Selenium WebDriverでWebアプリのテストが変わる(後編):

Selenium WebDriverのブラウザ自動テストを実践する

https://www.atmarkit.co.jp/ait/articles/1211/07/news009.html

Chrome、Firefox、Internet Explorer、Opera、Android、iOSといったブラウザに対応し、Java、C#、Python、Rubyなどが使えるWebテスト自動化ツールを簡単に紹介する連載。後編は、サンプルを例に、テストケースの作成、クラス設計、テスト手順の整理、コード化などを実践していく【2017年の情報に合うように更新】。

2017年07月20日 05時00分 更新

[後藤正規, ビーブレイクシステムズ]

Selenium WebDriverのテスト例として従業員の情報を管理するサンプルを使う

前編の「iPhone／Android含むブラウザ自動テストの最終兵器Selenium WebDriverとは」では、Selenium WebDriverの概要とサンプルを動かす上での前提環境と実装方法の概要を説明しました。ここからは、サンプルアプリケーションとテストケースについて説明します。

本稿でのテスト対象とするWebサイトとして、従業員の情報を管理するサンプルWebサイトを用意しました。Webサイトのサンプルは下記のリンクからダウンロードできます。ファイルは通常のwarファイルなので、実行環境のWebサーバに展開すると使用可能になります。

- warファイルへのリンク

また、サンプルコードのWebサイトをEclipseにプロジェクトとしてインポートする場合は、下記ファイルを使用します。

- プロジェクトのzipファイルへのリンク

今回の主な内容

サンプルWebサイトの概要

ダウンロードしたテスト対象のWebサイトについて簡単に説明します。Webサイトにアクセスするためにはwebブラウザで下記のURLを開いてください(※異なるサーバおよびポートを使用する場合は、適宜読み替えてください)。

- http://localhost:8080/EmployeeManager/index.jsp

Webアプリケーション内で保管されているデータは、アプリケーションの起動のたびに初期化されます(永続化されません)。アプリケーション起動中にデータを初期化する場合は、「http://localhost:8080/EmployeeManager/index.jsp」内の「従業員情報初期化ボタン」をクリックするか、Webブラウザに以下のURLを入力します。

- http://localhost:8080/EmployeeManager/ initEmployee.do

社員情報を0件でデータを初期化する場合は、Webブラウザで下記のURLを入力してください。

- http://localhost:8080/EmployeeManager/initEmployee.do?initdata=0

初期化すると、初期化結果画面が表示されます。

Webサイトは、表1の画面、機能を持ち、図1の画面遷移を行うアプリケーションとなっています。

表1 画面と機能概要

	画面名	機能

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

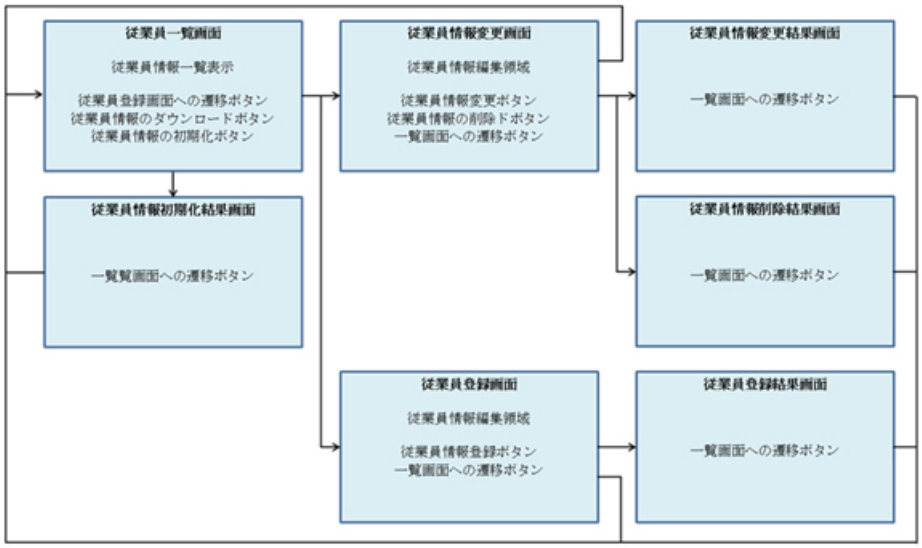

図1　画面遷移概要

テストケースの作成

　次に、前節のサンプルWebサイトにおいて機能ごとに行うテストを整理します。主に表2のようなテストケースが考えられるでしょう。

表2 テストケース一覧

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

No	テスト対象画面	テスト内容
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

ここからは、前節までで紹介してきた実装方法とサンプル概要を踏まえ、テストケースの実装方法を解説していきます。

クラス設計

テストコードは、前編で紹介した通り、5言語のいずれかで記述できますが、今回のサンプルではJavaを使用しています。テストコードは下記のリンクからダウンロードしてください。

