Wagby User Group

超高速開発コミュニティ Wagby分科会

Wagby Testing Framework

OUTLINE

- Wagby Testing Framework(WTF)とは
- テストクラスの作成 – 簡単なテスト
- WTFが提供する機能
- ワークフローのテスト

Wagby Testing Framework(WTF)とは

- Wagbyアプリケーションのテストに特化したFW
  - Selenideをベースとしている(SelenideはSeleniumベース)
  - ブラウザを経由したデータの入力や確認をプログラムで記述
  - E2Eテストを自動で行うことができる

テストクラスの作成

- 事前準備
  - Wagby のインストール
  - eclipse のインストール
  - eclipse上でWagbyが動作するように設定
  - Google Chrome用のWebDriver（ChromeDriver）をダウンロードし、customizeフォルダ直下に配置。

テストクラスの作成

- テストクラスを作成する
  - JUnitテストクラスとして作成
  - ソースフォルダ：customize/test/java
  - パッケージ名及びファイル名は任意
  - setUpBeforeClass()でselenide用の初期設定を記述

```
/** テストの前処理を行います。 */
@BeforeClass
public static void setUpBeforeClass() {
    Configuration.baseUrl = "http://localhost:8921/wagby";
    // 利用するブラウザ
    Configuration.browser = WebDriverRunner.CHROME;
    // Selenium(WebDriver) 用ドライバファイル
    System.setProperty("webdriver.chrome.driver",
            "customize/chromedriver.exe");
}
```

テストの実装

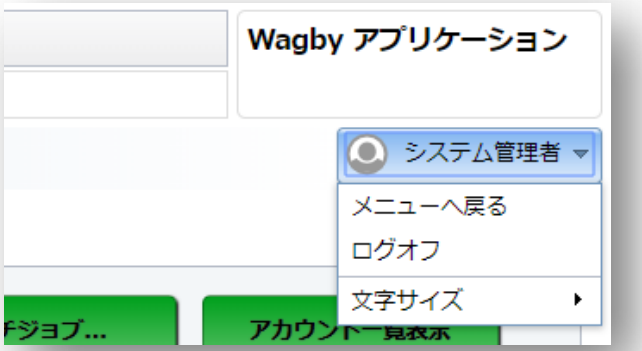
- test()メソッドにテストを記述する
  - 多用するクラスは事前に import を記述しておく

```
import static com.codeborne.selenide.CollectionCondition.*;
import static com.codeborne.selenide.Condition.*;
import static com.codeborne.selenide.Selenide.*;

import static jp.jasminesoft.jfc.test.support.selenide.Operations.*;
import static org.hamcrest.CoreMatchers.*;
import static org.junit.Assert.*;
...
    @Test
    public void test01_シンプルなテスト() {
        logon("admin", "admin"); // ログオン処理を行う。
        pageTitle().shouldHave(exactText("メニュー")); // ページタイトルを確認
        logoff(); // ログオフ処理を行う。
        // HTMLのタイトルを確認
        assertThat(title(), is("Wagby アプリケーション ログオン"));
    }
```

操作と確認

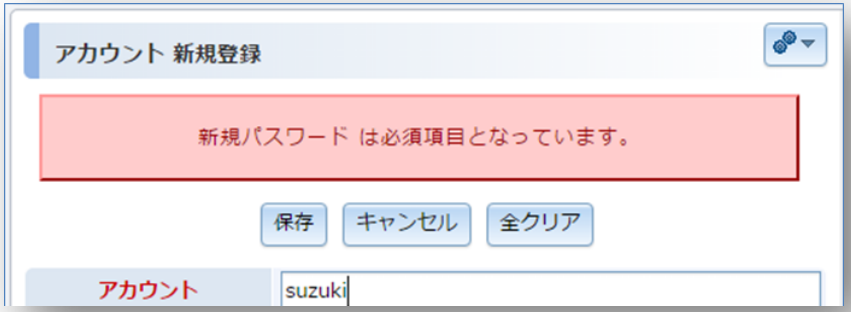
- テストは操作と確認の繰り返し
  - ログオン操作 : `logon("admin", "admin")`
  - 操作後のページタイトルの確認 :
    `pageTitle().shouldHave(exactText("メニュー"))`
  - ログオフ操作 : `logoff()`
  - 操作後のHTMLタイトルの確認 :

    `assertThat(title(), is("Wagby アプリケーション ログオン"))`
- 画面遷移操作後は必ずページタイトルを確認する



暗黙的な確認

- 暗黙的な確認事項
  - Operations クラスが提供する操作用メソッドは以下の暗黙的な確認事項を自動的に実施
    - エラーメッセージが表示されていないことを確認
    - JavaScriptのエラーが発生していないことを確認
  - 例) ログオン操作 : `logon("admin", "admin")`
    - logon()メソッド内部でcheckNoErrors()メソッドを自動的に呼び出す
    - checkNoErrors()メソッドでは以下を実行
      - *assertNoErrorMessages()*
      - *assertNoJavascriptErrors()*



