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. 本書の構成
- 3. 開発環境のセットアップ
  - 3.1. プロジェクトの作成と設定
  - 3.2. モジュールの依存関係について
  - 3.3. サンプルプロジェクトのインポートと設定
  - 3.4. サンプルについて
  - 3.5. テンプレートについて
- 4. ユーザアプリケーション開発
  - 4.1. ワークフローエンジンとワークフローアプリケーション
  - 4.2. IM-Workflow とユーザアプリケーション
  - 4.3. ワークフローデータとユーザアプリケーションデータ
- 5. ユーザコンテンツ画面の開発
  - 5.1. 申請時の画面処理の流れ
  - 5.2. 承認時の画面処理の流れ
  - 5.3. 全体の作業の流れ
  - 5.4. Formクラスの作成
  - 5.5. Controllerクラスの作成
  - 5.6. 画面の作成
  - 5.7. 作成した画面の設定方法
- 6. ユーザプログラムの開発
  - 6.1. DIコンテナの呼び出し
  - 6.2. アクション処理
  - 6.3. 到達処理
  - 6.4. 案件終了処理・案件終了処理（トランザクションなし）
- 7. 付録
  - 7.1. 案件処理系APIの実行について
  - 7.2. サンプルプログラム

改訂情報

変更年月日	変更内容
2015-10-23	初版
2016-12-01	第2版 下記を変更しました。 - 「 *開発環境のセットアップ* 」 のサンプルプロジェクトを更新しました。 - 「 *ユーザプログラムの開発* 」 のトランザクション制御について説明を追加しました。 - 「 *サンプルプログラム* 」 からDB2に関する記述を削除しました。 - 「 *サンプルプログラム* 」 のマッピングファイルを修正しました。
2017-04-01	第3版 下記を変更しました。 - 「 *開発環境のセットアップ* 」 のサンプルプロジェクトを修正しました。 - 「プロジェクトのインポート」のインポートしたモジュール・プロジェクトでエラーになる問題を修正しました。
2017-12-01	第4版 下記を変更しました。 - 「 *アクション処理* 」 に自動処理・一括処理時のUserParameterに関するコラムを追加しました。
2019-12-01	第5版 下記を変更しました。 - 「 *作成した画面の設定方法* 」 の画面定義のパス種別「JSP or Servlet」を「URL」に変更しました。

はじめに

本書の目的

本書では、TERASOLUNA Server Framework for Java (5.x) を利用して IM-Workflow のユーザアプリケーションを開発するための手順・注意点を説明します。
TERASOLUNA Server Framework for Java (5.x) に特化した内容を扱うため、 IM-Workflow のプログラミングの全ての内容については本書のみで扱うことができません。
不足点については適宜以下のドキュメントを参照してください。

- 「IM-Workflow 仕様書」
- 「IM-Workflow プログラミングガイド」

また、 TERASOLUNA Server Framework for Java (5.x) については以下のドキュメントを参照してください。

- 「TERASOLUNA Server Framework for Java (5.x) プログラミングガイド」

本書で使用するサンプルプログラムはあくまでも、 IM-Workflow の機能および API 等の使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もあえて採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。

対象読者

本書では次の開発者を対象としています。

- IM-Workflow の各機能を利用したい開発者
- TERASOLUNA Server Framework for Java (5.x) for Accel Platform の開発者

なお、本書では次の内容を理解していることが必須です。

- IM-Workflow の基本機能
- TERASOLUNA Server Framework for Java (5.x) for Accel Platform 基礎

本書の構成

- *開発環境のセットアップ*

  開発環境の構築方法とサンプルプロジェクトについて説明します。

- *ユーザアプリケーション開発*

  IM-Workflow におけるユーザアプリケーション開発の位置づけについて説明します。

- *ユーザコンテンツ画面の開発*

  ユーザコンテンツ画面の開発方法について説明します。

- *ユーザプログラムの開発*

  ユーザプログラムの開発方法について説明します。

- *付録*

  案件処理系APIの実行について説明します。
  サンプルプログラムについて説明します。

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。

- 「 IM-Workflow 」
- 「 TERASOLUNA Server Framework for Java (5.x) for Accel Platform 」

スマートフォン向けの申請画面・承認画面を開発する場合は、以下のモジュールへの依存関係も必要です。

- 「 IM-Workflow スマートフォン 」

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、サンプルフロー「TERASOLUNA Server Framework開発モデル」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プロジェクト時に、追加リソース（設定ファイル等）として「 TERASOLUNA Server Framework for Java （5.x） 設定ファイル（iAP 2015 Spring 8.0.10以降） 」・「 TERASOLUNA Server Framework for Java （5.x） for MyBatis3設定ファイル（iAP 2015 Spring 8.0.10以降） 」を選択します。
  3. Jugglingプロジェクトのモジュール構成には、「 IM-Workflow 」「 IM-Workflow スマートフォン 」「 TERASOLUNA Server Framework for Java (5.x) for Accel Platform 」を含めます。
  4. Jugglingプロジェクトのユーザモジュールタブから「 **im_workflow_tsfw_sample-8.0.1.imm** 」を取り込みます。
  5. TypeAlias 設定ファイル「 **mybatis-config.xml** 」をダウンロードし、%Jugglingプロジェクト%/classes/META-INF/mybatis/mybatis-config.xmlに配置します。
  6. 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。
  7. 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。
  8. 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。
  9. 申請一覧にアクセスすることで「TERASOLUNA Server Framework開発モデル」が表示されます。

- 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_workflow_tsfw_sample-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_workflow_tsfw」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_workflow_tsfw」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

