目次

- 1. 改訂情報
- 2. はじめに
  - 2.1. 本書の目的
  - 2.2. 対象読者
  - 2.3. サンプルコードについて
- 3. 概要
  - 3.1. 履歴・コメントモジュール とは
  - 3.2. 用語
    - 3.2.1. アプリケーション
    - 3.2.2. イベントグループ
    - 3.2.3. イベント
    - 3.2.4. イベントタイプ
    - 3.2.5. アクション
- 4. 開発環境のセットアップ
  - 4.1. プロジェクトの作成と設定
  - 4.2. モジュールの依存関係について
  - 4.3. サンプルプロジェクトのインポートと設定
  - 4.4. サンプルについて
- 5. 履歴・コメントモジュールの利用方法
  - 5.1. アプリケーションへの組み込み
    - 5.1.1. スクリプト開発モデルで履歴・コメントモジュールを組み込む
  - 5.2. 操作権限設定の実装
    - 5.2.1. 操作権限の設定を行うJavaファイルの作成
    - 5.2.2. 操作権限の設定を行うクラスの登録
- 6. カスタムイベント
  - 6.1. カスタムイベントとは
  - 6.2. カスタムイベントの実装
    - 6.2.1. 履歴・コメントモジュールへのイベント登録
    - 6.2.2. 表示用テンプレート
      - 6.2.2.1. 表示用テンプレートとは
      - 6.2.2.2. テンプレートの実装方法
      - 6.2.2.3. 表示用テンプレートの作成
      - 6.2.2.4. 設定ファイルの作成
- 7. イベントハンドラ
  - 7.1. イベントハンドラとは
  - 7.2. イベントハンドラの実装方法
    - 7.2.1. イベントハンドラの処理を行うJavaファイルの作成
    - 7.2.2. イベントハンドラの処理を行うクラスの登録
- 8. REST API
  - 8.1. REST APIについて
  - 8.2. 認証方式
  - 8.3. 認可
  - 8.4. エンドポイント
  - 8.5. Swagger
  - 8.6. REST APIドキュメント
  - 8.7. イベントグループキーについて

- 9. 付録
  - 9.1. OAuth認証を利用した外部アプリケーション連携
    - 9.1.1. intra-mart Accel Platform上にてクライアントアプリケーションの追加を行う
    - 9.1.2. 外部アプリケーションより履歴・コメントモジュールのREST APIのエンドポイントを呼び出す

改訂情報

変更年月日	変更内容
2020-08-01	初版
2020-12-01	第2版 下記のページを追加しました。 ▪ 「*REST API*」 ▪ 「*付録*」を追加しました。 ▪ 「*OAuth認証を利用した外部アプリケーション連携*」を追加しました。
2023-10-01	第3版 下記を変更しました。 ▪ 「*表示用テンプレート*」の journal_template_sample.js のプロファイル画像取得部分を修正しました。

はじめに

本書の目的

本書では、履歴・コメントモジュールにおけるそれぞれの機能を拡張する仕組の詳細を説明します。

対象読者

本書では、次のユーザを対象としています。

- アプリケーションで、履歴・コメントモジュールを利用する方法を知りたい。

サンプルコードについて

本書で使用するサンプルプログラムはあくまでも、履歴・コメントモジュール機能の利用方法を理解することに主眼をおいています。
そのため、必ずしもベストなコーディング方法とはいえない方法をあえて取っている箇所があります。
あくまでも、サンプルとしての位置付けでとらえるようにしてください。

概要

項目

- 履歴・コメントモジュール とは
- 用語

履歴・コメントモジュール とは

履歴・コメントモジュールは、タイムライン形式で履歴の表示やコメントの投稿を行う機能です。

- 履歴・コメントモジュールを利用することで、業務画面に履歴・コメント機能を簡単に追加できます。
- 「*カスタムイベント*」を作成することで、独自のレイアウトで履歴・コメントを表示できます。
- 「*イベントハンドラ*」を作成することで、コメントやカスタムイベントの登録を検知して、任意の処理を行うことができます。

> **注意**
>
> 履歴・コメントモジュールは、アプリケーションに組み込んで利用するモジュールであり、単体では機能しません。

用語

履歴・コメントモジュールには、以下のような構成要素があります。

アプリケーション

履歴・コメントモジュールを組み込む対象のアプリケーションを指します。

- intra-mart Accel Platformが提供するアプリケーションでは、「 チケットモジュール」「 IM-BPM for Accel Platform」などが挙げられます。

イベントグループ

履歴・コメントモジュールの個々のタイムラインを指します。

- 1つのイベントグループに対して 1 つのアプリケーションが紐付きます。
- 1つのイベントグループ（タイムライン）に対して複数のイベント（履歴）が所属します。