異常系のテスト

- エラーメッセージの内容確認
  - 例)初期パスワードでのログオン


```
import static com.codeborne.selenide.CollectionCondition.*;
  …
// ログオン処理を行う(checkNoErros()は行わない)。
logon("user01", "user01", false);
// JavaScriptのエラーが無いことを確認。
assertNoJavascriptErrors();
// エラーメッセージの完全一致チェック。
errors().shouldHave(exactTexts("初期パスワードがセットされています。パスワードの変更を行ってください。パスワードを変更するまでは、操作が制限される可能性があります。"));
```

異常系のテスト

- 複数のエラーメッセージ

```
import static com.codeborne.selenide.CollectionCondition.*;
 …
// 複数エラーメッセージの完全一致チェック。
errors().shouldHave(exactTexts(
        "エラーメッセージ01",
        "エラーメッセージ02",
        "エラーメッセージ03"));

// 複数エラーメッセージの部分一致チェック。
errors().shouldHave(texts(
        "メッセージ01",
        "メッセージ02",
        "メッセージ03"));
```

異常系のテスト

- エラーコードでのチェック
  - エラーメッセージが変更されるとテストに失敗する
  - エラーメッセージのHTMLにはエラーコードが含まれている(Wagbyの独自属性)

エラーメッセージのHTML

```
<div  class="errormsg"  msgcode="error.init.passwd">
   初期パスワードがセットされて...
</div>
```

```
// エラーコードでのチェック。
errors().shouldHave(msgcodes("error.init.passwd"));
```

Operationsクラスが提供するメソッド

- 登録・更新・コピー登録画面用

メソッド名	処理内容
save()	「保存」ボタンをクリックする。
cancel()	「キャンセル」ボタンをクリックする。

Operationsクラスが提供するメソッド

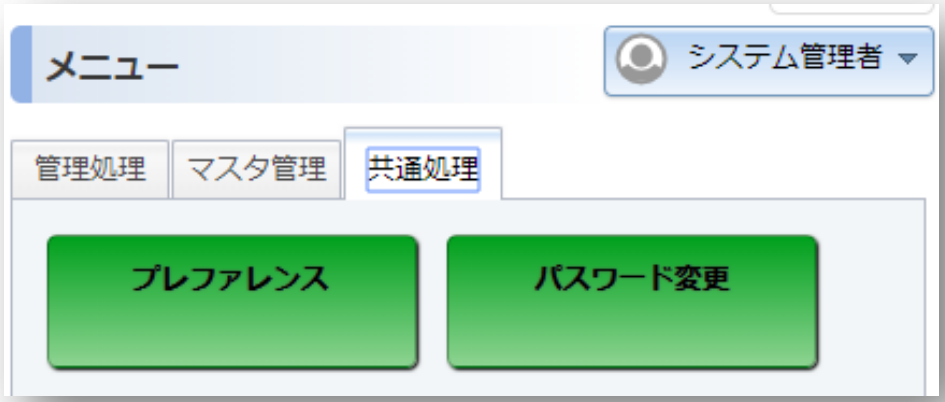
- 詳細･検索･一覧画面用

メソッド名	処理内容
clickNewButton()	「登録画面へ」ボタンをクリックする。
clickEditButton()	「更新画面へ」ボタンをクリックする。
clickDeleteButton()	「削除」ボタンをクリックする。
clickCopyButton()	「コピー登録へ」ボタンをクリックする。
clickSearchPageButton()	「検索画面へ」ボタンをクリックする。
clickListPageButton()	「一覧画面へ」ボタンをクリックする。
search()	検索画面の「検索の実行」ボタンをクリックする。

Operationsクラスが提供するメソッド

- メニュー画面用

メソッド名	処理内容
selectMenu(“タブ名”, “メニュー名”)	指定のメニューを選択する。



Operationsクラスが提供するメソッド

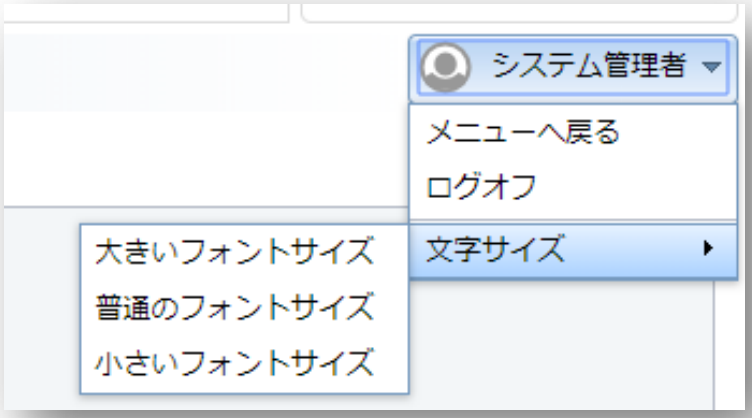
- ウィザード画面用

メソッド名	処理内容
clickWizardNextButton()	「次画面へ画面へ」ボタンをクリックする。
clickWizardPrevButton()	「前画面へ画面へ」ボタンをクリックする。