サンプルフローの仕様を簡単に紹介します。

- フロー「TERASOLUNA Server Framework開発モデル」
  - 申請画面にて、ユーザアプリケーションデータを入力します。
  - 入力されたユーザアプリケーションデータは、アクション処理にてテナントデータベース上のテーブルへ登録します。
  - 未処理一覧から受け取ったユーザデータIDをキーにテーブルを検索します。
  - 取得したユーザアプリケーションデータを承認画面に表示します。
  - 案件終了処理にて、テーブルの案件終了フラグを更新します。
  - 案件退避処理リスナーにて、テーブルのアーカイブフラグを更新します。

サンプルユーザアプリケーションについては、 *サンプルプログラム* を参照してください。

テンプレートについて

ユーザプログラムを作成する際のテンプレートが提供されています。

- サンプルプロジェクトの以下のディレクトリに配置されています。
  - <%サンプルプログラムディレクトリ%/jp/co/intra_mart/sample/im_workflow/tsfw/template>

テンプレート一覧

処理	物理名
案件開始処理	MatterStartProcess.java
案件終了処理	MatterEndProcess.java
アクション処理	ActionProcess.java
到達処理	ArriveProcess.java
分岐開始処理／分岐終了処理	RuleCondition.java
未完了案件削除処理リスナー	WorkflowActvMatterDeleteListener.java
完了案件削除処理リスナー	WorkflowCplMatterDeleteListener.java
過去案件削除処理リスナー	WorkflowArcMatterDeleteListener.java
案件退避処理リスナー	WorkflowMatterArchiveListener.java
処理対象者プラグイン	WorkflowAuthorityExecEventListener.java
案件終了処理（トランザクションなし）	MatterEndProcessNoTran.java

ユーザアプリケーション開発

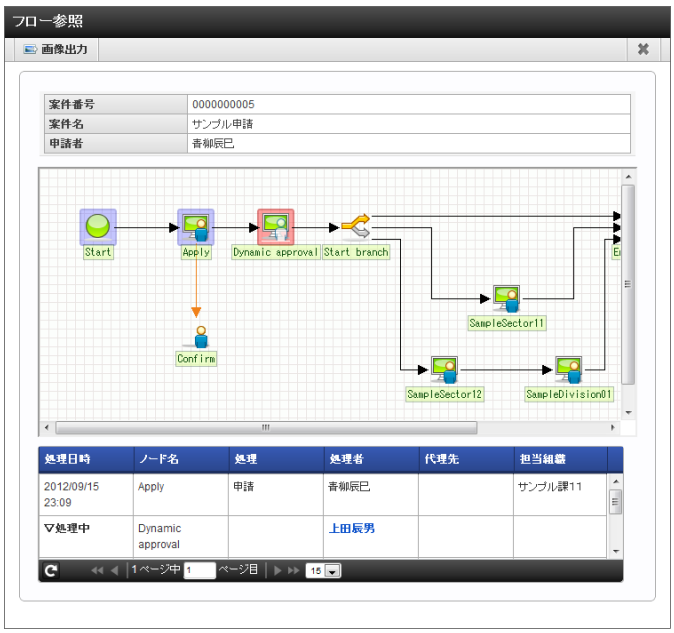
本章では、 IM-Workflow におけるユーザアプリケーション開発の位置づけについて説明します。

- IM-Workflow ではワークフローエンジンの機能を提供しており、連携する業務アプリケーション（ユーザアプリケーション）を開発することでワークフローシステムを構築できます。
- ユーザアプリケーションは、ユーザコンテンツ画面・ユーザプログラム・ユーザアプリケーションデータの3つで構成されます。
- ユーザコンテンツ画面で入力された業務データはユーザアプリケーションデータとして、ユーザアプリケーション内で管理します。

ワークフローエンジンとワークフローアプリケーション

ワークフローシステムとは、あらかじめ決められた承認経路に基づき、複数の担当者が電子化された申請書を回覧していくシステムです。 ワークフローエンジンとワークフローアプリケーションの2つで構成されます。

- 「ワークフローエンジン」
  - 設定された承認経路に従い、タスクを各担当者へ展開します。
  - 案件のステータス・タスクの処理履歴を管理します。



- 「ワークフローアプリケーション」
  - 担当者がタスクを実行した際に申請書（入力画面）を表示します。
  - 申請書（入力画面）に入力された業務データを管理します。

IM-Workflow とユーザアプリケーション

IM-Workflow はワークフローエンジンとしての機能を提供しており、業務要件に合わせてワークフローアプリケーションを開発することでワークフローシステムを構築することが可能です。
本書では、 IM-Workflow が提供するワークフローエンジンに対してユーザが開発するワークフローアプリケーションのことを「ユーザアプリケーション」と表記しています。

ユーザアプリケーションは、以下の3つで構成されます。

- ユーザコンテンツ画面
  - ワークフローの各ノードにて IM-Workflow から呼び出される申請書（入力画面）を表示するためのプログラムです。
  - TERASOLUNA Server Framework for Java (5.x) が提供するSpring MVCを利用して開発する手順を説明します。
- ユーザプログラム
  - 案件処理の様々なタイミングにて IM-Workflow から実行される業務ロジックのプログラムです。
  - ユーザコンテンツ画面にて入力された業務データのデータベース・ストレージへの保存などを行います。
  - TERASOLUNA Server Framework for Java (5.x) が提供するSpringのDIコンテナを利用して開発する手順を説明します。
- ユーザアプリケーションデータ
  - ユーザコンテンツ画面にて入力され、ユーザプログラムを通して加工・管理される業務データです。
  - データベース・ストレージ等を利用し、ユーザアプリケーション内で管理します。