イベント

履歴・コメントモジュールのタイムラインに表示される個々のイベント（履歴）データを指します。
イベントは、以下の情報から構成されます。

- イベント操作日時
- イベント操作ユーザ
- イベントタイプ
- アクション

イベントタイプ

イベントの種類を指します。

- イベントタイプは、イベントをタイムラインに表示する際のテンプレートを定義します。
- 履歴・コメントモジュールで提供されるイベントタイプは、「コメント/添付ファイルイベント」と「添付ファイル削除イベント」の2種類です。
- カスタムイベントを用意することで、独自のテンプレートを持ったイベントをタイムライン上に表示することが可能です。

アクション

イベント内で行った操作を表すデータを指します。

- 1つのイベントデータに対して複数のアクションを所属させることが可能です。
- 例えば、「コメント/添付ファイルイベント」では、添付ファイルを登録した場合に「コメント」と「添付ファイル」の2つのアクションのデータが登録されます。

開発環境のセットアップ

プロジェクトの作成と設定

intra-mart e Builder for Accel Platform 上にモジュール・プロジェクトを作成し、プロジェクトの設定を行います。
プロジェクトの作成・設定の方法に関しては、「 intra-mart e Builder for Accel Platform アプリケーション開発ガイド」の「モジュール・プロジェクト作成」、および「プロジェクトの設定」を参照してください。

モジュールの依存関係について

作成したモジュール・プロジェクトに、以下のモジュールへの依存関係を追加してください。

- 「 履歴・コメントモジュール 」

モジュールID	jp.co.intra_mart.im_journal
バージョン	8.0.2

> **コラム**
>
> 本書で解説している 履歴・コメントモジュール の説明は、バージョン 8.0.2 時点の動作に基づいて記載されています。
> 必要に応じて、 8.0.2 以上のバージョンを指定できます。

モジュールへの依存関係追加の方法に関しては、「 intra-mart e Builder for Accel Platform アプリケーション開発ガイド 」の「module.xml」を参照してください。

サンプルプロジェクトのインポートと設定

- 開発環境のセットアップ

  以下の手順により、サンプルコードの動作確認を行うことが可能です。

  1. IM-Jugglingを起動し、Jugglingプロジェクトを作成します。
  2. Jugglingプロジェクトのモジュール構成には、「 履歴・コメントモジュール 」を含めたままにします。
  3. Jugglingプロジェクトのユーザモジュールタブから「 **im_journal_examples-8.0.1.imm** 」を取り込みます。
  4. Jugglingプロジェクトからwarファイルを作成します。サンプルを含めるにチェックを入れます。
  5. 作成されたwarファイルをデプロイし、デバッグサーバを起動します。
  6. 起動完了後に、システム管理者としてログインし、テナント環境セットアップを実施します。
  7. `%ベースURL%/sample/journal/examples` を開きます。

- プロジェクトのインポート

  サンプルのユーザアプリケーションを含んだモジュール・プロジェクト「 **im_journal_examples-8.0.1.zip** 」をダウンロードし、以下の手順で e Builder にインポートします。

  1. e Builder を起動
  2. ツールバーメニュー[ファイル]-[インポート]よりインポートウィザード画面を開く
  3. 項目[General]-[既存プロジェクトをワークスペースへ]を選択し次へ
  4. [アーカイブ・ファイルの選択]項目よりダウンロードしたzipファイルを選択し、[終了]ボタンをクリック

5. 以上の手順で、モジュール・プロジェクト「im_journal_examples」がインポートされます。

- ビルドパスの設定

  モジュール・プロジェクト「im_journal_examples」を修正後に以下の手順を実施することで、デバッグサーバの環境に適用できます。

  1. モジュール・プロジェクトのプロパティにてWebアーカイブディレクトリを設定します。
  2. プロジェクトのクリーンを実行します。
  3. クリーンの完了後に、デバッグサーバを起動します。

サンプルについて

サンプルモジュールの仕様を簡単に説明します。

- 「*履歴・コメントモジュールの利用方法*」
  - スクリプト開発モデルでの履歴・コメントモジュールの記述方法を説明します。
  - 履歴・コメントモジュールの操作権限の制御をJavaで行うための説明をします。
- 「*カスタムイベント*」
  - Javaでカスタムイベントを登録する記述方法を説明します。
  - 登録されたカスタムイベントの表示用のテンプレートの記述方法を説明します。
- 「*イベントハンドラ*」
  - Javaでイベントハンドラを実装する記述方法を説明します。

履歴・コメントモジュールの利用方法

この章では、スクリプト開発モデルで履歴・コメントモジュールを利用する手順や注意事項を説明します。

実装の流れ

- アプリケーションへの組み込み
  - スクリプト開発モデルで履歴・コメントモジュールを組み込む
- 操作権限設定の実装
  - 操作権限の設定を行うJavaファイルの作成
  - 操作権限の設定を行うクラスの登録

アプリケーションへの組み込み

スクリプト開発モデルで履歴・コメントモジュールを組み込む

埋め込みたいモジュールのHTMLファイルに、以下の記述を追加します。
ヘッダ部（<imart type="head"> ～ </imart>）に、以下を記述します。

