目次

改訂情報

変更年月日	変更内容
2012-10-01	初版
2015-08-01	第２版　下記を変更しました。 ■ 「 *実装の詳細* 」の説明を修正

IM-共通マスタの拡張について

IM-共通マスタは以下の下記の3つの方法で拡張することが出来るようになっています。

- APIのリスナー
  IM-共通マスタのAPIには登録や更新などのイベントが発生したタイミングに同期して処理を実行するリスナーを任意の数だけ定義することが出来ます。
  リスナーを使用することでIM-共通マスタに対して更新や削除が実行されたタイミングで、独自の操作や制御を追加することができるようになっています。
  トランザクションは一括して管理されますので、リスナーの処理を含めて成功しなければコミットされません。
- IM-共通マスタ メンテナンス画面のタブ
  IM-共通マスタメンテナンスの編集画面にはタブ型のインタフェースがありますが、拡張情報を編集する画面を新たなタブとして追加することが出来ます。
  これにより拡張情報のテーブルを追加し、IM-共通マスタの編集と同時に更新することが出来るようになっています。
  検索画面でも同様に、検索用のタブを追加することが出来ます。
- 共通検索画面のタブ
  マスタメンテナンス同様、共通検索画面にもタブインターフェースがあり、検索処理を追加する事ができます。
  独自の検索処理を追加のタブとして実装する事で統合された操作を実現しやすくなります。

本ガイドでは上記の拡張それぞれの作成方法を説明します。

前提となる知識

本ドキュメントではIM-共通マスタで実際に提供している機能を元に、サンプルとなるコードを提示して実装方法を説明しています。
その前提として、APIのリスナーを作成する際には一般的なJavaの知識、また画面のタブ拡張を作成する際にはスクリプト開発モデルについての知識が必要になります。
その他の技術的な要素として、PluginManagerとim-JSPackmanについての知識が必要になりますが、これらについて本ガイドでは詳細に解説していません。適時、次に上げるドキュメントを参照して下さい。

Plugin Manager

IM-共通マスタのマスタメンテナンス画面やAPIはプラグインという形で機能を拡張できるようになっています。
プラグインを追加する場合には、拡張ポイントに応じた内容でプラグインの実装を作成し、対象の拡張ポイントへPluginするための設定ファイルを記述します。
拡張ポイントと、プラグインの関係はPlugin Managerによって管理されます。
PluginManagerは同じ拡張ポイントへの拡張に競合などがあった際は優先度の高いプラグインを自動的に選択します。
IM-共通マスタのマスタメンテナンス画面やAPIの拡張を作成する際、PluginManagerについて理解しておく必要があります。
Plugin Managerの詳細についてはAPIリストのPlugin Managerについての項を参照してください。

im-JSPackman

im-JSPackmanは弊社で開発されたクライアントサイドJavascriptのフレームワークです。
im-JSPackmanを利用する事によってクライアントサイドJavascrtiptのクラス化・パッケージ管理が可能になります。
plugin構造という方針をとるにあたりim-JSPackmanを使用する事によりplugin間での実装の衝突の回避や、依存関係の明確化を図っています。

表記について

本ガイドの中で、環境や実装によって変わる記述について以下のような表記を用いています。
それぞれインストールした環境や、作成した実装に読み替えてください。

表記	意味
%plugin_id%	プラグインのID。プラグイン実装者が定義する値です

APIの拡張

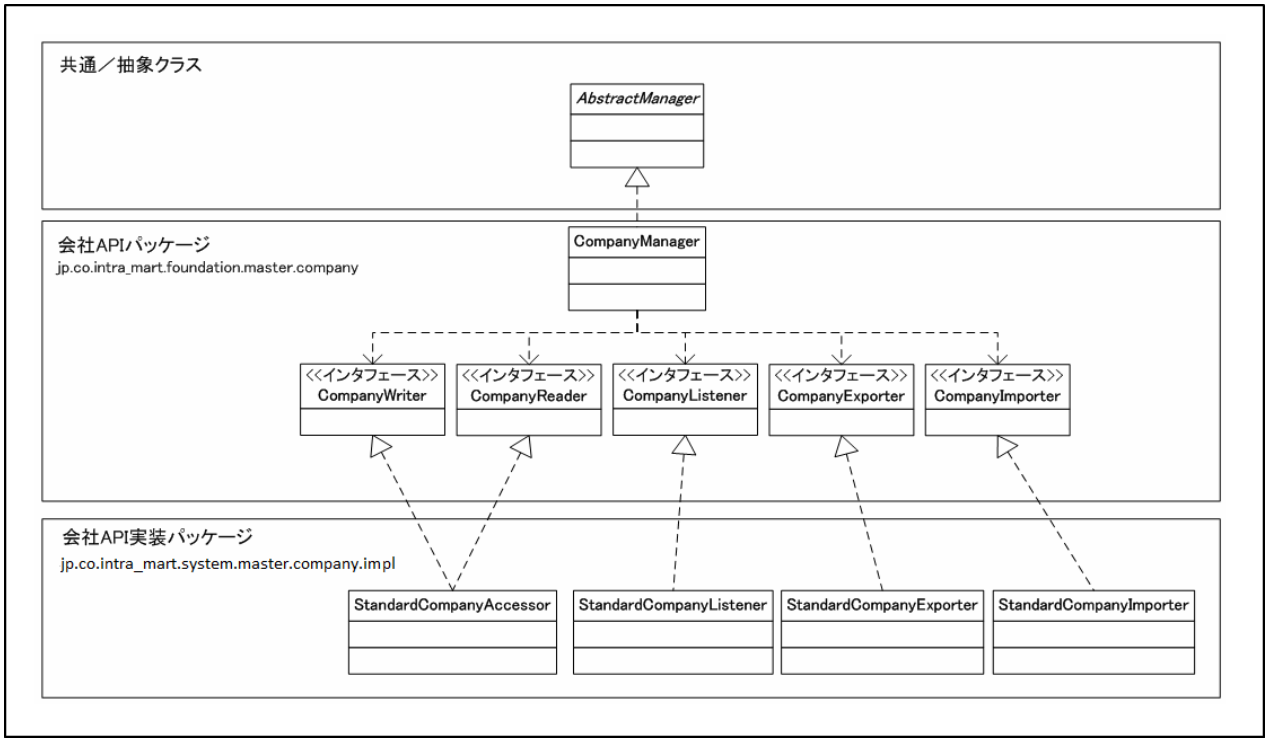
APIの概観

IM-共通マスタのAPIの構造について説明します。
マネージャの内部は実際にデータアクセスを行う実装とは分離されており、Plug-inの仕組みを利用してデータベースへのアクセサを組み込むことでマネージャとしての機能を提供しています。
これにより独自にPlug-inとして実装クラスを作り、インタフェースを変更せずに全く異なる実装を動作させることも出来ます。

マネージャクラスと実行クラスのインタフェース

マネージャは全てAbstractManagerを継承した形になっています。
Manager自身は直接データベースへアクセスすることはありません。
データベースへ直接アクセスを行う実体のクラスはインタフェースを介して別のパッケージへ分離されています。

会社組織マネージャでのクラス構造を【図：会社組織マネージャの例】に示します。
図のように各マネージャに対して、実装クラスのインタフェース5種（Writer, Reader, Listener, Exporter, Importer）が必ず定義されています。



【図：会社組織マネージャの例】

- 共通／抽象クラス
  - AbstractManager
    各マネージャの共通処理を管理する抽象クラスです。
- 会社APIパッケージ
  - CompanyManager
    会社組織マネージャの実装クラスです。
  - CompanyWriter
    会社組織エンティティへの書き込みをサポートするインタフェース。
    実装クラスが複数定義されていた場合は最も優先度の高いものを一つだけ採用します。
  - CompanyReader
    会社組織エンティティからの読み出しをサポートするインタフェース。

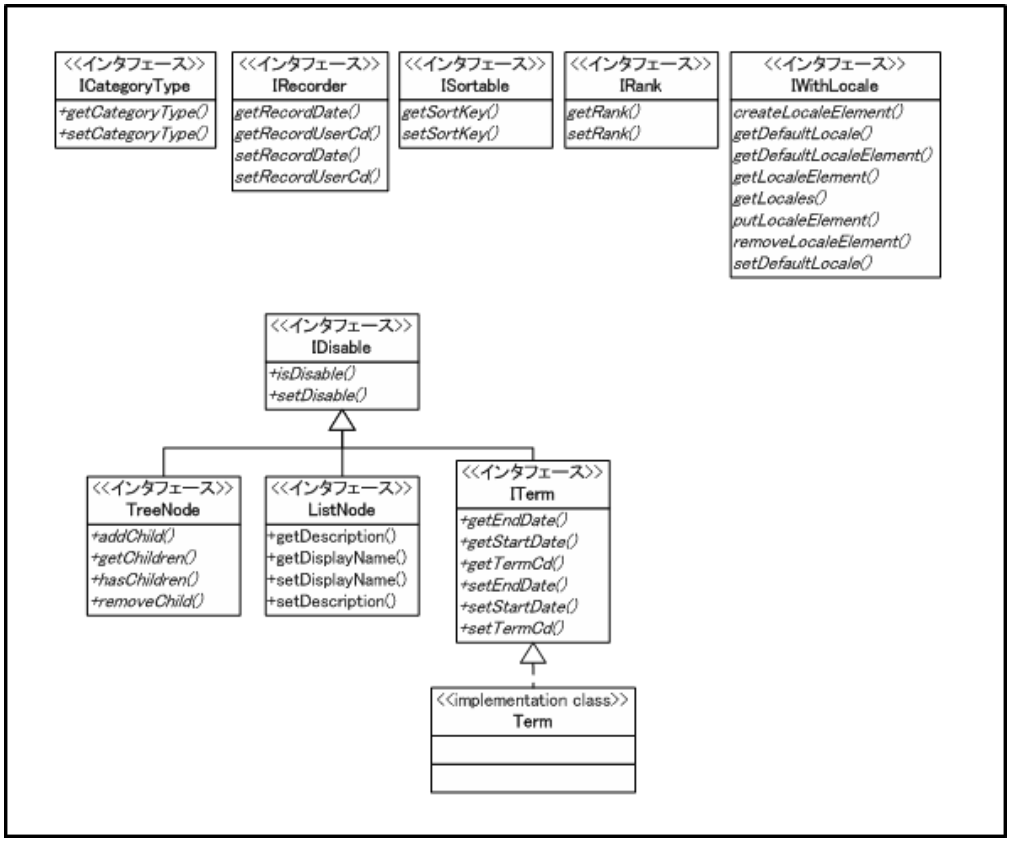
実装クラスが複数定義されていた場合は最も優先度の高いものを一つだけ採用します。

- CompanyListener
  会社組織マネージャにおけるリスナのインタフェース。
  Plug-in として定義されている実装クラス全てを使用します。
- CompanyExporter
  会社組織エンティティのエクスポート処理をサポートするインタフェース。
  実装クラスが複数定義されていた場合は最も優先度の高いものを一つだけ採用します。
- CompanyImporter
  会社組織エンティティのインポート処理をサポートするインタフェース。
  実装クラスが複数定義されていた場合は最も優先度の高いものを一つだけ採用します。

各インタフェースの詳細についてはIM-共通マスタのAPI リストを参照してください。

モデルクラスのインタフェース

マネージャのメソッドを実行した際に返されるモデルオブジェクトは数種類のインタフェースを組み合わせて実装されています。
モデルとして使用されているインタフェース群の構成を【図：モデルクラスのインタフェース群】に示します。