ワークフローデータとユーザアプリケーションデータ

ワークフローシステムを通じて生成されるトランザクションデータは、以下の2つに分けられます。

- ワークフローデータ
  - 案件のステータスなどワークフローの処理に関する情報・ユーザコンテンツ画面やユーザプログラムのパスなど案件にひもづくユーザアプリケーションの情報です。
  - 「システム案件 ID」によって、 IM-Workflow 内で管理されます。
- ユーザアプリケーションデータ
  - ユーザコンテンツ画面にて入力され、ユーザプログラムを通して管理される業務データです。
  - 「ユーザデータ ID」によって、ユーザアプリケーション内で管理します。

システム案件 ID

システム案件 ID とは、 IM-Workflow 内で案件（ワークフローデータ）を一意に識別するキーです。

- 案件作成時に（申請または起票を行った際に）、 IM-Workflow 内で採番されます。外部から指定することはできません。

ユーザデータ ID

ユーザデータID とは、案件にひもづく業務データ（ユーザアプリケーションデータ）を一意に識別するキーとして、ユーザアプリケーション内で採番するキーです。

- 採番のタイミングは任意です（案件の開始前に採番してもかまいません）。一時保存または申請または起票を行う際に IM-Workflow 側に渡します。
- IM-Workflow に保存されたユーザデータIDは、ユーザアプリケーション（ユーザコンテンツ画面・ユーザプログラム）の呼び出し時にパラメータとして受け取ることができます。

申請時の画面処理の流れ

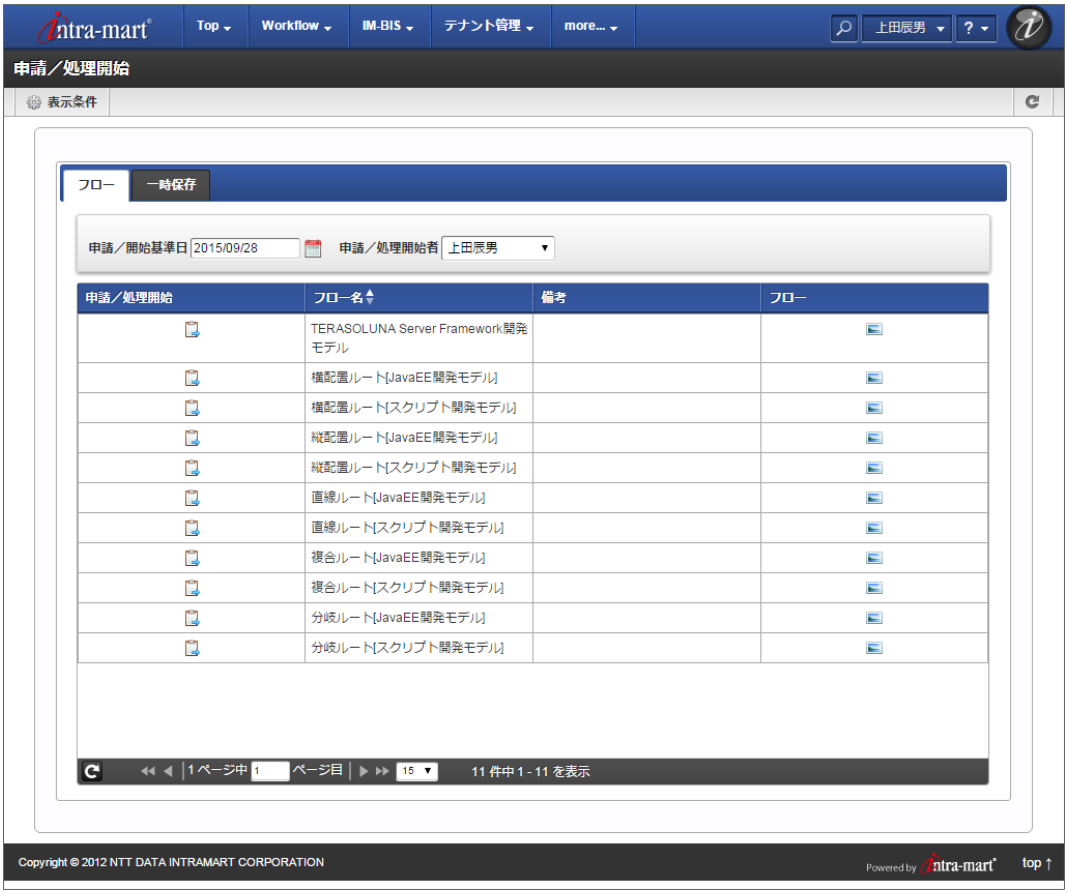
申請時の処理の流れを説明します。

項目

- 申請一覧画面
- ユーザコンテンツ画面
- 標準申請画面
- 申請完了後の遷移画面

申請一覧画面

- 申請可能なフローの一覧が表示されます。
- フローを選択することで、ユーザコンテンツ画面へ遷移します。
- 申請基準日やフローIDなど申請に必要なワークフローデータが送信されます。