- <meta http-equiv="X-Intramart-Secure-Token" content="<imart type="imSecureToken" mode="value" />"> CSRF（クロスサイト・リクエスト・フォージェリ）対策用のSecureTokenを出力するmetaタグ
- src="im_journal/bundles/js/journal.bundle.js" を指定したscriptタグ

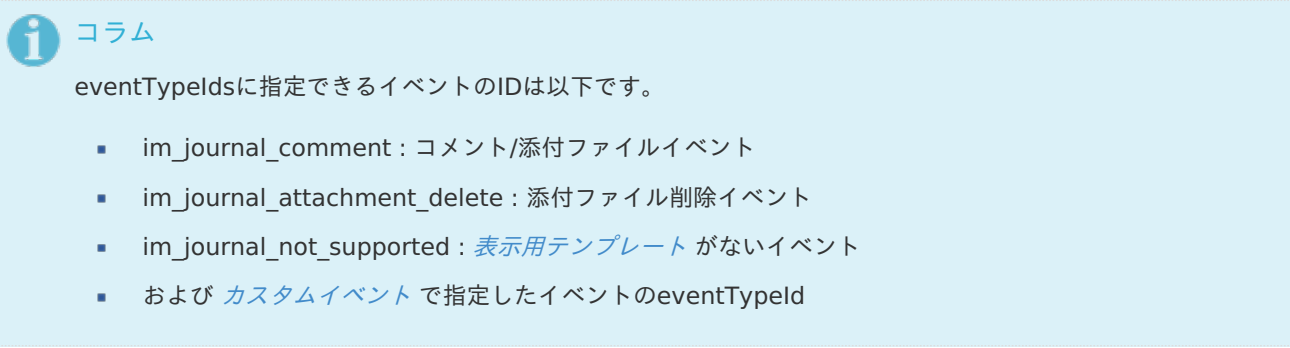
script内で、window.ImJournal.Journal を実行します。

```
<imart type="head">
  <meta http-equiv="X-Intramart-Secure-Token" content="<imart type="imSecureToken" mode="value" />">
  <script type="text/javascript" src="im_journal/bundles/js/journal.bundle.js"></script>
</imart>
<!-- 埋め込むアプリケーションの記述 -->

<div id="im-journal"></div>

<!-- 埋め込むアプリケーションの記述 -->
<script>
   const elementId = 'im-journal';

   const eventGroupKey = { journalSampleId: '1' };

   const application = 'im-journal-sample';

   const options = { sortOrder: 'asc',  eventTypeIds: [ 'im_journal_comment', 'im_journal_sample_log' ] };

   new window.ImJournal.Journal(elementId, eventGroupKey, application, options);
</script>
```

引数については次の通りです。

引数名	必須/任意	型	説明	省略時の動作
elementId	必須	string	埋め込む場所に置いたdivタグのid属性を指定します。	なし

引数名	必須/任意	型	説明	省略時の動作
eventGroupKey	必須	object	イベントグループを一意に設定するためのキーです。 オブジェクトのKeyは任意の値を指定します。	なし
application	必須	string	埋め込むアプリケーションを指定します。	なし
options	任意	object	履歴情報の昇順および降順の切り替えや、履歴に表示するイベントの絞り込みを行います。 利用可能なオプションについては以下のとおりです。 ・ **sortOrder** : string 履歴の昇順および降順を指定するオプションです。 昇順を asc、降順を desc で指定します。 ・ **eventTypeIds** : Array<string> 履歴に表示するものを絞り込むためのオプションです。 ・ **showEventCount** : boolean 履歴件数の表示非表示を制御するためのオプションです。 ・ **showAttachmentCount** : boolean 取得した添付ファイルの件数の表示非表示を制御するためのオプションです。 ・ **indicatorStyle** : string 履歴・コメントモジュール自体のインジケータの表示制御をするためのオプションです。 表示しない場合は hide、表示する場合は show で指定します。	**sortOrder** 降順で表示されます。 **eventTypeIds** 絞り込みは行われません。 **showEventCount** 履歴件数を表示します。 **showAttachmentCount** 添付ファイルの件数を表示します。 **indicatorStyle** インジケータを表示します。



コラム

eventTypeIdsに指定できるイベントのIDは以下です。

- im_journal_comment : コメント/添付ファイルイベント
- im_journal_attachment_delete : 添付ファイル削除イベント
- im_journal_not_supported : *表示用テンプレート* がないイベント
- および *カスタムイベント* で指定したイベントのeventTypeId

図: 履歴・コメントモジュール

操作権限設定の実装

履歴・コメントモジュールは、各アプリケーションの業務データに従って、操作権限の設定が必要です。

操作権限の設定はJavaで実装します。

操作権限の設定を行うJavaファイルの作成

操作権限の設定を行うJournalSamplePolicyクラスを作成します。

getApplication の戻り値は HTMLファイル で指定した application と同じ値にしてください。
getPermissions の引数 JournalEventGroupKey eventGroupKey は HTMLファイル で指定した eventGroupKey と同じ値です。
getPermissions の戻り値として、操作権限の配列を返却してください。
操作権限については、サンプルコードの次の表を参照してください。