【図：モデルクラスのインタフェース群】

各インタフェースの概要を以下に説明します。詳細についてはAPI リストを参照してください。

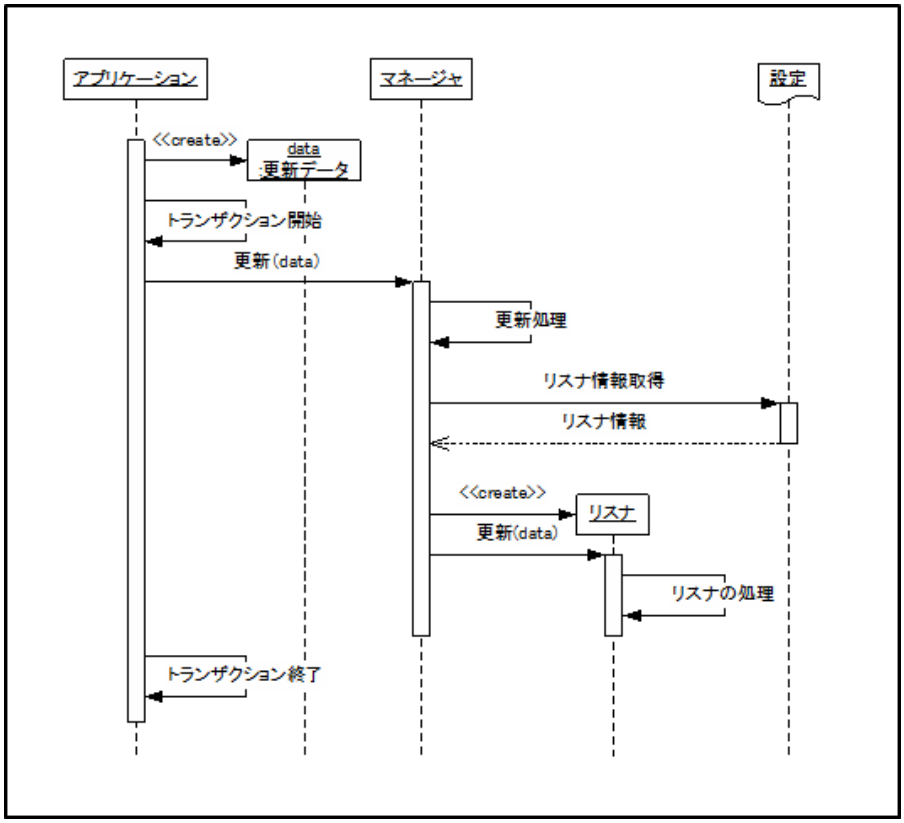
- ICategoryType
  分類(ユーザ分類、組織分類、パブリックグループ分類)の付属するモデルであることを表します。
- IRecorder
  更新日、更新ユーザを記録するモデルです。
- ISortable
  ソートキーを保持しており、任意の順序を設定できるモデルであることを表します。
- IRank
  ランクによる順位付けが可能なモデルであることを表します。
- IWithLocale

国際化情報を取り扱えるモデルであることを表します。

- IDisable
  削除フラグを保持しており、論理削除を行うことが可能なモデルであることを表します。
- TreeNode
  階層型構造を保持できるモデルであることを表します。
- ListNode
  一覧化されたモデルのビューであることを表します。
- ITerm
  期間化可能なモデルであることを表します。

API（マネージャ）へのリスナーの追加

ここではリスナーの追加方法について説明します。
マネージャがデータベースの情報に何らかの変更をしようとするタイミングで、任意の処理を追加することが出来ます。これをリスナーと呼びます。
各アプリケーションからIM-共通マスタへのデータの変更は各マネージャを通して行う必要がありますが、リスナーを追加しておけばIM-共通マスタの情報の更新と連動して独自のデータも更新したり、追加のチェック処理を実装したりすることが出来ます。
リスナーは各マネージャに対応するリスナーインタフェースを使用して実装します。
リスナーインタフェースに定義されているメソッドは大きく追加・変更・削除のものがあり、それぞれマネージャがデータベースを操作するタイミングで呼び出されます。
リスナーの動作の概要を 【図：リスナーの動作概要】 に示します。



【図：リスナーの動作概要】

用意されているリスナーインタフェースについては「*マネージャの拡張に関する情報*」に一覧を掲載していますが、各メソッドがどのようなタイミングで呼び出されるかについてはIM-共通マスタのAPIリスト、及び「IM-共通マスタ Listener定義一覧」に詳細に記載していますので、そちらを参照して下さい。

リスナーの実装の為に必要な作業は次の二つです。

- リスナークラスの実装
- plugin.xml の作成

以降、順に説明します。

リスナークラスの実装

リスナーインタフェースを実装したリスナークラスを実装します。

リスナーの処理はJavaで実装します。マネージャによってそれぞれリスナーのインタフェースが定義されていますので、行いたい処理によって対応するインタフェースを実装する必要があります。

下記の【リスト：会社組織マネージャのリスナーの例】では会社組織マネージャのリスナーの例です。会社組織の役職が登録される際に別の処理を実装するために、CompanyListenerインタフェースを実装し、createCompanyPost メソッドをオーバーライドして処理を記述しています。

```
public class SynchronousCompanyListener implements CompanyListener {
/**
* 役職を作成した際に呼び出されるListener メソッド
*/
@Override
public void createCompanyPost(IAppCmnInfo info, List<CompanyPost> list)
  throws BizApiException {
try{
  ICompanyPostBizKey postBizKey = list.get(list.size() - 1);
  if(!postBizKey.getCompanyCd().equals(postBizKey.getDepartmentSetCd())){
    return;
  }
  /*・・略・・*/
  } catch (AccessSecurityException ex) {
    throw new BizApiException(ex);
  }
}
  /*・・略・・*/
```

【リスト：会社組織マネージャのリスナーの例】

各リスナーインタフェースにはいくつかのメソッドが定義されており、特定のタイミングでマネージャから呼び出されます。マネージャと対応するリスナーインタフェースの詳細は、別途公開されているドキュメント「IM-共通マスタ Listener定義一覧」にまとめられていますので、そちらを参照して必要なインタフェースを実装して下さい。