ユーザコンテンツ画面

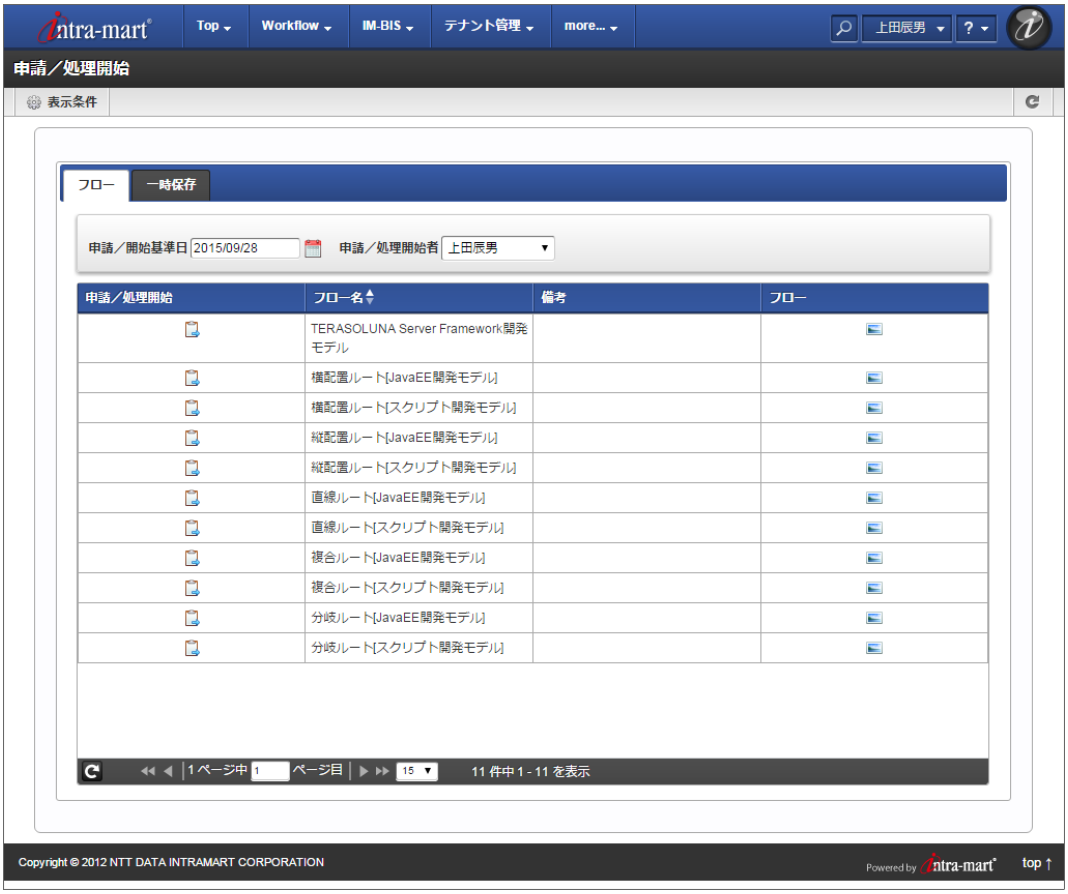
- フローに設定されたユーザコンテンツ画面が呼び出されます。
- 入力完了時に、入力値に対してクライアントサイド・バリデーションを行います。
- 入力完了時にCSJS API workflowOpenPageを実行して、標準申請画面を呼び出します。

標準申請画面

- 申請ボタンの押下時にAjaxによる申請処理が実行され、フローに設定されたアクション処理 ユーザプログラムが呼ばれます。
- アクション処理では、入力値のサーバサイド・バリデーションとデータベースへの登録などを行います。
- 申請処理の完了後、標準申請画面の呼び出し時に指定した画面に自動で遷移します。

申請完了後の遷移画面

- API workflowOpenPageの実行時に指定した画面に自動で遷移します。
- 指定がない場合は、特定の画面へ遷移することなく、ただ標準申請画面が閉じられます。



