目次

改訂情報

変更年月日	変更内容
2012-12-21	初版
2013-07-01	第2版　下記を追加・変更しました ■ 「*メッセージ種別追加プログラム*」のサンプルを修正
2014-04-01	第3版　下記を追加・変更しました ■ 「*メッセージ種別追加プログラム*」のサンプルを修正、タイムライン表示用SSJSの追記
2014-05-01	第4版　下記を追加・変更しました ■ IMBox 拡張プログラミングガイドから IMBox プログラミングガイドに名称を変更 ■ 「*送信側 IM-Propagation 追加プログラム*」を追記

はじめに

本書の目的

本書では IMBox の機能を拡張するユーザプログラムを開発する場合の基本的な方法や注意点等について説明します。

対象読者

次の開発者を対象としてます。

- 独自に作成したアプリケーションを IMBox と連携させたい。
- IMBox に独自のメッセージ種別を追加し、 IMBox 内で使用したい。
- IM-Propagation を利用した通知するプログラム（トリガ）を実装したい。

次の内容を理解していることが必須となります。

- IMBoxの仕様
- Javaを理解している開発者
- サーバサイドJavaScriptを理解している開発者

本書の構成

本書は上記の対象読者に応じて次の２つの構成を取っています。

- *アプリケーション連携プログラム*
  アプリケーション連携を実現するための設定やプログラミング方法について説明します。
- *メッセージ種別追加プログラム*
  メッセージ種別追加を実現するための設定やプログラミング方法について説明します。
- *送信側 IM-Propagation 追加プログラム*
  IM-Propagation を利用して、IMBox連携を実現するための送信側のプログラミング方法について説明します。

アプリケーション連携プログラム

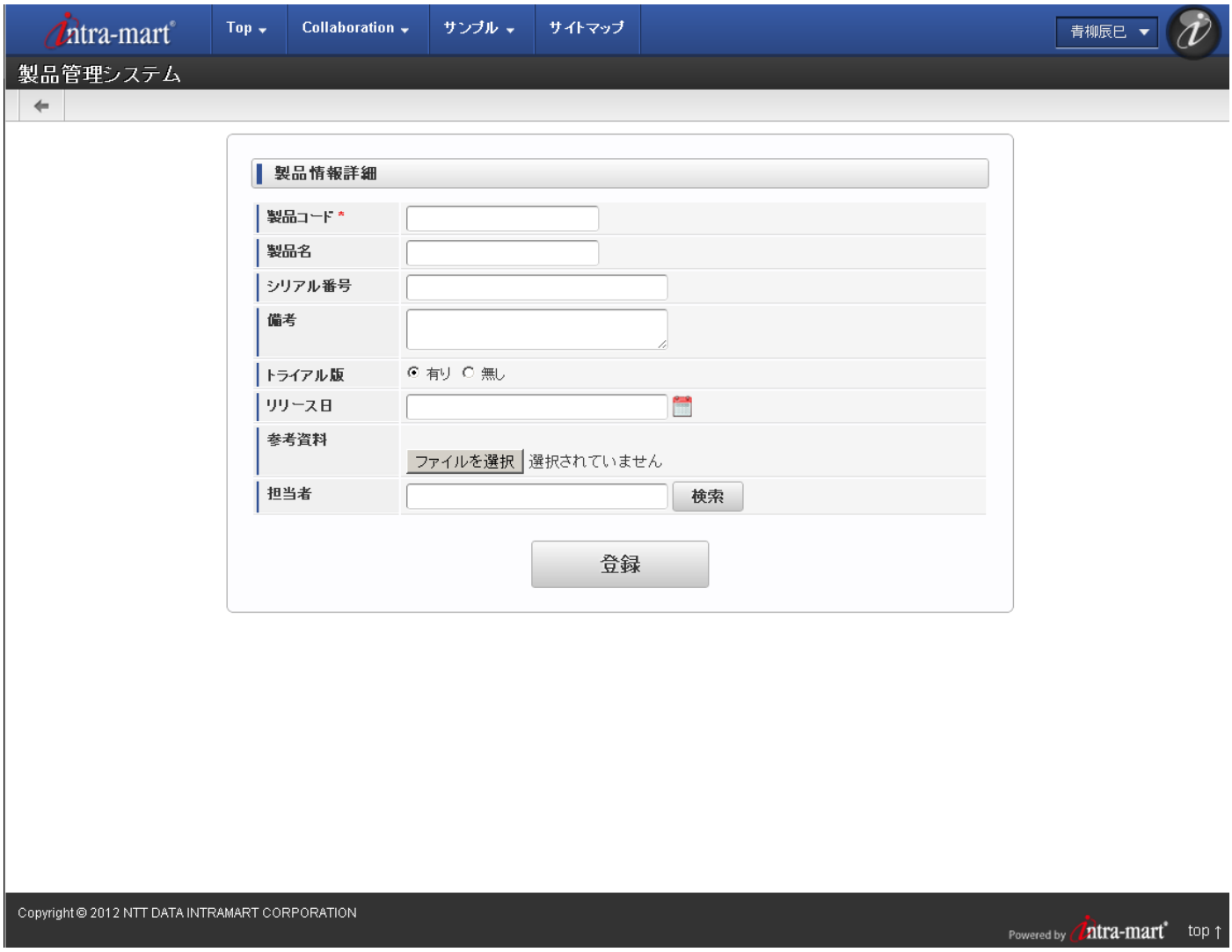
独自に作成したアプリケーションを、intra-martが提供するAPIを利用することで、次に挙げる例のような処理ができます。

- アプリケーションからの通知メッセージをApplicationBoxに投稿する。
- アプリケーション情報をウォッチする。
- ウォッチされたアプリケーション情報からメッセージをApplicationBoxに投稿する。