実装したリスナークラスのクラスファイルは intra-mart Accel Platform のclasspath の通ったパスに配備します。

plugin.xmlの作成

APIのリスナーはPlug-inの形で追加しますのでplugin.xmlを作成してPluginManagerによって読み込めるようにします。
プラグインはあらかじめ用意された拡張ポイントに対して機能を追加する機能です。
リスナーを追加する際は各マネージャが用意している拡張ポイントに対してプラグインを設定するという形で追加します。
各マネージャが用意している拡張ポイントについては「*マネージャの拡張ポイント一覧* 」を参照してください。
会社組織マネージャの拡張ポイントへプラグインを設定する設定ファイル「plugin.xml」の例を【リスト：会社組織マネージャへの拡張を行うplugin.xml】に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension
    point="jp.co.intra_mart.foundation.master.accessor.company" >
    <accessor
      name="standard"
      id="jp.co.intra_mart.standard"
      version="8.0.0"
      rank="1" >
      <listener class="jp.co.intra_mart.system.master.sync.SynchronousCompanyListener" />
    </accessor>
  </extension>
</plugin>
```

【リスト：会社組織マネージャへの拡張を行うplugin.xml】

plugin.xml の書き方や構成についての詳細はPluginManager のAPI ドキュメントを参照してください。
ここではリスナーの追加に当たって必要な範囲で説明します。

- extension タグ
  - point 属性： プラグインを追加する拡張ポイントを指定します。この例では会社組織マネージャの拡張ポイントを指定しています。
- accessor タグ
  - name、id、version、rank などの属性はPluginManager によって依存関係の管理などに使用されます。PluginManager のドキュメントを参照して下さい。
- listener タグ
  - listener として追加する実装の情報を記述するタグです。
  - class 属性にはPlug-in として追加するリスナーインタフェースを実装したクラスのFQDNを指定します。

このxmlファイルをPluginManagerの管理するディレクトリに配置します。具体的には以下のパスになります。

`<（展開したwar）/WEB-INF/plugin/%plugin_id%/plugin.xml>`

plugin.xml を変更した場合は、intra-mart の再起動が必要になります。

以上でリスナーの実装は完了です。

マスタメンテナンス画面の拡張

マスタメンテナンス機能の詳細情報の更新画面や、検索条件を入力する画面の多くはタブを持ったインタフェースになっています。
これらの画面はタブを追加することで拡張をすることが出来ます。

- 詳細画面にタブを追加することで編集する情報を拡張することが出来るようになります。
- 検索画面にタブを追加することによって既存とは異なる独自の検索を行うことが出来るようになります。

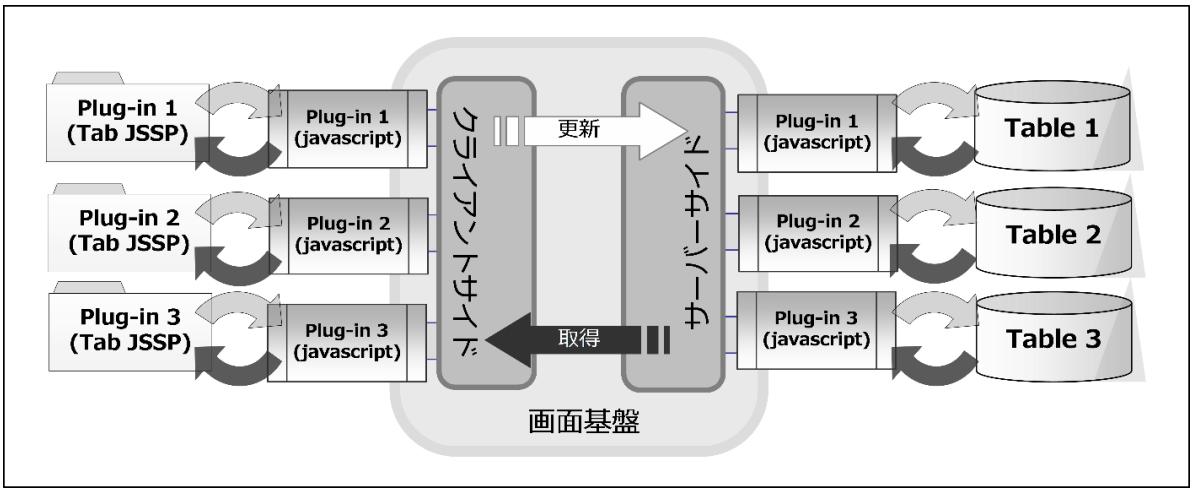
まずタブ画面の動作の概要を説明した上で、実装の方法について説明します。

詳細画面（登録・編集画面）の拡張

詳細画面のタブ拡張について、まず動作の概要を説明し、実装の詳細について説明します。

詳細タブの動作概要

IM-共通マスタメンテナンス画面はプラグインをコントロールする基盤部分と、画面（タブ内）にデータを表示したり、実際にテーブルへのデータ入出力を行ったりするプラグインという構成で成り立っています。
詳細画面の構成の概要を【図：詳細タブの動作概要】に示します。



【図：詳細タブの動作概要】

- クライアント側へ情報を表示する際、まずサーバサイドで画面基盤が各プラグインを呼び出します。
  プラグインはそれぞれデータを取得し、画面基盤は各プラグインからデータを受け取り、受け取ったデータをまとめてクライアントサイドへ転送します。
  クライアントサイドでは転送したデータをそれぞれ対応するプラグイン毎に受渡します。
  クライアントサイドのプラグイン処理は受け取ったデータをタブに描画するなどの処理を行い、ユーザがデータを編集できるようにします。
- ユーザがデータの編集を行い、画面の更新ボタンが押されると、画面基盤はクライアントサイドの各プラグインに永続化する情報を収集するよう依頼します。
  クライアントサイドの各プラグイン処理はそれぞれのタブに入力された情報を収集して画面基盤に返します。
  画面処理基盤は全てのタブのデータを集めて一括してサーバサイドへ転送しクライアントサイドから受け取ったデータをサーバサイドの各プラグイン処理に渡します。
  サーバサイドのプラグイン処理は受け取ったデータをもとにテーブルの更新処理などを行います。

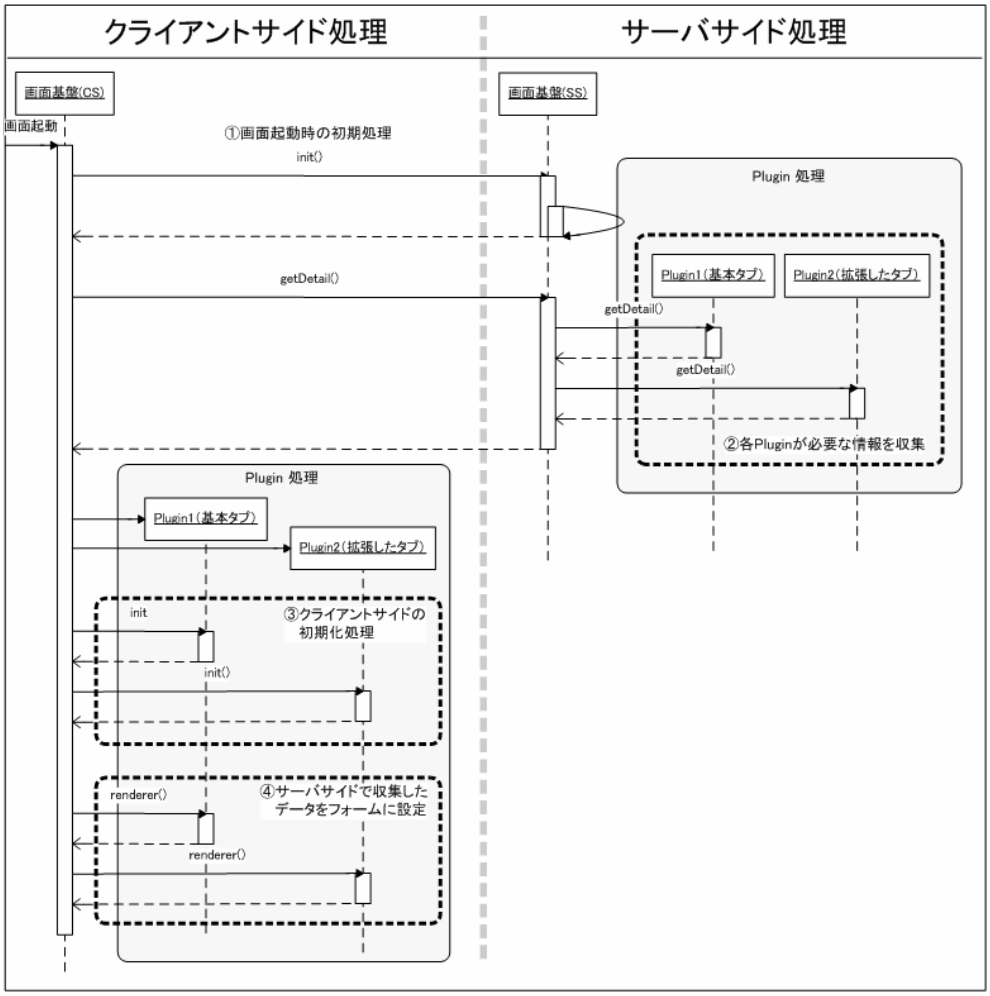
以上から、プラグインとしてタブを追加するために、以下のような実装が必要となります。

- タブを描画するための HTML,またはJSSP
- クライアントサイドでタブと画面処理基盤とのデータの受け渡しを行うjavascript
- サーバサイドで永続化されたデータを入出力する javascript

他にpluginの設定ファイル(plugin.xml)が必要になりますが、詳細は「*詳細タブの実装*」で解説します。

画面起動時のシーケンス

画面起動時の動作シーケンスを【図： タブ画面起動時のシーケンス】に示します。



【図：タブ画面起動時のシーケンス】

上記はタブが二つ（基本タブと拡張したタブ）ある場合の例です。処理の順番に以下に概要を説明します。

1. init (画面起動時の初期処理／サーバサイド処理)
   クライアントサイドからの画面を表示するリクエストです。ユーザによって画面の表示が行われた場合に、まずこの処理から実行されます。
   画面処理基盤が初期化やプラグイン情報の収集処理を行います。

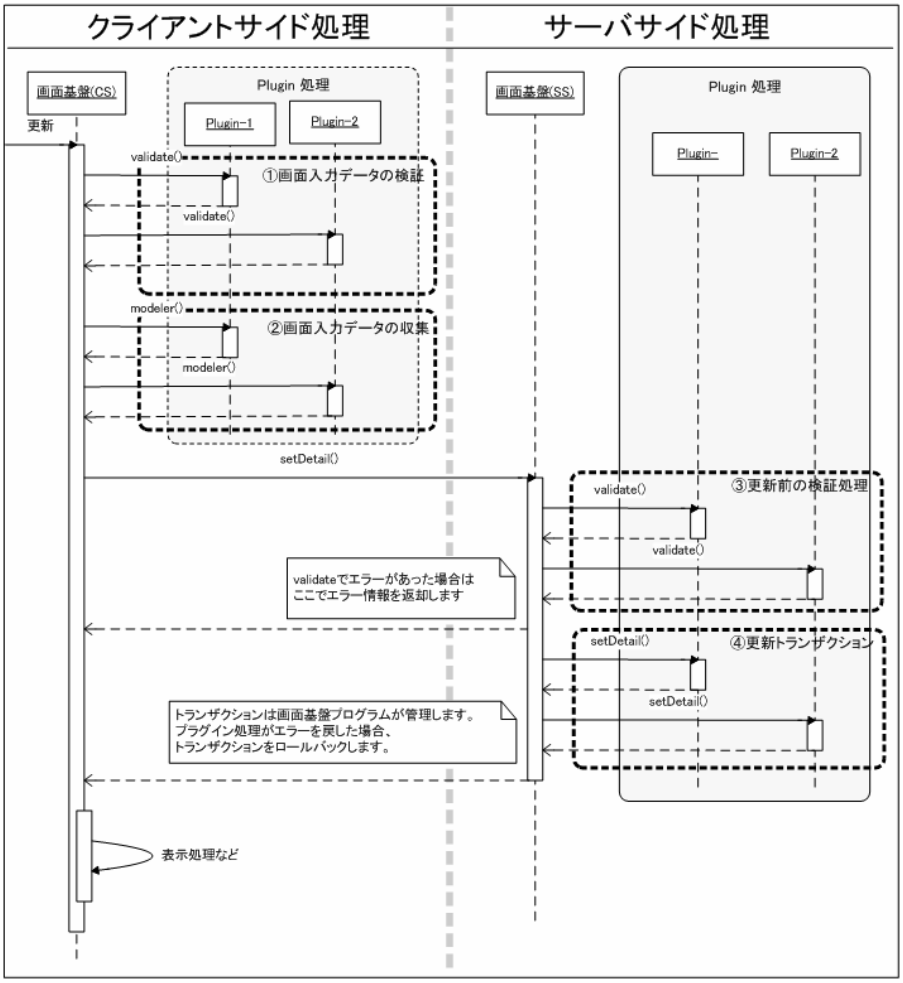
2. getDetail (表示情報の取得処理／サーバサイド処理) :
   詳細情報取得のリクエスト。画面起動時には自動的に発生します。ほかにも画面で最新の情報が必要になった場合等に発生します。
   このタイミングで各プラグインのサーバサイドの処理が呼び出されます。引数に前提条件となる情報が渡されますので、各プラグインは画面に表示するための情報を取得して返します。

3. init (画面の初期化処理。クライアントサイド処理)
   初期化処理を実行するために、各プラグインのクライアント処理が呼び出されます。
   特に画面処理基盤側から期待する処理や戻り値はありません。

4. renderer(画面の内容表示処理クライアントサイド処理)
   各タブの情報を表示するために各プラグインのクライアント処理が呼び出されます。
   上記 2 で述べたサーバサイドでの処理が返したデータがそのまま引数に渡されます。

更新時のシーケンス

ユーザが更新のアクションを起こした際に発生する、更新処理のシーケンスを【図：更新処理実行時のシーケンス】に示します。



【図：更新処理実行時のシーケンス】

1. validate(入力値検証処理／クライアントサイド処理)
   ユーザの操作により更新処理が実行された場合に、まずクライアントサイドで各タブの入力チェック処理の契機としてvalidate関数が実行されます。プラグインで各タブの内容をチェックし、問題がある場合はエラーを返します。
   validateは全てのタブに対して実行され、一つでもエラーが返された場合、画面処理基盤はエラーメッセージを表示して更新処理を中断します。

2. modeler(入力値収集処理／クライアントサイド処理)
   各タブのユーザ入力値を収集するために、画面処理基盤が各プラグイン処理のmodeler を実行します。この処理は各タブでユーザが入力した値をJSON 文字列で表現可能なObject として返す必要があります。これはサーバサイドのプラグイン処理にそのまま渡されます。
   画面処理基盤は全てのタブに対して modeler を呼び出し、全ての情報を一括してサーバサイドへ送信します。

3. validate(入力値検証処理／サーバサイド処理)
   データベースに保管されているデータとの関連チェックなどのために、再度plugin に対して入力値の検証を行う機会があります。
   サーバサイドでもクライアントサイド同様にvalidate 関数（実際の関数名はplugin.xml ファイルによってマッピングされます）が呼び出されます。

4. setDetail(更新処理／サーバサイド処理)
   クライアントサイドから収集してきたデータを更新するために各プラグインのsetDetail 関数（これも実際の関数名はplugin.xml ファイルによりマッピングされます）が呼び出されます。
   引数には全てのプラグインのデータが渡されます。処理が成功した場合は最新の情報を取得し直して返します。

詳細タブの実装

これまでに説明した通り、タブを追加するためには次のものを準備する必要があります。

- プラグインの構成情報を記述する plugin.xml
- タブ内の UI を構築するJSSP(js+html)
- クライアントサイドでデータの入出力やイベントをハンドリングするjs ファイル
- サーバサイドでデータにアクセスするロジック(js ファイル)

以下では、この他の実装の詳細を順を追って説明します。

プラグインの構成情報を記述する plugin.xmlの作成

ここでは品目詳細画面の所属タブを例にとって説明します。
品目詳細画面の所属タブの例を【リスト：品目詳細画面所属タブのplugin.xml】に示します。

