ダイアログの実装（コンシューマ画面）

TCIより提供されたdialog.jsを利用してダイアログを開く際の実装方法について記載する。
下記内容はatd-csm-webにおけるPC画面でのみ適用可能なので注意。

親画面のボタンクリックなどでダイアログを開く処理

させたい動作の概要

1. 親画面に、以下のようにダイアログ用コンテンツ表示領域を用意しておく（不可視）
   - イメージ表示イメージ表示

```
<%--　ダイアログ描画領域　-%>
<div id="modal-dialog-form" class="dialog　dialog-forı" style="display:none;"></div>
```

2. ダイアログを開くボタンがクリックされたら、Ajaxで子画面（ダイアログ）用のコントローラの初期表示メソッドを呼び出し、
   JSP（HTML）を返させて、親画面の上記「ダイアログ用コンテンツ表示領域」に埋め込む（この時点ではまだ不可視）
   - Ajaxイメージ表示Ajaxイメージ表示

```
function handleDialogOpen(event)　{
  var $dialog　=$(this);
  var $popupButton　=$(event.currentTarget);
  var　dialogContentUrl　$popupButton.data('atd-dialog-content-url');
//　画面から子画面（ダイアログ）用コントローラの初期表示メソッドのURLを取得

  return　atc.ajax(dialogContentUrl,　{
//　Ajaxで呼び出してHTML取得　【注】Ajax呼び出しの完了を待ったうえでダイアログを表示させる必要があるので、必ず　atc.ajaxの戻り値　(Promiseオブジェクト)　を返すこと
    type: 'GET'
     }).doıfunction(dialogContentHtml)　{
    $dialog.html(dialogContentHtml);　//　取得したHTMLを「ダイアログ用コンテンツ表示領域」に埋め込む
     });
}
```

   - 子画面コントローラ初期表示メソッドイメージ表示子画面コントローラ初期表示メソッドイメージ表示

```
    @GetMapping("init")
    public　String　init(Model　model)　{
        final　DSBS0260P01PInitForm　formnew　DSBS0260P01PInitForm();
                model.addAttribute(form);
        return "DSBS0260P01P"; //　ふつうに画面表示するときと同じにすればJSP(HTML)を返せる
          }
```

3. dialog.jsがモーダルを開き、「ダイアログ用コンテンツ表示領域」の内容が表示される

対応内容

1. 親画面に、以下のようにダイアログ用コンテンツ表示領域を用意しておく（不可視）

```
<%--　ダイアログ描画領域　-%>
<div id="modal-dialog-form" class="dialog　dialog-forı" style="display:none;"></div>
```

2. 親画面のダイアログを開くためのボタンのdata属性に、以下のように定義する
※実際はTCIデザイン時点である程度記載されているはず
- PC版PC版

```
<input type="button" value="上記の内容で次へ進む" class="省略"
  data-dialog-popup-form="#modal-dialog-form"          <!-- <1>  >
      data-dialog-fadespeed="500"                      <!--  <2>  ->
      data-atd-dialog-content-url="${dialogContentUrl}" <!--  <3>  -->
      data-callback-func-open="handleDialogOp<>        <!--  <4>  -->
```

  - <0> data-callback-func-submit属性は、TCIデザインで定義されていてもここには含めないこと
    data-callback-func-close属性は、後述する「ダイアログクローズ前に処理を挟みたい場合」のみ使用するため、その場合以外は含めないこと
  - <1> 手順1で設定したIDと合わせる
  - <2> TCIデザインのままにする（他にも、この<0>～<4>に登場しないdata要素はTCIデザインのままにすること）
  - <3>
    子画面（ダイアログ）コンテンツ取得用コントローラーの初期表示メソッドのURLを定義する（後でspring-urlで仕込むので上記のように定義で良い）
  - <4> このボタンをクリックしてからダイアログを開くまでの間に実行されるcallback関数名を指定する
    単に子画面（ダイアログ）コンテンツを取得して表示するだけで良い場合は「handleDialogOpen」を指定すること。

- SP版SP版

```
<input type="button" value="上記の内容で次へ進む" class="省略"
  data-toggle="modal"                              <!-- <1>  >
      data-target="#dialogSelectWindow1"           <!--  <2>  ->
      data-callback-func-open="handleDialogOpen1"  <!--  <3>  ->
      data-form="true"
      data-popup-speed="300"
      data-atd-dialog-content-url="${dialogContentUrl}" <!--  <4>  -->
```