項目

サンプル実装

ここではサンプルとして、製品管理システムからApplicationBoxへの通知を行う機能の実装方法について説明します。



作成する資材は以下となります。

1. *設定ファイルの作成*
2. *メッセージ種別投稿欄の表示画面の作成*
3. *アプリケーション情報のウォッチ処理の作成*

4. *ApplicationBoxへのメッセージ投稿処理の作成*

設定ファイルの作成

アプリケーション連携を行うために、作成したアプリケーションに以下のファイルを設定してください。

設定ファイル例

src/main/conf/imbox-application-config/seihin.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <tns:imbox-application-config
3     xmlns:tns="http://www.intra-mart.jp/imbox/imbox-application-config"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://www.intra-mart.jp/imbox/imbox-application-config ../../schema/imbox-application-
6  config.xsd ">
7
8    <tns:application
9        applicationCd="seihin"
10       applicationName="CAPTION.IMBOX.SEIHIN"
11       iconPath="application/appli.png"
12       messageTypeCd="MESSAGE_TYPE_MESSAGE" />
   </tns:imbox-application-config>
```

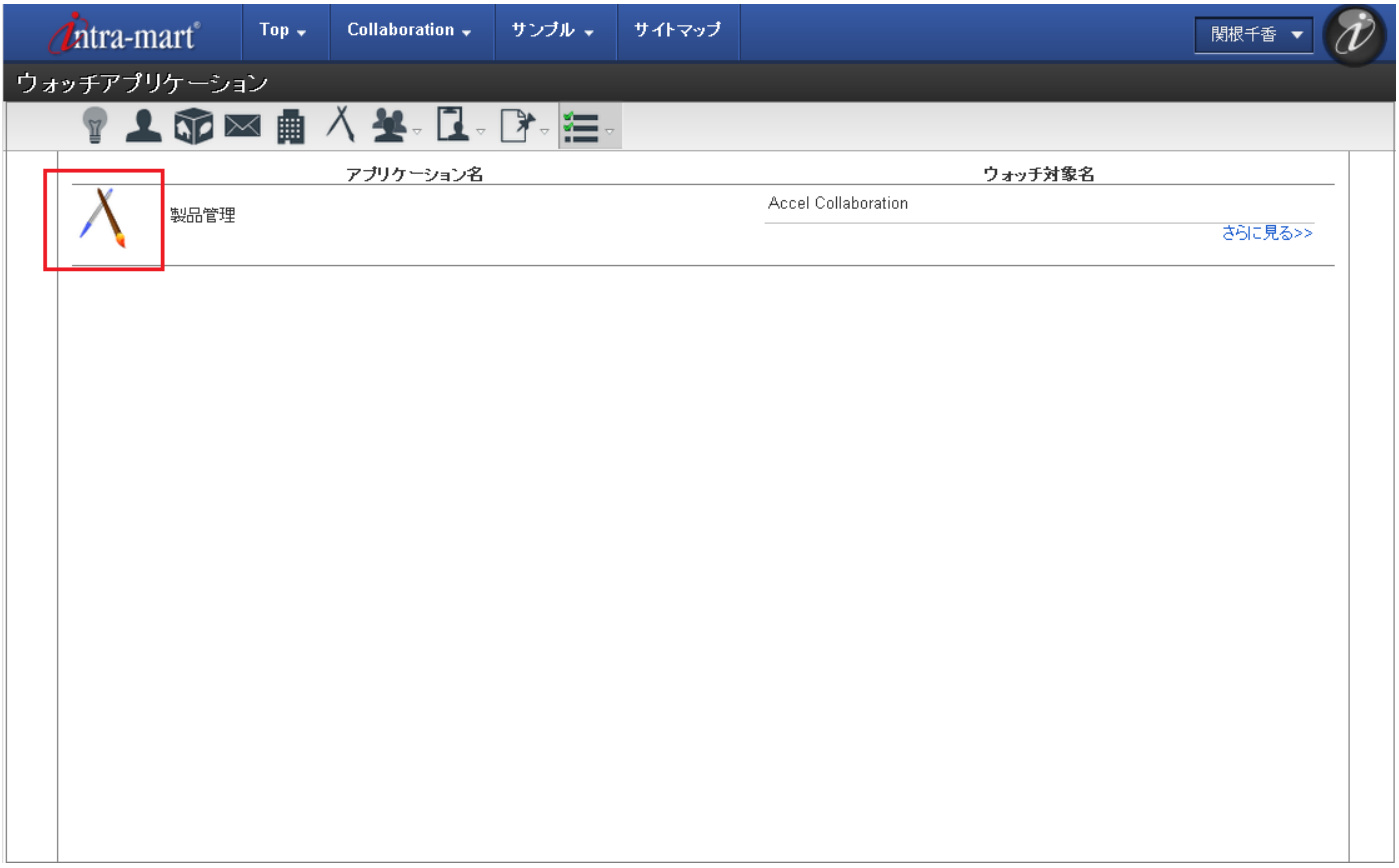
<tns:application> タグでは以下の属性について設定を行います。

属性	内容
applicationCd	アプリケーションの識別ID
applicationName	アプリケーション表示名のプロパティID
iconPath	アプリケーション表示画像のパス
messageTypeCd	メッセージ種別識別CD

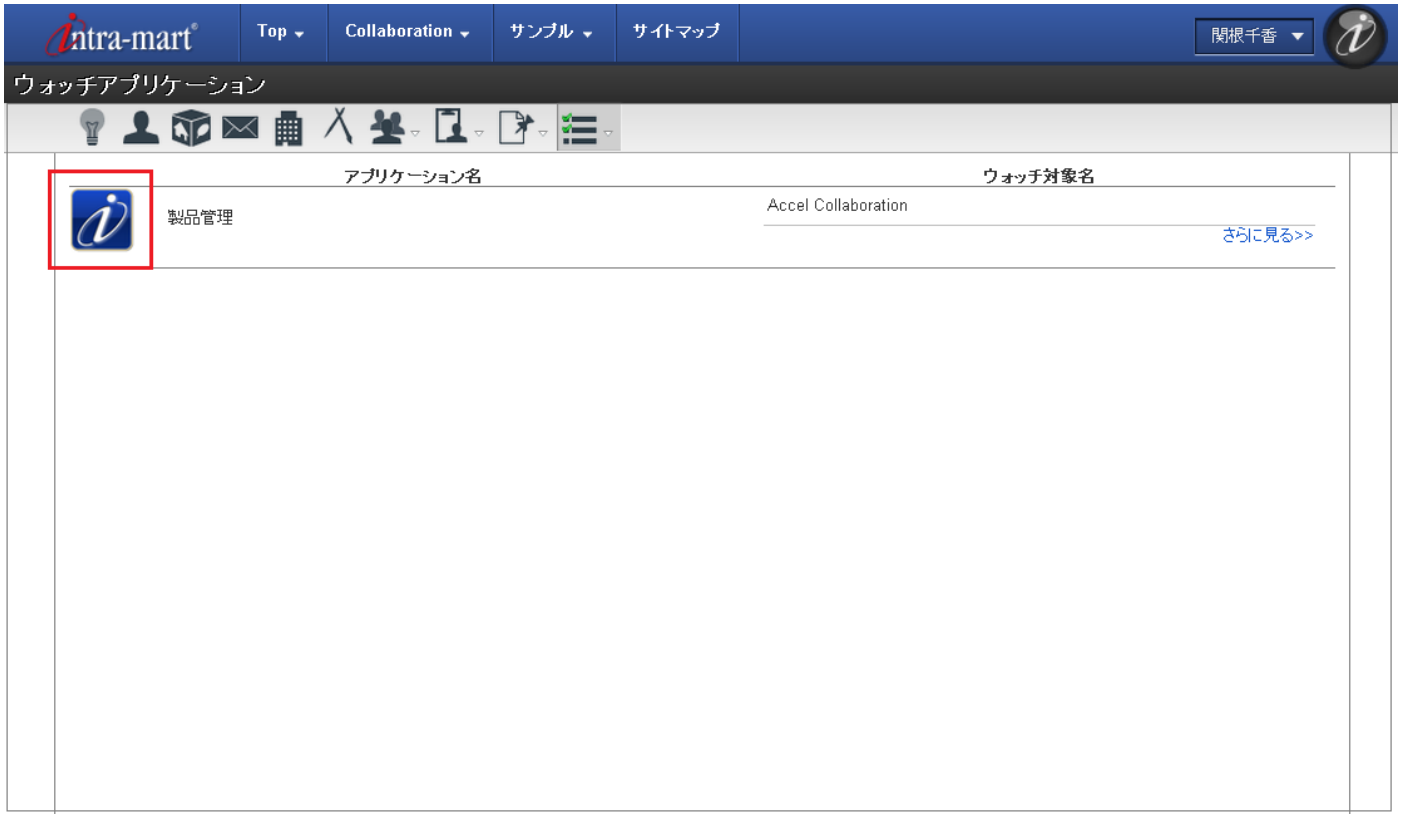
コラム

- **iconPath** が未指定の場合は、 IMBox で用意しているデフォルトのアイコン画像が適用されます。
- **iconPath** を指定する場合、Storage領域に画像を設定してください。
- **messageTypeCd** に *MESSAGE_TYPE_MESSAGE* を設定した場合は、 IMBox で用意しているメッセージ形式でタイムラインに表示されます。
- 独自の形式でApplicationBoxに表示を行いたい場合はメッセージ種別を追加し、追加したメッセージ種別識別CDを設定してください。
- メッセージ種別の追加については *メッセージ種別追加プログラム* を参照してください。

iconPathを指定しない場合



iconPathを指定した場合



ApplicationBoxへの投稿処理の作成

アプリケーション連携を設定した、アプリケーションからの通知メッセージをApplicationBoxに投稿する。
アプリケーションから通知をしたいタイミングで、以下の処理を実装してください。

実装例

IMBox にメッセージを投稿するために、 **imbox.ApplicationBoxService#sendNoticeMessage()** を利用します。

src/main/jssp/src/kintai/register.js

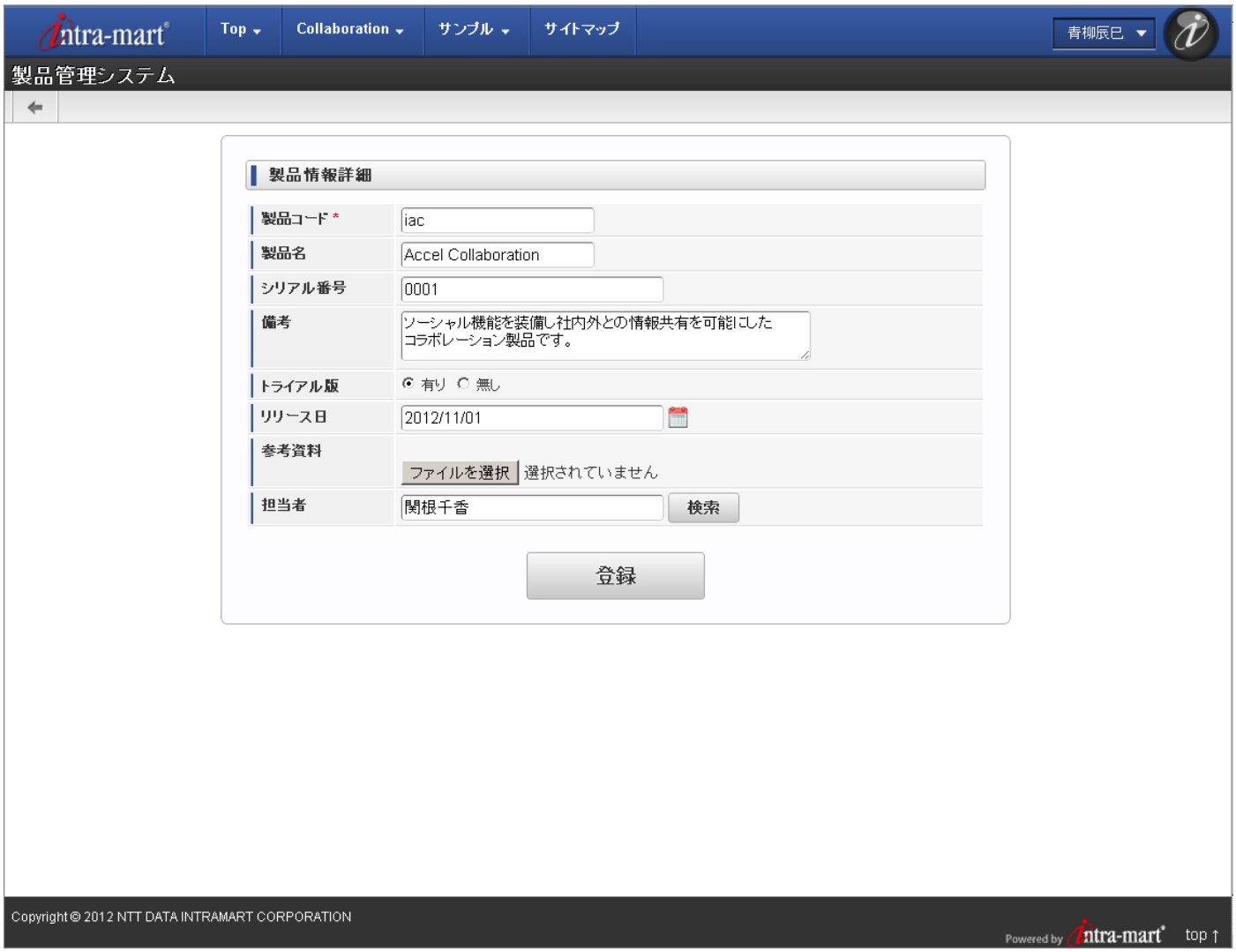
```
function send(){
  var applicationBoxService = new imbox.ApplicationBoxService();
  var message = {
     applicationCd : 'seihin',