- テストコードのプロジェクトファイルへのリンク

※テストコードはEclipseのプロジェクトファイルとなっているので、Eclipseにインポートすると、JUnitによってテストを実行できる形になっています。

「seleniumWebDriver」のプロジェクトはそのままではクラスパスのエラーが発生するので、「lib」フォルダに「selenium-java-3.x.x.zip」（http://www.seleniumhq.org/download/から取得）を解凍した中身の「client-combined-3.x.x-nodeps.jar」と「libs」フォルダ以下に含まれるjarファイルを全てコピーし、それらをビルドパスに追加してから利用してください。

また、テスト対象のブラウザに対応するドライバをプロジェクト直下に配置してください。

クラスの関係と大まかな役割に関しては図2を参照してください。

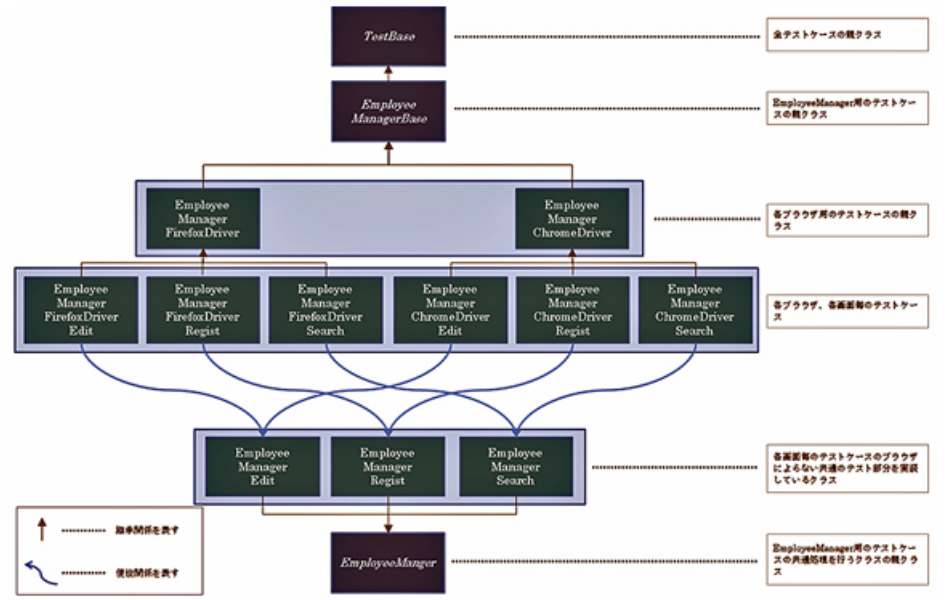

図2　テストケースのクラス図と関係

今回のサンプルでは、Webブラウザに依存するテスト処理とWebブラウザに依存しないテスト処理を別クラスで実装する形にしてあります。これはSeleniumの設計指針が、テストケースをWebブラウザによらず実装できるように、APIと各Webブラウザのdriverを作成していく設計になっているため容易に実現できます。

クラス名と役割が少々分かりにくくなってしまっていますが、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。

テスト手順の整理

一般的にWebアプリケーションのテストでは、「Webページ内の要素に対して、まず操作を行い、そして操作結果を想定値と比較する」形をとります。そのため、テストクラスの実装は下記のような処理の手順を踏むことになります。