承認時の画面処理の流れ

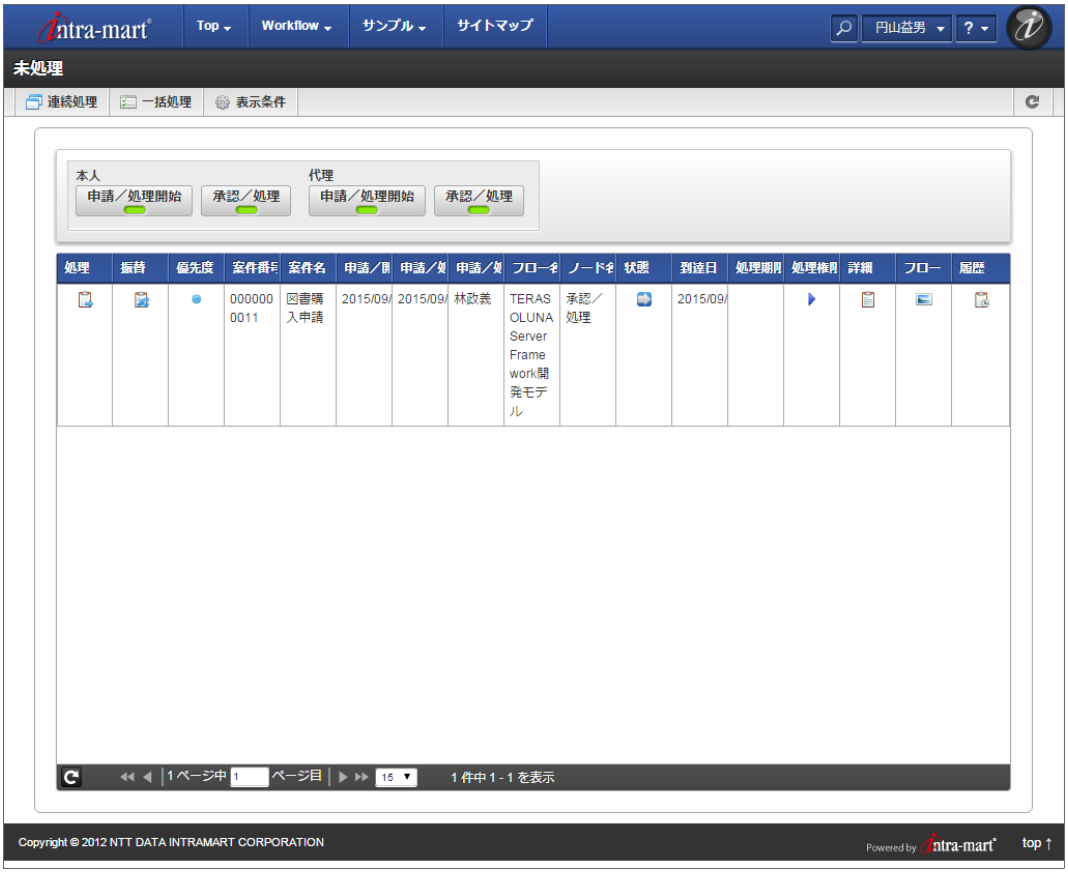
承認時の処理の流れを説明します。

項目

- 未処理一覧画面
- ユーザコンテンツ画面
- 標準処理画面
- 承認完了後の遷移画面

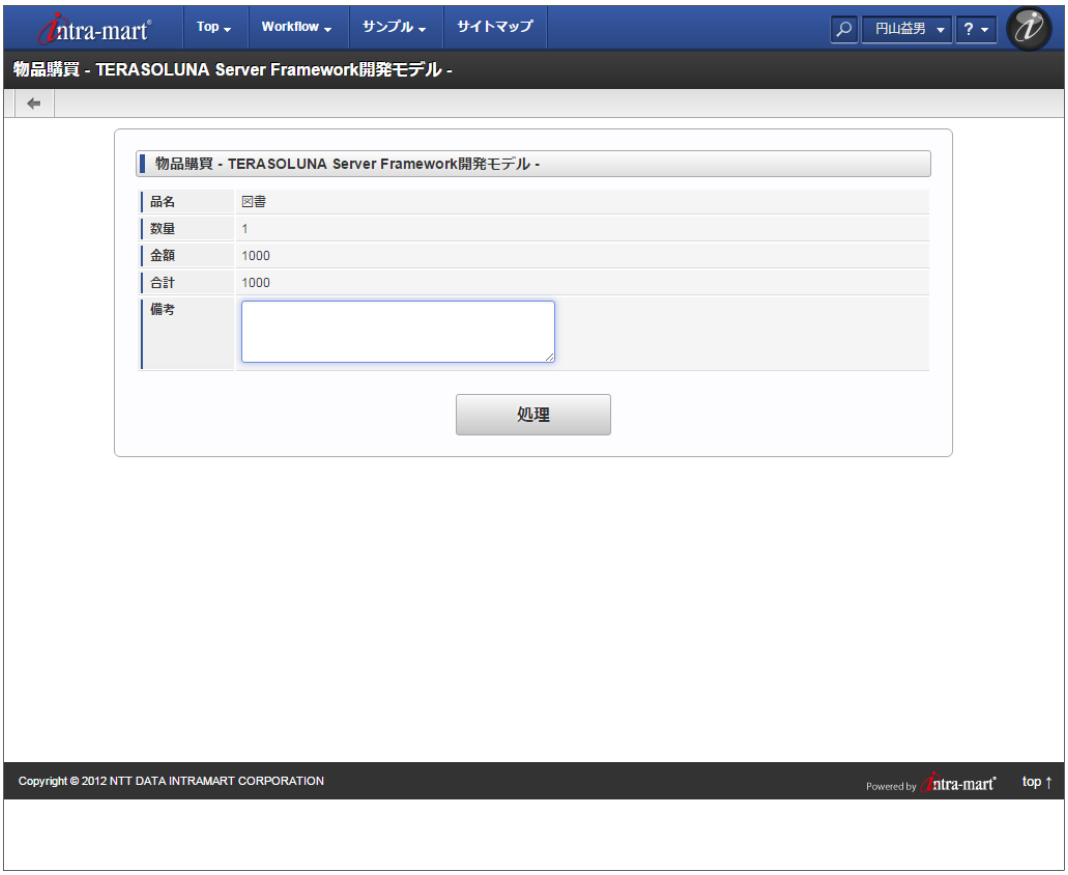
未処理一覧画面

- 処理可能な案件の一覧が表示されます。
- 案件を選択することで、ユーザコンテンツ画面へ遷移します。
- システム案件IDやユーザデータIDなど案件を特定するために必要なワークフローデータが送信されます。



ユーザコンテンツ画面

- 案件に設定されたユーザコンテンツ画面が呼び出されます。
- Controllerクラスにてユーザデータ IDをキーにデータベースから申請時の入力したユーザアプリケーションデータを取得し、画面へバインドします。
- 入力完了時に、更新項目の入力値に対してクライアントサイド・バリデーションを行います。
- 入力完了時に、CSJS API workflowOpenPageを実行して、標準処理画面を呼び出します。