     sendUserCd    : 'aoyagi',
     messageText   : '製品情報が登録されました。',
     messageTypeCd : 'MESSAGE_TYPE_MESSAGE',
   };
  applicationBoxService.sendNoticeMessage(message, ['sekine']);
}
```

通知メッセージ情報として、以下のパラメータを設定しています。

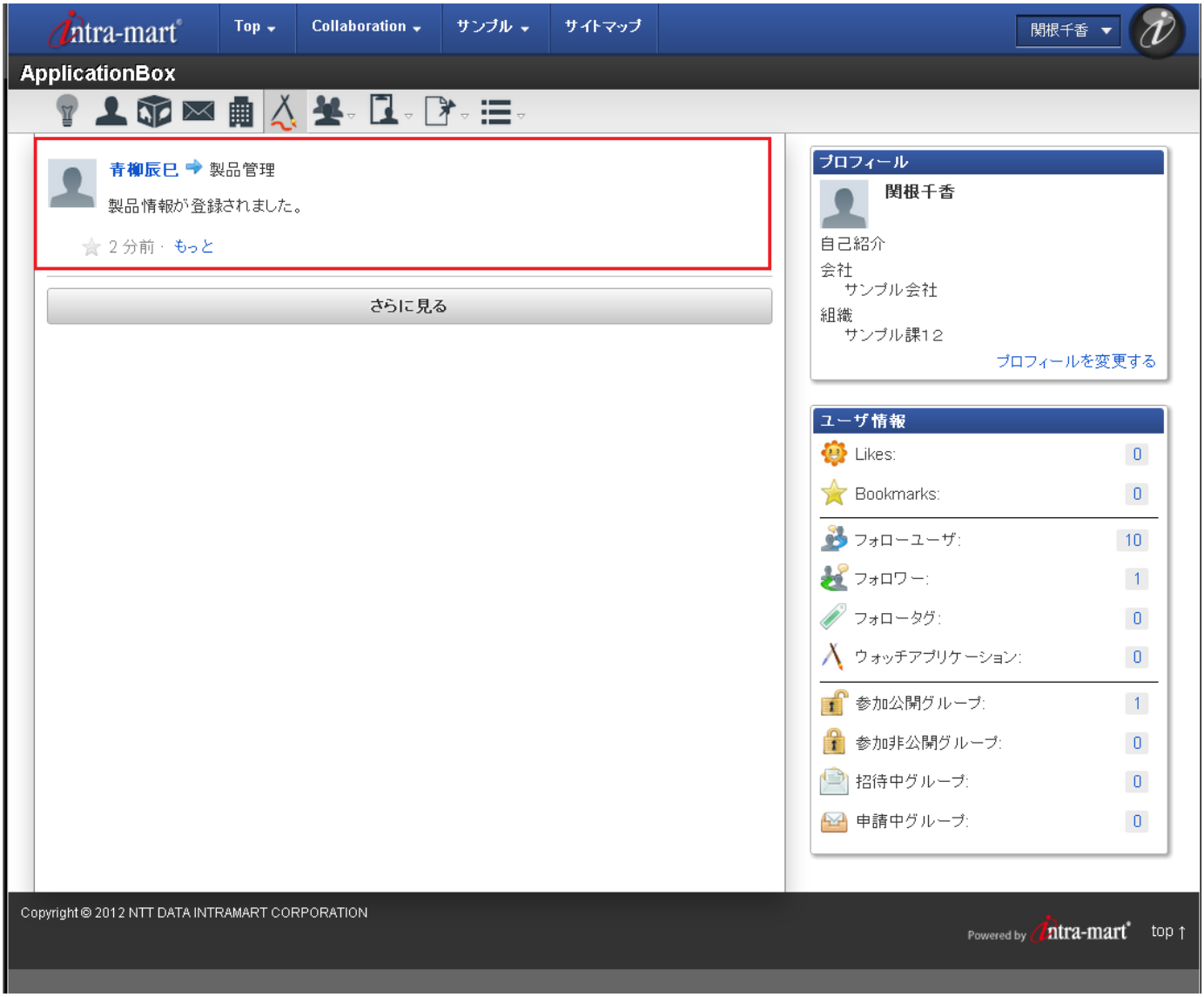
パラメータ	内容
applicationCd	アプリケーションの識別ID
sendUserCd	送信ユーザコード
messageText	表示するメッセージ
messageTypeCd	メッセージ種別識別CD

コラム

- 第二引数には、アプリケーション内のウォッチ行う各情報毎の識別IDを指定します。
- 上記サンプルでは 1 件だけ設定していますが、複数の識別IDを指定することができます。



上記の処理が行われた場合、関根さん(sekine)のApplicationBoxに製品情報の登録通知が以下のように表示されます。



アプリケーション情報のウォッチ処理の作成

アプリケーション情報をウォッチする実装のサンプルです。
アプリケーションのウォッチを行った場合、ウォッチを行ったアプリケーション情報に変更があった際にApplicationBoxへ通知されます。
ウォッチを行いたいアプリケーション情報に対して、以下の処理を実装してください。

実装例

アプリケーションをウォッチするために、 **imbox.ApplicationOperations#watch()** を利用します。

src/main/jssp/src/kintai/watch.js

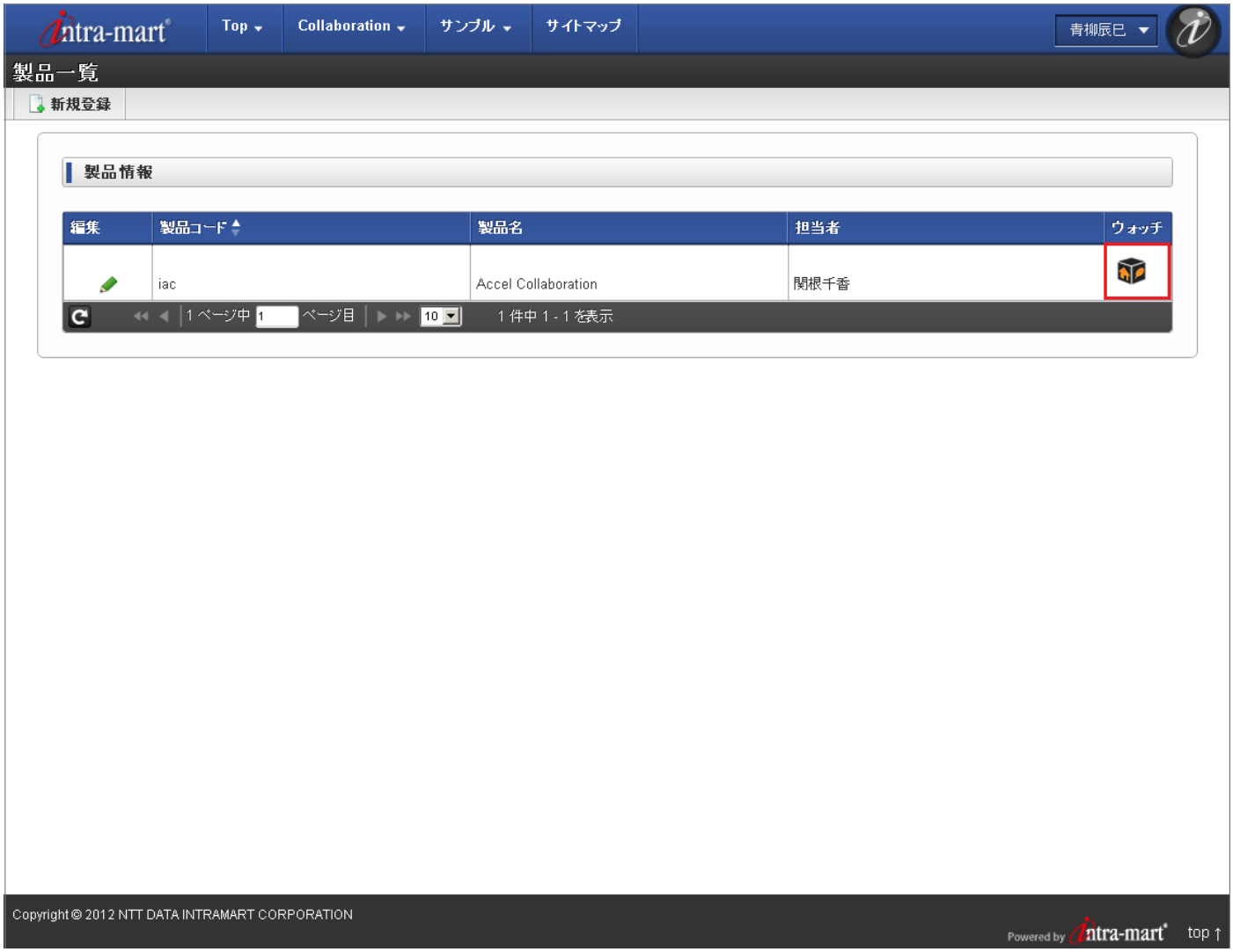
```
1  function watch(){
2    var applicationOperations = new imbox.ApplicationOperations();
3    var applicationCd = 'seihin';
4    var entry4Targets = [];
5    entry4Targets.push({
6       targetId   : 'iac',
7       targetName : 'Accel Collaboration'
8     });
9    applicationOperations.watch(applicationCd,entry4Targets);
10 }
```

ウォッチ情報として、以下のパラメータを設定しています。

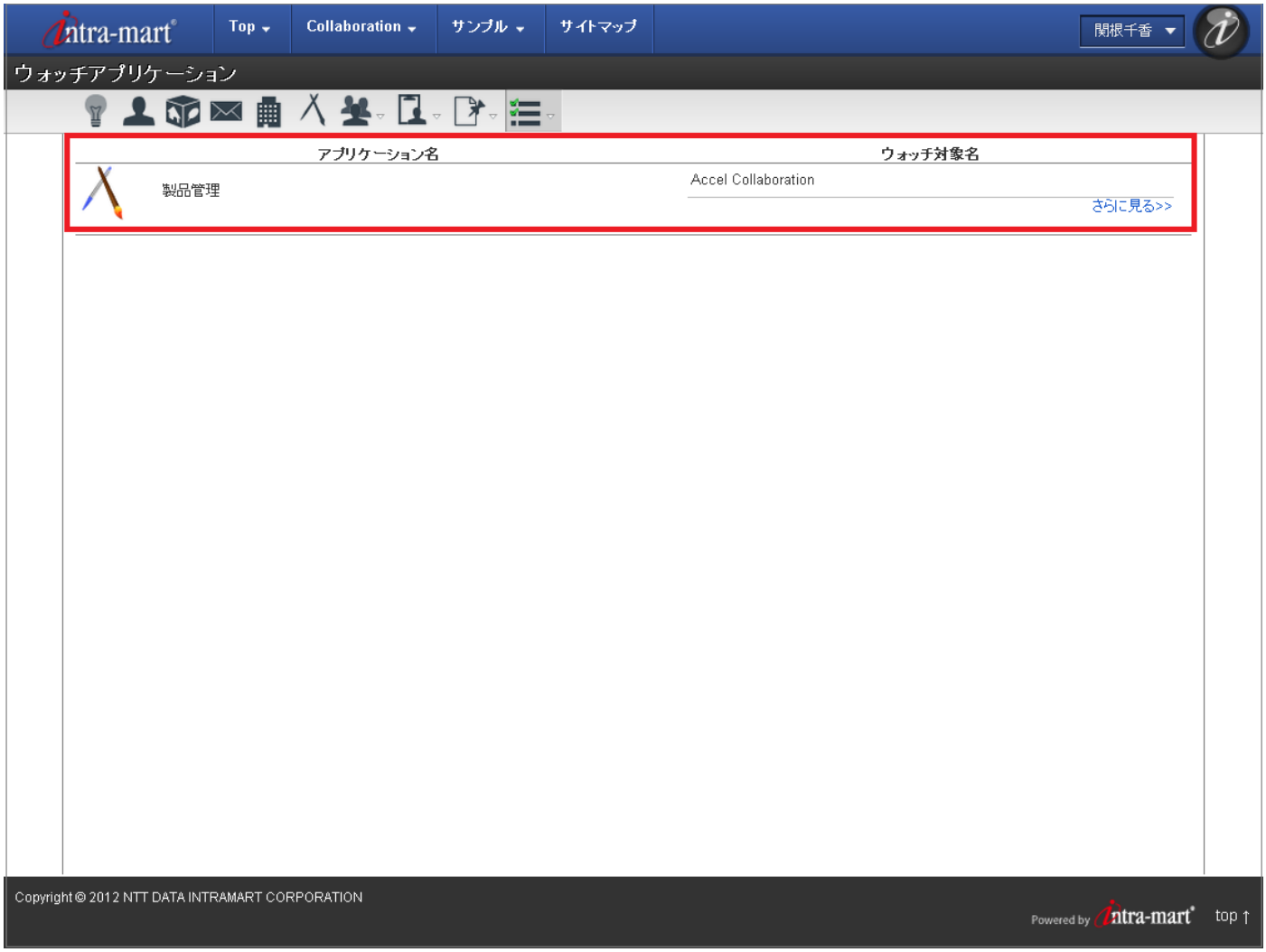
パラメータ	内容
applicationCd	アプリケーションの識別ID
targetId	ウォッチ対象のID
targetName	ウォッチ対象の名称

コラム

- 第二引数には、アプリケーション内のウォッチを行う各情報毎の識別IDを指定します。
  - 上記サンプルでは 1 件だけ設定していますが、複数の識別IDを指定することができます。
- ウォッチを解除する場合は **imbox.ApplicationOperations#unwatch()** を利用してください。



上記の処理が行われた場合、[IMBox の一覧] > [ウォッチアプリケーション画面] に以下のように表示されます。



ApplicationBoxへのメッセージ投稿処理の作成

ウォッチしたアプリケーション情報から、メッセージをApplicationBoxに投稿する実装のサンプルです。
ウォッチを実装したアプリケーション情報に変更があった際にApplicationBoxに通知します。
アプリケーション情報の更新タイミングで、以下の処理を実装してください。

実装例

IMBox にメッセージを投稿するために、 **imbox.ApplicationBoxService#sendWatchMessage()** を利用します。

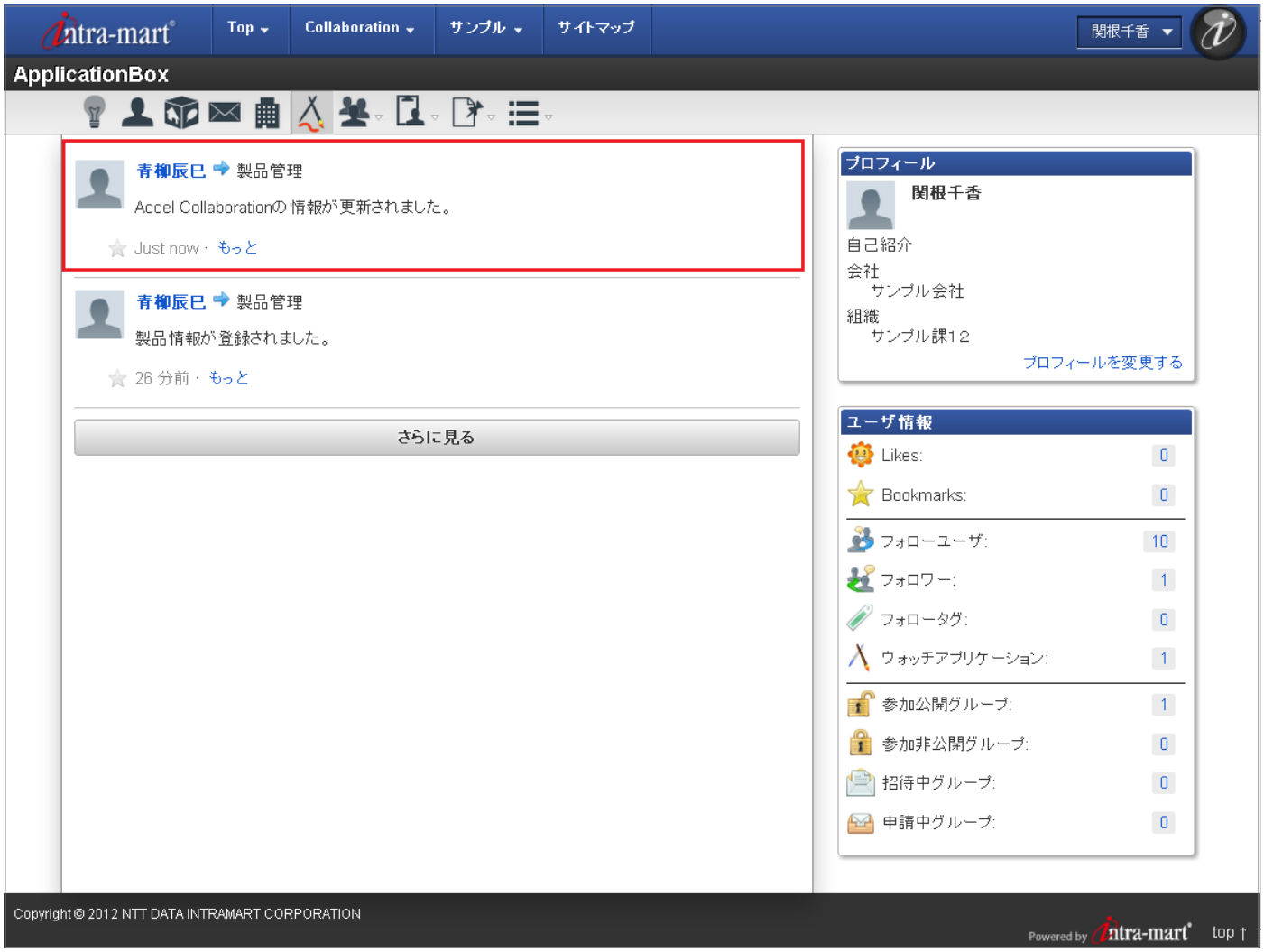
src/main/jssp/src/kintai/update.js