```
<?xml version="1.0" encoding="utf-8"?>
<plugin>
  <extension point="jp.co.intra_mart.foundation.master.setting.item.detail.contents">
    <detail_item_attach_config
      name="standard"
      id="jp.co.intra_mart.master.item.detail.attach"
      version="8.0.0"
      rank="3">
      <content
        title="%tabname"
        csjs="im.app.master.plugins.item.detail.tabs.attach.ItemAttach"
        page="master/plugins/maintenance/item/detail/tabs/attach/item_attach"
        min_height="280"
        />
      <operations page="master/plugins/maintenance/item/detail/tabs/attach/item_attach" object="">
        <operation id="get_detail" method="getDetail"></operation>
        <operation id="set_detail" method="setDetail"></operation>
        <operation id="validate" method="validate"></operation>
        <operation id="remove" method="remove"></operation>
      </operations>
    </detail_item_attach_config>
  </extension>
</plugin>
```

【リスト：品目詳細画面所属タブのplugin.xml】

plugin.xml の書き方や構成についての詳細はPlugin Manager のマニュアルを参照してください。
ここではリスナーの追加に当たって必要な範囲で説明します。

- extension タグ
  - point属性：タブを拡張したい対象の拡張ポイントを指定します。
    IM-共通マスタのメンテナンス画面で用意している拡張ポイントについては「*マスタメンテナンス画面の拡張に関する情報*」で説明していますので参照してください。
- detail_item_attach_config タグ
  - このタグ名は任意の名称が使用可能です。
  - name、id、version、rank などの属性はPlugin Manager によって依存関係の管理などに使用されます。詳細はPlugin Manager のドキュメントを参照してください
- content タグ
  - title 属性：追加したタブに表示されるタイトルを指定します。%表記は国際化メッセージキーを表します。国際化に関してはPlugin Manager のドキュメントを参照してください。
  - csjs 属性：クライアントサイドでタブの情報をハンドリングする実装を指定します。この処理はim-jspackman で管理可能なクラス形式である必要があります。ここにはそのクラスのFQDN を指定します。
  - page 属性：タブ内のUI を構築するJSSP のパスを指定します（拡張子無し）
  - min_height 属性：最低限必要な表示高さをpixel で指定します。特に指定しなければ他のTab の要求サイズや実際の表示領域の大きさを勘案して自動的に決定します。
- operations タグ

    - page 属性：サーバサイドでデータにアクセスするロジックのjs ファイルのパスを指定します(拡張子無し)
- operation タグ：処理と実装をマッピングする情報です。画面処理基盤側が必要とするid に対してmethodのマッピングを定義します。
    - id属性：詳細・更新タブで定義する必要のあるidは 【表：サーバサイドスクリプトで実装する必要のある処理】で挙げているget_detail、set_detail、validate、removeの4 つです。operationタグが4 つ必要になります。
    - method 属性：operations タグのpage 属性で指定したファンクションコンテナ内のメソッド名を指定します。上記のID に対応する形で記述してください。

この xml ファイルをPlugin Manager の管理するディレクトリに配置します。具体的には以下のパスになります。

`<（展開した war）/WEB-INF/plugin/%plugin_id%/plugin.xml>`

plugin.xml の変更を行った場合には再起動が必要になります。

タブ内のUIを構築するJSSP(js+html) の作成

JSSP内の処理に関してはタブの初期表示に必要な処理以外には特別な記述を行う必要はありません。初回のinitの引数が通常と異なりますので注意が必要です。
initの引数には詳細画面が受け取っている引数がObjectでとして渡されます。内容は機能によって異なりますが、概ね以下の内容が渡されます。

- 表示中の基準日やロケールなど、画面表示に使用する基本情報（basicInfo）
- タブ内を描画するにあたり渡されるパラメータ(parameters)
    - 画面が新規登録として開かれているか、編集として開かれているかを表すフラグ(isEdit)
    - 現在選択中の期間情報（term）
    - 編集として開いている場合、編集する対の情報のビジネスキー(recordInfo)
    - 所属構造を持っており、画面操作の流れ上所属先が選択されている場合であればその所属先のビジネスキー(parentInfo)

詳細な内容は機能によって異なります。詳細はIM-共通マスタで提供する各拡張ポイントについて、詳細を記載したドキュメント「IM-共通マスタ 拡張インタフェース定義一覧」をご覧確認下さい。

ここで作成したファイルは 「*プラグインの構成情報を記述する plugin.xmlの作成*」でcontentタグのpageプロパティと一致する場所に配置します。

クライアントサイドでデータの入出力やイベントをハンドリングするjsファイルの作成

データの出し入れに関しては画面基盤プログラムがサーバサイドとクライアントサイドでデータの通信を一括して取り扱います。このため、クライアントサイドではデータの表示や、ユーザの入力したデータの収集を各Pluginのクライアントサイドスクリプトに依頼します。
画面基盤プログラムは画面表示の更新が必要になったタイミングや、更新ボタンが押された等、特定のタイミングでクライアントサイドスクリプトの特定のメソッドを呼び出します。
クライアントサイドスクリプトはim-JSPackmanの実装方式にしたがって定義したクラスである必要があります（plugin.xmlに指定したクラス名から、画面基盤が動的にロードしてインスタンス化します）。
クライアントサイド処理で必要な処理を 【表：クライアントサイドスクリプトに必要なメソッドの一覧】 に示します。

	メソッド名	戻り値	説明
1	init(window)	void	タブを初期表示した時点で呼び出されます。引数にはタブ自身を表すwindowオブジェクトが渡されます。 戻り値は必要ありません。

	メソッド名	戻り値	説明
2	renderer(window,model)	void	画面の表示を更新する必要がある場合（ユーザ操作による画面表示更新操作など）に呼び出されます。引数にはタブ自身を表すwindowオブジェクトと、サーバサイド処理の取得処理(plugin.xmlでid:get_detailに指定された処理）を実行して返された内容(model)をそのまま渡します。 戻り値は必要ありません。
3	validate	Object	次で説明するmodelerが呼び出される直前に呼び出されます。 クライアントサイドで可能な範囲で入力値検証処理を実装できます。 戻り値の形式は以下のとおりです。 プロパティ / 型 / 説明 error / boolean / エラーの有無を表すフラグ message / Array / 下記オブジェクトの配列 1_n (配列添字) / code / number / Plugin任意のエラーコード 1_n (配列添字) / message / string / エラーメッセージ とくに問題がなければerror をtrue に設定して返して下さい。
4	modeler(window)	Object	画面のデータを収集する必要がある場合（更新時など）に呼び出されます。この関数の戻り値の内容を各タブのplugin idをキーとした連想配列に収めた物がサーバサイドの更新処理(plugin.xmlでid:set_detailに指定された処理)に渡されます。 特に形式はありませんがJSONで表せる内容である必要があります。
5	isUpdated(window)	boolean	画面基盤側がタブの内容がユーザによって変更されているかチェックしたい場合に呼び出されます。（画面遷移時などに編集中データが残っていないかの確認に使用）。画面の内容が変更されているか否かをtrue/falseで返してください。 画面基盤側はtrueを受け取ると編集中の情報を破棄して良いかをユーザに問い合わせるダイアログを表示します。

【表：クライアントサイドスクリプトに必要なメソッドの一覧】

これらの処理を、「*プラグインの構成情報を記述する plugin.xmlの作成*」でcontentタグのcsjs属性に指定するクラスに実装します。
以下に品目詳細画面 所属タブの例を示します。

```
Package("im.app.master.plugins.item.detail.tabs.attach");

/**
* 品目詳細画面 所属タブのクライアントサイドスクリプト
*
*/

Class("im.app.master.plugins.item.detail.tabs.attach.ItemAttach").define(

  im.app.master.plugins.item.detail.tabs.attach.ItemAttach = function () {

/* ・・・ 略 ・・・ */

    /**
    * init
    * クライアントサイドの初期化処理
    * タブのロードが完了したタイミングで呼び出されます。
    **/
  this.init = function(tabwindow,basicInfo,param) {

  };

  /**
```

```
   * renderer
   * クライアントサイドの描画処理
   * 初期表示、及び、タブ内の再描画が必要なタイミングで呼び出されます。
   **/
  this.renderer = function(tabwindow,model,parameters) {

    tabwindow.attach_term_data = model.data ? model.data : new Array();
   /* ・・・ 略 ・・・ */

  };

  /**
   * modeler
   * クライアントサイドの情報取得処理
   * クライアントサイドの情報をサーバサイドに転送する際に呼び出されます。
   * このメソッドはタブ内で編集された情報をオブジェクトに纏めて返します。
   **/
  this.modeler = function(tabwindow) {
    var o = new Object();
    o.termsInfo = tabwindow.attach_term_data;
    o.removedParents = tabwindow.removedParents;
    return o;
  };

  /**
   * validate
   * クライアントサイドの検証処理
   * modelerの直前に呼ばれ、クライアントサイドで可能な範囲で値の検証を行います。
   * エラーがあればエラーフラグ・エラーメッセージを設定したオブジェクトを返します。
   **/
  this.validate = function(contentWindow){
    return {};
  };

  /**
   * isUpdated
   * 画面処理基盤が画面を閉じようとしたり遷移しようとした場合に
   * タブ内に編集中のデータが無いか確認するために呼び出します。
   * trueを返すと、ユーザに対し編集中のデータがあるが続行して良いか
   * 問い合わせるダイアログを表示します。
   **/
  this.isUpdated = function(tabwindow){
    var propertyCount = 0;

    for( var i=0; i<tabwindow.attach_term_data.length; i++  ){
      if(tabwindow.attach_term_data[i].modified || tabwindow.attach_term_data[i].created) {
        return true;
        }
      }
      if(tabwindow.removedParents && tabwindow.removedParents.length > 0){
        return true;
      }

      return false;
    };