標準処理画面

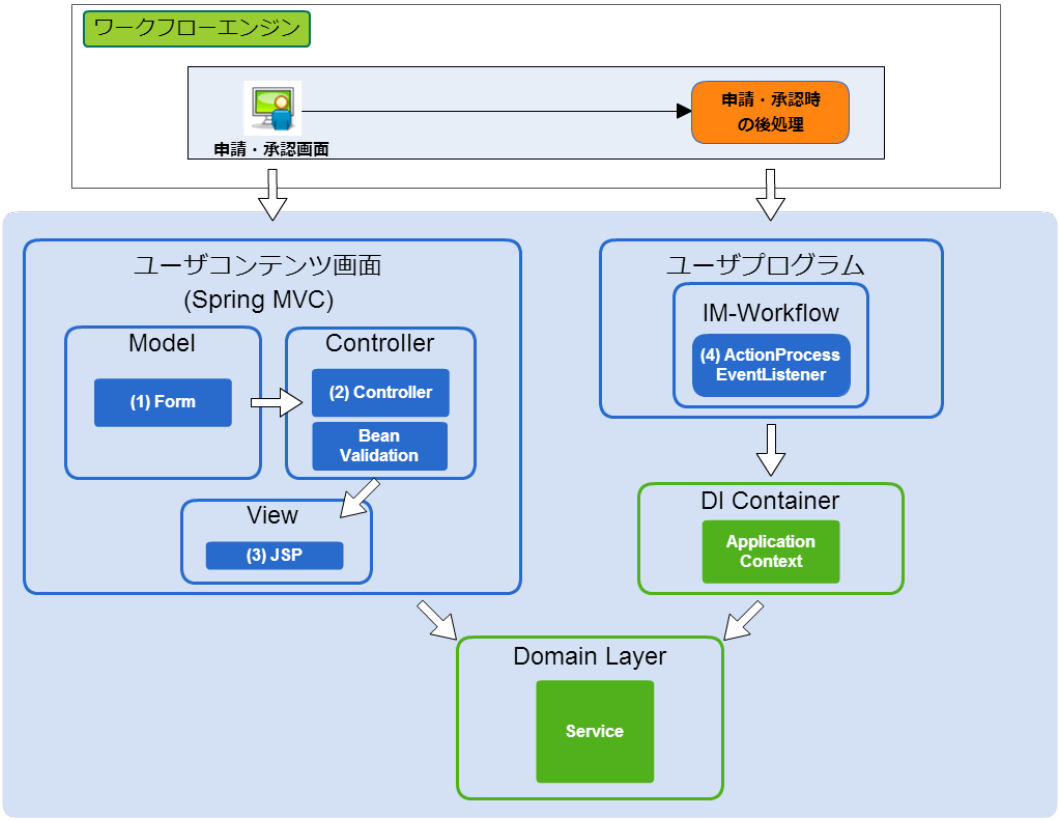
- 承認ボタンの押下時のAjaxによる承認処理が実行され、案件に設定されたアクション処理 ユーザプログラムが呼ばれます。
- アクション処理では、入力値のサーバサイド・バリデーションとデータベースへの更新などを行います。
- 承認処理の完了後、標準処理画面の呼び出し時に指定した画面に自動で遷移します。

承認完了後の遷移画面

- API workflowOpenPageの実行時に指定した画面に自動で遷移します。
- 指定がない場合は、特定の画面へ遷移することなく、ただ標準処理画面が閉じられます。

全体の作業の流れ

ユーザコンテンツ画面は、以下の流れで開発していきます。



Formクラス

- ワークフローデータとユーザアプリケーションデータを格納するモデルを作成します。

Controllerクラス

- Formクラスの内容をバインドした上で、画面（JSP）を呼び出す処理を作成します。
- 申請時以外はユーザデータIDをキーにデータベースからユーザアプリケーションデータを取得し、Formクラスに格納します。

画面（JSP）

- Formクラスから受け取った情報に基づいてユーザコンテンツ画面を表示させるためのJSPを作成します。
- 入力後に実行されるバリデーション処理と標準申請画面（標準処理画面）の呼び出し処理を記述する必要があります。

アクション処理

- 標準申請画面（標準処理画面）から送信されたユーザパラメータに対してバリデーション・データベースへの格納処理などを作成します。

> **コラム**
>
> アクション処理 ユーザプログラム の詳細については *アクション処理* を参照してください。

作成した画面の設定

- 作成した画面のURLをコンテンツ定義に登録します。

Formクラスの作成

ユーザコンテンツ画面にパラメータをバインドするためのモデルを作成します。

ユーザアプリケーションデータ

ユーザコンテンツ画面として作成した業務画面で入力される各項目を格納できるようにします。

ワークフローデータ

各種一覧画面からユーザコンテンツ画面が呼び出される際に送信されるパラメータです。

案件の処理を行うために必要なメタデータが格納されています。

各パラメータの詳細については、「IM-Workflow プログラミングガイド」-「リクエストパラメータ」を参照してください。

Formクラスの実装例（ユーザアプリケーションデータ）

```
public class PurchaseForm extends WorkflowForm {

    /* 品名 */
    private String item_name;

    /* 数量 */
    private String item_amount;

    /* 金額 */
    private String item_price;

    /* 合計 */
    private String item_total;

    /* 備考 */
    private String item_comment;

    public String getItem_name() {
        return item_name;
    }

    public void setItem_name(final String item_name) {
        this.item_name = item_name;
    }

    public String getItem_amount() {
        return item_amount;
    }

    public void setItem_amount(final String item_amount) {
        this.item_amount = item_amount;
    }

    public String getItem_price() {
        return item_price;
    }

    public void setItem_price(final String item_price) {
        this.item_price = item_price;
    }

    public String getItem_total() {
        return item_total;
    }

    public void setItem_total(final String item_total) {
        this.item_total = item_total;
    }

    public String getItem_comment() {
        return item_comment;
    }

    public void setItem_comment(final String item_comment) {
        this.item_comment = item_comment;
    }
}
```

