Wagby User Group

超高速開発コミュニティ Wagby分科会

Wagby Testing Framework

OUTLINE

- 回帰テスト
- テストの自動化
- Wagbyアプリケーションに対するテスト
- テスト用フレームワーク

回帰テスト

超高速開発

- 企業のスピード経営を実現するための手法
  - 現在の企業経営はシステムと密接に関連
  - 経営方針の変更に合わせてシステムの変更も必須
  - システムの改修が遅れるとビジネスチャンスを逃すことに
  - 経営の変化に迅速に対応する手法として**超速開発**が生まれた

超高速開発とテスト

- システムの変更にはテストが必須
  - 「システム変更は早い」が「不具合も多い」ではNG
  - 超高速開発では大規模な修正を短時間で実現できる
    - しかし、テストの範囲が広がり、テスト工数が増える
- 超高速開発には効率のよいテストが求められる
  - 「テストをしっかりやります。ただし、納期も伸びます」はNG
  - テスト時間がボトルネックになり、システム変更に消極的になるのは望ましくない
    - 逆にテストがしっかりしていると積極的な変更も可能

Wagbyアプリケーションへのテストの考え方

- Wagbyの標準機能は基本的にテスト不要
  - 複雑な自動計算式や式を利用した画面制御、複雑な機能の組み合わせは念のためテストが必要
  - 式が誤っている可能性
  - 解釈の違いにより動作が想定と異なる
    - この機能とこの機能を組み合わせればこのような動作となる(はずだ)
    - この機能を組みわせても今までの動作は変わらない(はずだ)
  - 最初の定義の時のみテストすればよいのか
    - ではなく、他の機能を追加しても動作が変わらないことを確認するテストが必要

回帰テストとは

回帰テスト

読み方:かいきテスト

別名:レグレッションテスト，退行テスト

【英】regression test

回帰テストとは、コンピュータプログラムに手を加えたことによる影響を確認するテスト操作のことである。

プログラムが大規模化で複雑化になってくると、何も関係がないかのように見えるプログラムが相互に関係しあっているのを見落とす場合も少なくない。ある箇所を改善しようとして加えた修正が、思いもよらない部分に影響してバグを呼び起こしてしまう、といった場合も珍しくない。

バグフィックスやリビジョンアップなどが行われた場合、システム全体のチェック作業に立ち返って(回帰して)コード改変の影響が確認される。想定外の異常が発生していないかを調べるためテストに立ち返るテストは、ほとんど必要不可欠な作業であるといえる。

出典：www.weblio.jp

回帰テストの必要性

- 予想していない部分への悪影響を確認するためのもの
  - 変更部分は誰もが注意してテストする
    - ただし、影響がない(だろう) 部分のテストが疎かになる
  - Wagbyはシステムの動作変更が容易
    - しかし、その影響範囲はわかりにくくなっている
    - 他の開発手法より回帰テストは重要

回帰テストの分類

- ユニット(単体)テスト
  - 小さな機能に対して細かなパターンを多くテストする
  - 自動化が容易

- E2Eテスト(結合テスト/総合テスト)

※ (総合テスト = 統合テスト = システムテスト)

  - End　to End テスト
  - 画面遷移や複数機能の組合せ等の大きな流れをテスト
  - テストは手動で行われることも多い（ただし、人為的ミスも）

回帰テストの実施方法

- ユニットテスト
  - プログラムでテストを記述
  - 短時間で多くのテストを実施可能
  - Wagby では基本不要
    - カスタマイズコードや複雑な自動計算などを実装した場合にのみ行う
- E2Eテスト
  - ブラウザを使って画面操作を行いながらの確認
  - 一つのテストに時間がかかることが多い

テストの自動化

テストの自動化

- 自動化の範囲
  1. とりあえず全部(理想)
  2. 優先度の高いものから
  3. 必要最低限
  4. 自動化せず手動で行う

テストプログラムのメンテナンス

- メンテナンスコスト
  - テストコードも(当然)メンテナンスコストがかかる
  - 保守しやすいテストコードとなるように心がける
  - 回帰テストの適用範囲を「すべて」にしてしまうと(当然)メンテナンスコストも上がる
    - 手動テストと自動テストのバランスを考える

安定したテスト

- 自動テストには安定性も必要
  - テストが不安定だと誰もテスト結果を見なくなる
- テストにかかる時間も重要
  - 1回のテストに何十時間もかかるのはNG
    - テストサーバーの分散により解決は可能

テストの自動化フレームワーク

- ユニットテスト
  - Junit
    - Javaテスティングフレームワークのデファクト
- E2Eテスト
  - Selenium
    - ブラウザテスト自動化のデファクト

Selenium

- Selenium (http://www.seleniumhq.org)
  - ブラウザテストの世界標準
  - W3Cで標準化も進められている(Selenium2 WebDriver)
  - マルチプログラミング言語(Java,Ruby,php,JavaScript等)
  - マルチプラットフォーム(Windows,Mac,Linux,モバイルOS)
  - クロスブラウザ(IE,Edge,Google Chrome,Firefox,Safari)