Operationsクラスが提供するメソッド

- グローバルリンク

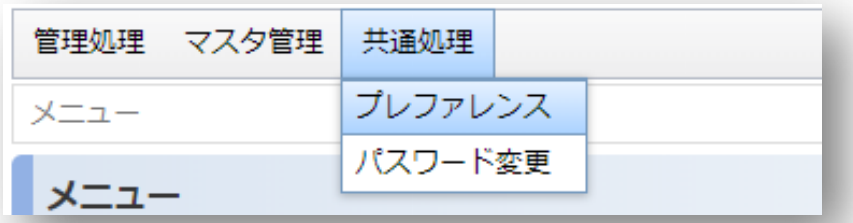
メソッド名	処理内容
menu()	「メニューへ戻る」をクリックする。
logoff()	「ログオフ」をクリックする。
changeFontSize(size) * sizeはlarge,medium,smallを指定	「XXXフォントサイズ」をクリックする。



Operationsクラスが提供するメソッド

- サブメニュー用

メソッド名	処理内容
selectSumMenu(“グループ名”, “メニュー名”)	指定のメニューを選択する。



Operationsクラスが提供するメソッド

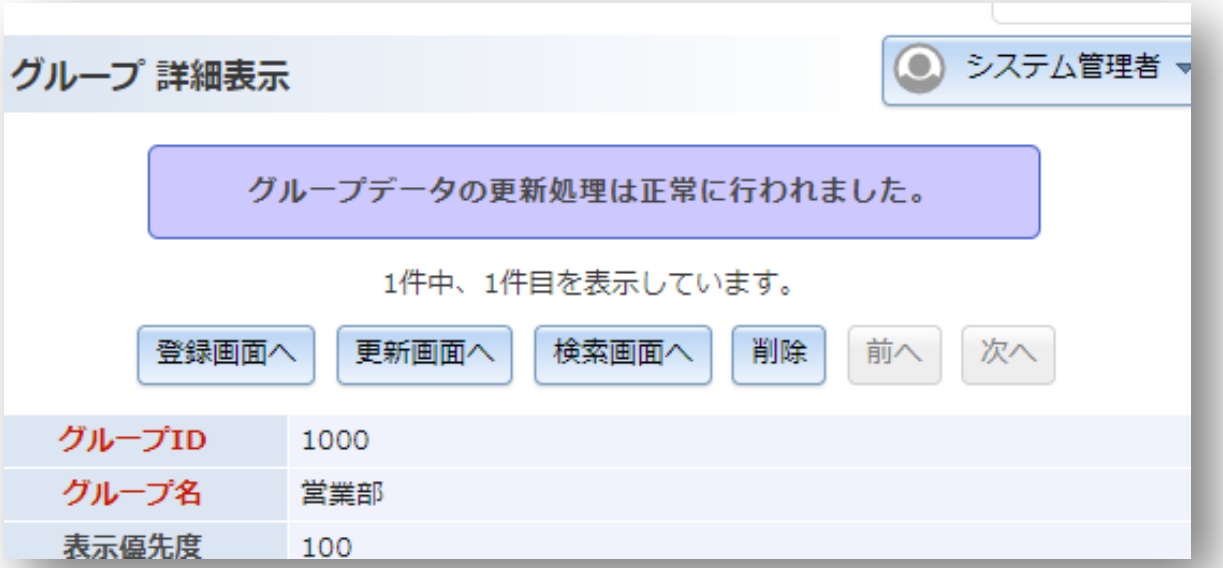
- 全画面共通

メソッド名	処理内容
clickButton(“ボタン表示名”)	指定のボタンをクリックする。
getButton(“ボタン表示名”)	指定のボタンを取得する。
pageTitle()	ページタイトルを取得する。
infos()	infoメッセージを取得する。
warns()	warnメッセージを取得する。
errors()	errorメッセージを取得する。