Formクラスの実装例（ワークフローデータ）

```
public class WorkflowForm {

    /* ログイングループID */
    private String imwGroupId;

    /* ログインユーザコード */
    private String imwUserCode;

    /* 画面種別 */
    private String imwPageType;

    /* ユーザデータID */
    private String imwUserDataId;

    /* システム案件ID */
    private String imwSystemMatterId;

    /* ノードID */
    private String imwNodeId;

    /* 到達種別 */
    private String imwArriveType;

    /* 代理元ユーザコード */
    private String imwAuthUserCode;

    /* 申請基準日 */
    private String imwApplyBaseDate;

    /* コンテンツID */
    private String imwContentsId;

    /* コンテンツバージョンID */
    private String imwContentsVersionId;

    /* ルートID */
    private String imwRouteId;

    /* ルートバージョンID */
    private String imwRouteVersionId;

    /* フローID */
    private String imwFlowId;

    /* フローバージョンID */
    private String imwFlowVersionId;

    /* 呼出元パラメータ */
    private String imwCallOriginalParams;

    /* 呼出元ページパス */
    private String imwCallOriginalPagePath;

    /* システム日で対象者を展開するフラグ */
    private String imwSysDateTargetExpandFlag;

    /* ショートカットフラグ */
    private String imwShortCutFlag;

    public String getImwGroupId() {
        return imwGroupId;
    }

    public void setImwGroupId(final String imwGroupId) {
        this.imwGroupId = imwGroupId;
    }

    public String getImwUserCode() {
        return imwUserCode;
    }

    public void setImwUserCode(final String imwUserCode) {
```

```
    this.imwUserCode = imwUserCode;
}

public String getImwPageType() {
    return imwPageType;
}

public void setImwPageType(final String imwPageType) {
    this.imwPageType = imwPageType;
}

public String getImwUserDataId() {
    return imwUserDataId;
}

public void setImwUserDataId(final String imwUserDataId) {
    this.imwUserDataId = imwUserDataId;
}

public String getImwSystemMatterId() {
    return imwSystemMatterId;
}

public void setImwSystemMatterId(final String imwSystemMatterId) {
    this.imwSystemMatterId = imwSystemMatterId;
}

public String getImwNodeId() {
    return imwNodeId;
}

public void setImwNodeId(final String imwNodeId) {
    this.imwNodeId = imwNodeId;
}

public String getImwArriveType() {
    return imwArriveType;
}

public void setImwArriveType(final String imwArriveType) {
    this.imwArriveType = imwArriveType;
}

public String getImwAuthUserCode() {
    return imwAuthUserCode;
}

public void setImwAuthUserCode(final String imwAuthUserCode) {
    this.imwAuthUserCode = imwAuthUserCode;
}

public String getImwApplyBaseDate() {
    return imwApplyBaseDate;
}

public void setImwApplyBaseDate(final String imwApplyBaseDate) {
    this.imwApplyBaseDate = imwApplyBaseDate;
}

public String getImwContentsId() {
    return imwContentsId;
}

public void setImwContentsId(final String imwContentsId) {
    this.imwContentsId = imwContentsId;
}

public String getImwContentsVersionId() {
    return imwContentsVersionId;
}

public void setImwContentsVersionId(final String imwContentsVersionId) {
    this.imwContentsVersionId = imwContentsVersionId;
}
```

```
    public String getImwRouteId() {
        return imwRouteId;
    }

    public void setImwRouteId(final String imwRouteId) {
        this.imwRouteId = imwRouteId;
    }

    public String getImwRouteVersionId() {
        return imwRouteVersionId;
    }

    public void setImwRouteVersionId(final String imwRouteVersionId) {
        this.imwRouteVersionId = imwRouteVersionId;
    }

    public String getImwFlowId() {
        return imwFlowId;
    }

    public void setImwFlowId(final String imwFlowId) {
        this.imwFlowId = imwFlowId;
    }

    public String getImwFlowVersionId() {
        return imwFlowVersionId;
    }

    public void setImwFlowVersionId(final String imwFlowVersionId) {
        this.imwFlowVersionId = imwFlowVersionId;
    }

    public String getImwCallOriginalParams() {
        return imwCallOriginalParams;
    }

    public void setImwCallOriginalParams(final String imwCallOriginalParams) {
        this.imwCallOriginalParams = imwCallOriginalParams;
    }

    public String getImwCallOriginalPagePath() {
        return imwCallOriginalPagePath;
    }

    public void setImwCallOriginalPagePath(final String imwCallOriginalPagePath) {
        this.imwCallOriginalPagePath = imwCallOriginalPagePath;
    }

    public String getImwSysDateTargetExpandFlag() {
        return imwSysDateTargetExpandFlag;
    }

    public void setImwSysDateTargetExpandFlag(String imwSysDateTargetExpandFlag) {
        this.imwSysDateTargetExpandFlag = imwSysDateTargetExpandFlag;
    }

    public String getImwShortCutFlag() {
        return imwShortCutFlag;
    }

    public void setImwShortCutFlag(String imwShortCutFlag) {
        this.imwShortCutFlag = imwShortCutFlag;
    }
}
```

Controllerクラスの作成

Formクラスの内容をバインドした上で、画面（JSP）を呼び出す処理を作成します。

項目

- URL設計
- 詳細画面のポップアップ表示
- Controllerクラスの実装例（クライアントタイプ「PC」）
- Controllerクラスの実装例（クライアントタイプ「スマートフォン」）