Selenium の問題点

- 複雑なことをするとコードの記述量が増える
- テストの安定性に欠ける
  - テストコードが速く動きすぎて描画が追いつかずエラー
    - JavaScriptを使って描画を行う(非同期)アプリケーションでは特に顕著
    - Implicit Wait(暗黙的な待機)機能では対応できないケースが多い
    - 解決策 : テストコードに wait を明示

Selenide

- Selenide (http://selenide.org)
- Seleniumを拡張したJavaテスティングフレームワーク
  - Selenium: ブラウザを操作することを目的(Low-Level API)
  - Selenide : ブラウザテストに特化
  - 少ない記述量で実装可能
  - 非同期操作のサポート(waitの記述が不要)
    - デフォルトで4秒(変更可)waitするようになっている
      - 描画が行われれば4秒待たずにテストが実行される
  - 異常時にスクリーンショットとhtmlが自動保存される

サンプルコード(Selenide)

- jQueryライクな記述

```
public static void main0(String[] args) {
    open("https://wagby.com/event/wdd2016/form.jsp");
    $("input[name=¥"organization¥"]").val("ジャスミンソフト");
    $(“input[name=¥”name¥“]”).val(“ジャスミン太郎);
    $("input[name=¥"dept¥"]").val("営業部");
    $(“input[name=¥”title¥“]”).val(“なし);
    $("input[name=¥"tel¥"]").val("000-111-2222");
    $("input[name=¥"mailaddress¥"]").val("taro@example.com");
    $("input[name=¥"agreement¥"]").click();
    //$("#apply")).click();
}
```

waitの記述

- Selenium

```
WebDriverWait wait = new WebDriverWait(driver, 4000); //待ち時間を指定
By button = By.id("apply");
wait.until(
    ExpectedConditions.visibilityOfElementLocated(button));
driver.findElement(button).click();
```

- Selenide

```
$("#apply").click(); // waitの記述は不要
```

Wagbyアプリケーションへの自動テスト

WagbyでのE2E自動テスト

- シンプルなHTMLではない
  - JavaScriptライブラリ「Dojo Toolkit」を採用
    - Dojoがもつ豊富なUIパーツを利用
    - ボタンなども <input type="button" ..> ではなく、の入れ子で表現されている
  - HTML構造が複雑なのでE2Eテストの難易度は高い

Wagby側での工夫

- 各項目、ボタンにはidを割り当てる
  - HTMLが複雑でもidを指定することでテストコードをシンプルにすることができる
    - idはユニーク性が保証されているのでテストミスが発生しない

保存ボタンのHTML

```
<div class="action_button">
  <span class="..." role="presentation" widgetid="btnSend">
    <span class="..." data-dojo-attach-event="ondijitclick:__onClick" role="presentation">
      <span class="..." role="button" id="btnSend" style="user-select: none;">
        <span class="..." data-dojo-attach-point="iconNode"></span>
        <span class="...">●</span>
        <span class="...” id=“btnSend_label”>保存</span>
      </span>
    </span>
    <input name="btnSend" type="button" value="" class="...">
  </span>
</div>
```

入力フィールド

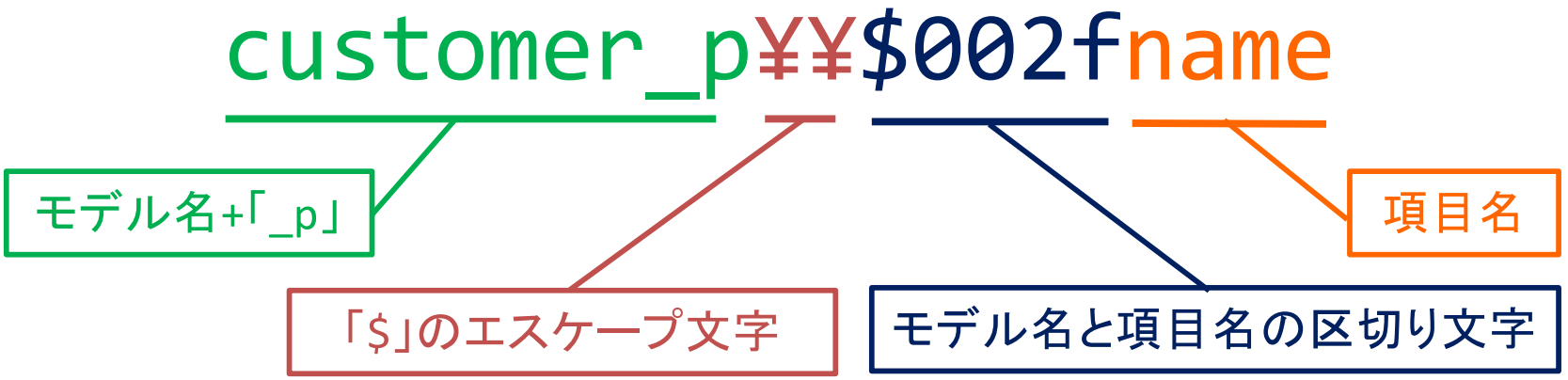
- 通常の入力フィールド、テキストエリア

```
$("#customer_p¥¥$002fname").val("ジャスミン太郎");
```