```
1  function  sendWatcher(){
2    var applicationBoxService = new imbox.ApplicationBoxService();
3    var targetId = 'iac';
4    var message = {
5       applicationCd : 'seihin',
6       sendUserCd    : 'aoyagi',
7       messageText   : 'Accel Collaborationの情報が更新されました。',
8       messageTypeCd : 'MESSAGE_TYPE_MESSAGE',
9     };
10   applicationBoxService.sendWatchMessage(message, [targetId]);
11 }
```

ウォッチ通知メッセージ情報として、以下を設定しています。

パラメータ	内容
applicationCd	アプリケーションの識別ID
sendUserCd	送信ユーザコード
messageText	表示するメッセージ
messageTypeCd	メッセージ種別識別CD
targetId	更新されるアプリケーション情報のID（ウォッチで使用するウォッチ対象のID）

上記の処理が行われた場合、ウォッチを行ったユーザのApplicationBoxにAccel Collaborationの更新通知が以下のように表示されます。



コラム

- 通知メッセージ、 ウォッチ通知メッセージの送信時に今回は引数に **threadSummarizeCd** を使用していませんが、
- **threadSummarizeCd** の指定を行うと、 **threadSummarizeCd** 単位でメッセージをスレッド形式で表示することも可能です。

メッセージ種別追加プログラム

IMBox ではメッセージ種別追加プログラムを実装することにより、次に挙げる例のような処理ができます。
作成したメッセージ種別を IMBox に追加することで、タイムラインに投稿することができます。

項目

サンプル実装

ここではサンプルとして、インフォメーションという新しいメッセージ種別を追加する実装方法について説明します。
作成する資材は以下となります。

1. *設定ファイルの作成*
2. *メッセージ種別投稿欄の表示画面の作成*
3. *投稿処理の作成*
4. *タイムライン表示画面*

設定ファイルの作成

メッセージ種別を追加するために、以下のファイルを設定してください。

設定ファイル例

src/main/conf/imbox-message-config/imbox-message-config_information.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<imbox-message-config
   xmlns="http://www.intra-mart.jp/imbox/imbox-message-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/imbox/imbox-message-config ../schema/imbox-message-config.xsd
">

  <public-message>
    <message-config
      display_flag="true"
      enabled_flag="true"
      sort_no="3"
      message_type_name="CAP.INFOMATION"
      post_type_path="app/message_type/type_information"
      display_path="app/message_type/information_timeline"
      message_type_cd="MESSAGE_TYPE_INFOMATION"/>
  </public-message>
</imbox-message-config>
```