  - <0> data-callback-func-submit属性は、TCIデザインで定義されていてもここには含めないこと
    data-callback-func-close属性は、後述する「ダイアログクローズ前に処理を挟みたい場合」のみ使用するため、その場合以外は含めないこと
  - <1> TCIデザインのままにする（他にも、この<0>～<4>に登場しないdata要素はTCIデザインのままにすること）
  - <2> 手順1で設定したIDと合わせる
  - <3> このボタンをクリックしてからダイアログを開くまでの間に実行されるcallback関数名を指定する
    単に子画面（ダイアログ）コンテンツを取得して表示するだけで良い場合は「handleDialogOpen」を指定すること。
  - <4>
    子画面（ダイアログ）コンテンツ取得用コントローラーの初期表示メソッドのURLを定義する（後でspring-urlで仕込むので上記のように定義で良い）

3. 親画面に、子画面（ダイアログ）用コントローラの初期表示メソッドのURLを定義する

```
<%--  ダイアログコンテンツ取得用URL  -->%
<spring:url var="dialogContentUrl" value="/search/DSBP0260/init"/>  <%-- <1>  --%>
```

- <1> varは手順2-<3>で指定した名前と合わせる

4. 子画面（ダイアログ）用コントローラの初期表示を実装
- Controller

```
@Controller
@RequestMapping("search/DSBP0260")
@DeviceMapping(DeviceType.PC)
public class DSBP0260PController  {   //  <1>
```

```
    @GetMapping("init")
    public String  init(Model  model)  {
        return "DSBS0260P01P";          //  <2>
      }
}
```

- <1> 親画面用のControllerと分ける（基本設計通りの機能IDの割り振りになるはず）
- <2> 通常の画面と同様、viewを返す

◦ tiles（子画面）

```
<definition name="DSBS0260P01P" extends="dialog-content-layout">                    <!--  <1>  -->
  <put-attribute name="content" value="/WEB-INF/views/search/DSBS0260P01P/content.jsp" />   <!--  <2>  -->
</definition>
```

- <1> ダイアログ用の共通レイアウト（dialog-content-layout）を継承
- <2> 通常の画面と同様、contentにjspを差し込む

◦ JSP

- コード例省略。ダイアログ内の要素をjsp可。

5. 親画面のtiles定義にダイアログオープン用の共通jsと子画面用のjsを追加
   ◦ tiles（親画面）

```
<definition name="DSBS0250P01P" extends="optional-plan-layout">
  <put-attribute name="content" value="/WEB-INF/views/search/DSBS0250P01P/content.jsp" />

  <put-list-attribute name="jsContentList" inherit="true">
    <add-attribute value="/resources/js/search/DSBS0250P01P/content.js" />
    <!--  モーダル画面で使用するjs  -->
    <add-attribute value="/resources/js/atd-dialog-common.js" />                <!--  <1>  -->
    <add-attribute value="/resources/js/search/DSBS0260P01P/content.js" />          <!--  <2>  -->
  </put-list-attribute>
</definition>
```

- <1> ダイアログ用共通js（handleDialogOpenが実装されている）
- <2> 子画面のjs

以上。ダイアログを開く前に処理を挟みたい場合は下記「ダイアログを開く前に処理を挟みたい場合」を参照。
（e.g. 自動割付可能な場合は画面遷移、自動割付不可能な場合はダイアログオープン　など）

ダイアログ内の「閉じる」ボタンなどのダイアログを閉じる処理

1. ダイアログのjspでデザイン通りに記載すれば閉じられる。

```
<p class="asw-btn-area-left"><input type="submit" value="閉じる" class="
asw-btn-base  asw-btn-cancel  asw-btn-width-variable  asw-btn-vertical-main  asw-btn-w200  asw-btn-h43  dial"></p>e
```