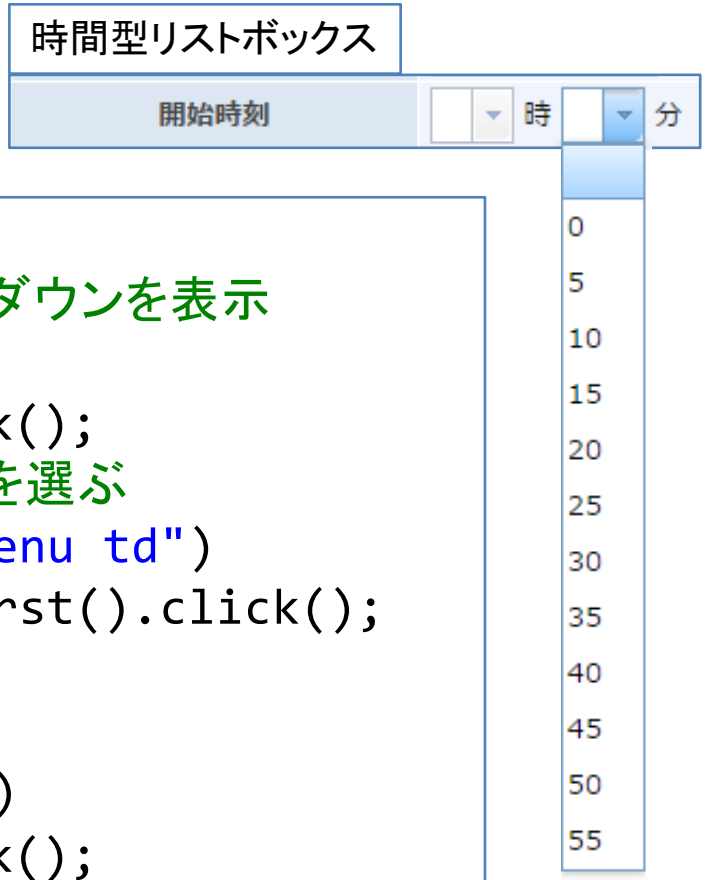
- 入力フィールドのid属性の構造

```
<input type="text" id="customer_p$002fname" name="customer_p$002fname" ..>
```

  - idに含まれる「$」記号はバックスラッシュでエスケープ
    (CSSセレクタのルール。jQueryも同じ)



リストボックス




```
//開始時刻の選択。「6:30」を指定する。
//時間の選択。STEP1:▼をクリックしてプルダウンを表示
$("#customer_p¥¥$002fstart_hour")
        .find(byValue("▼ ")).click();
//時間の選択。STEP2:プルダウンから時間を選ぶ
$$("#customer_p¥¥$002fstart_hour_menu td")
        .filter(exactText("6")).first().click();

//分の選択
$("#customer_p¥¥$002fstart_minute")
        .find(byValue("▼ ")).click();
$$("#customer_p¥¥$002fstart_minute_menu td")
        .filter(exactText("30")).first().click();
```

パターン化されたHTML

- 複雑なHTMLの中にもパターンがある
  - idの命名規則
    - 入力フィールド: cucstomer_p¥¥$002fname
    - 時間リストボックスの「時」: customer_p¥¥$002fstart_hour
    - 時間リストボックスの「分」: customer_p¥¥$002fstart_minute
  - リストボックスの操作パターン
    - STEP1：▼をクリックしてプルダウンを表示
    - STEP2：プルダウンから時間を選ぶ

Wagby Testing Framework

- Wagbyに特化したTesting Frameworkを提供
  - パターン化された共通部分は記述不要に
  - 編集画面での入力もシンプルに
    - 長いIDの記述は不要
    - 通常項目は、項目名と値のみを指定

ログオン(selenide)

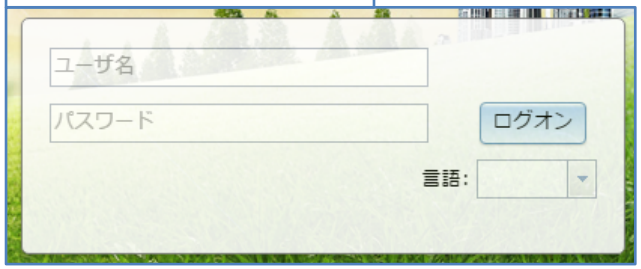
- ユーザー名とパスワードを入力してログオン

```
// http://localhost:8921/wagby/ へアクセス
open("/");
// JavaScriptにエラーが無いことを確認
assertNoJavascriptErrors();
// ユーザー名とパスワードを入力後、ログオン
$("#user").val("admin");
$("#pass").val("admin").submit();
```

ログオン画面の簡易HTML

```
<html><head>
    <title>Wagby アプリケーション ログオン</title>
  </head><body>
  <form ...>
    <input id="user" name="user" type="text">
    <input id="pass" name="pass" type="password">
  </form>
</body></html>
```

ログオンフォーム



ログオン(selenide)

- ログオン後の確認

```
// ページタイトルを確認
$("div.pagetitle").shouldHave(exactText("メニュー"));
// エラーが無いことを確認
$("div.errormsg").shouldNot(exist);
assertNoJavascriptErrors();
```