<message-config> タグで以下の属性の設定を行います。

属性	内容
display_flag	メッセージ種別セレクトボックス表示フラグ
enabled_flag	使用可否フラグ
message_type_cd	メッセージ種別識別CD
message_type_name	メッセージ種別名称のプロパティID

属性	内容
post_type_path	投稿欄表示用ファイルパス
display_path	タイムライン表示用ファイルパス
iconPath	アプリケーションの表示画像のパス
sort_no	メッセージ種別ソートナンバー

コラム

- **メッセージ種別セレクトボックス表示フラグ** に *true* を設定した場合、 メッセージ種別のセレクトボックスに表示されるようになります。通知のみに使用する場合は *false* を設定してください。
- **使用可否フラグ** を *true* にした場合、使用可能となります。 *false* を設定した場合は、使用不可となります。
- **投稿欄表示用ファイルパス** は投稿欄の表示として使用したいjsspのパスを設定してください。メッセージ種別セレクトボックスを切り替えた際に、表示されます。
- **投稿欄表示用ファイルパス** は **メッセージ種別セレクトボックス表示フラグ** が *false* の場合は必要ありません。
- **タイムライン表示用ファイルパス** はタイムラインに表示として使用したいjsspのパスを設定してください。
- **メッセージ種別ソートナンバー** は既存のメッセージ種別が *1* となっています。メッセージ種別情報返却の際は、このソートナンバーで返却される保障はありません。

