー時にロールバックされるか
-

9. まとめ

新人エンジニアが押さえるべき5つのポイント

1. HTML構造を理解する

- Webページは階層構造になっている
- 親・子・孫の関係を意識する

2. 基本の要素取得をマスターする

- まずID、NAME、CLASS_NAMEから
- 慣れたらXPathやCSSセレクタに挑戦

3. 待機処理を必ず入れる

- WebDriverWaitで要素が表示されるまで待つ
- 安定したテストを書くための必須技術

4. ページオブジェクトパターンを活用

- コードの再利用性が高まる
- メンテナンスが楽になる

5. エラーハンドリングを丁寧に

- try-finallyで必ずブラウザを閉じる

- エラーメッセージをよく読む



作成日: 2025年9月30日



対象: 新人エンジニア



タグ: Selenium, テスト自動化, Python, WebDriver, 図解