Selenlum

Junit

Selenlum

Junit

1. ドライバの利用と画面キャプチャの方法

- テスト処理方法
- 初期処理などの記載
- Junitとseleniumを活用

```
    private WebDriver driver;
    private static String baseUrl;
    private ScreenShot screenShot;

    /**
     * クラスで1回の起動
     */
    @BeforeClass
    public static void Setup() {
        System.out.println("BeforeClass");
        System.setProperty("webdriver.chrome.driver", "./driver/chromedriver.exe");

    }

    /**
     * テスト毎の起動
     */
    @Before
    public void testBefore() {
        System.out.println("Before");
        baseUrl = "https://www.yahoo.co.jp/";
        driver = new ChromeDriver();
        screenShot = new ScreenShot();
    }

    /**
     * テストごとの後処理
     */
    @After
    public void testAfter() {
        System.out.println("After");
        driver.quit();
        screenShot = null;
    }

    /**
     * クラスの終了処理
     */
    @AfterClass
    public static void testAfterClass() {
        System.out.println("AfterClass");
    }
```


1. ドライバの利用と画面キャプチャの方法

- テスト部分
- ScreenShotは実ソースを参照

```
    @Test
    public void testApp4() {
        driver.get("https://www.post.japanpost.jp/index.html");
        WebElement element = driver.findElement(By.xpath("//*[@id=¥"select¥"]"));
        Select sel = new Select(element);
        sel.selectByIndex(12);

        try {
            screenShot.takeScreenShot(driver);
        } catch (InterruptedException e1) {
            // TODO 自動生成された catch ブロック
            e1.printStackTrace();
        } catch (IOException e1) {
            // TODO 自動生成された catch ブロック
            e1.printStackTrace();
        }

        try {
            Thread.sleep(5000);
        } catch (Exception e) {

        }

    }
```

xpathで必要とするデータを取得する

ドライバを渡すことによって入力された画面をキャプチャする

Selenlum

基本的なコマンド

5

1.要素を特定する方法

- findElementに指定する引数によって操作対象のHTML要素を特定します。

ストラテジー	ロケータ	説明
idによる特定	By.id	
CSSセレクタによる特定	By.cssSelector	
nameによる特定	By.name	要素のname属性で特定
タグ名による特定	By.tagName	要素のHTMLタグ名で特定
CLASSNAMEによる特定	By.className	要素のclass属性で特定
リンクテキストによる特定	By.linkText	リンクテキストで特定
リンクテキスト(部分一致)による特定	By.partialLinkText	リンクテキスト(部分一致)で特定
XPathによる特定	By.xpath	
idもしくはnameによる特定	ByIdOrName	id属性もしくはname属性で特定

6

1.要素を特定する方法

- 見本Htmlソース

```
<html>
 <body>
  <a href="...">セレニウム確認用リンク</a>
  <form id="testForm">
   <input class="required" name="username" type="text" />
   <input class="required passfield" name="password"
type="password" />
   <input id="action" name="loginbutton" type="submit"
value="Login" />
   <input id="action" name="clearbutton" type="button"
value="Clear" />
  </form>
 </body>
<html>
```


1.要素を特定する方法

idによる特定	driver.findElement(By.id("testForm")).click();

CSSセレクタによる特定	//タグ名="form"を検索
	driver.findElement(By.cssSelector("form")).click();
	//クラス名="login"を検索
	driver.findElement(By.cssSelector(".login")).click();
	//id名="form"を検索
	driver.findElement(By.cssSelector("#testForm")).click();
	//タグ名="input"、クラス名="login"を検索
	driver.findElement(By.cssSelector("input.login")).click();
	//タグ名="form"、id名="testForm"を検索
	driver.findElement(By.cssSelector("form#testForm")).click();

nameによる特定	//name="loginbutton"を検索
	driver.findElement(By.name("loginbutton")).click();

1.要素を特定する方法

タグ名による特定	//タグ名="form"を検索
	driver.findElement(By.tagName("form")).click();

CLASSNAMEによる特定	//class名="clear"を検索
	driver.findElement(By.className("clear")).click();