メニュー画面の簡易HTML

```
<html>
  <body>
    <div class="pagetitle">メニュー</div>
  …
  </body>
</html>
```

メニュー画面

ログオン(testing framework)

- 共通処理を行うOperationsクラスを用意

```
// ログオン処理を行う。
Operations.logon("admin", "admin");
// ページタイトルを確認
Operations.pageTitle().shouldHave(exactText("メニュー"));
```

  - static インポートで更に省略

```
import static com.codeborne.selenide.Condition.*;
import static jp.jasminesoft.jfc.test.support.selenide.Operations.*;

    …
    // ログオン処理を行う。
    logon("admin", "admin");
    // ページタイトルを確認
    pageTitle().shouldHave(exactText("メニュー"));
```

エラーチェック(selenide)

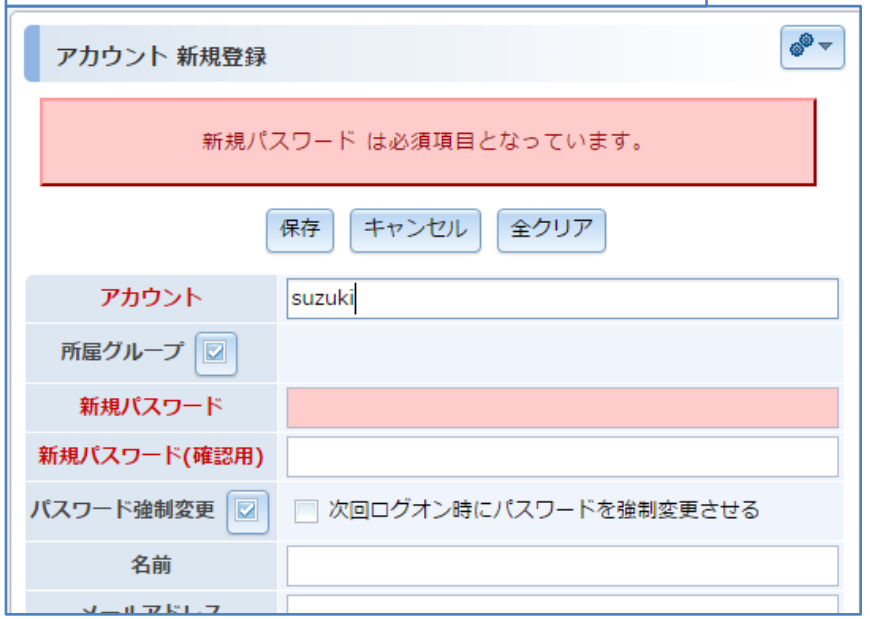
- エラーメッセージの存在チェック

```
// エラーメッセージが表示されていないことを確認
$("div.errormsg").shouldNot(exist);
// JavaScriptのエラーが無いことを確認。
assertNoJavascriptErrors();
```

エラーメッセージのHTML

```
<div class="errormsg“>
   新規パスワード は必須項目となっています。
</div>
```

エラーメッセージが表示されている画面



エラーチェック(testing framework)

- Operations#checkNoErrors() メソッド
  - 画面上にエラーメッセージが無いこと及びJavaScriptのエラーが発生していないことを確認。

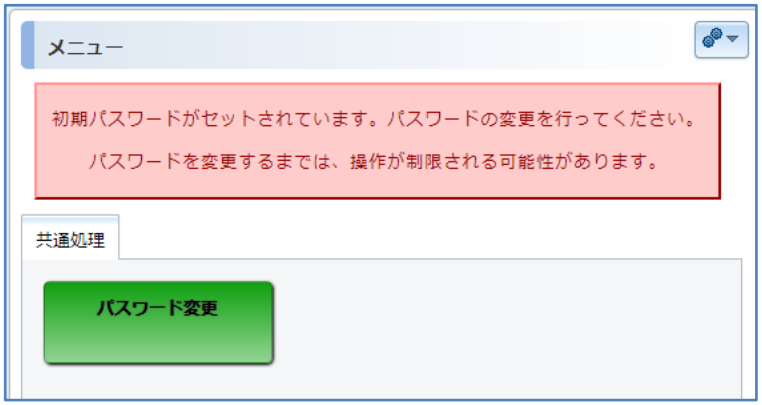
```
// エラーが無いことを確認します。
checkNoErrors();
```

- checkNoErrors() の自動呼び出し
  - ログオン時やデータの保存時処理の呼出時は自動的にcheckNoErrors()が実行される

```
// ログオン処理を行う(内部でcheckNoErros()を実行)。
logon("admin", "admin");
// checkNoErrors(); // 個別実行は不要。
```

異常系のテスト(testing framework)

- 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(testing framework)

- 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(testing framework)