サンプルコードでは、src/main/java/jp/co/intra_mart/journal/sample/policy ディレクトリ配下に JournalSamplePolicy.java を作成して実装します。

```
package jp.co.intra_mart.journal.sample.policy;

import java.util.HashSet;
import java.util.Set;

import jp.co.intra_mart.foundation.journal.event.JournalEventGroupKey;
import jp.co.intra_mart.foundation.journal.policy.JournalPermission;
import jp.co.intra_mart.foundation.journal.policy.JournalPolicyProvider;

public class JournalSamplePolicy implements JournalPolicyProvider{

   @Override
   public String getApplication() {
      return "im-journal-sample";
   }

   @Override
   public Set<JournalPermission> getPermissions(final JournalEventGroupKey eventGroupKey) {
      final Set<JournalPermission> permissions = new HashSet<>();
         permissions.add(JournalPermission.READ_EVENT);
         permissions.add(JournalPermission.REGISTER_COMMENT);
         permissions.add(JournalPermission.UPDATE_COMMENT);
         permissions.add(JournalPermission.DELETE_COMMENT);

      return permissions;
   }
}
```

操作権限は以下の通りです。

操作権限	説明
READ_EVENT	履歴情報の参照権限です。 履歴情報の参照が許可されます。 この権限のみの場合は読み取り専用となり、コメントの投稿や添付ファイルの追加ができません。
REGISTER_COMMENT	コメントの登録権限です。 この権限では、コメントの投稿が許可されます。

操作権限	説明
UPDATE_COMMENT	コメントの更新権限です。 この権限では、自分が投稿したコメントの編集が許可されます。 ただし、この権限には登録権限は含まれません。
DELETE_COMMENT	コメントの削除権限です。 この権限では、自分が投稿したコメントの削除が許可されます。 ただし、この権限には登録権限は含まれません。
READ_ATTACHMENT	添付ファイルの参照権限です。 この権限では、履歴・コメントモジュールに添付ファイルタブが表示され、添付ファイル一覧の参照が許可されます。
REGISTER_ATTACHMENT	添付ファイルの追加権限です。 この権限では、コメント投稿時にファイルの添付が許可されます。
DELETE_ATTACHMENT	添付ファイルの削除権限です。 この権限では、自分が添付したファイルの削除が許可されます。
DENY	参照を拒否します。 この権限では、履歴・コメントモジュールが表示されません。

操作権限の設定を行うクラスの登録

作成したJournalSamplePolicyクラスは、ServiceLoaderクラスを利用して読み込まれるサービスプロバイダとして扱います。
その為、ServiceLoaderとして読み込みを行う為のプロバイダ構成ファイルを配置します。
プロバイダ構成ファイルはクラスパス上の `/META-INF/services/jp.co.intra_mart.foundation.journal.policy.JournalPolicyProvider` ファイルです。
intra-mart e Builder for Accel Platform を利用する場合には、プロジェクト配下より**src/main/resources/META-INF/services** ディレクトリを作成した後、 **jp.co.intra_mart.foundation.journal.policy.JournalPolicyProvider** ファイルを作成してください。

プロバイダ構成ファイルには、作成したパッケージ指定クラスのFQDNを指定します。

```
jp.co.intra_mart.journal.sample.policy.JournalSamplePolicy
```

カスタムイベント

カスタムイベント情報の登録方法および独自の表示用テンプレートの作成方法について説明します。

カスタムイベントとは

カスタムイベントとは、任意のアプリケーションに対して独自に作成できる1つの履歴データです。
任意のアプリケーションに対して紐づけを行うことで、データの変更などを検知し、履歴・コメントモジュールに自動で反映します。