    /* constructor */
    {
      this.superclass();
    }
  }

);
```

【リスト：品目詳細画面所属タブのクライアントサイドjs の例】

クライアントサイドスクリプトの配置場所は <（展開したwar）/csjs> 配下に、パッケージ名にあわせてディレクトリを作成して配置して下さい。
上記の品目詳細画面 所属タブの例では、パッケージがim.app.master.plugins.item.detail.tabs.attach、クラス名がItemAttachなので、実際には以下のファイルになります。

<（展開したwar）/csjs/im/app/master/plugins/item/detail/tabs/attach/ItemAttach.js>

サーバサイドでデータにアクセスするロジック(jsファイル) の作成

クライアントサイドでmodeler()が収集した情報をデータベースに反映したり、要求された情報を検索して返す処理を実装します。
実装する処理のファイル名や関数名は「*プラグインの構成情報を記述する plugin.xmlの作成*」で定義している値と一致していなければなりません。
operationsタグのpage属性と一致する場所にjsファイルを作成し、operationタグのmethod属性に指定した名前と同じ名前で関数を実装して下さい。
詳細画面で実装する必要のあるメソッドは以下の通りですが、引数や期待される戻り値の詳細な定義は機能ごとに異なります。別途公開されている「IM-共通マスタ 拡張インタフェース定義一覧」をご確認下さい。

	処理（plugin.xml上のid）	**説明**
1	取得処理 (id:get_detail)	編集対象の詳細情報を取得して画面に表示する必要がある場合にこの取得処理が呼び出されます。 この処理の結果がクライアントサイドのrenderer()にそのまま渡されますので、連携の取れるようにデータを返してください。 また、エラーが発生した場合は戻り値のエラーフラグをtrueに設定しておくことで、処理を中断させることが出来ます。
2	検証処理 (id:validate)	編集画面で編集された情報をデータベースに更新する直前に、内容を検証するために呼び出されます。引数にはクライアントサイドの全てのタブについて、modeler()が集めた形式のオブジェクトがplugin-idをキーとした連想配列として渡されます。Plugin側ではこの連想配列から自身のPlugin-idに一致するデータを取り出して処理する必要があります。 エラーが発生した場合は戻り値のエラーフラグをtrueに設定しておくことで、処理を中断させることが出来ます。 ※値検証は複数のエラーメッセージを同時に返せるようにmessageプロパティが配列型になっています。
3	更新処理 (id:set_detail)	編集画面で編集された情報をデータベースに更新するなど、永続化を行う処理です。引数の形式は検証処理と同じ内容が渡されます。処理が完了した場合は更新後のデータを戻してください。（取得処理と同じ形式） エラーが発生した場合は戻り値のエラーフラグをtrueに設定しておくことで、画面基盤処理がロールバックをし、エラーメッセージ表示を行います。
4	削除処理 (id:remove)	対象のエンティティを削除する処理です。引数の形式は更新処理と同じ内容が渡されます。 エラーが発生した場合は戻り値のエラーフラグをtrueに設定しておくことで、画面基盤側では処理をロールバックし、エラーメッセージを表示させることが出来ます。

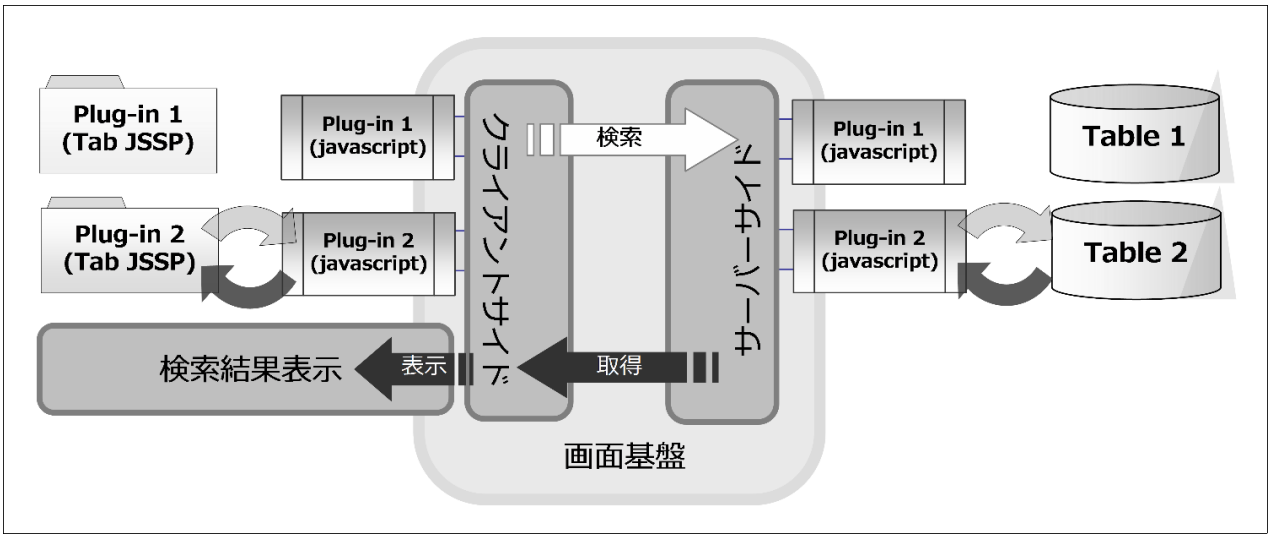
以上で詳細画面のタブ拡張の実装は完了です。

検索画面の拡張

マスタメンテナンス画面の検索機能を拡張する方法を説明します。
ポップアップ型の共通の検索画面の拡張については、「*共通検索画面の拡張*」を参照して下さい。

検索条件タブの動作概要

検索画面ですでに提供されている検索方法では問題がある場合、新たな検索条件入力画面をタブの形で追加することが出来ます。
追加の仕方や動作の概要は詳細画面の場合と似ていますが、タブ横断的な処理は行わない点が大きく異なります。
検索条件タブの動作概要を【図：検索条件タブの動作概要】に示します。



【図：検索条件タブの動作概要】

- 検索処理を行う場合は、クライアントサイドで現在入力中のタブが入力値を収集します。
  画面処理基盤は該当タブのデータのみをサーバへ転送し、サーバサイドの対応するプラグイン処理にその情報を渡します。
  サーバサイドのプラグイン処理は受け取ったデータを元にテーブルの検索処理などを行います。
- 検索結果が画面処理基盤によって、検索結果一覧に表示されます。
  検索結果の一覧は画面基盤によって制御されます。

詳細画面のタブ同様、検索画面のプラグインとしてタブを追加するために、必要なものは以下です。

- タブを描画するためのHTML,またはJSSP
- クライアントサイドでタブと画面処理基盤とのデータの受け渡しを行うjavascript
- サーバサイドで永続化情報からデータを検索するjavascript

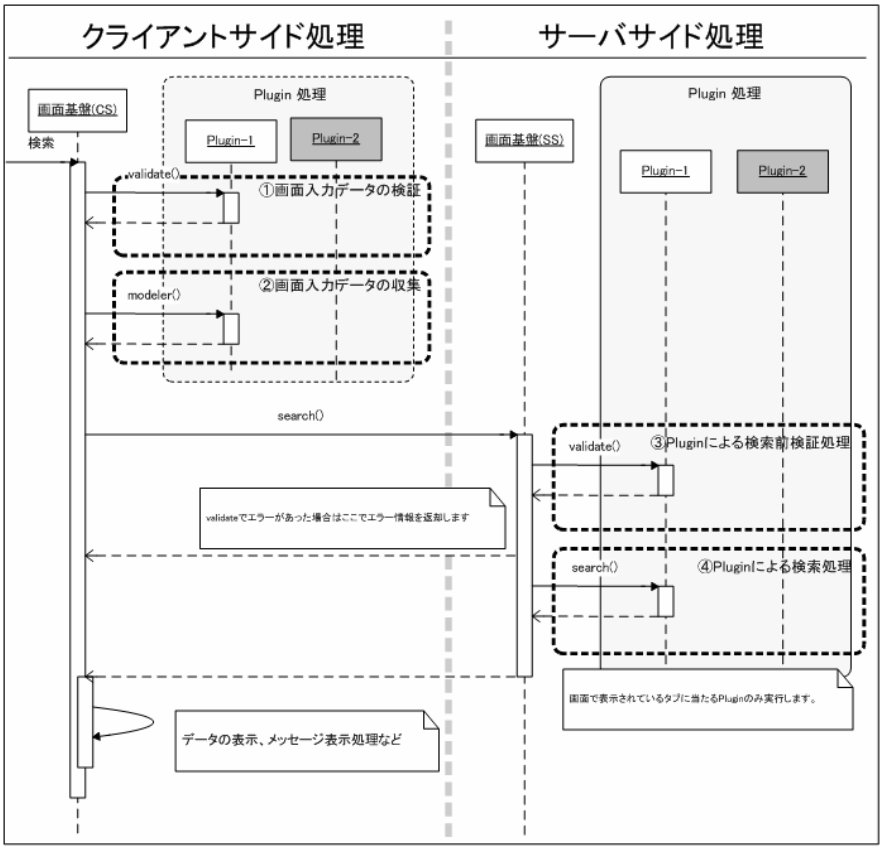
このほかに、pluginとして追加するために設定ファイルを書く必要がありますが、詳細は「*検索タブの実装* 」で解説します。

画面起動時のシーケンス

画面起動時の動作シーケンスは詳細画面と同様です。「*画面起動時のシーケンス* 」を参照してください。

検索時のシーケンス

検索時の動作シーケンスを【図：検索処理実行時の動作シーケンス】に示します。



【図：検索処理実行時の動作シーケンス】

1. validate(入力値検証処理／クライアントサイド)
   ユーザによって検索処理が実行されると、まず現在表示中のタブの入力検証処理を行います。
   画面処理基盤はプラグインのvalidateメソッドを呼び出し、入力内容にエラーがないかを確認します。

2. modeler(入力値収集処理／クライアントサイド)
   入力値の検証にひとまず問題がなければ、ユーザが入力した条件を収集するためにプラグインのmodelerメソッドが呼び出されます。
   このメソッドの処理中にユーザが入力した情報を集め、JSONで表現可能なObjectの形式で戻り値に返します。

3. validate(入力値検証処理／サーバサイド)
   サーバサイドで再度入力値チェックの機会があります。データベース上のデータとの比較が必要な検証処理はこちらで行います。
   エラーがある場合は所定の形式で戻り値を戻します。

4. search(検索処理／サーバサイド)
   実際の検索処理を実行します。ここで返した結果は画面処理基盤によって検索結果欄に表示されます。
   戻り値はオブジェクトの配列ですが、表示するプロパティやどのように表示するかについてはplugin.xmlで指定します。

検索タブの実装

詳細タブ同様、次のものを準備する必要があります。

- プラグインの構成情報を記述する plugin.xml
- タブ内のUIを構築するJSSP(js+html)
- クライアントサイドでデータの入出力やイベントをハンドリングするjsファイル
- サーバサイドでデータにアクセスするロジック(jsファイル)

実装の形式としては詳細画面のタブと同様ですが、使用するメソッドや一部やり取りされる引数が異なります。
ただし検索結果は検索した後で何らかの処理に使用される情報になる場合が多いと思われますので結果として返す形式には注意する必要があります。
以降に詳細に説明します。

ここでは品目検索画面の基本タブを例にとって説明します。

```
<?xml version="1.0" encoding="utf-8"?>
<plugin>
  <extension point="jp.co.intra_mart.foundation.master.setting.item.search.criteria">
  <search_criteria_config name="standard" id="jp.co.intra_mart.master.item.criteria.main" version="8.0.0"
rank="1">
    <content
      title="%jp.co.intra_mart.foundation.master.setting.item.search.criteria.content.tabname"
      csjs="im.app.master.plugins.item.search.tabs.main.SearchMain"
      page="master/plugins/maintenance/item/search/tabs/main/itemcategory_main"
      min_height = "150"
    />
      <operations page="master/plugins/maintenance/item/search/tabs/main/itemcategory_main" object="">
      <operation id="search" method="search"></operation>
      <operation id="validate" method="validate"></operation>
    </operations>
    <list>
      <columns>
        <column
          id="edit"
          type="icon"
          icon="images/icons/16x16/report--pencil.png"
          name="%jp.co.intra_mart.foundation.master.setting.item.search.criteria.content.list.edit"
          title="%jp.co.intra_mart.foundation.master.setting.item.search.criteria.content.list.edit"
          jsaction="im.app.master.plugins.item.search.frame.item#clickList"
          width="40"
          />
        <column id="code"
          name="%jp.co.intra_mart.foundation.master.setting.item.search.criteria.content.list.code"
          width="150"></column>
        <column id="description"
          name="%jp.co.intra_mart.foundation.master.setting.item.search.criteria.content.list.name"
          width="250" sortable="false"></column>
        <column id="deleteFlag" name="delete flag" width="200" visible="false" indicator="true"
color="#ff0000" />
      </columns>
    </list>
    </search_criteria_config>
  </extension>
</plugin>
```

【リスト：品目検索画面 基本タブのplugin.xml】

plugin.xmlの書き方や構成についての詳細はPlugin Managerのマニュアルを参照してください。
ここではタブの追加に当たって必要な範囲で説明します。

- extensionタグ
  - point属性 ：タブを拡張したい対象の拡張ポイントを指定します。
    IM-共通マスタのメンテナンス画面で用意している拡張ポイントについては「*マスタメンテナンス画面の拡張に関する情報* 」で説明していますので参照してください。
- search_criteria_configタグ
  - name、id、version、rank などの属性はPlugin Managerによって依存関係の管理などに使用されます。
    詳細はPlugin Managerのドキュメントを参照してください
  - タグ名は任意です
- content タグ
  - title属性：追加したタブに表示されるタイトルを指定します。%表記は国際化メッセージキーを表します。国際化に関してはPlugin Managerのマニュアルを参照してください。
  - csjs属性：クライアントサイドでタブの情報をハンドリングする処理を設定します。この処理はim-jspackman

で管理可能なクラス形式である必要があります。
クラスのFQDNを指定します。
    - page属性：タブ内のUIを構築するJSSPのパスを指定します（拡張子無し）
    - min_height属性：最低限必要な表示高さをpixelで指定します。
特に指定しなければ他のTabの要求サイズや実際の表示領域の大きさを勘案して自動的に決定します。
- operationsタグ
    - page属性：サーバサイドでデータにアクセスするロジックのjsファイルのパスを指定します(拡張子無し)
- operationタグ：処理と実装をマッピングする情報です。
画面側が必要とするidに対してmethodのマッピングを定義します。
    - id属性：検索条件タブで定義する必要のあるidは上記「サーバサイドでデータにアクセスするロジック」の部分で挙げているsearch、validateの2つです。operationタグが2つ必要になります。
    - method属性：operationsタグのpage属性で指定したファンクションコンテナ内のメソッド名を指定します。
上記のIDに対応する形で記述してください。
- list タグ：検索結果に関する設定を行います。属性はありません。
- columns タグ：属性はありません。配下に画面に表示するカラムの設定を列挙します。
- column タグ：検索結果を表示する欄のカラムを表します。検索結果をどのように表示するかを設定します。
    - name属性：カラムヘッダに表示する名称を設定します。
    - id属性：検索処理の実装が返す検索結果と、カラムをマッピングします。
検索結果はオブジェクトの配列である必要がありますが、そのオブジェクトのプロパティのうち、このカラムの表示に使用するプロパティ名を設定します。
    - width属性（省略可）：このカラムの初期表示幅を指定します。デフォルトではクライアント表示領域を勘案してなるべく均等に幅を取ります。一覧の一番右側になるカラムは自動的に表示上一覧の右端まで延長されます。
    - visible属性（省略可）：このカラムの表示・非表示を設定します。デフォルトでは表示されます。falseを設定しておくと表示されません。
    - indicator属性（省略可）：次のcolor属性と関連します。表示色を変更するフラグを表すプロパティのプロパティ名を設定します。
この属性を設定しておくと、画面処理基盤は検索結果の各オブジェクトのindicator属性で指定された名前のプロパティをチェックし、trueであった場合、その行の文字をcolor属性に指定された色で表示します。
    - color属性（省略可）：indicator属性と関連します。検索結果のオブジェクトのindicator属性の指すプロパティがtrueの場合の表示色を指定します。スタイルシートの色指定として有効な文字列を指定してください。

このxmlファイルをPlugin Managerの管理するディレクトリに配置します。具体的には以下のパスになります。