- 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"));
```

ログオフ(testing framework)

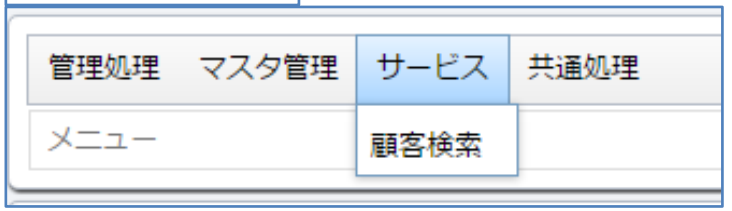
```
import static com.codeborne.selenide.Selenide.*;

  // ログオフ処理を行う。
  logoff();
  // checkNoErrors(); // 不要。
  // 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"));
```

メニュー(testing framework)

```
// メインメニューでの画面遷移
// 「サービス」タブを選択後、「顧客検索」メニューをクリック
selectMenu("サービス", "顧客検索");
// ページタイトルを確認
pageTitle().shouldHave(exactText("顧客　検索"));

// サブメニューでの画面遷移
// 大項目「サービス」を選択後、プルダウンから「顧客検索」を選択する
selectSubMenu("サービス", "顧客検索");
pageTitle().shouldHave(exactText("顧客　検索"));
```

メインメニュー

サブメニュー

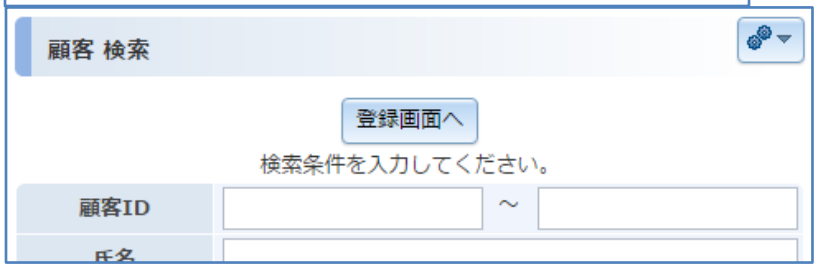


登録画面への遷移

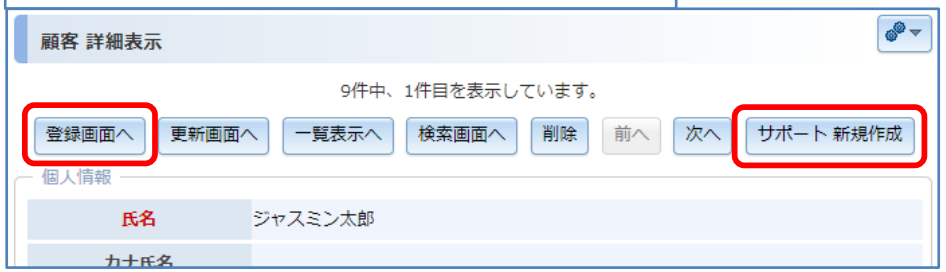
- 登録画面への遷移ボタン
  - ボタンのid値は”btnNewInsert**Customer**”

```
// 「登録画面へ」ボタンをクリック
clickNewButton("customer");

// 登録画面用ボタンが一つの場合は"customer"は省略可
clickNewButton();

//clickEditButton(); //更新用(ルールは「登録画面へ」のボタンと同じ)
```

登録画面への遷移ボタンが一つのみ存在



登録画面への遷移ボタンが複数存在



保存/キャンセル処理

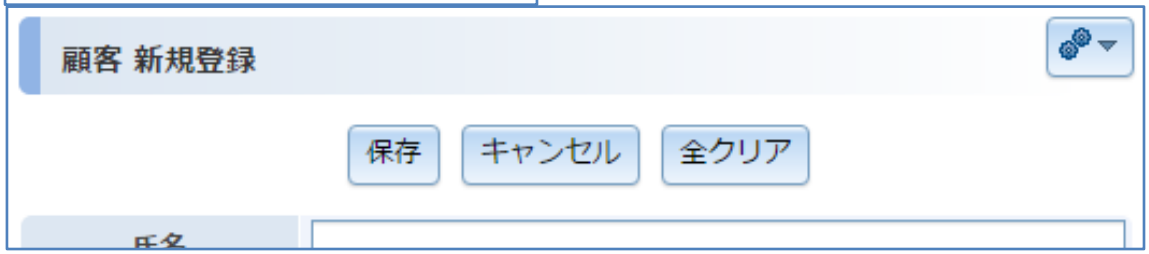
– 保存処理

```
// 「保存」ボタンをクリック
save();