カスタムイベントの実装

実装は、履歴・コメントモジュールへのイベント登録と、カスタムイベント情報の設定に分けて行います。
カスタムイベントの処理はJavaで実装します。

履歴・コメントモジュールへのイベント登録

プロジェクトのsrc/main/java/ 配下に、任意のフォルダを作成します。その配下にJavaファイルを作成し、以下のように実装を行います。
サンプルコードでは、jp/co/intra_mart/journal/sample/endpoint ディレクトリ配下に MyService.java というファイルを作成します。

```
package jp.co.intra_mart.journal.sample.endpoint;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

import jp.co.intra_mart.foundation.journal.JournalServiceFactory;
import jp.co.intra_mart.foundation.journal.action.JournalActionRegisterModel;
import jp.co.intra_mart.foundation.journal.action.JournalActionType;
import jp.co.intra_mart.foundation.journal.event.JournalEventGroupKey;
import jp.co.intra_mart.foundation.journal.event.JournalEventRegisterModel;
import jp.co.intra_mart.foundation.journal.event.JournalEventService;
import jp.co.intra_mart.foundation.journal.exception.JournalException;

public class MyService {
   // カスタムイベント情報の設定
   private JournalEventRegisterModel registerJournal() throws JsonProcessingException {

      // 独自のイベントを作成する
      final JournalEventRegisterModel event = new JournalEventRegisterModel();

      // eventGroupKeyを設定する
      final JournalEventGroupKey eventGroupKey = new JournalEventGroupKey();
      eventGroupKey.put("journalSampleId", "1");
      event.setEventGroupKey(eventGroupKey);
      event.setApplication("im-journal-sample");

      // eventTypeId （設定ファイルのevent-type-idに設定した値）
      event.setEventTypeId("im_journal_sample_log");

      // アクションを設定する
      final Set<JournalActionRegisterModel> actions = new HashSet<JournalActionRegisterModel>();
```

```
        final JournalActionRegisterModel action = new JournalActionRegisterModel();
        action.setKey("my-action-key");
        action.setType(JournalActionType.REGISTER);

        final Map<String, String> actionData = new HashMap<String, String>();
        final ObjectMapper jsonMapper = new ObjectMapper();
        actionData.put("myData", "123456");

        final String jsonString = jsonMapper.writeValueAsString(actionData);

        action.setData(jsonString);
        actions.add(action);
        event.setActions(actions);

        return event;
    }

    // イベント登録
    public void registerMyEvent() throws JsonProcessingException, JournalException {

        // サービスクラスを呼び出す
        final JournalEventService service = JournalServiceFactory.getJournalEventService();

        // 関数の呼び出し
        final JournalEventRegisterModel event = registerJournal();

        // カスタムイベントを登録する
        service.registerEvent(event);
    }
}
```

> **コラム**
>
> イベント操作日時とイベント操作ユーザの初期値には、それぞれシステム時刻とログインしているユーザのユーザコードが設定されています。そのため、時刻およびユーザコードを変えたい場合は、operationDateおよびoperationUserCdを設定してください。

表示用テンプレート

表示用テンプレートとは

表示用テンプレートとは、追加されたカスタムイベントを履歴・コメントモジュール内で表示するための部品です。

テンプレートの実装方法

表示用テンプレートは スクリプト開発モデル に準じた実装ができます。
HTML と サーバサイドJavaScript を用いてテンプレートを作成し、設定ファイルでカスタムイベントと関連付けることで、履歴・コメントモジュールに表示できます。

表示用テンプレートの作成

表示用テンプレートは HTML で実装します。
プロジェクトの src/main/jssp/ 配下に任意のフォルダを作成し、その配下に HTML ファイルを作成し、以下のように実装します。
サンプルコードでは src/sample/journal ディレクトリ配下に journal_template_sample.html というファイルを作成します。

```
<div class="journal-template-sample">
  <!-- アイコンが表示される左のエリア -->
```

```
  <div class="journal-template-sample-user-header">
    <span class="journal-template-sample-user">
      <img class="journal-template-sample-user-img" src="<imart type="string" value=operationUserProfileImg
escapeJs="false" escapeXml="true"/>" />
    </span>
  </div>

  <!-- 本体 -->
  <div class="journal-template-sample-container" data-event-id="<imart type="string" value=eventId />">
    <!-- 本体ヘッダ部分 -->
    <div class="journal-template-sample-header">
      <div class="journal-template-sample-header-left">
        <span class="journal-template-sample-header-name" title="<imart type="string" value=operationUserCd
escapeJs="false" escapeXml="true"/>">
          <imart type="string" value=operationUserName escapeJs="false" escapeXml="true"/>
        </span>
      </div>
    </div>

    <!-- 本体ボディ部分 -->
    <div class="journal-template-sample-body">
      <div class="im-journal-message-content">
        <p id="comment-display-area-<imart type="string" value=eventId />" class="journal-template-sample-body-
text">
          <imart type="string" value=comment escapeJs="false" escapeXml="true"/>
        </p>
      </div>
    </div>

    <!-- 本体フッタ部分 -->
    <div class="journal-template-sample-footer">
      <span class="journal-template-sample-header-date">
        <imart type="string" value=operationDateLabel />
      </span>
    </div>
  </div>
</div>
<style type="text/css">
  .journal-template-sample {
    display: flex;
  }
  .journal-template-sample-user {
    display: block;
    border: 1px solid #ccc;
    background: #fff;
    border-radius: 50%;
    margin-right: 5px;
  }
  .journal-template-sample-user-img {
    width: calc(1.6em + 10px);
    height: calc(1.6em + 10px);
    border-radius: 50%;
    vertical-align: middle;
  }
  .journal-template-sample-container {
    border: 1px solid #ddd;
    border-radius: 4px;
    width: calc(100% - 1.6em - 20px);
  }
  .journal-template-sample-header {
    border-radius: 4px 4px 0 0;
    height: 2.2em;
    padding: 0 10px;
    display: flex;
    background: #f7f7f7;
```

```
    background: #f1f1f1;
    border-bottom: 1px solid #ddd;
  }
  .journal-template-sample-body {
    padding: 8px 20px;
  }
  .journal-template-sample-body-text {
    white-space: pre-wrap;
    line-height: 1.6;
    word-wrap: break-word;
    max-width: 100%;
  }
  .journal-template-sample-header > * {
    display: flex;
    align-items: center;
  }
  .journal-template-sample-footer {
    border-top: 1px solid #ddd;
    padding: 2px 10px;
    text-align: right;
    font-size: 0.95em;
  }
</style>
```