リンクテキストによる特定	//リンクテキストがセレニウム確認用リンクのリンクを検索
	driver.findElement(By.linkText("セレニウム確認用リンク")).click();
	//リンクテキストがセレニウムを含むリンクを検索
	driver.findElement(By.partialLinkText("セレニウム")).click();

XPathによる特定	//全ての要素からid=testFormを検索し、その配下のinputタグの2番目をクリック
	driver.findElement(By.xpath("//*[@id='testForm']/input[2]")).click();

9

1.要素を特定する方法

idもしくはnameによる特定	driver.findElement(new ByIdOrName("testForm")).click();

10

2. Page Object Design Patternの利用方法

- seleniumのテスト用サイトを利用させていただきます。
  - http://example.selenium.jp/reserveApp/

2. Page Object Design Patternの利用方法

- テスト対象ページ毎のクラスとテストケースクラスを分離し、変更に強いテストを作成するための仕組みとなります。
- テスト用サイトの「入力画面」、「確認画面」、「エラー画面」を対象にテストを作成する場合は、
  「入力画面」のページクラス(PageObject)、「入力画面」のテストクラス
  「確認画面」のページクラス、「確認画面」のテストクラス
  「エラー画面」のページクラス、「エラー画面」のテストクラス
  の６クラスを作成します。
- 厳密に言うと「確認画面」と「エラー画面」は同じURLですので同じクラスとしてもOKですが、今回は明示的に異なるページクラスにマッピングしてみます。

2. Page Object Design Patternの利用方法

- ページクラスのインスタンス化とWebElementの特定を行うために以下の２つの仕組みが提供されいます。
- PageFactory(ページファクトリー)
  - ページクラスの作成時にはPageFactoryのinitElementsメソッドを利用します。
  - この操作を行うことによってSeleniumが提供しているページオブジェクトパターンが利用できるようになります。
    - HogePage pageObject = PageFactory.initElements(driver, HogePage.class);

2. Page Object Design Patternの利用方法

- FindByアノテーション
  - org.openqa.selenium.WebElement型のクラスメンバに@FindByアノテーションを付与することにより、クラスメンバにアクセスした時にSeleniumがアノテーションで指定されたHTML要素をインジェクションしてくれます。

```
public class HogePage {
   @FindBy(name="hoge")
   private WebElement hogeElement;
}
```

  - 上記の例ではhogeElementにアクセスすると、HTMLのname属性がhogeの要素が返却されます。

3.非同期処理を考慮したテストの作成方法

- 待つ方法としては
  - ① 単純にスリープを行う。
  - ② findElementのタイムアウト時間を調整する。
  - ③ 任意の条件を満たすまで待つ。

15

3.非同期処理を考慮したテストの作成方法

- findElementのタイムアウト時間を調整する。
  - findElement、findElementsに共通な設定となります。
  - なお、ページリクエスト時のタイムアウトは別な設定が存在しており、この設定とは異なります。

- ページリクエスト時のタイムアウト設定は以下のように設定できます。
  下記の例では60秒にセットしています。

```
driver.manage().timeouts().pageLoadTimeout(60,TimeUnit.SECONDS);
```

16

3.非同期処理を考慮したテストの作成方法

- 任意の条件を満たすまで待つ。
  - findElement相当な処理

```
//タイムアウトを60秒に設定したWaitを作成
Wait<WebDriver> wait = new WebDriverWait(driver, 60);
//ボタンクリック等の処理を記述
//id="hoge"の要素を検索、60秒後でも要素が存在しない場合は例外がスローされる。
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("hoge")));
```

  - 要素が非表示になるまで待つ処理

```
//タイムアウトを60秒に設定したWaitを作成
Wait<WebDriver> wait = new WebDriverWait(driver, 60);
//非表示になる要素を取得
WebElement element = driver.findElement(By.id("hoge"));
//ボタンクリック等の処理を記述
//表示になるまでウエイト
wait.until(ExpectedConditions.stalenessOf(element));
```

17

4 よく利用する基本的なコマンド

- WebDriverのよく利用するメソッドは以下の通りです。

メソッド名	説明
get	引数で指定したURLを開く
findElement	org.openqa.selenium.WebElementを返却
findElements	org.openqa.selenium.WebElementの配列を返却
quit	テストで起動したブラウザを終了