```
<（展開したwar）/WEB-INF/plugin/%plugin_id%/plugin.xml>
```

plugin.xmlを変更したら、intra-martの再起動が必要になります。

タブ内のUIを構築するJSSP(js+html)

JSSP内の処理に関しては特別な処理を行う必要はありません。
また初回のinitの引数には、機能によって異なりますが、概ね以下の内容が渡されます。

- 表示中の基準日やロケールなど、画面表示に使用する基本情報（basicInfo）

詳細な内容は機能によって異なります。詳細なメソッドや引数の内容については別途「IM-共通マスタ 拡張インタフェース定義一覧」として公開していますので、あわせてご覧下さい。

このファイルは【リスト：品目検索画面 基本タブのplugin.xml】 のcontentタグのpageプロパティと一致する場所に配置します。

クライアントサイドでデータの入出力やイベントをハンドリングするjsファイルの作成

詳細・更新タブの拡張と同様に、データの出し入れに関しては画面の基盤プログラムがサーバサイドとクライアントサイドでデータの通信を一括して取り扱います。
画面の基盤プログラムは画面表示の更新が必要になったタイミングや、検索ボタンが押された等、特定のタイミングでクライアントサイドスクリプトの特定のメソッドを呼び出します。

	メソッド名	戻り値	説明
1	init(window)	void	タブを初期表示した時点。引数にはタブ自身を表すwindowオブジェクトが渡されます。戻り値は必要ありません。
2	renderer(window, model)	void	画面のデータを更新する必要がある場合（ユーザ操作による画面表示更新操作など）に呼び出されます。引数にはタブ自身を表すwindowオブジェクトと、サーバサイド処理の取得処理(plugin.xmlでid:get_detailに指定された処理）を実行して返された内容をそのまま渡します。戻り値は必要ありません。
3	validate(window)	Object	画面データをサーバに送信する場合、クライアントサイドでの値の検証の目的で呼び出します。原則としてmodelerの直前に呼び出されます。戻り値の形式は以下のように成ります。 プロパティ / 型 / 説明 error / boolean / エラーの有無を表すフラグ message / Array / 下記オブジェクトの配列 1_n (配列添字) / code / number / Plugin任意のエラーコード 1_n (配列添字) / message / string / エラーメッセージ 問題がない場合はエラーフラグをfalseにした値を返して下さい。
4	modeler(window)	Object	画面のデータを収集する必要がある場合（検索実行時など）。この関数の戻り値の内容を各タブのplugin idをキーとして連想配列の形式に収め、そのままサーバサイドの更新処理(plugin.xmlでid:get_detailに指定された処理)に渡されます。JSONで表せる内容である必要があります。
5	hasValidInput(window)	boolean	大量データモードが設定されている場合に、検索実行の契機で呼び出されます。 検索を実行するに足る条件が設定されているかどうかをboolean値で返して下さい。 falseが返された場合、処理を中断します。
6	clear(window)	void	画面にクリアボタンがある場合に、クリア処理を実行するために呼び出します。戻り値は特に必要ありません。

【表：クライアントサイドスクリプトに必要なメソッドの一覧】

これらの処理を、「*プラグインの構成情報を記述する plugin.xml*」でcontentタグのcsjs属性に指定しているクラスに実装します。
以下に品目検索画面 基本タブの例を示します。

```
Package("im.app.master.plugins.item.search.tabs.main");

/**

* 品目検索画面 基本タブのクライアントサイドスクリプト
*
 **/
Class("im.app.master.plugins.item.search.tabs.main.SearchMain").define(

 im.app.master.plugins.item.search.tabs.main.SearchMain = function () {

  var initmodel;

  /**
   * init
   * 初期化処理。タブのロードが完了したタイミングで呼び出されます。
   **/
  this.init = function(parentWindow,basicInfo) {
    var parentElement = parentWindow.document;
  };
```

```
};

/**
 * renderer
 * 画面の描画処理。初期表示時や再表示の必要がある場合に呼び出されます。
 **/
this.renderer = function(parentWindow,model) {
  var parentElement = parentWindow.document;

  var targets = searchElementByName(parentElement,"target")[0];
  /*・・・略・・・*/
};

/**
 * modeler
 * 検索実行時の入力値の収集処理。
 * 画面で検索ボタンが押下された際に呼び出されます。
 **/
this.modeler = function(parentWindow) {
  var parentElement = parentWindow.document;
  var model = new Object();

 /*・・・略・・・*/

  }
  model.category = searchElementByName(parentElement,"category")[0].value;
  model.categoryset = searchElementByName(parentElement,"categoryset")[0].value;

/*・・・略・・・*/

  model.searchname = parentElement.getElementById("searchname").checked;
  model.ignore = searchElementByName(parentElement,"ignore")[0].checked;
  return model;
};

/**
 * clear
 * 検索条件入力欄のクリア処理。
 * 画面でクリアボタンが押下された際に呼び出されます。
 **/
this.clear = function(parentWindow) {
  var parentElement = parentWindow.document;

  searchElementByName(parentElement,"searchbelow")[0].checked = false;

/*・・・略・・・*/

  searchElementByName(parentElement,"searchname")[0].value = "";
};

/**
 * validate
 * 検索実行時の入力値の検証処理。modelerの直前に呼び出されます。

**/
this.validate = function(parentWindow) {
  var parentElement = parentWindow.document;
  var o = new Object();
  o.error = false;
  o.messages = new Array();
  //o.messages[o.messages.length] = {"message" : "error!", "code" : 1};

  return o;
};

/**
```

```
  * hasValidInput
  * 検索時に有効な検索条件が与えられているかの判断処理。
  * 全件検索の防止機能を実現するために、タブ内に有効な条件が入力されているか
  * 判断するために呼び出されます。
  **/
 this.hasValidInput = function(parentWindow){
   var parentElement = parentWindow.document;

   if(searchElementByName(parentElement, "keyword")[0].value.replace(/ /g, "") == ""){
     return false;
   }else{
     return true;
   }
 };

 /*・・・略・・・*/
}
);
```

【リスト： 品目検索画面基本ダブのクライアントサイドjsの例】

クライアントサイドスクリプトの配置場所は <（展開したwar）/csjs> 配下に、パッケージ名にあわせてディレクトリを作成して配置して下さい。
上記の品目検索画面 基本タブの例ではパッケージ名がim.app.master.plugins.item.search.tabs.main、クラス名がSearchMainなので実際には以下の場所に配置します。

<（展開したwar）/csjs/im/app/master/plugins/item/search/tabs/main/SearchMain.js>

サーバサイドでデータにアクセスするロジック(jsファイル) の作成

クライアントサイドでmodeler()で収集した検索条件を元に情報を検索して返す処理を実装します。
実装する処理のファイル名や関数名は「*プラグインの構成情報を記述する plugin.xmlの作成*」で定義している値と一致しなければなりません。
下記の説明に従ってそれぞれ実装して下さい。引数や戻り値については、拡張ポイント毎に詳細を定義した「IM-共通マスタ 拡張インタフェース定義一覧」を別途公開していますので、あわせてご確認下さい。

	処理（plugin.xml上のid）	**説明**
1	検索処理(id:search)	検索が実行された際に呼び出されます。 画面で入力された条件を元に、検索を行う処理を実装します。引数にはクライアントサイドでmodeler()が作成した形式のオブジェクトが渡されます。検索の場合、現在表示中のタブPluginの実装のみ呼び出されます。この結果のオブジェクトは後述するplugin.xmlで定義するcolumnの設定と一致していなければなりません。 もし検索処理にエラーがある場合は、戻り値のエラーフラグにtrueを設定しておくことで、処理を中断することが出来ます。
2	検証処理(id:validate)	編集画面で編集された情報をデータベースで検索する前に、内容を検証するために呼び出されます。引数にはクライアントサイドのタブでmodeler()が作成した形式のオブジェクトが渡されます。

【表：サーバサイドスクリプトで実装する必要のある処理】

共通検索画面の拡張

共通検索画面はIM-共通マスタの検索処理をアプリケーションから共用できるよう実装された検索機能です。

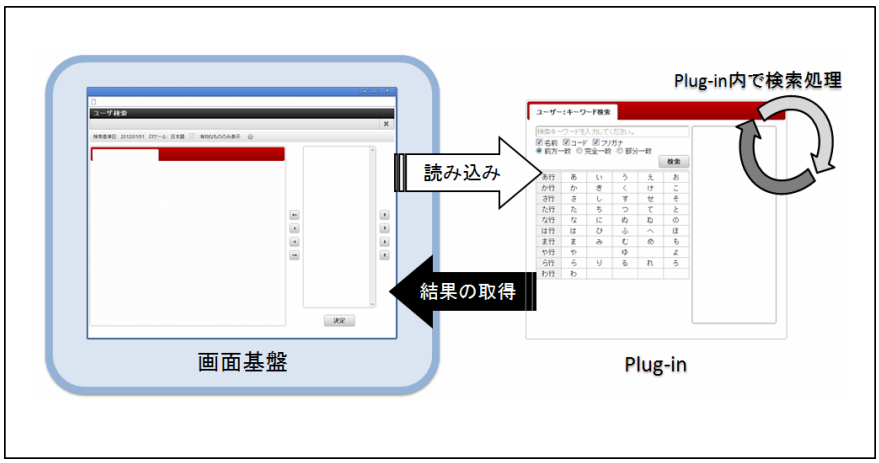
共通検索画面は画面処理基盤、基本情報エリア、タブプラグインの3つの要素で構成されています。
タブをプラグインとして追加することで、共通検索画面に統合された検索処理を作成することが出来ます。

動作の概要

共通検索画面の基盤とプラグインのインタフェースを中心に、実装の構成と動作の概要を説明します。

共通検索画面 タブの動作概要

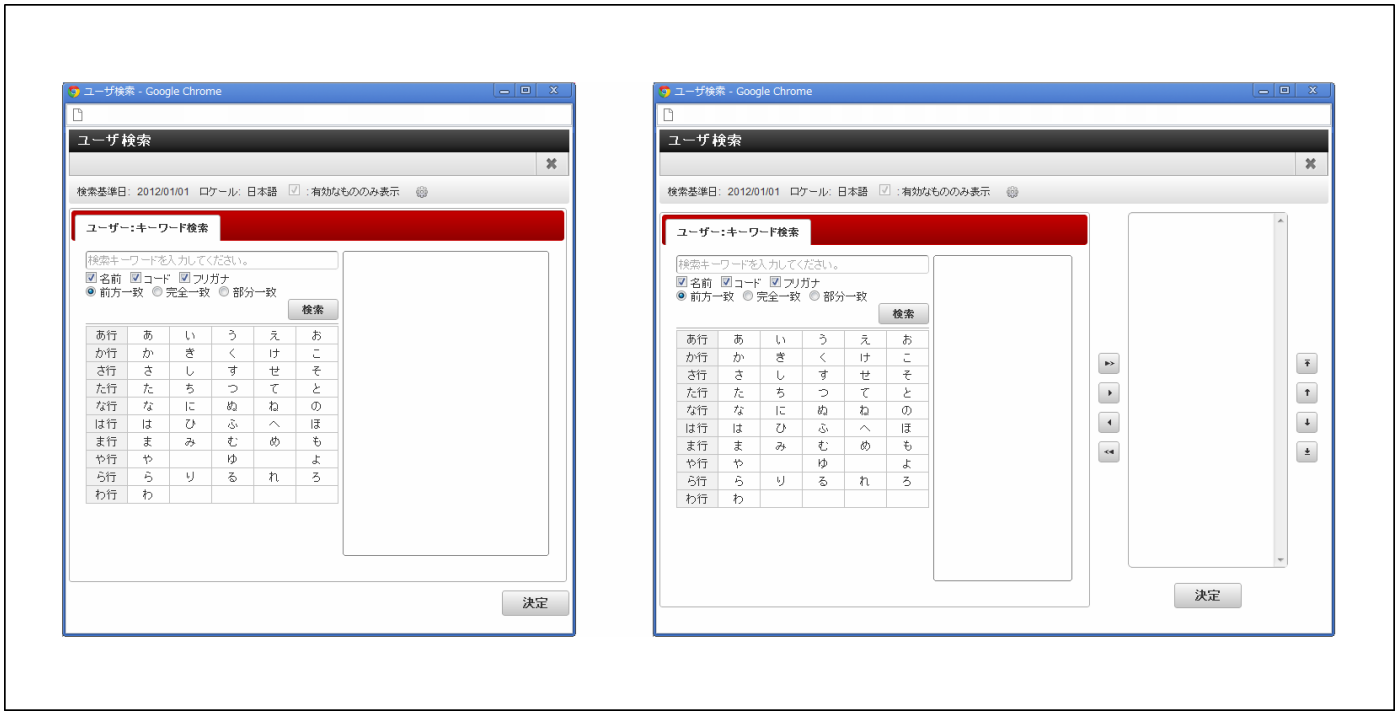
共通検索画面は、検索画面のウィンドウを作成し、タブ拡張の呼び出しや、検索結果の受け渡しを行う基盤部分と、実際に検索処理を行い、基盤に対してユーザの選択値を渡すタブのプラグインという構成で成り立っています。
マスタメンテナンス画面のタブとは異なり、プラグインはUIとDBアクセス処理に分断されてはいません。初期表示から検索条件の受付、検索、結果の表示までを全てプラグインの処理として実装します。
基盤部分では、基本となる条件（検索基準日、ロケール、削除情報を含めるか否か等）の管理と、検索結果の受け渡しや取りまとめを行います。
共通検索画面の構成の概要を【図：共通検索タブの動作概要】に示します。



【図：共通検索タブの動作概要】

単一選択モードと複数選択モード

画面基盤は起動時のパラメータにより、検索画面で結果を複数選択するか、単一で選択するか処理を切り変えることが出来ます。
単一選択の場合、タブ内で選択された物が選択結果として使用されますが、複数選択の場合、タブ欄の右側に選択中の項目一覧が表示され、そこに選択項目を貯めておくことができるようになります。



【図：単一選択モードと複数選択モード】

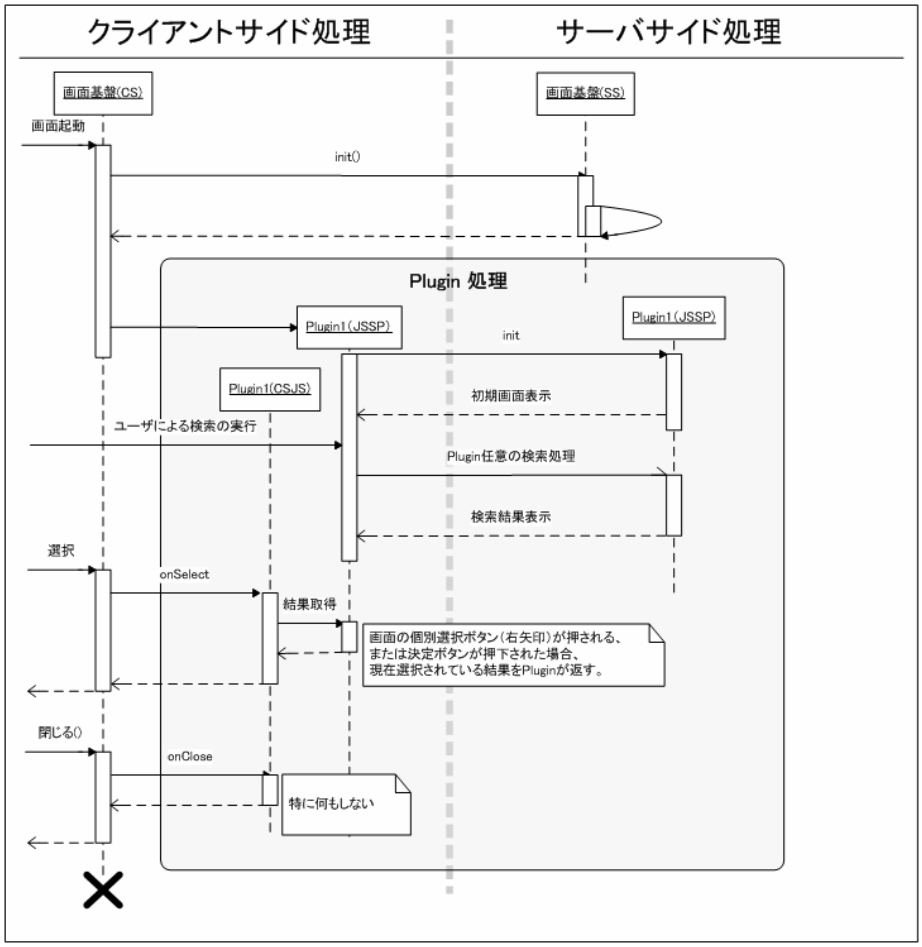
この単一選択か複数選択かによってユーザのインタラクションが異なっており、タブ側では双方の要求に答えられる必要があります。

- 単一選択の場合は一つしか選択出来ない。画面基盤から結果を要求された場合に単一の値を返す必要がある。
- 複数選択の場合は個別選択のボタンと全選択のボタンがあるため、複数の結果を返せる必要がある。
  個別選択ボタンは単一選択ではなく、タブ内で複数の値が同時に選択されていた場合、複数の値を返して良い。

これらの要求に答えるためにいくつか規定の処理を実装する必要があります。
以降で詳細に説明します。

画面起動時から結果選択までのシーケンス

共通検索画面の動作シーケンスを 【図：共通検索画面の動作シーケンス】 に示します。
クライアントサイドでinitの呼び出しがありますが、基本的にはプラグイン側で初期表示から検索処理、結果の表示を行う必要があり、画面基盤側は関与しません。
基盤側では検索基準日やロケールの設定等を管理しており、これはユーザにより変更されることがありますが、基本条件が変更された場合にはタブのプラグインは初期表示と同じシーケンスで再読み込みされます。



【図：共通検索画面の動作シーケンス】

基盤部分から発生するイベントのハンドリング

【図：共通検索画面の動作シーケンス】 の後半、クライアントサイドで画面基盤側からonSelect()が呼び出されていますが、これはユーザが検索結果から値を選択するアクションを起こした際に発生します。
この他にもいくつかのイベントが規定されており、プラグインではそれぞれの要求に応えられる必要があります。
画面基盤側がプラグインへ処理を要求するイベントは下記の通りです。

1. タブの初期化時(init)
   タブの読み込みが完了した時点で呼び出されます。

2. 個別選択ボタン押下時(onSelect)
   複数選択モードの場合は個別の選択ボタン、単一選択モードの場合は決定ボタン押下時に呼び出されます。

3. 全選択ボタン押下時(onSelectAll)
   画面が複数選択モードで起動されている場合にのみ発生する要求です。
   全選択ボタンが押下された際に呼び出されます。

4. 個別解除ボタン押下時(onDeselect)
   画面が複数選択モードで起動されている場合にのみ発生する要求です。
   既に選択項目の一覧に入っている項目を解除する場合に呼び出されます。項目自体は基盤によって削除されます。

5. 全解除ボタン押下時(onDeselectAll)
   画面が複数選択モードで起動されている場合にのみ発生する要求です。
   選択項目の一覧に入っている項目すべてを解除する場合に呼び出されます。項目自体は基盤によって削除されます。

6. 決定ボタン押下時(onDecide)
   画面下側の決定ボタンを押下した際に呼び出されます。

7. ウィンドウクローズ時(onClose)
   画面右上の閉じるボタンを押下した際に呼び出されます。

これらの処理をクライアントサイドのjsとして実装し、適切にハンドリングしなければなりません。

実装の詳細

共通検索タブの実装

共通検索画面で使用するタブを追加するためには次のものを準備する必要があります。

- プラグインの構成情報を記述するplugin.xml
- タブ内で検索処理を行うJSSP （html+js）
- クライアントサイドで基盤部分からのイベントに応答するためのjs(csjs)

プラグインの構成情報を記述するplugin.xmlの作成

ここではユーザ検索（キーワード）タブのplugin.xmlを例に説明します。