カスタムイベントの情報をテンプレートで使用するためには、サーバサイドJavaScript で取得する必要があります。
init関数 の引数を利用して登録した情報が取得できるので、それを HTML で使用できるようにします。
表示用テンプレートの HTML ファイルと同階層に サーバサイドJavaScript ファイルを作成し、以下のように実装します。

```
let eventId;
let operationUserCd;
let operationUserName;
let operationDateLabel;
let operationUserProfileImg;
let eventGroupId;
let eventGroupKey;
let application;
let comment;

function init(event) {
    eventId = event.eventId;
    operationUserCd = event.operationUserCd;
    operationUserProfileImg = 'api/immaster/user/image/' + encodeURIComponent(operationUserCd) + '/original?
cacheControl=private&noImage=true';
    operationUserName = event.operationUserName || event.operationUserCd;
    operationDateLabel = event.operationDateLabel;
    eventGroupId = event.eventGroupId;
    eventGroupKey = event.eventGroupKey;
    application = event.application;

    const commentAction = event.actions.filter(function (a) { return a.key === 'my-action-key' })[0];
    comment = JSON.parse(commentAction.data).myData;
}
```

実装イメージは以下の通りです。

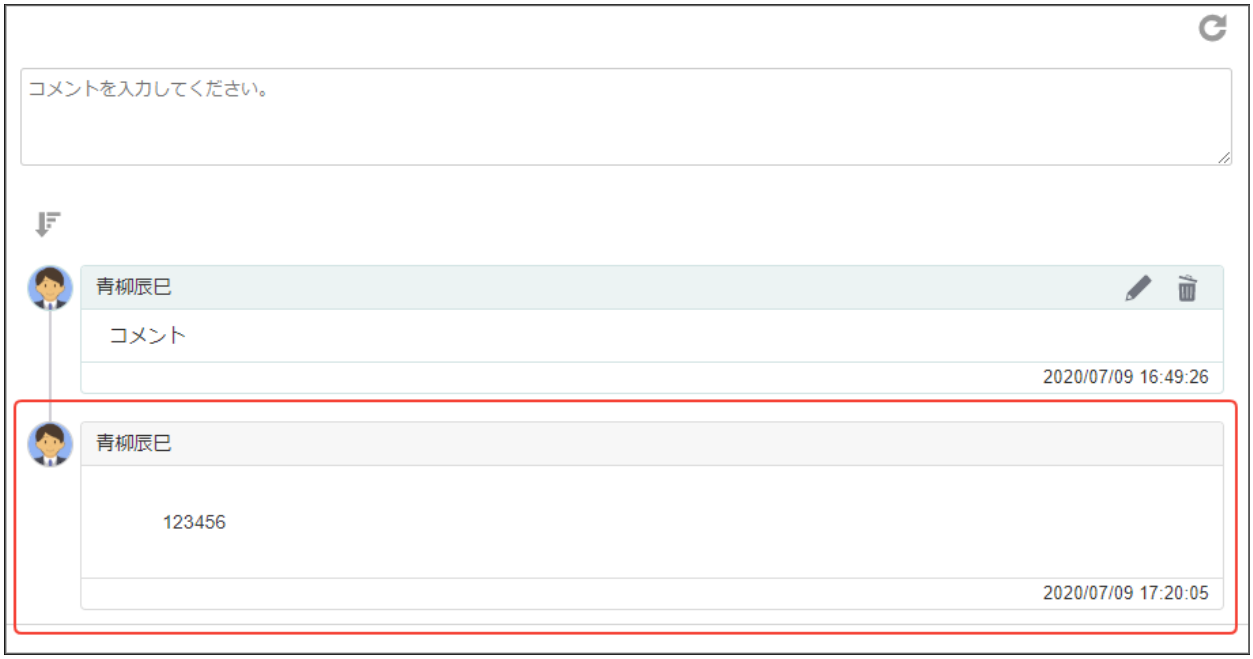


図: カスタムイベント

設定ファイルの作成

作成したテンプレートをシステムに読み込ませるための設定ファイルを追加します。
プロジェクト配下より src/main/conf/im-journal-template-config ディレクトリを作成し、xmlファイルを作成し、以下のように実装します。

```
<?xml version="1.0" encoding="UTF-8"?>
<im-journal-template-config xmlns="http://intra-mart.co.jp/system/journal/im-journal-template-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://intra-
mart.co.jp/system/journal/im-journal-template-config ../schema/im-journal-config.xsd ">
    <!-- カスタムイベントの eventTypeId とテンプレートのファイルパスを指定して紐づけます。 -->
    <template event-type-id="im_journal_sample_log" path="sample/journal/journal_template_sample" />
</im-journal-template-config>
```