【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

「【1】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

「【2】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

「【3】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。

テスト手順のコード化

ここからは、この手順を踏んだ形でテストケースの中身を抜粋して見ていきましょう。

サンプルコード1は「表2　テストケース一覧」のテストケース6とテストケース7を合わせたテストケースを実装したものです。

```
// ユーザ名21文字でのエラーを発生させた後、20文字で入力して変更を成功させる
private void editInvalidTest_21char_20char() {
    driver.get(initialURL);
    // 【1】 編集画面に遷移
    driver.findElement(By.xpath("//table[@id='emptable']/tbody/tr[2]/td/a")).click();
    // 【2】始まり
    // ユーザー名など入力
    driver.findElement(By.xpath("//table[@id='edittable']/tbody/tr[2]/td[2]/input")).clear();
    driver.findElement(By.xpath("//table[@id='edittable']/tbody/tr[2]/td[2]/input"))
    .sendKeys("テスト変更_XX#?1234567890＋");
    new Select(driver.findElement(By.xpath("//table[@id='edittable']/tbody/tr[3]/td[2]/select")))
    .selectByVisibleText("開発部1部");
    driver.findElement(By.xpath("//table[@id='edittable']/tbody/tr[4]/td[2]/input[2]")).click();
    driver.findElement(By.xpath("//table[@id='edittable']/tbody/tr[5]/td[2]/input")).click();
    // 変更ボタンクリック
    driver.findElement(By.xpath("//div[@id='editpanel']/form/input[6]")).click();
    // 【2】終わり
    // 【3】 確認ダイログでOKを押す
 driver.switchTo().alert().accept();
 // 【4】始まり
 // エラーメッセージのアサート
 assertThat(driver.findElement(By.xpath("//div[@id='editpanel']//ul/li")).getText(), is(containsString("社員名は20文字以内で入力してください。")));
 // 画面キャプチャーも撮っておく
 String filePath_20over = CaputureUtils.getFilePath(getClass().getName(), browserName, "editInvalidTest_21char_20char_1st");
 CaputureUtils.getScreenshot((TakesScreenshot) driver, filePath_20over);
 Logger logger = Logger.getLogger(getClass());
 logger.info(filePath_20over + "のキャプチャー内エラー文言が「・社員名は20文字以内で入力してください。 」となっていることを確認してください。");
 // 【4】終わり
 // 【5】始まり
 // 20文字で入力
 driver.findElement(By.xpath("//table[@id='edittable']/tbody/tr[2]/td[2]/input")).clear();
 driver.findElement(By.xpath("//table[@id='edittable']/tbody/tr[2]/td[2]/input"))
 .sendKeys("テスト変更_X#?1234567890＋");
 // 変更ボタンクリック
 driver.findElement(By.xpath("//div[@id='editpanel']/form/input[6]")).click();
 // 確認ダイログでOKを押す
 driver.switchTo().alert().accept();
 // メッセージのアサート
 assertThat(driver.findElement(By.xpath("//center")).getText(), is(containsString("テスト変更_X#?1234567890＋の情報を変更しました。")));
 // 画面キャプチャーも撮っておく
 String filePath_20 = CaputureUtils.getFilePath(getClass().getName(), browserName, "editInvalidTest_21char_20char_2nd");
 CaputureUtils.getScreenshot((TakesScreenshot) driver, filePath_20);
 logger.info(filePath_20 + "のキャプチャー内文言が「テスト変更_X#?1234567890＋の情報を変更しました。 」となっていることを確認してください。");
 // 【5】終わり
 // 【6】一覧画面に戻る
 driver.findElement(By.linkText("検索")).click();
}
```

サンプルコード1　EmployeeManagerEdit#editInvalidTest_21char_20char()

それぞれの処理は下記のようになっています。

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

サンプルコード1ではキャプチャーを保存する際に、「CaputureUtils」というクラスを使用していますが、これは独自で作成したユーティリティークラスです。

処理内容はサンプルコード2を参照してください。下記【1】のようにしてWebブラウザのdriver（TakesScreenshot）を使用することで、キャプチャーを保存できます。

```
public class CaputureUtils {
   public static void getScreenshot(TakesScreenshot driver, String filePath) { // 【1】
      File scrFile = driver.getScreenshotAs(OutputType.FILE);
      try {
         FileUtils.copyFile(scrFile, new File(filePath));
      } catch (IOException e) {
         Logger logger = Logger.getLogger(CaputureUtils.class);
         logger.warn(e);
      }
   }

   public static String getFilePath(String... params) {
      StringBuilder builder = new StringBuilder();
      builder.append("c:¥¥tmp¥¥");
      builder.append(getYYYYMMDD());
      builder.append("¥¥screenshot¥¥");
      builder.append(getYYYYMMDD());
      for (String param : params) {
         builder.append("_");
         builder.append(param);
      }
      builder.append(".png");
      return builder.toString();
   }
（中略）
}
```