Operationsクラスが提供するメソッド

- 登録･更新･削除後の画面

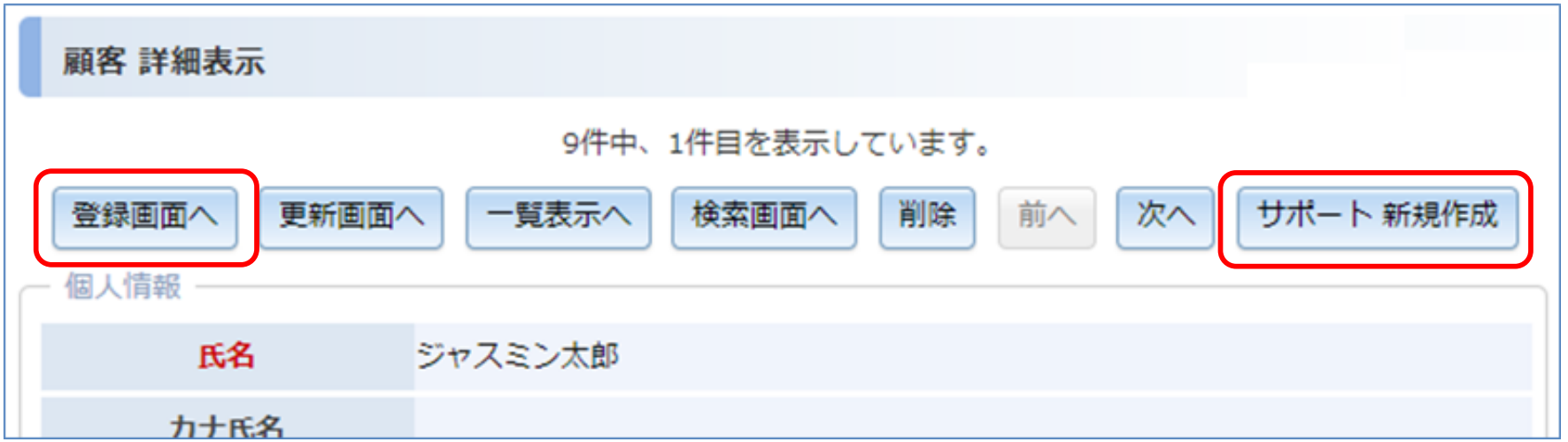
メソッド名	処理内容
checkSuccessMessages()	正常終了メッセージが表示されていることを確認する。 ※ただし、このメソッドはsave()メソッド呼び出し時に自動実行されるため、通常は明示的に呼び出す必要はない。



複数の登録ボタン

- 同一画面に登録ボタンが複数存在する場合
  - 引数にモデルIDを指定する

```
clickNewButton("customer");
```



入力フィールド

- 項目への入力 : val() メソッド

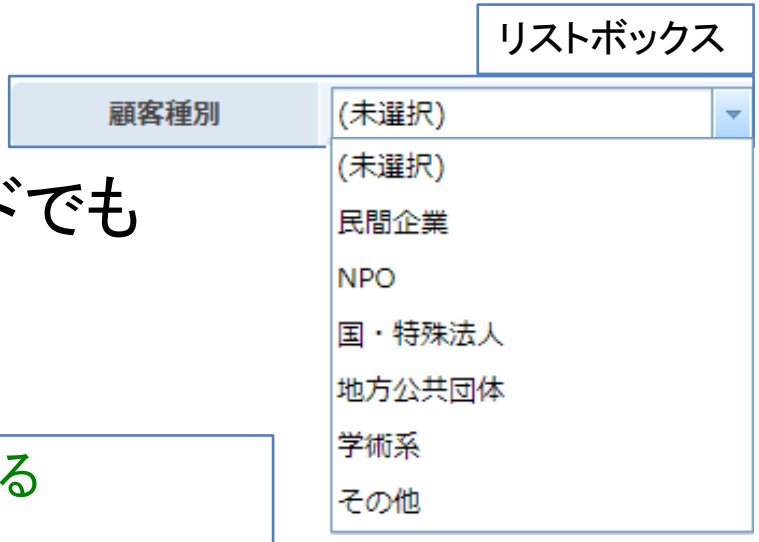
```
// モデルの情報を保持するインスタンス
WebModel model = new WebModel("customer");
// customerモデルのname項目への入力
new WebModelitem<>(model, "name").val("ジャスミン太郎");
```

- 入力値の確認 : shouldHave() メソッド

```
// 「ジャスミン太郎」と入力されていることを確認
new WebModelitem<>(model, "name").shouldHave("ジャスミン太郎");
```

リストボックス

- ListBoxクラスを利用
  - 通常項目と同様に val() メソッドでも値のセットは可能。

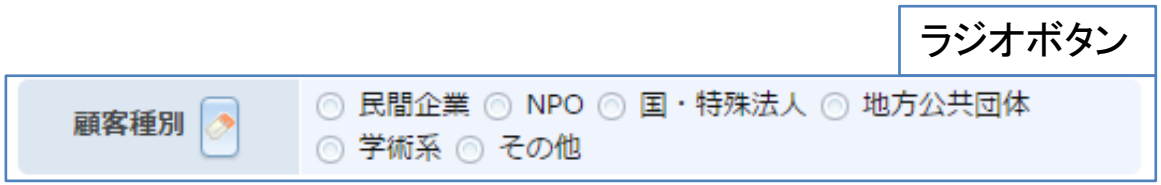



```
// リストボックス:「地方公共団体」を選択する
new ListBox(model, "customertype")
        .dropdown()
        .select("地方公共団体");


// こちらでも可(内部処理は同じ)
new ListBox(model, "customertype")
        .val("地方公共団体");
```