18

4 よく利用する基本的なコマンド

- org.openqa.selenium.WebElementのよく利用するメソッド
  - click：要素をクリックするメソッドです。
  - sendKeys：テキストボックスに文字列を入力するときに利用するメソッドです。
  - clear：テキストボックスに入力されている内容をクリアします。
  - isSelected:初期画面表示時のチェックボックス、セレクトのオプション、ラジオボタンのデフォルト状態を検証する時に利用します。

```
//id="chckboxId"の要素がチェックされているかチェック
if (driver.findElement(By.id("chckboxId")).isSelected()) {
  //checked
}else {
  //not checked
}else {
  //not checked
}
```

19

4 よく利用する基本的なコマンド

- getText:要素のインナーテキストを取得します。テキストの前後の空白は除去されます。

```
//aタグのリンクテキストを取得
String linkText = driver.findElement(By.tagName("a")).getText();
```

- getCssValue:CSSプロパティを取得するメソッドです。

```
//name="password"のCSSプロパティのwidthを取得
driver.findElement(By.name("password")).getCssValue("width");
```

20

4 よく利用する基本的なコマンド

- セレクトのオプションの選択
  - 以下のような画面を例に説明させていただきます。

```
<select name="jobSelect" id="jobSelectId" name="jobSelectName">
<option value="10" selected="selected">会社員</option>
<option value="20">自営業</option>
<option value="30">その他</option>
</select>
```

  - 以降の例は全て「自営業」のoptionを選択する例となります。//CSSセレクタでoptionを特定しクリック

WebElementでoptionを検索して選択する方法

```
//CSSセレクタでoptionを特定しクリック
driver.findElement(By.cssSelector("#jobSelectId > option:nth-child(2)")).click();
```

org.openqa.selenium.support.ui.Selectを利用する方法

```
//id="jobSelectId"の要素を検索しそのオブジェクトを引数にしてSelectを作成
Select select = new Select(driver.findElement(By.id("jobSelectId")));
//テキストが"自営業"のoptionを選択
select.selectByVisibleText("自営業");
//valueが"20"のoptionを選択
select.selectByValue("20");
//1番目のoptionを選択
select.selectByIndex(1);
```


4 よく利用する基本的なコマンド

- ラジオボタンの選択

```
<label><input type="radio" name="sexName" value="man">man</label>
<label><input type="radio" name="sexName" value="woman">woman</label>
```

```
// 上記画面の「woman」を選択する実装例は以下の通りです。
driver.findElement(By.xpath("(//input[@name='sexName'])[2]")).click();
```

4 よく利用する基本的なコマンド

- チェックボックスの選択

```
//id="checkboxId"の要素を検索
WebElement checkBox = driver.findElement(By.id("checkboxId"));
//チェックされていなければ
if ( !checkBox.isSelected() )
{
 //クリックしてチェック
 checkBox.click();
}
```