サンプルコード2　CaputureUtils（一部）

　サンプルコード1は一切Webブラウザに依存しない部分についてのみ記述しています。Webブラウザごとに異なる実装をする必要がある場合には、それぞれのWebブラウザ用のテストケースでテストコードを実装することになります。

　例えば、Firefox用の従業員検索画面テストクラスのコード（下記サンプルコード3）は、ファイルダウンロードに関するテストは独自で実装しています。

　またサンプルコード3に関しては、ファイルダウンロードに関するテスト以外、「共通のテスト実行用の処理のみ」のテストが可能となっていることに注目してください。

　Selenium WebDriverでは、テストコードにおいてWebブラウザごとに異なる記述をする場面が少なくなるようにAPIが設計されていることが実感できる例といえるでしょう。

```
package employeeManager.firefox.search;

import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.firefox.FirefoxDriver;

import employeeManager.common.search.EmployeeManagerSearch;
import employeeManager.firefox.EmployeeManagerFirefoxDriver;

/**
 *    Firefoxの従業員一覧画面テスト
 */
public class EmployeeManagerFirefoxSearch extends EmployeeManagerFirefoxDriver {

    private EmployeeManagerSearch employeeManagerSearch;

    @Override
    protected void setupProfile() {
    }

    @Override
    public void preTest() {
        super.preTest();
        employeeManagerSearch = new EmployeeManagerSearch("Firefox", getDriver(), getinitialURL());
    }

    // 遷移テスト
    @Test
    public void transitionTest() throws InterruptedException {
        employeeManagerSearch.transitionTest();
    }
    // ダウンロードテスト
    @Test
    public void downloadTest() throws InterruptedException {
        if (driver != null) {
            driver.quit();
        }
// NOTE Firefoxはダウンロードリンクをクリックするだけだと、そのまま開くか、どこにダウンロードするかのダイアログが開かれてしまう
// このダイアログを回避するため、profileの設定を行い直接ダウンロードできるようにする

        // ダウンロードするファイルの保存先フォルダを指定　0:デスクトップ 1：ダウンロードフォルダ 2:ダウンロードに指定された最後のフォルダ
        profile.setPreference("browser.download.folderList", 0);
        // ダウンロードするファイルの保存先フォルダが指定してあればそれを使う
        profile.setPreference("browser.download.useDownloadDir", true);
        // 指定したmimeタイプは有無を言わさずダウンロードする
        profile.setPreference("browser.helperApps.neverAsk.saveToDisk", "application/octet-stream;");
        // 作成したプロファイルでFirefox（のドライバ）を起動する
        driver = new FirefoxDriver(profile);
        driver.get(getinitialURL());
        driver.findElement(By.xpath("//div[@id='downloadpanel']/form/input")).click();
        Thread.sleep(3000);
    }

}
```

サンプルコード3　Firefox用の従業員検索画面のテストクラス

非同期処理をテストする

　今までテストコードを書いてきた機能は全て画面遷移を含む同期的なテストでした。しかし、現在Webアプリでは非同期処理を用いて画面遷移せずにページの内容を書き換えることが多くなってきています。

　非同期処理で値を書き変えるような機能に対して、今までのコードでは正しくテストすることができません。非同期処理が実行され、値が変更される前に要素の値が検証されてしまうからです。

　非同期処理をテストするためには、WebDriverWaitというクラスを用いて、明示的に非同期処理が完了するまで待機する必要があります。

　それでは、具体的に「id=button1であるボタンをクリックした後、id=text1であるテキスト部品がhogeとなる」処理について、同期処理と非同期処理のテストコードの違いを見比べてみましょう。

```
WebElement button1Element= driver.findElement(By.id("button1"));
button1Element.click();
WebElement text1Element = driver.findElement(By.id("text1"));
assertThat(text1Element.getAttribute("value"), is("hoge"));
```

同期処理の場合

```
WebElement button1Element= driver.findElement(By.id("button1"));
button1Element.click();
final WebElement text1Element = driver.findElement(By.id("text1"));
// 非同期処理に対応した明示的な待機処理
// WebDriverWaitのコンストラクタの第2引数はタイムアウト秒数
// untilメソッドに待機する条件を指定。
(new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
            // この条件が満たされたら処理が続行される
      return text1Element.getAttribute("value").equals("hoge");
   }
});
assertThat(text1Element.getAttribute("value"), is("hoge"));
```