ラジオボタン/チェックボックス

- それぞれの型に応じたクラスを利用

```
// ラジオボタン:「地方公共団体」を選択する
new RadioButton(model, "customertype").val("地方公共団体");

// チェックボックス:「民間企業」と「地方公共団体」を選択する
// チェックボックスは複数の値の指定が可能
new CheckBox(model, "customertype")
        .val("民間企業", "地方公共団体");
```



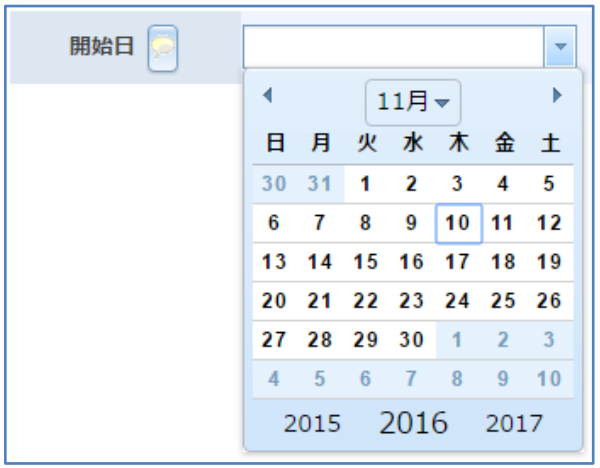
さまざまな種類の項目への対応

- 全て val() で入力、shouldHave()で確認可能

項目の種類	対応するJavaクラス
入力フィールド	WebModelitem
リストボックス	ListBox
ラジオボタン	RadioButton
チェックボックス	CheckBox
検索画面(サブウィンドウ)	SearchList
日付入力フィールド(DatePicker)	DateTextBox
日付リストボックス	DateListBox
時間リストボックス	TimeListBox
追記型リストボックス	ComboBox
郵便番号	Postcode
繰返し項目	WebMultiModelitem

DatePicker

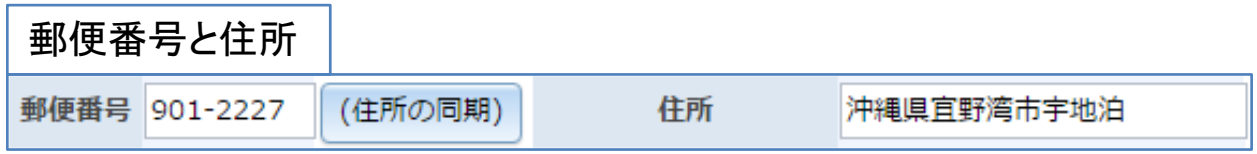
```
// DatePickerを操作するオブジェクトを取得する
DatePicker datePicker
        = new DateTextBox(model, "startdate").datePicker();
datePicker.nextMonth();   // 翌月へ
datePicker.prevMonth();   // 前月へ
datePicker.nextYear();    // 翌年へ
datePicker.prevYear();    // 前年へ
datePicker.selectDate(1); // 1日を選択
```



郵便番号と住所の同期

- Postcodeクラスを利用
  - sync()メソッドで住所の同期を行う

```
// 郵便番号と住所の同期
new Postcode(model, "postcode").val("901-2227").sync();

// 住所項目の値を確認
new WebModelitem<>(model, "address")
        .shouldHave("沖縄県宜野湾市宇地泊");
```



読み込み専用項目

- shouldBeReadOnly()メソッドを利用
  - 読込専用となっていない場合はエラーとなる

```
// 読込専用項目の確認:最終更新者
new WebModelitem<>(model, "updatedBy")
        .shouldBeReadOnly()
        .shouldHave("admin");

// 読込専用項目の確認:データ作成日
new WebModelitem<>(model, "createdAt")
        .shouldBeReadOnly()
        .shouldHave("2017-01-01 00:00:00");
```