// ページタイトルを確認
pageTitle().shouldHave(exactText("顧客　詳細表示"));
```

– キャンセル処理

```
//「キャンセル」ボタンをクリック
//「キャンセル」ボタンクリック後に表示される確認ダイアログも
//自動的にOKをクリックします。
cancel();
// ページタイトルを確認
pageTitle().shouldHave(exactText("顧客　詳細表示"));
```

保存/キャンセルボタン



入力フィールド

- 通常項目への入力

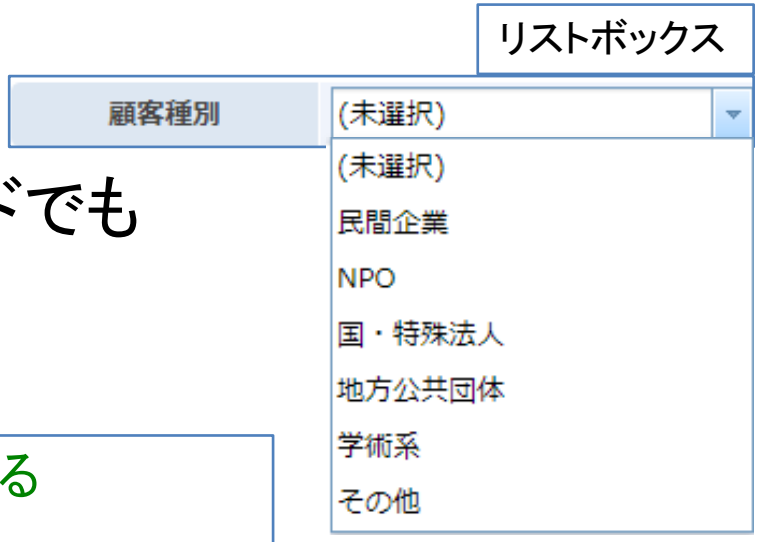
```
// モデルの情報を保持するインスタンス
WebModel model = new WebModel("customer");
// customerモデルのname項目への入力
new WebModelitem<>(model, "name").val("ジャスミン太郎");
```

- 入力値の確認

```
// 「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。

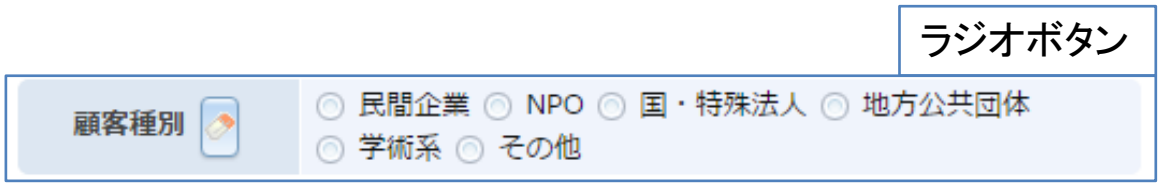



```
// リストボックス:「地方公共団体」を選択する
new ListBox(model, "customertype")
        .dropdown()
        .select("地方公共団体");


// 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

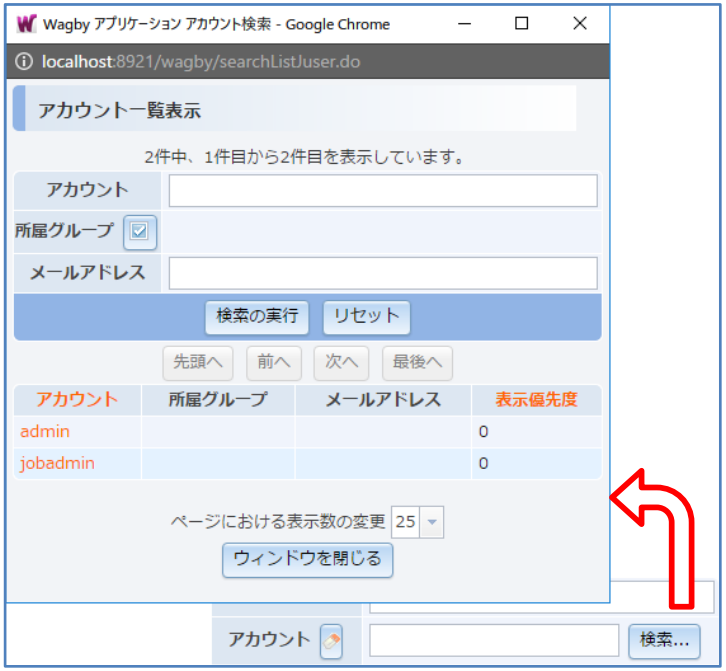
```
// ラジオボタン:「地方公共団体」を選択する
new RadioButton(model, "customertype").val("地方公共団体");

// 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");
```



他モデルの参照(検索画面)

- サブウィンドウを開く

```
SearchList account = new SearchList(model, "account");
// サブウィンドウを操作するオブジェクトを取得。
SubWindow subWindow = account.subWindow();
// サブウィンドウを表示する。
subWindow.show();
```



他モデルの参照(検索画面)

- サブウィンドウの一覧表示部から値を選ぶ

```
// サブウィンドウの一覧画面に表示されている "admin" のリンクをクリック。
subWindow.clickLink("admin");
// フォーカスをメインウィンドウに戻す。
WebElementUtils.focusMainWindow();
```

アカウント	所属グループ	メールアドレス	表示優先度
admin			0
jobadmin			0

– サブウインドウの表示から値の選択までを一括で処理