閉じる前に処理を挟みたい場合は下記「ダイアログを閉じる前に処理を挟みたい場合」を参照。
（e.g. 閉じる前に閉じてもよろしいですか？のアラートを表示するなど）

ダイアログ内の「決定」ボタンなどのformをサブミットする処理

1. 子画面JSPの「決定」ボタンのclassからdialog-closeを削除する（TCIデザインに含まれているはず）
2. 子画面のjsに以下のようにsubmitのイベントハンドラを記載する
   ◦ 例1:Ajaxで処理を行い、処理結果を基にダイアログを閉じるか否かを切り分ける
   例1:Ajaxで処理を行い、処理結果を基にダイアログを閉じるか否かを切り分ける

```
$(function()  {
    'use  strict';
```

```
    var contextPath = $('meta[name="contextPath"]').attr('content');
    var $dialog = $('#modal-dialog-form');                    // <1>

    // submit
    $dialog.on('submit', '#dialog-form', function() {     // <2>
        var $form = $(this);
                    atc.ajax(con'/search/DSBP0260/writeCartSession', {
            type: 'POST',
            data: $form.serialize(),
            context: {
                errorMessageArea: '#dialog-error-message-area', // <3>
                form: $form
                            }
                function() {
            // ダイアログを閉じる
            $dialog.find('.dialog-close').click();          // <4>
                    });

        return false;                                       // <5>
          });

});
```

<1> 親画面のボタンクリックなどでダイアログを開く処理-対応手順2<1>で指定したidを指定
<2> 2つ目の引数は子画面のformのidを指定
<3> エラー時にダイアログ画面上部へエラー文言を出したい場合は Ajaxのエラーハンドリング に従う
<4> ダイアログを閉じる場合はこの通りに記載する
<5> return false;することでsubmitさせない

Appendix

ダイアログを開く前に処理を挟みたい場合

1. 親画面のダイアログを開くためのボタンのdata-callback-func-open属性に、挟み込むcallback関数名を記載する

```
<input type="button" value="上記の内容で次へ進む" class="省略"
  data-dialog-popup-form="#modal-dialog-form"
  data-dialog-fadespeed="500"
  data-atd-dialog-content-url="${dialogContentUrl}"
  data-callback-func-open="handleAssignDialogOpen">          <!-- <1> -->
```

   - <1> ダイアログを開く前に実行したいcallback関数名を指定する。 関数名は「handle～～DialogOpen」とすること。
2. 親画面のcontent.js内に以下の例のようにglobal関数を定義する。
   - 例1:Ajaxで業務的な判定を行い、判定結果を基に画面遷移かダイアログオープンを切り分ける
     例1:Ajaxで業務的な判定を行い、判定結果を基に画面遷移かダイアログオープンを切り分ける

```
$(function() {
  'use stri';

  var contextPath = $('meta[name="contextPath"]').attr('content');

  /**
       * PKG割付ダイアログオープンハンドラ.
       * <p>
       * 自動割付可能な場合は○○画面へ遷移、不可能な場合は割付ダイアログを開く。
       */
    windchandleAssignDialogOpen =function(event) {                    // <1>
    var that this;

    return atc.ajax(contextPath '/search/DSBP0250/checkAutoAssign', {          // <2>
```

```
        type: 'POST',
        data:  {
          dummy: 'dummy'
                }
            }).function(autoAssignEnabled)  {
        if  (autoAssignEnabled)  {                                          //  <3>
          //  自動割付可能な場合、○○画面へ遷移
          var  queryStr  $.param({ deptDt:'2016-11-29', useDd:'2016-12-01', relUseDd: 5});
                    window.location.href  =  c'/search/DSBP0250/init?'  +  queryStr  ;
//  TODO:遷移先設定(仮で自画面遷移させている)  //  <4>
          return $.Deferred().reject(); //  ダイアログオープンはキャンセル                    //  <5>
          else  {}
          //  自動割付不可能な場合、割付ダイアログのコンテンツを取得・設定して表示
          return  window.handleDialogOpen.call(that,  event);                  //  <6>
                }
            });
    }

});
```

<1> 手順1-<1>で指定したcallback関数名に合わせる。window.を付けることでglobalに公開する。※
<2> 業務的な判定結果をAjaxで取得
<3> 判定切り分け
<4> 遷移させる場合はlocation.hrefで遷移
<5> こう記載するとダイアログオープンをキャンセルできる(dialog.js仕様)
<6> ダイアログのコンテンツを取得してダイアログを開く際はこの通りに記載