最終更新者	admin
データ作成日	2017-01-01 00:00:00
データ更新日	2017-01-01 00:00:00

入力フィールドの存在確認

- 権限により項目が非表示となっていることを確認
  - 登録･更新･コピー登録画面

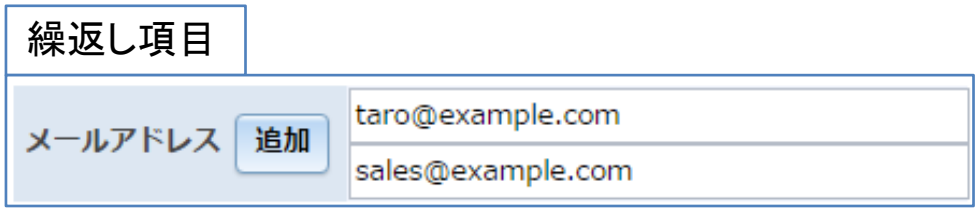
```
// 権限により非表示となっている
new WebModelitem<>(model, "secret_memo")
        .element().shouldNotBe(exist);
```

  - 詳細表示画面

```
// 権限により非表示となっている
new WebModelitem<>(model, "secret_memo")
        .contentElement().shouldNotBe(exist);
```

繰返し項目

```
// 繰り返し項目「email」を操作するオブジェクトを作成。
new WebMultiModelitem<>(model, "email")
        //.clear() // 既存の入力値全てを削除
        .get(1) // 1番目のテキストフィールドを取得
        .val("taro@jasminesoft.co.jp")
        .add()  // 「追加」ボタンをクリック
        .val("sales@jasminesoft.co.jp");

// こちらでも可(既存の入力は全て削除してから、値をセットする)
new WebMultiModelitem<>(model, "email")
        .val("taro@jasminesoft.co.jp", "sales@jasminesoft.co.jp");
```



繰り返しコンテナ

```
// 繰り返しコンテナを操作するオブジェクトを作成。
WebContainerModelitem report = new WebContainerModelitem(model, "report");
// コンテナの1行目があればこれを取得する(なければエラー)。
WebContainerModelitem report01 = report.get(1);
// コンテナ1行目の「rnote」項目への入力
new WebModelitem<>(report01, "rnote").val("Wagby 購入");

// コンテナの2行目がない状態で実行するとエラーとなる。
//WebContainerModelitem report02 = report.get(2);
// コンテナの「追加」ボタンをクリックして、新規行を取得。
WebContainerModelitem report02 = report.add();
// コンテナ2行目の「rnote」項目への入力
new WebModelitem<>(report02, "rnote").val("Wagby 保守契約更新");
```

繰り返しコンテナ

商談履歴

追加		商談日	商談内容	商談状況
1 挿入	削除	2016-04-01	Wagby購入	A
2 挿入	削除	2017-04-01	Wagby保守契約更新	A

詳細画面での入力値の確認

```
// 通常項目
new WebModelitem<>(model, "name").shouldHave("ジャスミン太郎");
// リストボックス
new ListBox(model, "customertype").shouldHave("地方公共団体");
// チェックボックス
new CheckBox(model, "customertype")
        .shouldHave("民間企業", "地方公共団体");
// 他モデルの参照(検索画面)
new SearchList(model, "account").shouldHave("admin");
// 時間型リストボックス
new TimeListBox(model, "start").shouldHave("15:30");
// 追記型リストボックス
new ComboBox(model, "companyname").shouldHave("ジャスミンソフト");
// 郵便番号
new Postcode(model, "postcode").shouldHave("901-2227");
```

モデル情報の分離

- PageObjectデザインパターン
  - 画面を1つのクラスとして定義
  - 画面内のボタンや入力項目をPageObjectに保持
  - テストシナリオクラスを別に用意し、同クラスからPageObjectを操作する方式
- WTFをこの方式に当てはめると
  - 登録/更新/詳細表示画面は、ほぼ同じPageObjectとなる
  - この考えを更に推し進めモデルクラスを作成
    - 登録/更新/詳細画面で同じモデルクラスを利用する
    - ボタンなどの操作はOperationsクラスが補助

モデルクラス

- 項目名と型情報はモデルクラスが保持

```
public class Customer extends WebModel {

    /**
     * コンストラクタ。
     */
    public Customer() {
        super("customer");
    }
    // 通常項目
    public final WebModelitem<?> name = new new WebModelitem<>(model, "name");
    // リストボックス
    public final ListBox<?> customertype = new ListBox(model, "customertype");
    // 他モデルの参照(検索画面)
    public final SearchList<?> account = new SearchList(model, "account");
    // 時間型リストボックス
    public final TimeListBox start = new TimeListBox(model, "start");
    // 追記型リストボックス
    public final ComboBox companyname = new ComboBox(model, "companyname");
    // 郵便番号
    public final Postcode postcode = new Postcode(model, "postcode");
```

シナリオクラス(テストクラス)

- 入力値や画面の操作情報を実装

```
Customer model =  new Customer(); // モデルオブジェクトの作成
// 通常項目
model.name.val("ジャスミン太郎");
// リストボックス
model.customertype.val("地方公共団体");
// チェックボックス
model.customertype.val("民間企業", "地方公共団体");
// 他モデルの参照(検索画面)
model.account.val("admin");
// 時間型リストボックス
model.start.val("15:30");
// 追記型リストボックス
model.companyname.val("ジャスミンソフト");
// 郵便番号
model.postcode.val("901-2227");
save(); // 保存(Operationsクラスのメソッド)
```