23

Selenlum

Selenium WebDriverのブラウザ自動テストを実践する

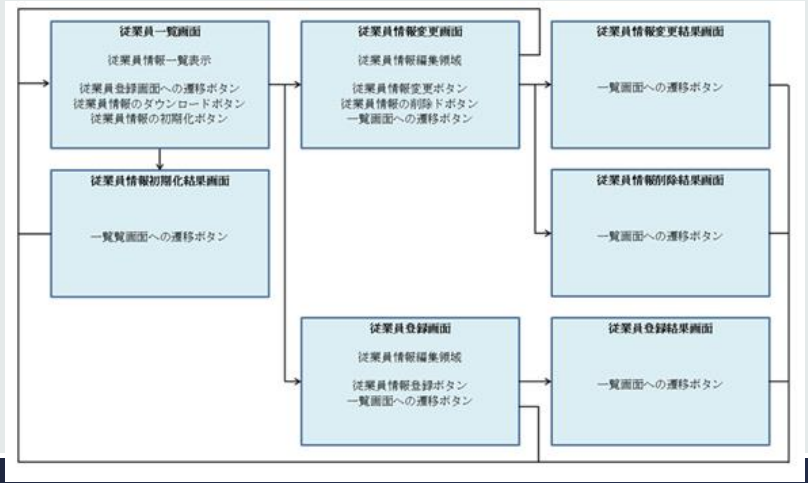
24

1. 画面機能一覧

画面名	機能
従業員一覧	従業員情報一覧表示
	従業員情報ファイルのダウンロード
	従業員情報の初期化
	従業員情報変更画面への画面遷移
	従業員情報登録画面への画面遷移
従業員情報変更	従業員情報の変更
	従業員情報一覧画面への遷移
従業員情報変更結果	従業員情報変更結果の表示
	従業員情報一覧画面への遷移
従業員情報削除結果	従業員情報削除結果の表示
	従業員情報一覧画面への遷移
従業員情報登録	従業員情報の初期化
	従業員情報一覧画面への遷移
従業員情報登録結果	従業員情報登録結果の表示
	従業員情報一覧画面への遷移
従業員情報初期化結果	従業員情報初期化結果の表示
	従業員情報一覧画面への遷移

25

2. 遷移図



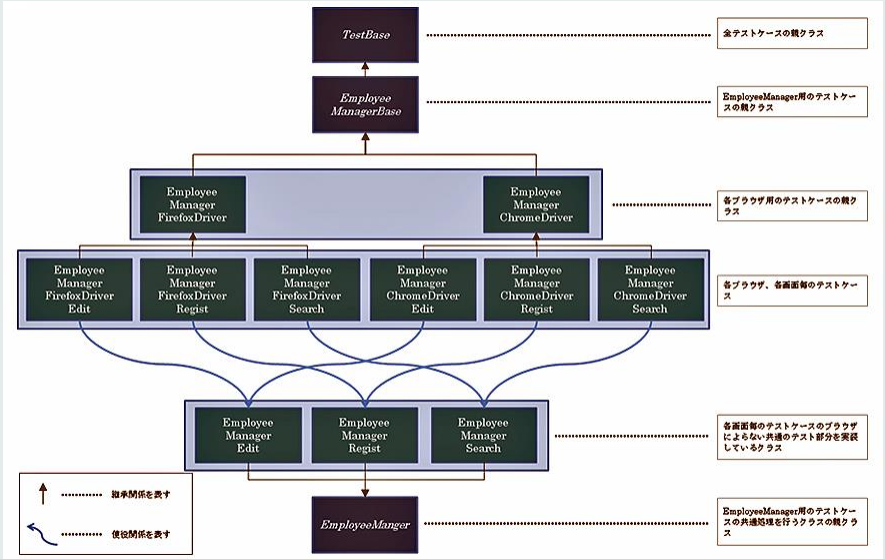
26

3. テストケースの作成

No	テスト対象画面	テスト内容
1	従業員一覧	一覧情報の従業員のIDをクリックすると、対象の従業員情報の編集画面に遷移することを確認
2	従業員一覧、従業員情報変更、従業員情報変更結果	従業員情報変更画面→変更結果画面→一覧画面という順序で画面遷移後、Webブラウザの「戻る」「進む」処理を行った場合に、正常に画面遷移がされることを確認
3	従業員一覧	従業員情報ダウンロードボタンをクリックすると、一覧表示されている従業員の情報がCSVファイルでダウンロードされることを確認
4	従業員情報登録	従業員情報登録時のチェック処理でエラーとならない情報を登録すると、正常に登録されることを確認
5	従業員情報登録	従業員情報登録時のチェック処理でエラーとなる情報を登録すると、エラーメッセージが表示され、画面遷移せず再入力を求められることを確認
6	従業員情報変更	従業員情報変更時のチェック処理でエラーとならない情報を登録すると、正常に変更されることを確認
7	従業員情報変更	従業員情報変更時のチェック処理でエラーとなる情報を登録すると、エラーメッセージが表示され、画面遷移せず再入力を求められることを確認
8	従業員情報変更	従業員情報の削除処理を行うと、正常に従業員情報が削除されることを確認
9	従業員情報変更	従業員情報変更時に表示される確認メッセージが正常に表示されることを確認
10	従業員情報変更	従業員情報削除時に表示される確認メッセージが正常に表示されることを確認