メッセージ種別投稿欄の表示画面の作成

設定ファイルに追加したメッセージ種別の「投稿欄表示用ファイルパス」に指定した画面を作成します。

実装例

投稿欄表示用HTMLの作成

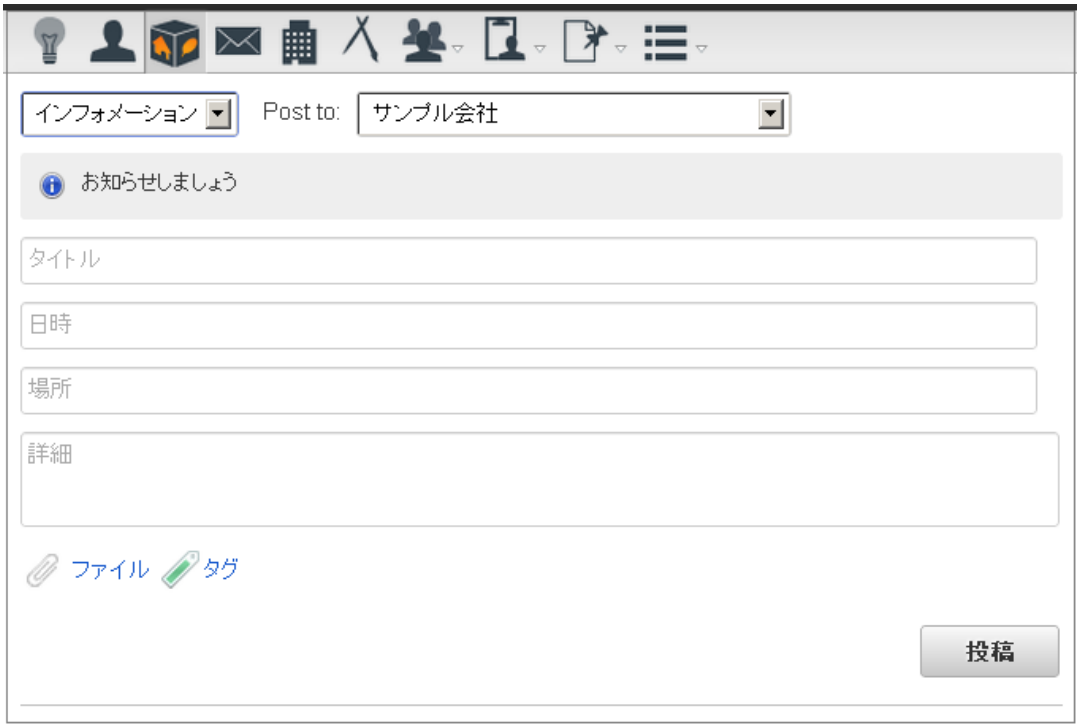
src/main/jssp/src/app/message_type/type_information.html

```
<style type="text/css">
#imui-container #imbox-message-type textarea.imbox-textarea{
  width: 98%;
  height: 45px;
  resize: none;
  overflow: hidden;
}
#imui-container #imbox-message-type  input.imbox-text {
  width: 96%;
}
#imui-container #imbox-message-type div.imbox-button{
 padding-top: 10px;
 text-align: right;
}
</style>
<div>
 <div class="imui-box-supplementation mt-10">
  <span class="im-ui-icon-common-16-information float-L"></span>
  <p class="imui-pgh-section imbox-timeline-question-info">
   <imart type="string" value="お知らせしましょう" />
  </p>
 </div>
 <form id="information_form">
  <ul class= "mt-10">
   <li class= "mt-10"> <imart type="imuiTextbox" id="information-title" class="imbox-text" name="information-
title-name" placeholder="タイトル"/></li>
   <li class= "mt-10"><imart type="imuiTextbox" id="information-date" class="imbox-text" name="information-
date-name" placeholder="日時" /></li>
   <li class= "mt-10"><imart type="imuiTextbox" id="information-place" class="imbox-text" name="information-
place-name" placeholder="場所"/></li>
  </ul>
  <div class="mt-10">
   <textarea id="information-detail" class="imbox-textarea" name="information-detail-name" placeholder="詳
細"></textarea>
  </div>
 </form>
</div>
<form id="imbox_timeline_send_form" action="ui/filer/upload" method="POST" enctype="multipart/form-data">
 <imart type="include" page="imbox/views/timeline/item/add_options" />
</form>
<div id="information_button" class="imbox-button mt-20 align-R">
 <button type="button" id="information_send_button" class="imui-medium-button">投稿</button>
</div>
```

タグ・添付ファイルを表示する場合は、作成するHTMLに以下のタグを追加してください。

```
<form id="imbox_timeline_send_form" action="ui/filer/upload" method="POST" enctype="multipart/form-data">
 <imart type="include" page="imbox/views/timeline/item/add_options" />
</form>
```

この位置に表示されます。



投稿処理の作成

メッセージ種別を独自に追加した場合、対応する投稿処理をCSJSにて実装する必要があります。

投稿処理のCSJS実装例

```
<script type="text/javascript">
(function($){
  $('#information_send_button').click(function(){
    var button = $(this);
    var attributes = {
      informationDate : $('#information-date').val(),
      informationPlace: $('#information-place').val(),
      informationDetail : $('#information-detail').val()
    };
    var title = $('#information-title').val();
    var messageTypeCd = $('.imbox-timeline-sendtype-list').children(':selected').val();
    var boxCd = ($imbox.timeline.clientType === $imbox.constants.CLIENT_TYPE_LIST['COMPONENTS']) ?
$imbox.timeline.boxCd: $('.imbox-timeline-grouptype-list').children(':selected').val();
    var url = 'imbox/send/' + encodeURIComponent(boxCd);
    var attachFlag = '0';
    var files = [];
    var tagNames = [];
    $('.imbox-timeline-send-tag-list-hidden').each(function(){
      tagNames.push($(this).val());
    });

    if(tagNames.length){
      attachFlag = '1';
    }

    for(i = 0, length = $imbox.fileName.length; i < length; i++){
        if($imbox.fileName[i].key === 'imbox_timeline_send_form'){
          files.push($imbox.fileName[i].name);
        }
      }

    var data = {
      'send_message': title,
      'messageTypeCd': messageTypeCd,
      'displayId': $imbox.timeline.displayId,
      'timelineType': $imbox.timeline.timelineType,
      'clientType': $imbox.timeline.clientType,
      'attributes': ImJson.toJSONString(attributes, false),
      'attachName[]':  files,
      'attachPath': $('#imbox_timeline_send_attach_file_name').data('store_to'),
      'attachFlag': attachFlag,
      'tag_name':   tagNames
    };

    if(!imuiValidate('#imbox_timeline_send_form', imboxTimelineSendRules, imboxTimelineSendMessage, '')){
      return false;
    }
    var success = $imbox.ajax.send('POST', url, data, $imbox.timeline.sendCallback, button);

    if (success) {
      $('#information-date').val('');
      $('#information-place').val('');
      $('#information-detail').val('');
      $('#information-title').val('');
      imuiResetForm('#imbox_timeline_send_form');
    }
  });
})(jQuery);
</script>
```

url パラメータは以下のようにエンコーディング処理を行う必要があります。

```
url = 'imbox/send/' + encodeURIComponent(boxCd);
```

$imbox.ajax.send メソッドを使用する際に渡すデータにおいて、以下のパラメータは必須となります。

パラメータ	内容
send_message	投稿内容（任意の文字列を入力する箇所で利用してください。）
messageTypeCd	メッセージ種別識別CD
displayId	画面ID
timelineType	投稿した画面のタイムライン種別（スレッド形式で表示している画面は「TIMELINE_TYPE_THREAD」の固定値、メッセージ形式で表示している画面は「TIMELINE_TYPE_MESSAGE」の固定値となります。）
clientType	操作しているクライアントの種別（PC版IMBoxを利用している場合は「IMBOX」の固定値、スマートフォン版IMBoxを利用している場合は「MOBILE」の固定値、IMBox連携用画面を利用している場合は「IMBOX」の固定値となります。）

コラム

- 拡張項目を使用する場合は **attributes** オブジェクトを作成し、作成したい情報を設定してください。
- SSJSの投稿処理を記述する必要はありません。

タイムライン表示画面

設定ファイルに追加したメッセージ種別の「タイムライン表示用ファイルパス」に指定した画面を作成します。
拡張項目として記述した **attributes** オブジェクトを表示する場合は、対応する表示処理をSSJSにて実装する必要があります。

実装例

タイムライン表示用SSJSの作成

src/main/jssp/src/app/message_type/information_timeline.js

```
1  var $imbox = {};
2
3  /**
4   * インフォメーション用タイムライン画面初期処理
5   * @param  {Object} request リクエストパラメータ
6   */
7  function init(request){
8    let message = request.message;
9    let attributes = request.message.attributes;
10   $imbox.informationTitle = message.messageText;
11   $imbox.informationDate = attributes.informationDate;
12   $imbox.informationPlace = attributes.informationPlace;
13   $imbox.informationDetail = attributes.informationDetail;
14   $imbox.message = message;
15 }
```