- 例2:非Ajaxで判定を行い、判定結果を基に何もしないかダイアログオープンを切り分ける
例2:非Ajaxで判定を行い、判定結果を基に何もしないかダイアログオープンを切り分ける

```
$(function()  {
  'use  strict';

  var  contextPath  $('meta[name="contextPath"]').attr('content');

    windchandleAssignDialogOpen  =function(event)  {  //  <1>
    var  that  this;

    if  false /*  判定  )  {          //  <2>
      //  ダイアログオープンしない
      return false;                    //  <3>
          }
    //  ダイアログのコンテンツを取得・設定して表示
    return  window.handleDialogOpen.call(that,  event//  <4>
    }

});
```

<1> 手順1-<1>で指定したcallback関数名に合わせる。window.を付けることでglobalに公開する。※
<2> 判定切り分け
<3> 非Ajaxの場合こう記載するとダイアログオープンをキャンセルできる(dialog.js仕様)
<4> ダイアログのコンテンツを取得してダイアログを開く際はこの通りに記載

※注意：申請なしにglobal関数を定義するのは規約違反だが、dialog.jsの仕様上global関数とする必要があるため、この場合においてのみglobal関数定義を許可する。

「閉じる」でダイアログを閉じる前に処理を挟みたい場合

1. 親画面のダイアログを開くためのボタンにdata-callback-func-close属性を追加し、閉じる前に挟みたいcallback関数名を指定する

```
<input type="button" value="上記の内容で次へ進む" class="省略"
```

```
  data-dialog-popup-form="#modal-dialog-form"
  data-dialog-fadespeed="500"
  data-atd-dialog-content-url="${dialogContentUrl}"
  data-callback-func-open="handleAssignDialogOpen">
      data-callback-func-close="handleDialogClose"   <!--  <1>  -->
```

    - <1> ダイアログを閉じる前に実行したいcallback関数名を指定する。関数名は「handle〜〜DialogClose」とすること。
2. 子画面のjsに以下の例のようにglobal関数を定義する。※
   ※ほとんどこの要件はないと思われるので、confirmダイアログを出す簡単な例のみ記載

```
$(function()  {
  'use  strict';

    windhandleDialogClose  =function(event)  {      //  <1>
    //  falseが返却されると（＝confirmダイアログの「キャンセル」が押下されると）ダイアログが閉じない
    return  window.confirm('閉じてよろしいですか？');
    }
});
```

    - <1> ダイアログを閉じる前にこの名前で指定したcallback関数が実行される。

※注意：申請なしにglobal関数を定義するのは規約違反だが、dialog.jsの仕様上global関数とする必要があるため、この場合においてのみglobal関数定義を許可する。

How-To (SP版ダイアログのトラップを回避して実装する方法)

TCI提供のSP版ダイアログの設計・内部実装に問題があり、その問題を回避して何とか実装するための方法を提示します。
※ ここで示す実装例はあくまで回避方法です。通常はこのような実装をするべきではありません。

TCI提供のSP版ダイアログは、ダイアログコンテンツをcloneしてDOMツリー上に配置していしまいます。
つまり、ダイアログコンテンツ内の要素に付けたid属性値は基本的に重複してしまいます。(HTML仕様違反状態)
従って、ダイアログ上の要素をidでfindすることは無理です。（出来たとしてもブラウザ依存なのでダメです)

ダイアログコンテンツ上の要素に対するイベントハンドラの設定

```
$('ポップアップリンクのdata-targetに指定している要素').on('イベント', 'セレクタ', function()  {
  //  ...
});

//  例
$('#car-dialog').on('change', 'select[name="carType"]', function()  {
  var  $select  =$(this);
  //  ...
});
```

留意事項

- フォームコントロール(input[type="text"]など)のidは変わってしまうので、セレクタには `[name="xxx"]` や data属性を使用すること
    - 内部的には `modal-` + 元のid として付いていますが、この考慮不足な内部実装に依存してはいけません

エラーメッセージエリアの指定

```
            context:  {
    errorMessageArea: '.asw-modal-container  .asw-notice-massage',
            }
```

※ ダイアログ上にダイアログを表示するケースでは、この方法を利用できません

ダイアログを能動的に閉じる

```
        }).function()   {
    //   ダイアログを閉じる
    $('.asw-modal-container    .asw-modal-clos').click();
        });
```

※ ダイアログ上にダイアログを表示するケースでは、この方法を利用できません