27

3. クラス設計



28

3. クラス設計

- 下記6クラスをJUnitで実行すると、テストが実行される形式になっています。
  - EmployeeManagerFirefoxDriverEdit
  - EmployeeManagerFirefoxDriverRegist
  - EmployeeManagerFirefoxDriverSearch
  - EmployeeManagerChromeDriverEdit
  - EmployeeManagerChromeDriverRegist
  - EmployeeManagerChromeDriverSearch
- 「EmployeeManagerXxxDriverEdit」という名前のクラスはWebブラウザXxx用、かつ従業員情報編集画面用のテストクラス、「EmployeeManagerXxxDriverRegist」という名前のクラスはWebブラウザXxx用、かつ従業員登録画面用のテストクラス、「EmployeeManagerXxxDriverSearch」という名前のクラスはWebブラウザXxx用、かつ従業員一覧画面用のテストクラスとなっています。

29

4. テスト手順

① Webブラウザの初期化処理
org.openqa.selenium.WebDriverの実体オブジェクトを作成し、メソッド「get(String url)」を使用し、テスト対象のURLを読み込みます。

② 対象要素の取得
「Webブラウザの初期化処理」で読み込んだテスト対象のURLに対し、WebDriverのメソッド「findElement(By by)またはfindElements(By by)」を使用し、操作対象の要素を取得します。

③ 対象要素に対する操作
「対象要素の取得」で取得した要素に対して操作を行います（入力用の要素の場合、値を入力します。ボタンなどイベントトリガ用の部品の場合、クリック処理を行います）。

④ 操作結果と想定値との比較
「対象要素に対する操作」で操作を行った結果と、テストで想定される結果とを比較します。APIが用意されており、操作結果を簡単に取得できる場合はJUnitなどでテスト結果を自動で出力します。
操作結果が簡単に取得できない場合は、画面のキャプチャーを撮り、後で目視確認します。

⑤ 終処理
テスト操作対象のWebブラウザウィンドウを全て閉じます。テスト実行処理前の状態に戻す場合は、テスト操作前の状態に戻す処理を行い、その後、テスト操作対象のWebブラウザウィンドウを全て閉じます。

30

4. テスト手順
EmployeeManagerEdit#editInvalidTest_21char_20char()

① 従業員一覧画面を開き、従業員一覧の上から2番目の従業員のIDをクリック

② 従業員情報変更時にエラーとなる情報を入力し、変更ボタンを押す

③ 変更ボタン押下後、確認ダイアログが出るので、確認ダイアログに対して、OKを押す

④ 従業員情報変更画面にエラーメッセージが出るので、エラーメッセージをアサート。
画面キャプチャーも取得しておく

⑤ 従業員情報にエラーとならない値を入力し、変更処理を行う。変更後、変更完了のメッセージをアサートし、画面キャプチャーも取得しておく

⑥ 変更処理終了後従業員一覧画面に遷移

31

5. テストコードの保守性を高めるPAGE OBJECTSパターン

- テストコードにxpathを埋め込んでしまうと、テストするページのレイアウトに修正が入った場合、そのページをテストするテストコード全てに修正が必要となり、影響範囲が大きくなってしまいます。
- そこで、テストコードの保守性を高めるために、Page Objectsパターンという設計が推奨されています。
- Page Objectsパターンでは、テスト対象の画面に対応するPageクラスを作成します。Pageクラスでは、「名前を入力する」「入力されたデータを新規登録する」といった、その画面の機能に対応した処理をpublicなメソッドで提供するようにします。要素の取得に使うxpathなどはクラスの外に公開しないようにし、画面のレイアウトなどの変更がPageクラスの外に影響しないようにします。
- また、メソッドの返り値はPageクラスとし、メソッドチェインで処理を記述することができるようにします。画面遷移の際は遷移先の画面のPageクラスを返します。

32

5. テストコードの保守性を高めるPAGE OBJECTSパターン

- Page Objectsパターンを用いることで、格段にテストコードが読みやすくなります
- また、もしテスト対象の画面の構成が変わったとしても、対応するPageクラスのみの修正で対応できます。