非同期処理の場合

WebDriverWaitを間に差し込む以外は同期処理と非同期処理のテストコードに違いはありません。

また、「要素の属性が変更した」「要素が表示/非表示となった」といった典型的な判定であれば、ExpectedConditionsというクラスが判定用のメソッドを提供しています。

先ほどの例であれば、ExpectedConditionsのattributeContainsメソッドを使うことで、以下のように匿名クラスを使わずに記述できます。

```
(new WebDriverWait(driver, 10)).until(ExpectedConditions.attributeContains(text1Element, "value", "hoge"));
```

ExpectedConditonsに用意されているメソッドは他には以下のようなものがあります。また、第1引数にWebElementを指定しているものは、Byを指定することも可能です。

	メソッド	処理概要
	attributeContains(WebElement element, String attribute, String value)	要素の属性が指定した値であるかどうか
	elementToBeClickable(WebElement element)	要素がvisibleかつenableであるか
	visibilityOf(WebElement element)	要素がvisibleか
	invisibilityOf(WebElement element)	要素がvisibleでないか
	and(ExpectedCondition<?>... conditions)	複数の条件を満たすか
	or(ExpectedCondition<?>... conditions)	複数の条件のいずれかを満たすか
	not(ExpectedCondition<?> condition)	指定の条件を満たさないか

なおプログラム中に非同期処理が大量にある場合、上記の明示的な待機がテストコードに多発し、可読性が損なわれる可能性が高くなります。そのようなケースでは、「Selenide」というフレームワークを使うことを検討した方がいいでしょう。Selenideを使うと、同期処理、非同期処理を意識せずに要素のアサーションが行えるようになります。

TIPS「テストコードの保守性を高めるPAGE OBJECTSパターン」

ここまで記載したテストコードでは、以下のようにテストメソッドに直接xpathで要素を取得するコードを記述していました。

```
private void editInvalidTest_21char_20char() {
driver.get(initialURL);
      driver.findElement(By.xpath("//table[@id='emptable']/tbody/tr[2]/td/a")).click();
```

しかし、このようにテストコードに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。

前ページに記載したeditInvalidTest_21char_20charテストメソッドにPage Objectsパターンを適用したものを見てみましょう。

```
/**
 * 従業員一覧ページ用PageObject
 */
public class EmployeeSearchPage {
    private WebDriver driver;

    public EmployeeSearchPage(WebDriver driver) {
        this.driver = driver;
    }

    /**
     * n番目に表示されている従業員の編集ページに遷移する
     * @param n 編集対象の行数(1開始)
     * @return
     */
    public EmployeeEditPage editEmployee(int n){
        getEmployeeEditLinkElement(n).click();
        return new EmployeeEditPage(driver);
    }

    private WebElement getEmployeeEditLinkElement(int n){
        return driver.findElement(By.xpath("//table[@id='emptable']/tbody/tr[" + n + "]/td/a"));
    }
}
```

従業員一覧ページ用のPageクラス

```
/**
 * 従業員情報変更ページのPageObject
 */
public class EmployeeEditPage {
    private WebDriver driver;

    private By employeeIdLabel = By.xpath("//table[@id='edittable']/tbody/tr[1]/td[2]");
    private By nameTextBox = By.xpath("//table[@id='edittable']/tbody/tr[2]/td[2]/input");
    private By orgCombo = By.xpath("//table[@id='edittable']/tbody/tr[3]/td[2]/select");
    private By manRadio = By.xpath("//table[@id='edittable']/tbody/tr[4]/td[2]/input[1]");
    private By womanRadio = By.xpath("//table[@id='edittable']/tbody/tr[4]/td[2]/input[2]");
    private By invalidFlagCheckBox = By.xpath("//table[@id='edittable']/tbody/tr[5]/td[2]/input");
    private By updatebutton = By.xpath("//div[@id='editpanel']/form/input[6]");
    private By errorLabel = By.xpath("//div[@id='editpanel']//ul/li");

    public EmployeeEditPage(WebDriver driver) {
        this.driver = driver;
    }

    public String getEmployeeId(){
        return driver.findElement(employeeIdLabel).getText();
    }

    public EmployeeEditPage setName(String name){
        driver.findElement(nameTextBox).clear();
        driver.findElement(nameTextBox).sendKeys(name);
        return this;
    }

    public EmployeeEditPage selectOrg(String text){
        new Select(driver.findElement(orgCombo)).selectByVisibleText(text);
        return this;
    }

    public EmployeeEditPage selectMan(){
        driver.findElement(manRadio).click();
        return this;
    }

    public EmployeeEditPage selectWoman(){
        driver.findElement(womanRadio).click();
        return this;
    }

    public EmployeeEditPage setInvalidFlag(boolean invalidFlag){
        WebElement invalidFlagElem = driver.findElement(invalidFlagCheckBox);
        boolean nowInvalidFlag = invalidFlagElem.isSelected();
        // 現在のフラグの状態が指定の状態と異なればクリックして値を変える
        if(nowInvalidFlag != invalidFlag){
            invalidFlagElem.click();
        }
        return this;
    }

    public EmployeeEditResultPage update(){
        driver.findElement(updatebutton).click();
        // 確認ダイアログのOKを押す
        driver.switchTo().alert().accept();
        return new EmployeeEditResultPage(driver);
    }

    public String getErrorMessage() {
        return driver.findElement(errorLabel).getText();
    }
}
```