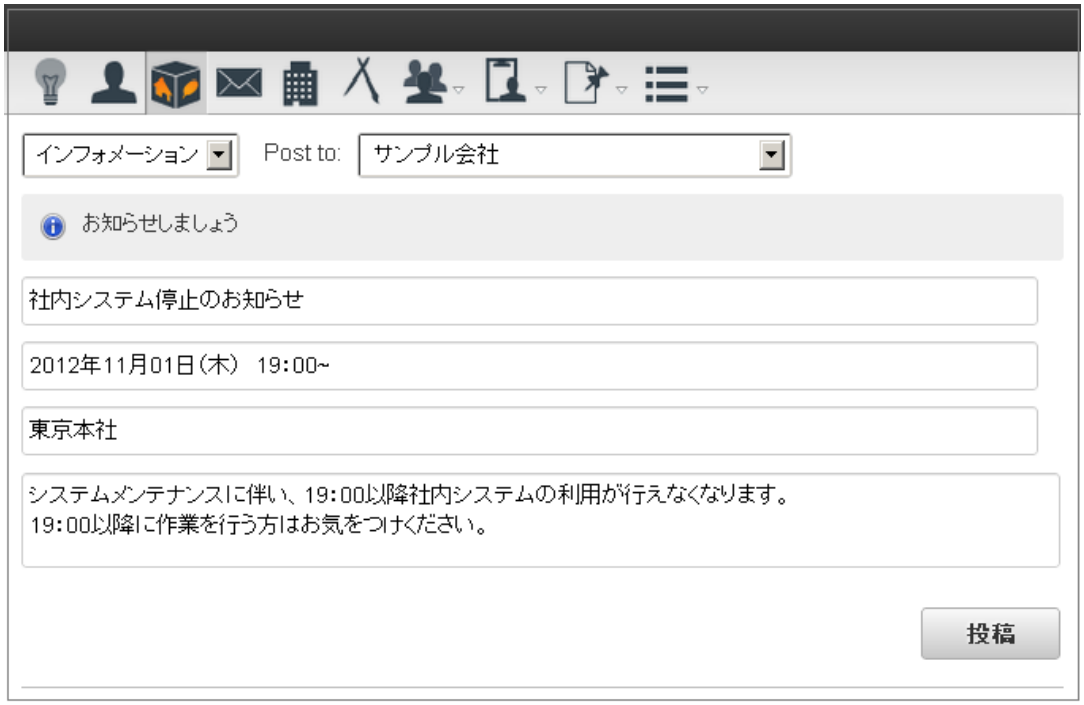
タイムライン表示用HTMLの作成

src/main/jssp/src/app/message_type/information_timeline.html

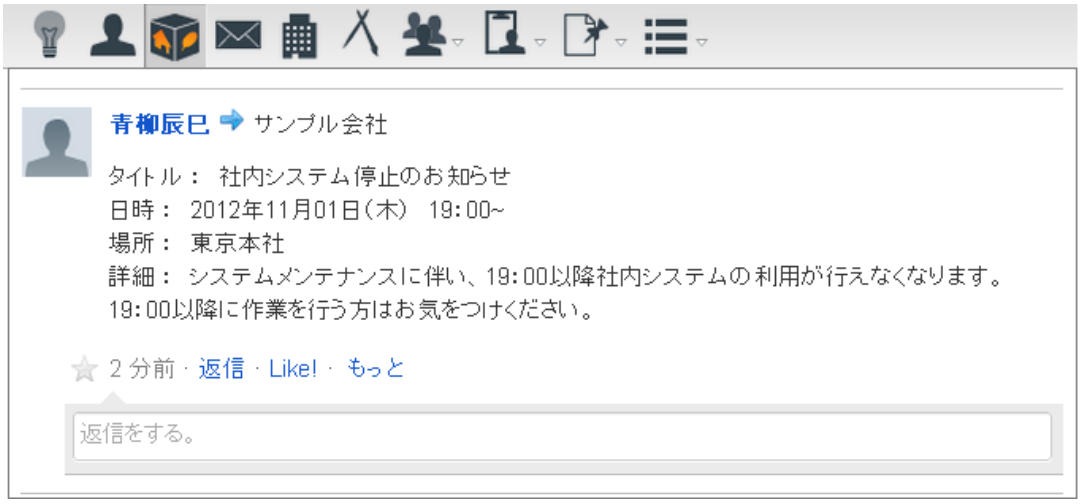
```
<div class="imbox-timeline-thread-post-right-body">
 <p>
  <a href="imbox/usermessage/<imart type="string" value=$imbox.message.encodePostUserCd escapeXml="true"
escapeJs="false" />" class="imbox-timeline-thread-post-user-name">
    <imart type="string" value=$imbox.message.postUserName escapeXml="true" escapeJs="false" />
  </a>
  <span class="imbox-timeline-thread-from-to-icon imbox-icon-common-16-arrow"></span>
  <imart type="string" value=$imbox.message.postTypeInfo.postToName escapeXml="true" escapeJs="false" />
 </p>
 <div class="imbox-timeline-thread-post-right-body">
  <div class="imbox-timeline-thread-message">
   <div class="imbox-timeline-thread-message-area">
    <div class="imbox-timeline-thread-message-text">
     <div class="cf">
      <div class="float-L mt-10">
       タイトル ： <imart type="string" value=$imbox.informationTitle /></span><br>
       日時 ： <imart type="string" value=$imbox.informationDate /></span><br>
       場所 ： <imart type="string" value=$imbox.informationPlace /></span><br>
       詳細 ： <imart type="string" value=$imbox.informationDetail /></span><br>
      </div>
     </div>
    </div>
   </div>
  </div>
 </div>
</div>
```

サンプルを実装すると以下のように利用することが出来ます。

インフォメーション投稿時



インフォメーションタイムライン表示時


青柳辰巳
サンプル会社
タイトル： 社内システム停止のお知らせ
日時： 2012年11月01日(木) 19:00~
場所： 東京本社
詳細： システムメンテナンスに伴い、19:00以降社内システムの利用が行えなくなります。
19:00以降に作業を行う方はお気をつけください。
2 分前 · 返信 · Like! · もっと
返信をする。

送信側 IM-Propagation 追加プログラム

IMBoxでは受信側の IM-Propagation を用意しています。
送信側の実装を行うことで、IM-Propagation を利用して「*受信側 IM-Propagation 一覧*」に記載している処理が実行可能となります。
IM-Propagation についての詳細は、以下のドキュメントを参照してください。

- IM-Propagation 仕様書
- IM-Propagation プログラミングガイド
- IM-Propagation 送受信設定一覧

受信側 IM-Propagation 一覧

IMBoxで利用可能な受信側の IM-Propagation は以下となります。

IMBox受信側 IM-Propagation 一覧表

データの操作種別	概要
WATCH	アプリケーションのウォッチを行います。
UNWATCH	アプリケーションのウォッチ解除を行います。
SEND_NOTICE_MESSAGE	ApplicationBoxへ通知メッセージを投稿します。（メッセージ形式で表示）
SEND_NOTICE_THREAD	ApplicationBoxへ通知メッセージを投稿します。（スレッド形式で表示）
SEND_WATCH_MESSAGE	ApplicationBoxへwatchユーザ宛てのメッセージを投稿します。（メッセージ形式で表示）
SEND_WATCH_THREAD	ApplicationBoxへwatchユーザ宛てのメッセージを投稿します。（スレッド形式で表示）

サンプル実装

ここではサンプルとして、「ApplicationBoxへのwatchユーザ宛てのメッセージ投稿（メッセージ形式で表示）」の送信側の実装方法について説明します。
作成する資材は以下となります。

1. *送信するデータを格納するためのクラスの作成*
2. *送信側のデータ変換クラス（Encoder）の作成*
3. *マッピング設定の作成*
4. *データ送信処理を作成*

コラム
- その他の「*受信側 IM-Propagation 一覧*」の送信側の実装方法も以下のサンプルと同様になります。

注意

- 本章では「 *アプリケーション連携プログラム* 」のサンプル実装と同様に、連携プログラムが存在する前提で記載しています。
- 実行される処理自体は、「 *ApplicationBoxへのメッセージ投稿処理の作成* 」と同様になります。