```
// こちらでも可(内部処理は同じ)
new SearchList(model, "account").val("admin");
```

日付項目

- 直接入力を行う場合は通常項目と同じ

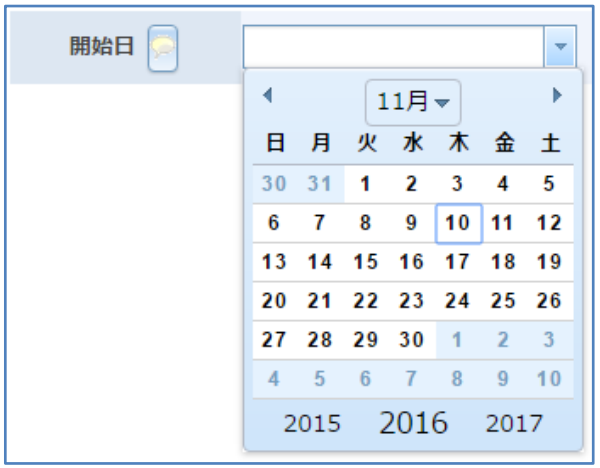
```
new WebModelitem<>(model, "startdate").val(“2017-01-01");
```

  - DatePicker を使って入力テストを行う場合はDateTextBox クラスを利用

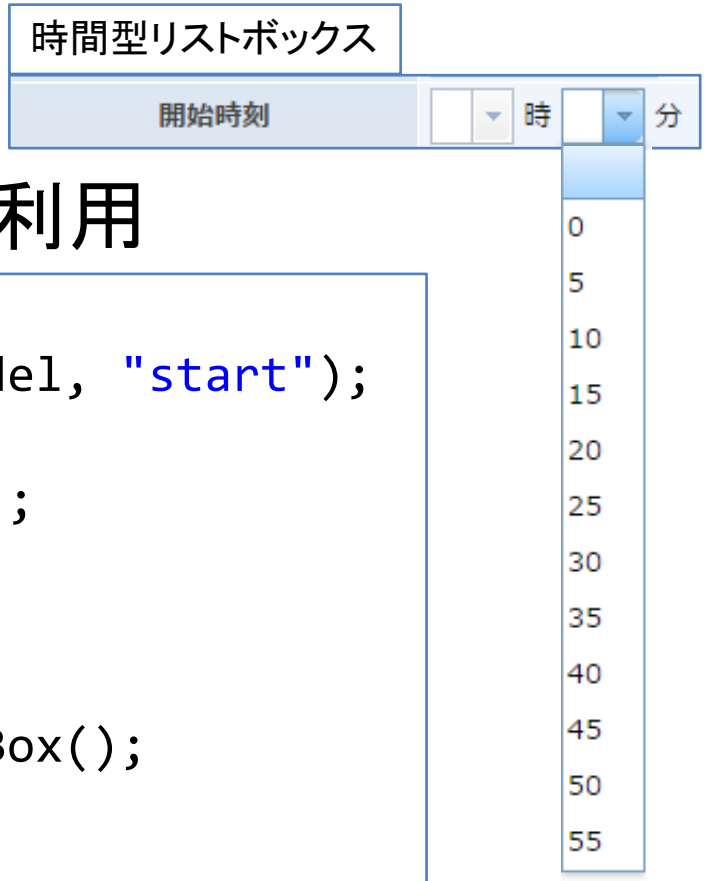
```
new DateTextBox(model, "startdate");
```

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
```



日付/時間型リストボックス

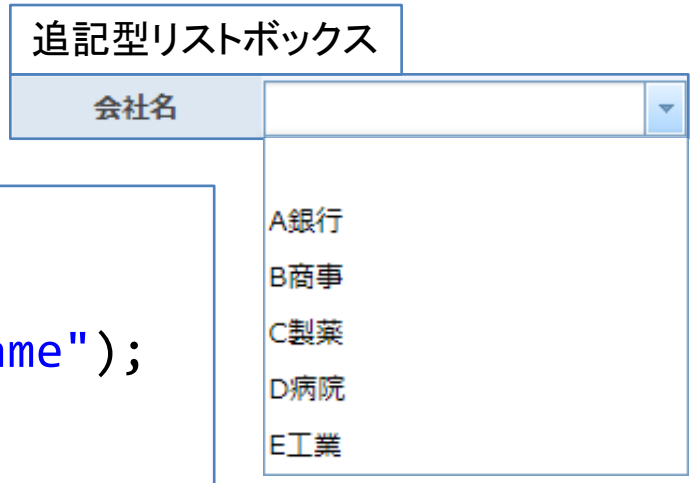


- DateListBox/TimeListBoxクラスを利用

```
// 時間型リストボックス
TimeListBox start = new TimeListBox(model, "start");
//「時」入力用のListBoxを取得します。
ListBox startHour = start.hourListBox();
// 15時をセット
startHour.dropdown().select(15);
//「分」入力用のListBoxを取得します。
ListBox startMinute = start.minuteListBox();
// 13分をセット
startMinute.val(30);

// こちらでも可(内部処理は同じ)
new TimeListBox(model, "start").val("15:30");
```

追記型リストボックス

- ComboBoxクラスを利用

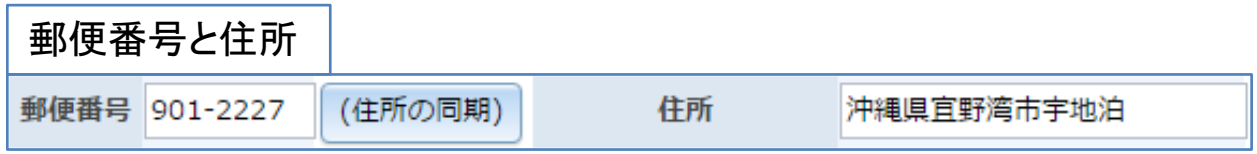



```
// 追記型リストボックス
ComboBox companyname
          = new ComboBox(model, "companyname");
// 直接入力を行う場合(存在しない選択肢も可)
companyname.val("ジャスミンソフト");


// ドロップダウンから選択する場合
// 存在しない選択肢を選んだ場合はエラー
companyname
     .dropdown()
     .select("ジャスミンソフト");
```