従業員情報変更ページ用のPageクラス

```
    public void editInvalidTest_21char_20char() {
        driver.get(initialURL);
        EmployeeSearchPage searchPage = new EmployeeSearchPage(driver);
        EmployeeEditPage editPage = searchPage.editEmployee(2);
        editPage.setName("テスト変更_XX#?1234567890+");
        editPage.selectOrg("開発部1部");
        editPage.selectWoman();
        editPage.setInvalidFlag(true);
        editPage.update();
        assertThat(editPage.getErrorMessage(), is(containsString("社員名は20文字以内で入力してください。")));
    }
```

Pageクラスを使ったテストメソッド

どうでしょうか。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ったと思います。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。テストコードを書く際はぜひPage Objectsパターンを使ってみてください。

テスト自動化は時代の必然。Selenium WebDriverの活用を

これまでのサンプル実装から、Selenium WebDriverを用いることでWebアプリケーションに対するテストがWebブラウザに依存せず容易に実装できることが分かったと思います。

近年、業務用のWebアプリケーションをSaaSの形態で提供するなど、Webアプリケーションに対する需要がますます高まってきています。また環境面では、FirefoxやChrome、Internet Explorerなど複数のWebブラウザがシェアを争っている状況もあり、Webアプリケーションはそれぞれの Webブラウザ上において共通的かつ安定した動作が求められます。

Webアプリケーションの動作保証の手段としては、一般的には複数のWebブラウザにおけるテストの実施ということになるでしょう。

さらに、Webブラウザのアップデートごとに繰り返し、かつ、Webブラウザごとに似たようなテストを行わなければならないことを考慮すると、「テストは、なるべく自動化する」という結論に達するでしょう。

Webアプリケーションに対するテスト自動化の実行環境を作成する手段として、Webブラウザに依存する部分を最小限にし、かつ容易にテストプログラムを作成できることが可能なSelenium WebDriverは有力な選択肢として検討に値するのではないでしょうか。

■更新履歴

【2012/11/7】初版公開(後藤正規、株式会社ビーブレイクシステムズ)。

【2017/7/20】2017年の情報に合うように更新(今泉俊幸、株式会社ビーブレイクシステムズ)。

筆者紹介

今泉 俊幸(いまいずみ としゆき)

2011年から、株式会社ビーブレイクシステムズに在籍。

快適に仕事をするためには教育と自動テストこそが大事、という思いを抱き活動中

関連記事

Selenium VBAを使って自動でブラウザーを操作してスクショをExcelに張り付けてみた
システム開発におけるソフトウェアテスト(結合テスト〜システムテスト)において重要視されるエビデンス(作業記録)。前後編の2回にわたって、エビデンスとしてスクリーンショットをキャプチャし、テスト仕様書や納品書に張り付けていく作業を自動化するためのVBA/マクロのテクニックを紹介する。前編はSelenium VBAのインストール方法と使い方、スクリーンショットを自動でExcelに張り付ける方法について。

SeleniumのUIテスト自動化をiOS/Androidにもたらすappiumの基礎知識とインストール方法、基本的な使い方
本連載では、AndroidおよびiOSアプリ開発における、システムテストを自動化するツールを紹介していきます。今回は、オープンソースのモバイルテスト自動化ツール「Appium」の特徴やインストール方法、基本的な使い方を説明します。

JavaScriptテストの基礎知識と使えるフレームワーク6選
しっかりとJavaScriptの“テスト”を行うために、最近のJavaScript事情やテストを取り巻く環境、今注目のテストフレームワークを6つ紹介する