イベントハンドラ

イベントハンドラとは

イベントハンドラとは、ユーザによる履歴・コメントモジュールへのコメントや、別のアプリケーションのイベントの登録を、組み込んだアプリケーションで検知する機能です。

イベントハンドラの実装方法

履歴・コメントモジュールでイベントの登録を検知した際に、任意の処理を行うことができます。
処理は Java で実装します。

イベントハンドラの処理を行うJavaファイルの作成

プロジェクトのsrc/main/java/ 配下に、任意のフォルダを作成します。その配下にJavaファイルを作成し、以下のように実装を行います。
サンプルコードでは、jp/co/intra_mart/journal/sample/event/handler ディレクトリ配下に JournalEventHandlerSample.java というファイルを作成します。

`getApplication` の返却値でどのイベントの登録を検知するのかを指定します。
`onRegisterJournalEvent` は検知したイベントの情報を受け取り、任意の処理を行うことができます。

```
package jp.co.intra_mart.journal.sample.event.handler;

import jp.co.intra_mart.common.platform.log.Logger;
import jp.co.intra_mart.foundation.journal.event.JournalEvent;
import jp.co.intra_mart.foundation.journal.handler.JournalEventHandler;

// JournalEventHandler インタフェースを実装したクラスを作成することで、イベントの登録を検知できる
public class JournalEventHandlerSample implements JournalEventHandler {

    private static final Logger LOGGER = Logger.getLogger(JournalEventHandlerSample.class);

    private static final String application = "*";

    // イベントの登録を検知するアプリケーションを指定
    // "*" を返却することで、すべてのアプリケーションに対して実行される
    @Override
    public String getApplication() {
        return application;
    }

    // イベントの登録を検知した際に実行される処理
    @Override
    public void onRegisterJournalEvent(final JournalEvent event) {
        LOGGER.info(event.toString());
    }
}
```

イベントハンドラの処理を行うクラスの登録

作成したJournalEventHandlerSampleクラスは、ServiceLoaderクラスを利用して読み込まれるサービスプロバイダとして扱います。
その為、ServiceLoaderとして読み込みを行う為のプロバイダ構成ファイルを配置します。
プロバイダ構成ファイルはクラスパス上の `/META-INF/services/jp.co.intra_mart.foundation.journal.handler.JournalEventHandler` ファイルです。

intra-mart e Builder for Accel Platform を利用する場合には、プロジェクト配下より**src/main/resources/META-INF/services** ディレクトリを作成した後、
jp.co.intra_mart.foundation.journal.handler.JournalEventHandler ファイルを作成してください。

プロバイダ構成ファイルには、作成したパッケージ指定クラスのFQDNを指定します。

```
jp.co.intra_mart.journal.sample.event.handler.JournalEventHandlerSample
```

REST API

REST APIについて説明します。

項目

- REST APIについて
- 認証方式
- 認可
- エンドポイント
- Swagger
- REST APIドキュメント
- イベントグループキーについて

REST APIについて

履歴・コメントモジュールが提供するREST APIは、HTTPプロトロルを使用し履歴・コメントモジュールに関する様々な処理を呼び出すことが可能です。
REST APIはコンテンツタイプとして、application/json (JSON)形式のみ対応しています。
REST APIとして利用可能な機能は以下のとおりです。

- 添付ファイル
- コメント
- イベント
- 認証情報

認証方式

REST APIは以下の認証方式に対応しています。

- Cookieに紐づくセッションの認証状態に依存する方式

  アプリケーションサーバが発行するセッションIDおよびアカウントコンテキストの状態に依存する方式です。
  コンテキストの状態を確認後、認可によるチェックが行われます。

- Basic認証

  Basic認証による認証方式です。
  認証後、ログイン状態として扱われ認可によるチェックが行われます。

- OAuth認証

  OAuth2.0の仕様に準拠した認証フローによる認証方式です。
  認証後、ログイン状態として扱われ認可によるチェックが行われます。

コラム
Basic認証を利用する場合には、httpsプロトコルの利用を強く推奨します。

認可

REST APIは全ての呼び出しに対し認可リソースを持ちます。
intra-mart Accel Platform認可設定において、「画面・処理」リソースの「履歴・コメントモジュール」で変更できます。

> **注意**
>
> 履歴・コメントモジュールの標準機能では、画面表示に必要となる情報を全てREST API経由で取得しています。
> その為、認可設定の変更により画面が正常に動作しなくなる場合があります。