```
<?xml version="1.0" encoding="utf-8"?>
<plugin>
  <extension point="jp.co.intra_mart.common.search.tabs">
    <search name="standard" id="jp.co.intra_mart.master.app.search.tabs.user.list_user" version="8.0.0"
rank="1">
      <content
        title="%tabtitle"
        csjs="im.app.search.plugins.tabs.user.keyword.Keyword"
        page="im_master/plugins/app/search/tabs/user/keyword/keyword"
        min_width="450"
        min_height="325"
      />
    </search>
  </extension>
</plugin>
```

【リスト：ユーザ検索（キーワード）タブのplugin.xml】

- extensionタグ
  - point属性 ：共通検索画面のタブを拡張する拡張ポイントを指定します。共通検索画面のタブであればこの値は同じです。
- searchタグ
  - name、id、version、rank などの属性はPlugin Managerによって依存関係の管理などに使用されます。詳細はPlugin Managerのドキュメントを参照してください。
  - タグ名は任意です
- content タグ
  - title属性：追加したタブに表示されるタイトルを指定します。%表記は国際化メッセージキーを表します。国際化に関してはPlugin Managerのマニュアルを参照してください。
  - csjs属性：クライアントサイドでタブの情報をハンドリングする処理を設定します。この処理はim-JSPackmanで管理可能なクラス形式である必要があります（im-JSPackmanの詳細はクライアントサイドライブラリのAPIを確認してください）。クラスのFQDNを指定します。
  - page属性：タブ内のUIを構築するJSSPのパスを指定します（拡張子無し）
  - min_height,, min_width 属性：最低限必要な表示高さ、幅をpixelで指定します。
    特に指定しなければ他のTabの要求サイズや実際の表示領域の大きさを勘案して自動的に決定します。

このxmlファイルをPlugin Managerの管理するディレクトリに配置します。
具体的には以下のパスが対象です。

`<（展開したwar）/WEB-INF/plugin/%plugin_id%/plugin.xml>`

plugin.xmlを変更した場合は、intra-martを再起動する必要があります。

タブ内で検索処理を行うJSSP（html+js）の作成

基本としてJSSPで検索処理の全てを実装します。
通常のJSSPのように最初はinit()が読み出されます。
引数には共通検索画面が受け取ったパラメータそのままのオブジェクトが渡されます。
共通検索画面が受け付けるオブジェクトの形式については IM-共通マスタ 検索画面仕様書 に定義されていますので、そちらを参照してください。

このファイルは 「*プラグインの構成情報を記述するplugin.xmlの作成* 」でcontentタグのpageプロパティと一致する場所に配置します。

html作成時の注意

使用するブラウザの種類とバージョンによって、初期処理が【図：共通検索画面の動作シーケンス】のとおりに動作しない場合があります。
そのため、検索タブのhtml側で以下の実装を行う必要があります。

```
<imart type="head">
<!--・・略・・-->
    // jspackmanクラスインポート
    Import("im.app.search.control.PageHandler");                                    ・・・・・・・①
    // 初期処理実行
    Main.invoke("this");

    function main(args) {
        pc = new im.app.search.control.PageHandler("global_instance", createPage);          ・・・・・・・②
        pc.execute();
    }

    function createPage() {
        try {
            // ここにpluginのメイン処理を記載する。
        /*・・略・・*/
        } finally {
            lwc.stopLoading();
            window.global_instance.completeNotice();                                    ・・・・・・・③
        }
    }
<!--・・略・・-->
</imart>
<!--・・略・・-->
```

【リスト：検索タブのhtmlのサンプル】

1. 検索タブのヘッド部で、PageHandlerをimportします。
2. PageHandlerのインスタンスを生成します。第2引数には、検索画面タブのメイン処理が書かれてあるメソッドを定義します。
3. 検索画面のメイン処理が終わった時に、completeNoticeを呼び出します。

クライアントサイドで基盤部分からのイベントに応答するためのjs の作成

クライアントサイドでは基盤部分でのユーザアクションに反応して検索結果を取り扱う必要があるため、特定のタイミングで、クライアントサイドスクリプトの特定のメソッドを呼び出します。
クライアントサイドスクリプトはim-JSPackmanの実装方式にしたがって定義したクラスである必要があります（plugin.xmlにクラス名を指定する事で画面基盤が動的にロードしてインスタンス化します）。
下記にクライアントサイド処理で必要な処理を示します。

	メソッド名	戻り値	説明
1	init(window, params)	void	タブのロードが終わったタイミングで呼び出されます。特に基盤側から期待する動作はありません。 intra-mart Accel Platform 2015 Spring 以前は、画面起動時にすべてのタブがロードされます。 intra-mart Accel Platform 2015 Summer 以降は、タブが選択されてからタブがロードされます。 引数の説明： window: タブのIFRAMEを表すwindowオブジェクト params : 起動引数
2	onSelect(window)	Array	ユーザが選択のアクションを行ったタイミングで呼び出されます。現在選択されている値を 4.2.1.3.1結果の形式について で説明している形式で、配列に詰めて返してください。 単一選択時の場合には「確定」ボタンを押下したタイミング、複数選択時の場合は個別選択ボタン（右矢印のアイコン）を押下したタイミングです。 引数の説明： window: タブのIFRAMEを表すwindowオブジェクト
3	onSelectAll(window)	Array	ユーザが全選択のアクション（右二重矢印のアイコンを押下）を行ったタイミングで呼び出されます。現在検索結果として表示している値を全て 4.2.1.3.1結果の形式についてで説明している形式にして、配列に詰めて返してください。 引数の説明： window: タブのIFRAMEを表すwindowオブジェクト
4	onDeselect(window, items)	boolean	ユーザが選択解除のアクション（左矢印のアイコンを押下）をしたタイミングで呼び出されます。 選択解除対象のオブジェクトが引数に渡され、全てのタブに対し呼び出されます。 項目の解除処理自体は画面処理基盤側で行いますので、特に基盤側から期待する動作はありません。 引数の説明： window: タブのIFRAMEを表すwindowオブジェクト items : 選択解除されたオブジェクトの配列
5	onDeselectAll(window, items)	boolean	ユーザが全解除のアクション（左二重矢印のアイコンを押下）をしたタイミングで呼び出されます。 選択解除対象のオブジェクトが引数に渡され、全てのタブに対し呼び出されます。 項目の解除処理自体は画面処理基盤側で行いますので、特に基盤側から期待する動作はありません。 引数の説明： window: タブのIFRAMEを表すwindowオブジェクト items : 選択解除されたオブジェクトの配列
6	onDecide(window)	boolean	ユーザが決定のボタンを押下した際に呼び出されます。 基盤側から期待する動作はありません。 引数の説明： window: タブのIFRAMEを表すwindowオブジェクト
7	onClose(window)	boolean	ユーザがタイトルバーの×ボタンを押下した際に呼び出されます。 基盤側から期待する動作はありません。 引数の説明： window: タブのIFRAMEを表すwindowオブジェクト

【表：クライアントサイドスクリプトに必要なメソッドの一覧】

また必要なメソッドを空実装した”im.app.search.abstracts.AbstractTab”クラスが存在します（実際のファイルは <（展開

したwar）/csjs/im/app/search/abstracts/AbstractTab.js>）。
このクラスを継承して実装すれば不要なメソッドをオーバーライドする必要がありません。