ワークフローのテスト

- テストの前提
  - 年休申請ワークフロー
  - フローパターンは「申請→承認→決裁」
  - 各処理を行うユーザー
    - 申請: applicationUser(申請ユーザー)
    - 承認: admitUser(承認ユーザー)
    - 決裁: approvalUser(決裁ユーザー)

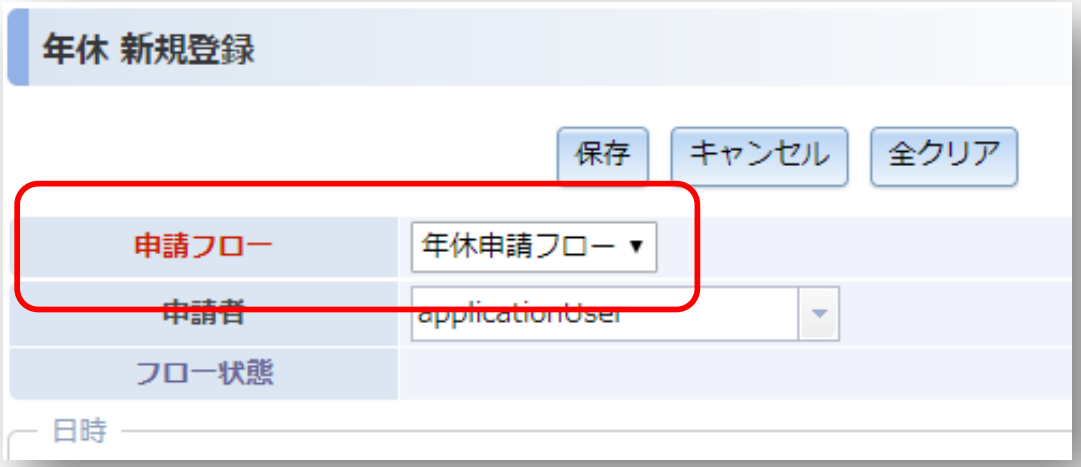
ワークフローのテスト

- テストシナリオを考える
  1. applicationUserでログオン
  2. 年休データの登録
  3. 申請処理を行う(ワークフロー開始)
  4. applicationUserはログオフし、admitUserでログオン
  5. 対象となる年休データの表示
  6. 承認処理を行う
  7. admitUserはログオフし、approvalUserでログオン
  8. 対象となる年休データの表示
  9. 決裁処理を行う
  10. approvalUserのログオフ

データの登録

- 申請フローの指定
  - 申請フロー項目はWagbyが自動的に作成した項目
  - 以下のコードで指定可能

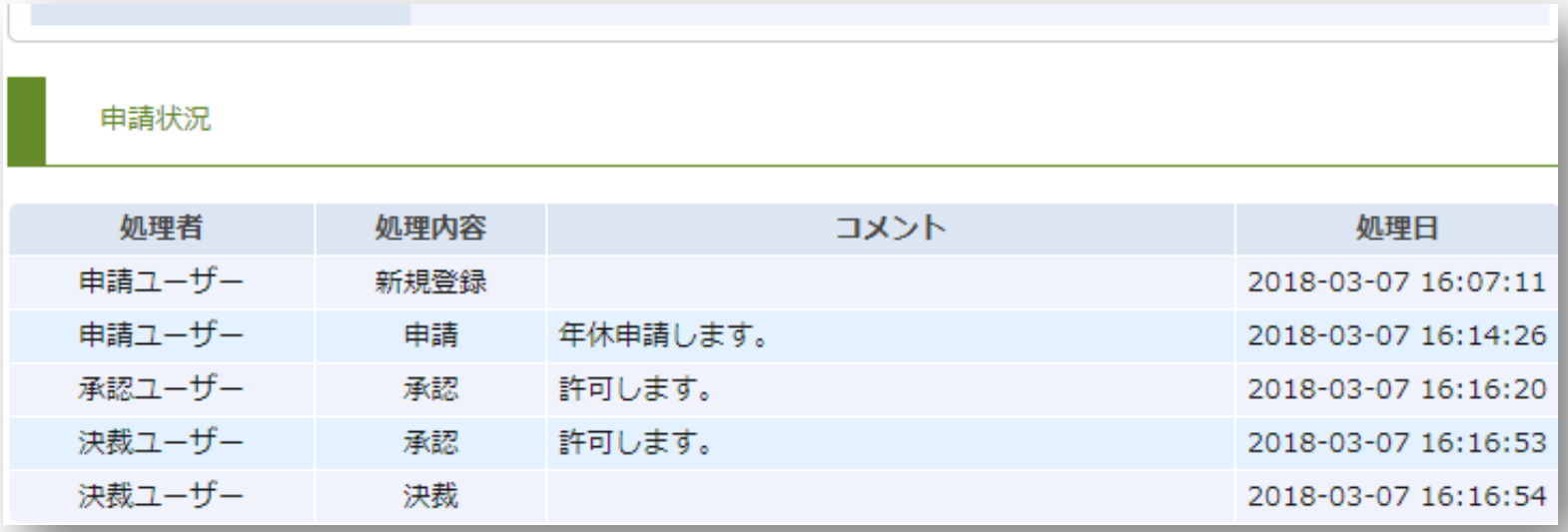
```
workflow().participants_id.val("年休申請フロー");
```