エンドポイント

REST APIは認証方式によって、呼び出し先のエンドポイントが異なります。

- Cookieに紐づくセッションの認証状態に依存する方式

  http(s)://{HOST}:{PORT}/{CONTEXT_PATH}/api/journal/....
  プロトコル、ホスト名、ポート番号、コンテキストパスは環境にあわせて置き換えてください。
  例:
  https://example.org/imart/api/journal/events

- Basic認証

  http(s)://{HOST}:{PORT}/{CONTEXT_PATH}/api/basic/journal/....
  プロトコル、ホスト名、ポート番号、コンテキストパスは環境にあわせて置き換えてください。
  例:
  https://example.org/imart/api/basic/journal/events

- OAuth認証

  http(s)://{HOST}:{PORT}/{CONTEXT_PATH}/api/bearer/journal/....
  プロトコル、ホスト名、ポート番号、コンテキストパスは環境にあわせて置き換えてください。
  例:
  https://example.org/imart/api/bearer/journal/events

Swagger

REST APIはSwagger Specに対応しています。
intra-mart Accel Platformに組み込まれているSwagger UIを通じてREST APIの確認、実行ができます。

Swagger UIは、以下のURLから確認できます。

http(s)://{HOST}:{PORT}/{CONTEXT_PATH}/swagger_ui/?url=/{CONTEXT_PATH}/api-docs/IM-journal
プロトコル、ホスト名、ポート番号、コンテキストパスは環境にあわせて置き換えてください。
例:
https://example.org/imart/swagger_ui/?url=/imart/api-docs/IM-journal

> **コラム**
>
> Swagger Specを元に、Swagger CodeGenを利用するとREST APIクライアントコードの自動生成を行うことが可能です。
> https://github.com/swagger-api/swagger-codegen
>
> Swagger CodeGenにより作成されたスタブコードの動作保証は行っておりません。

REST APIドキュメント

APIの詳細に関しては、「履歴・コメントモジュール Rest」を御覧ください。

イベントグループキーについて

イベントグループを一意に設定するためのキーとしてイベントグループキーをREST APIに送信する必要があります。
イベントグループキーは、リクエストパラメータに設定してください。

イベントグループキーは、リクエストパラメータの以下を除くキーとその値から生成されます。

- count
- index
- sortOrder
- eventTypeIds
- _（アンダスコア）
- application
- access_token

コラム

例:
https://example.org/imart/api/journal/event_count?application=APPLICATION1&eventGroupKey1=value1&eventGroupKey2=value2
`application` は除かれ、 その他の `eventGroupKey1` と `eventGroupKey2` でイベントグループキーが生成されます。
イベントグループキー: { eventGroupKey1: ‘value1’, eventGroupKey2: ‘value2’ }

付録

OAuth認証を利用した外部アプリケーション連携

OAuth認証機能を利用し、外部アプリケーションから履歴・コメントモジュールのREST APIを呼び出す方法を説明します。

> **コラム**
>
> OAuth認証の概要については「OAuth認証モジュール 仕様書」-「概要」を参照してください。

項目

- intra-mart Accel Platform上にてクライアントアプリケーションの追加を行う
- 外部アプリケーションより履歴・コメントモジュールのREST APIのエンドポイントを呼び出す

intra-mart Accel Platform上にてクライアントアプリケーションの追加を行う

「OAuth 管理者操作ガイド」-「クライアントアプリケーションの登録」を参照し、クライアントアプリケーションの登録を行います。この際、「アクセス範囲」にスコープ「journal」を指定します。
または、設定ファイルを使用してクライアントアプリケーションの登録を行います。
設定ファイルを使用したクライアントアプリケーションの設定の詳細は「設定ファイルリファレンス」-「OAuth認証機能」-「クライアント詳細設定」を参照してください。この場合も同様に「アクセス範囲」にスコープ「journal」を指定してください。

外部アプリケーションより履歴・コメントモジュールのREST APIのエンドポイントを呼び出す

「OAuth プログラミングガイド」-「クライアントアプリケーションからOAuth認証機能を利用する方法」を参照し、クライアントアプリケーションから履歴・コメントモジュールのREST APIのエンドポイントを呼び出します。

> **コラム**
>
> ユーザ（リソースオーナー）によりクライアントアプリケーションからのスコープ「journal」へのアクセスが許可された場合のみ、クライアントアプリケーションより履歴・コメントモジュールのREST APIのエンドポイントを呼び出すことができます。

> **コラム**
>
> 履歴・コメントモジュールのREST APIのエンドポイントの詳細は「APIドキュメント」-「履歴・コメントモジュール Rest」を参照してください。
> 履歴・コメントモジュールのREST APIでは、認証方式により呼び出し先のエンドポイントが異なります。
> OAuth認証を使用する場合、パスプレフィックスとして”/api/bearer/journal/...”を使用します。