```
/**
* @fileoverview タブをハンドリングするクラスを定義しています。<br/>
*/
Package("im.app.search.abstracts");

Class("im.app.search.abstracts.AbstractTab").define(
  im.app.search.abstracts.AbstractTab = function() {
    this.superclass();

    /* 画面構築基盤の提供する機能へアクセスするためのオブジェクトが格納される */
    this.parent;

    /**
    * 要override
    * タブのロードが完了した時点で呼び出される。戻り値は不要。
    * @param {window} window タブ内のフレームを表すウィンドウオブジェクト
    * @param {Object params} 起動引数オブジェクト
    **/
    this.init = function(window, params){
    };

    /**
    * 要override
    * 画面基盤側で選択ボタンが押下された場合など、現在選択中の情報が要求された場合に呼び出される。
    * タブ内で現在選択されている情報を規定形式のJSONオブジェクトとして返す。
    * 配列を返すことで複数一括の選択として使用できる。multipleでない場合は先頭のもののみ使用する。
    * 選択されている情報がない、または無効などの場合はnullを返す。
    *
    * @param {window} window タブ内のフレームを表すウィンドウオブジェクト
    **/
    this.onSelect = function(window){
    };

    /**
    * 要override
    * 画面基盤側で全選択ボタンが押下された場合など、現在の検索結果全てが要求された場合に呼び出される。
    * タブ内で現在選択されている情報を規定形式のJSONオブジェクトとして返す。
    * 配列を返すことで複数一括の選択として使用できる。multipleでない場合は先頭のもののみ使用する。
    * 選択されている情報がない、または無効などの場合はnullを返す。
    *
    * @param {window} window タブ内のフレームを表すウィンドウオブジェクト
    **/
    this.onSelectAll = function(window){
    };

    /**
    * 画面基盤側で項目が選択解除された際に呼び出される。
    * ほとんどのタブに於いて処理する必要はないはず。
    * 全てのタブに対して呼び出される。
    * @param {window} window タブ内のフレームを表すウィンドウオブジェクト
    * @param {Object} items  選択解除されたオブジェクト。複数の場合は配列。typeに関係なく渡される。
    **/
    this.onDeselect = function(window, items){
    };


    /**
    * 画面基盤側で項目が選択全解除された際に呼び出される。
    * ほとんどのタブに於いて処理する必要はないはず。
    * 全てのタブに対して呼び出される。
    * @param {window} window タブ内のフレームを表すウィンドウオブジェクト
    * @param {Object} items  選択解除されたオブジェクト。複数の場合は配列。typeに関係なく渡される。
```

```
    * @param {Object} items  選択解除された対象オブジェクト。複数の場合は配列。typeに関係なく渡される。
    **/
    this.onDeselectAll = function(window, items){
    };

    /**
    * 画面基盤側で項目が選択が確定された際に呼び出される。
    * ほとんどのタブに於いて処理する必要はないはず。
    * 全てのタブに対して呼び出される。
    * @param {window} window タブ内のフレームを表すウィンドウオブジェクト
    * @param {Object} items  選択決定されたオブジェクト。複数の場合は配列。typeに関係なく渡される。
    **/
    this.onDecide = function(window, items){
    };

    /**
    * 画面基盤側で閉じるボタンが押下された再に呼び出される。
    * ほとんどのタブに於いて処理する必要はないはず。
    * 全てのタブに対して呼び出される。
    * @param {window} window タブ内のフレームを表すウィンドウオブジェクト
    **/
    this.onClose = function(window){
    };
  }
);
```

【リスト： AbstractTab.js】

クライアントサイドスクリプトの配置場所は <（展開したwar）/csjs> 配下に、パッケージ名にあわせてディレクトリを作成して配置してください。

結果の形式について

結果は以下の形式のオブジェクトを配列にして返してください。

クライアントサイドでは基盤部分でのユーザアクションに反応して検索結果を取り扱う必要があるため、特定のタイミングで、クライアントサイドスクリプトの特定のメソッドを呼び出します。
クライアントサイドスクリプトはim-JSPackmanの実装方式にしたがって定義したクラスである必要があります（plugin.xmlにクラス名を指定する事で画面基盤が動的にロードしてインスタンス化します）。
下記にクライアントサイド処理で必要な処理を示します。

プロパティ名	型	説明
data	Object	実際にデータベースから取得したレコードの内容をオブジェクトとして設定してください。
type	Array	このオブジェクトの型を表します。主にプラグイン側から画面処理基盤へ型の判別ができるように提示するものです。 画面処理基盤ではtypeと、keyFieldsを同じ項目が選択されていないか判断するために使用しています。
keyFields	Array	文字列の配列。 data 内で一意性を表すキーとなるプロパティのプロパティ名を配列として設定してください。 画面処理基盤側で重複選択を避ける為の情報として使用します。 具体的にはdataからkeyFieldsに設定されたの名前のプロパティを取得し、同一typeかどうかを含めて比較して重複がないかを確認しています。
displayName	string	オブジェクトを画面に表示する際に使用する表示文字列

マネージャの拡張に関する情報

マネージャの拡張ポイント一覧

マネージャに対してリスナーを追加したり、実装クラスを変更するために公開されている拡張ポイントは【表：マネージャの拡張ポイント一覧】の通りです。

	マネージャ	拡張ポイント
1	CompanyGroupManager	jp.co.intra_mart.foundation.master.accessor.company_group
2	CompanyManager	jp.co.intra_mart.foundation.master.accessor.company
3	CorporationGroupManager	jp.co.intra_mart.foundation.master.accessor.corporation_group
4	CorporationManager	jp.co.intra_mart.foundation.master.accessor.corporation
5	CurrencyManager	jp.co.intra_mart.foundation.master.accessor.currency
6	CustomerManager	jp.co.intra_mart.foundation.master.accessor.customer
7	ItemCategoryManager	jp.co.intra_mart.foundation.master.accessor.item_category
8	ItemManager	jp.co.intra_mart.foundation.master.accessor.item
9	PrivateGroupManager	jp.co.intra_mart.foundation.master.accessor.private_group
10	PublicGroupManager	jp.co.intra_mart.foundation.master.accessor.public_group
11	UserManager	jp.co.intra_mart.foundation.master.accessor.user

【表：マネージャの拡張ポイント一覧】

リスナーインタフェースの一覧

リスナーを作成する場合には下記のいずれかのインタフェースを実装する必要があります。

	マネージャ	インタフェースFQDN
1	CompanyGroupManager	jp.co.intra_mart.foundation.master.company_group.CompanyGroupListener
2	CompanyManager	jp.co.intra_mart.foundation.master.company.CompanyListener
3	CorporationGroupManager	jp.co.intra_mart.foundation.master.corporation_group. CorporationGroupListener
4	CorporationManager	jp.co.intra_mart.foundation.master.corporation.CorporationListener
5	CurrencyManager	jp.co.intra_mart.foundation.master.currency.CurrencyListener
6	CustomerManager	jp.co.intra_mart.foundation.master.customer.CustomerListener
7	ItemCategoryManager	jp.co.intra_mart.foundation.master.item_category.ItemCategoryListener
8	ItemManager	jp.co.intra_mart.foundation.master.item.ItemListener
9	PrivateGroupManager	jp.co.intra_mart.foundation.master.private_group.PrivateGroupListener
10	PublicGroupManager	jp.co.intra_mart.foundation.master.public_group.PublicGroupListener
11	UserManager	jp.co.intra_mart.foundation.master.user.UserListener

【表：各マネージャのリスナーインタフェース一覧】

マスタメンテナンス画面の拡張に関する情報

画面の拡張ポイントと、各処理の引数の詳細

マスタメンテナンスの画面拡張時に必要になる拡張ポイントを【表：各画面の拡張ポイント一覧】 に示します。

	機能	画面	拡張ポイント
1	ユーザ	検索	jp.co.intra_mart.foundation.master.setting.user.search.criteria
		詳細	jp.co.intra_mart.foundation.master.setting.user.detail
2	会社組織	会社組織検索	jp.co.intra_mart.foundation.master.setting.company.search.criteria
		会社組織詳細	jp.co.intra_mart.foundation.master.setting.company.detail
		役職詳細	jp.co.intra_mart.foundation.master.setting.post.detail
3	パブリックグループ	パブリックG検索	jp.co.intra_mart.foundation.master.setting.publicgroup.search.criteria
		パブリックG詳細	jp.co.intra_mart.foundation.master.setting.publicgroup.detail
		役割詳細	jp.co.intra_mart.foundation.master.setting.post.detail
4	会社グループ	検索	jp.co.intra_mart.foundation.master.setting.companygroup.search.criteria
		詳細	jp.co.intra_mart.foundation.master.setting.companygroup.detail
5	品目カテゴリ・品目	品目カテゴリ・品目検索	jp.co.intra_mart.foundation.master.setting.item.search.criteria
		品目カテゴリ詳細	jp.co.intra_mart.foundation.master.setting.itemcategory.detail
		品目詳細	jp.co.intra_mart.foundation.master.setting.item.detail
6	法人グループ	検索	jp.co.intra_mart.foundation.master.setting.corporationgroup.search.criteria
		詳細	jp.co.intra_mart.foundation.master.setting.corporationgroup.detail
7	法人・取引先	法人・取引先検索	jp.co.intra_mart.foundation.master.setting.customer.search.criteria
		法人詳細	jp.co.intra_mart.foundation.master.setting.corporation.detail
		取引先詳細	jp.co.intra_mart.foundation.master.setting.customer.detail
8	通貨	通貨検索	jp.co.intra_mart.foundation.master.setting.currency.search.criteria
		通貨詳細	jp.co.intra_mart.foundation.master.setting.currency.detail
		通貨精度詳細	jp.co.intra_mart.foundation.master.setting.currency_precision.detail
		通貨換算コード詳細	jp.co.intra_mart.foundation.master.setting.currency_conversion.detail
		通貨レート詳細	jp.co.intra_mart.foundation.master.setting.currency_rate.detail
9	ユーザ分類	分類詳細	jp.co.intra_mart.foundation.master.setting.usercategory.detail
		分類項目詳細	jp.co.intra_mart.foundation.master.setting.usercategoryitem.detail
10	パブリックG分類	分類詳細	jp.co.intra_mart.foundation.master.setting.publicgroupcategory.detail
		分類項目詳細	jp.co.intra_mart.foundation.master.setting.publicgroupcategoryitem.detail
11	組織分類	分類詳細	jp.co.intra_mart.foundation.master.setting.departmentcategory.detail

機能	画面	拡張ポイント
	分類項目詳細	jp.co.intra_mart.foundation.master.setting.departmentcategoryitem.detail

【表：各画面の拡張ポイント一覧】

タブ拡張用のメソッドインタフェース

【表：各画面の拡張ポイント一覧】に挙げた拡張ポイント毎に必要なメソッドや引数が異なります。
別途公開している「IM-共通マスタ 拡張インタフェース定義一覧」に 拡張ポイント毎に詳細を説明していますので、そちらをご確認下さい。

共通検索画面の拡張に関する情報

共通検索画面の拡張ポイント

共通検索画面にタブを追加するための拡張ポイントは以下の一つだけです。

	拡張ポイント	
1	共通検索画面 追加タブ	jp.co.intra_mart.common.search.tabs