送信するデータを格納するためのクラスの作成

送信するデータを格納するためのクラス（以下、「独自モデル」）を作成します。

送信するデータを格納するためのクラス

```java
package jp.co.intra_mart.sample.model;

import java.io.Serializable;
import java.util.Map;

/**
 * ApplicationBoxへのwatchユーザ宛てのメッセージ投稿（メッセージ形式で表示） 用の
 * 送信するデータを格納するための独自モデルクラス
 */
public class SampleOriginalModel implements Serializable {

    /** バージョン番号（新しく採番してください） */
    private static final long serialVersionUID = 1234567890123456789L;

    /** アプリケーションCD */
    private String applicationCd;

    /** 通知元ユーザのユーザCD */
    private String sendUserCd;

    /** 通知内容 */
    private String messageText;

    /** 投稿uri */
    private String uri;

    /** 投稿uriタイトル */
    private String uriTitle;

    /** URI添付ID */
    private String uriAttachId;

    /** 投稿uri内容 */
    private String uriText;

    /** 投稿URI添付パス */
    private String uriAttachPath;

    /** メッセージ種別コード */
    private String messageTypeCd;

    /** その他の属性 */
    private Map<String, String> attributes;

    /** ウォッチ対象の識別CD（複数指定可能） */
    private String[] targetIds;

    public String getApplicationCd() {
        return applicationCd;
    }

    public Map<String, String> getAttributes() {
        return attributes;
    }

    public String getMessageText() {
        return messageText;
    }

    public String getMessageTypeCd() {
        return messageTypeCd;
    }

    public String getSendUserCd() {
        return sendUserCd;
    }

    public String[] getTargetIds() {
        return targetIds;
```

```
70      }
71
72      public String getUri() {
73          return uri;
74      }
75
76      public String getUriAttachId() {
77          return uriAttachId;
78      }
79
80      public String getUriAttachPath() {
81          return uriAttachPath;
82      }
83
84      public String getUriText() {
85          return uriText;
86      }
87
88      public String getUriTitle() {
89          return uriTitle;
90      }
91
92      public void setApplicationCd(final String applicationCd) {
93          this.applicationCd = applicationCd;
94      }
95
96      public void setAttributes(final Map<String, String> attributes) {
97          this.attributes = attributes;
98      }
99
100     public void setMessageText(final String messageText) {
101         this.messageText = messageText;
102     }
103
104     public void setMessageTypeCd(final String messageTypeCd) {
105         this.messageTypeCd = messageTypeCd;
106     }
107
108     public void setSendUserCd(final String sendUserCd) {
109         this.sendUserCd = sendUserCd;
110     }
111
112     public void setTargetIds(String[] targetIds) {
113         this.targetIds = targetIds;
114     }
115
116     public void setUri(final String uri) {
117         this.uri = uri;
118     }
119
120     public void setUriAttachId(final String uriAttachId) {
121         this.uriAttachId = uriAttachId;
122     }
123
124     public void setUriAttachPath(final String uriAttachPath) {
125         this.uriAttachPath = uriAttachPath;
126     }
127
128     public void setUriText(final String uriText) {
129         this.uriText = uriText;
130     }
131
132     public void setUriTitle(final String uriTitle) {
133         this.uriTitle = uriTitle;
134     }
135 }
```

送信側のデータ変換クラス（Encoder）の作成

「独自モデル」を「送受信モデル（Generic）」に変換する機能を提供するクラス（以下、「データ変換クラス」）を作成します。

送信側のデータ変換クラス

```java
package jp.co.intra_mart.sample.encoder;

import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Map.Entry;

import jp.co.intra_mart.foundation.propagation.exception.ConvertException;
import jp.co.intra_mart.foundation.propagation.model.generic.imbox.GenericSendWatchMessage;
import jp.co.intra_mart.foundation.propagation.sender.AbstractEncoder;
import jp.co.intra_mart.sample.model.SampleOriginalModel;

/**
 * ApplicationBoxへのwatchユーザ宛てのメッセージ投稿（メッセージ形式で表示） 用の
 * 送信側のデータ変換クラス（Encoder）です。
 */
public class SampleEncoder extends AbstractEncoder<SampleOriginalModel, GenericSendWatchMessage> {

    /**
     * @param model 変換前クラス
     * @return 変換後クラス
     * @throws ConvertException
     */
    @Override
    public GenericSendNoticeThread encode(final SampleOriginalModel model) throws ConvertException {

        final GenericSendWatchMessage generic = new GenericSendWatchMessage();
        generic.setApplicationCd(model.getApplicationCd());
        generic.setSendUserCd(model.getSendUserCd());
        generic.setMessageText(model.getMessageText());
        generic.setUri(model.getUri());
        generic.setUriTitle(model.getUriTitle());
        generic.setUriAttachId(model.getUriAttachId());
        generic.setUriText(model.getUriText());
        generic.setUriAttachPath(model.getUriAttachPath());
        generic.setMessageTypeCd(model.getMessageTypeCd());
        generic.setTargetIds(model.getTargetIds());
        if (null != model.getAttributes()) {
            final Map<String, String> map = new LinkedHashMap<String, String>();
            for (final Entry<String, String> attribute : model.getAttributes().entrySet()) {
                map.put(attribute.getKey(), attribute.getValue());
            }
            generic.setAttributes(map);
        }
        return generic;
    }

    @Override
    public Class<GenericSendWatchMessage> getGenericDataClass() {
        return GenericSendWatchMessage.class;
    }
}
```

> **コラム**
>
> IMBoxの「送受信モデル（Generic）」についての詳細は、「 APIドキュメント 」 を参照してください。

マッピング設定の作成

「独自モデル」「送受信モデル（Generic）」「データ変換クラス」を紐付けるためのマッピング設定を作成します。
マッピング設定は、以下の形式で記述します。

- *WEB-INF/conf/propagation-senders-config/{任意のファイル名}.xml*

マッピング設定

```
<?xml version="1.0" encoding="UTF-8"?>
<propagation-senders-config xmlns="http://www.intra-mart.jp/propagation/senders-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/propagation/senders-config propagation-senders-config.xsd">
  <sender source="jp.co.intra_mart.sample.model.SampleOriginalModel" operationType="SEND_WATCH_MESSAGE">
    <encoder class="jp.co.intra_mart.sample.encoder.SampleEncoder" />
  </sender>
</propagation-senders-config>
```

<sender> タグに以下の属性の設定を行います。

属性	内容	備考
source	「独自モデル」のパッケージ名を含むクラス名（完全修飾子）	*送信するデータを格納するためのクラスの作成* で作成した独自モデルを設定します。
operationType	データ変換対象の「データの操作種別」	ApplicationBoxへのwatchユーザ宛てのメッセージ投稿（メッセージ形式で表示）のデータの操作種別を設定します。

<encoder> タグに以下の属性の設定を行います。

属性	内容	備考
class	「データ変換クラス」のパッケージ名を含むクラス名（完全修飾子）	*送信側のデータ変換クラス（Encoder）の作成* で作成したデータ変換クラスを設定します。

> **コラム**
>
> データの操作種別についての詳細は、「 IM-Propagation 送受信設定一覧 」-「 IM-Propagation 受信側一覧 」を参照してください。

データ送信処理を作成

データの送信処理を作成します。
作成したデータ送信処理（以下）を、アプリケーションから通知をしたいタイミングでコールしてください。

データ送信処理

```java
package jp.co.intra_mart.sample.sender;

import jp.co.intra_mart.sample.model.SampleOriginalModel;

/**
 * データ送信処理のサンプル実装です
 */
public class SampleSender {


   /**
    * データ送信処理です。
    * アプリケーションから通知をしたいタイミングでコールしてください。
    *
    * @param sampleOriginalModel 独自モデル
    */
   public static void send(final SampleOriginalModel sampleOriginalModel) {
      final PropagationManager manager = PropagationManagerFactory.getInstance().getPropagationManager();
      try {
         // セッションを開始
         manager.begin();
         // データを送信
         manager.send("SEND_WATCH_MESSAGE", SampleOriginalModel.class, sampleOriginalModel,
EmptyObject.class);
        // セッションを確定
         manager.decide();
      } catch (final BeginException e) {
         // manager.begin() に失敗した場合
      } catch (final SendException e) {
         // manager.send() に失敗した場合
      } catch (final DecideException e) {
         // manager.decide() に失敗した場合
      } catch (final Exception e) {
         // その他例外が発生した場合
      } finally {
      // セッションを中断。 manager.decide() が成功した場合は何もしない
         manager.abort();
      }
   }
}
```

上記の処理がコールされた場合に、ウォッチを行ったユーザのApplicationBoxにwatchユーザ宛てのメッセージが表示されます。