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

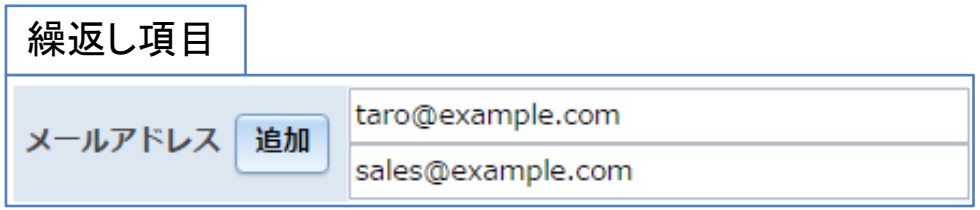
- shouldBeReadOnly()メソッドを利用
  - 読込専用となっていない場合はエラーとなる

```
// 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

繰返し項目

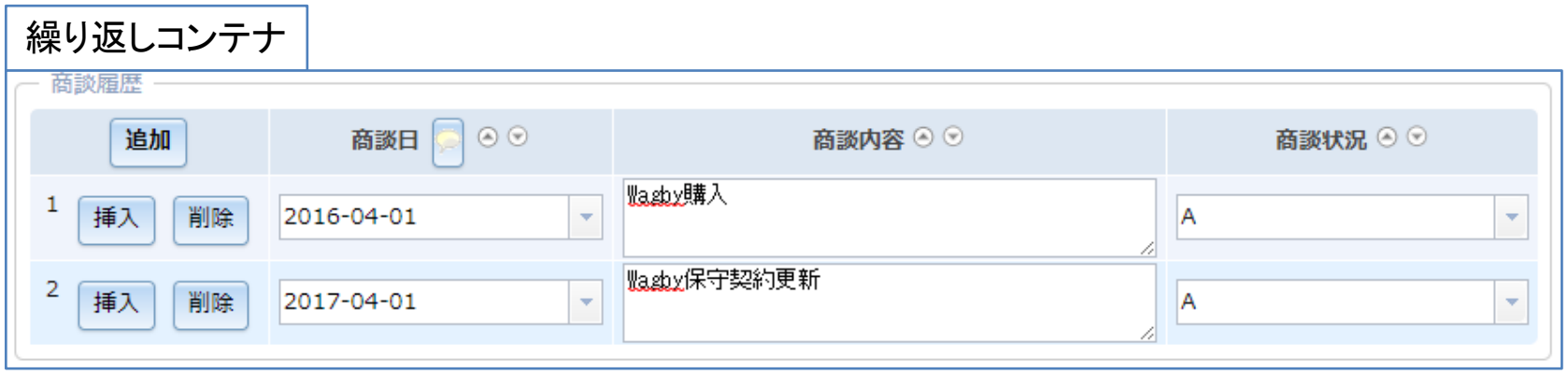
```
// 繰り返し項目「email」を操作するオブジェクトを作成。
new WebMultiModelitem<>(model, "email")
        //.clear() // 既存の入力値全てを削除
        .get(1) // 1番目のテキストフィールドを取得
        .val("taro@jasminesoft.co.jp")
        .add()  // 「追加」ボタンをクリック
        .val("sales@jasminesoft.co.jp");

// 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



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");
new DateTextBox(report01, "startdate").shouldHave("2017-07-01");
// 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");
```

詳細画面での入力値の確認

```
// 繰返し項目
new WebMultiModelitem<>(model, "email")
        .shouldHave("taro@jasminesoft.co.jp", "sales@jasminesoft.co.jp");
// 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行目を取得。
WebContainerModelitem report01 = report.get(1);
// コンテナ1行目の「rnote」項目
new WebModelitem<>(report01, "rnote").shouldHave("Wagby 購入");
// コンテナの2行目を取得。
WebContainerModelitem report02 = report.get(2);
// コンテナ2行目の「rnote」項目
new WebModelitem<>(report02, "rnote")
        .shouldHave("Wagby 保守契約更新");
```

テストコードの保守性向上

- バージョンアップによる変更はTesting Frameworkで吸収します
  - 自動生成されたjspファイルの変更
  - Dojo Toolkitのバージョンアップ
- Wagbyのバージョンアップ=Testing Frameworkのバージョンアップ
  - 開発者が記述したテストコードはそのままに新しいHTML構造に対応

まとめ

- 超高速開発には回帰テストは必須
- Wagbyアプリケーションに対するテスト
  - すべてをテストする必要はない
  - 自動計算式や複雑な設定を組み合わせたケースが対象
- Wagby Testing Framework
  - テストコードのシンプル化
  - 保守性の向上