申請状況の確認

- 申請状況
  - 年休モデル詳細画面の下部に表示される

```
// 申請状況一覧の1行目のデータを確認。
JfcworkstateLp record01 = workflow().getWorkstate(1);
record01.username.shouldHave("申請ユーザー");      // 処理者
record01.event.shouldHave(“新規登録”);             // 処理内容
record01.comment.shouldHave(empty);               // コメント
```



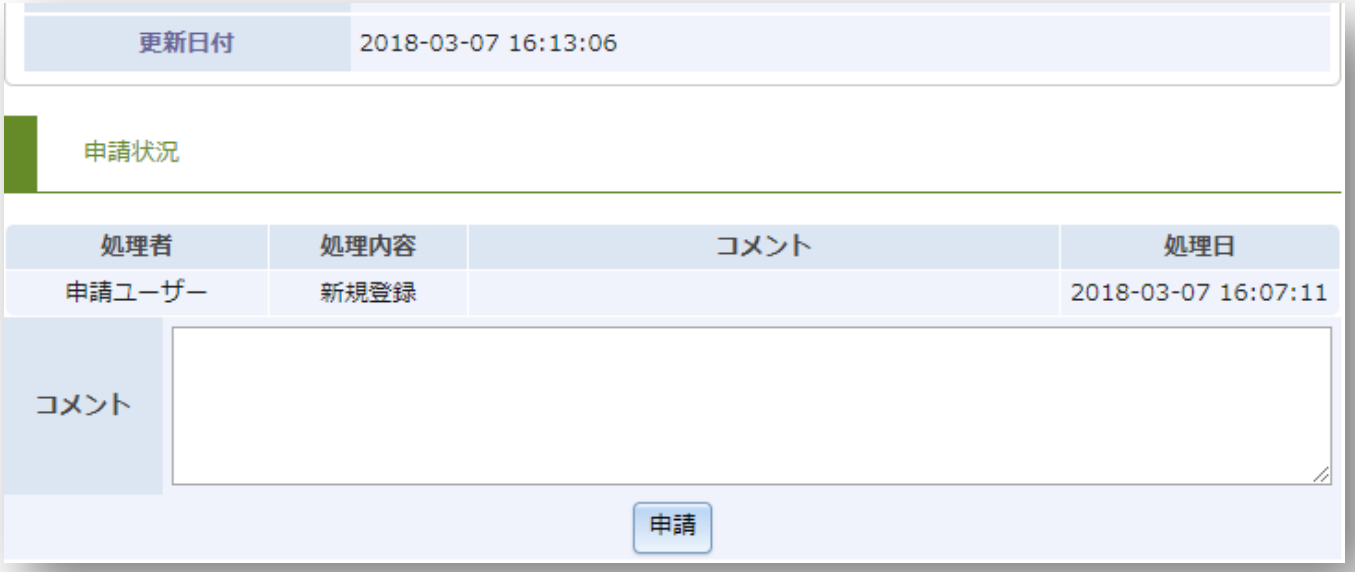
申請状況

処理者	処理内容	コメント	処理日
申請ユーザー	新規登録		2018-03-07 16:07:11
申請ユーザー	申請	年休申請します。	2018-03-07 16:14:26
承認ユーザー	承認	許可します。	2018-03-07 16:16:20
決裁ユーザー	承認	許可します。	2018-03-07 16:16:53
決裁ユーザー	決裁		2018-03-07 16:16:54

申請・承認・決裁処理

- 申請・承認・決裁処理の実行
  - 年休モデル詳細画面の下部に表示されるボタンの操作

```
workflow().comment.val("年休申請します。"); // コメント
workflow().application();                  // 申請ボタンをクリック。
```



コードレビュー及びデモ

まとめ

- Wagby Testing Framework のメリット
  - E2Eテストを自動化
  - テストコードの実装量を減らすことが可能
  - WagbyのバージョンアップによるHTML等の違いもWTFが吸収
    - R7からR8へアップデート後も同じテストコードが動作