URL設計

コンテンツ定義にて画面種別ごとに異なるURLを設定できるため、画面種別（ユーザコンテンツ画面の呼び出しパターン）によってURLを分離することが可能です。また、ワークフローのシステムパラメータ imwPageTypeとして画面種別を受け取ることが可能なので、同一のURLが設定されている場合でもControllerクラスにて画面種別を判定することが可能です。

サンプルのControllerクラスでは、以下のようにjspごとに異なるURLを設定しています。各URLのメソッド内で、imwPageTypeに応じてさらに処理を分岐しています。

クライアントタイプ「PC」のURL一覧

URL	jsp	画面種別
sample/im_workflow/tsfw/app/purchase/apply	apply.jsp	申請画面
		申請（起票案件）画面
		一時保存画面
		再申請画面
sample/im_workflow/tsfw/app/purchase/approve	approve.jsp	処理画面
sample/im_workflow/tsfw/app/purchase/confirm	confirm.jsp	確認画面
sample/im_workflow/tsfw/app/purchase/detail	detail.jsp	詳細画面

クライアントタイプ「SP」のURL一覧

URL	jsp	画面種別
sample/im_workflow/tsfw/app/smartphone/purchase/apply	apply_sp.jsp	申請画面（スマートフォン用）
		申請（起票案件）画面（スマートフォン用）
		一時保存画面（スマートフォン用）
		再申請画面（スマートフォン用）
sample/im_workflow/tsfw/app/smartphone/purchase/approve	approve_sp.jsp	処理画面（スマートフォン用）
sample/im_workflow/tsfw/app/smartphone/purchase/confirm	confirm_sp.jsp	確認画面（スマートフォン用）

コラム

画面種別の一覧については、以下のAPIドキュメントを参照してください。
「IM-Workflow CodeList 画面種別」
「jp.co.intra_mart.foundation.workflow.code.PageType」

注意

詳細画面については、クライアントタイプごとにURLが分かれておりません。
どちらのクライアントタイプでアクセスした場合でも共通のURLが呼び出されます。

詳細画面のポップアップ表示

- 詳細画面は各種一覧画面からポップアップにて表示されるため、グローバルナビが表示されないように設定する必要があります。

- テーマ設定

  グローバルナビが表示されないモードでテーマが表示されるように、詳細画面のURLに対してHeadOnlyThemeBuilderを適用します。
  本書のサンプルでは、<%サンプルプログラムディレクトリ%>/src/main/conf/theme-head-only-path-config/im_workflow_tsfw_sample.xml で設定を行っています。

```
<?xml version="1.0" encoding="UTF-8"?>
<theme-head-only-path-config xmlns="http://www.intra-mart.jp/theme/theme-head-only-path-config"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.intra-mart.jp/theme/theme-head-only-path-config theme-head-only-path-config.xsd">
 <path>/sample/im_workflow/tsfw/app/purchase/detail</path>
</theme-head-only-path-config>
```

> **コラム**
>
> テーマの仕様については、「テーマ仕様書」-「PageBuilder」を参照してください。

Controllerクラスの実装例（クライアントタイプ「PC」）

- ルートURLの設定

  @RequestMappingアノテーションによってControllerクラス全体のルートURLをセットします。

```
@Controller
@RequestMapping("sample/im_workflow/tsfw/app/purchase")
public class PurchaseController {
```

- 申請画面表示処理
  - 申請画面

    ユーザデータIDを採番して画面にバインドします。

  - 申請（起票）画面

    案件が作成されているため、ユーザデータIDは採番しません。

  - 一時保存画面

    ユーザデータIDをキーにデータベースから入力済みのレコードを取得し、画面にバインドします。
    レコードが取得できない場合は、エラー画面へ遷移させます。

  - 再申請画面

ユーザデータIDをキーにデータベースから入力済みのレコードを取得し、画面にバインドします。
レコードが取得できない場合は、エラー画面へ遷移させます。

```
/**
 * 申請画面表示処理
 * @param model モデル
 * @return 遷移先
 * @throws AccessSecurityException
 * @throws IOException
 */
@RequestMapping(value = "apply")
public String apply(final Model model, final PurchaseForm purchaseForm) throws AccessSecurityException, IOException {

    if (PageType.pageTyp_App.toString().equals(purchaseForm.getImwPageType())) {
        // 申請
        String userDataId = "";
        final Identifier identifier = new Identifier();
        userDataId = identifier.get();

        purchaseForm.setImwUserDataId(userDataId);
    } else if (PageType.pageTyp_UnApp.toString().equals(purchaseForm.getImwPageType())) {
        // 起票
    } else {
        // 一時保存 or 再申請
        SampleImwTPurchase matterData = repository.select(purchaseForm.getImwUserDataId());
        if (matterData == null) {
            final Message message = new Message();
            message.setTitle(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
            message.setMessage(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
            final String[] details = { MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.005") };
            message.setDetails(details);
            model.addAttribute(TransferView.MESSAGE, message);
            return TransferView.VIEW_NAME_ERROR;
        }

        purchaseForm.setItem_name(matterData.getItemName());
        purchaseForm.setItem_amount(matterData.getItemAmount());
        purchaseForm.setItem_price(matterData.getItemPrice());
        purchaseForm.setItem_comment(matterData.getItemComment());

    }

    model.addAttribute(purchaseForm);

    return "sample/im_workflow/tsfw/app/purchase/apply.jsp";
}
```

- 処理画面表示処理
  - 処理画面

    ユーザデータIDをキーにデータベースから入力済みのレコードを取得し、画面にバインドします。
    レコードが取得できない場合は、エラー画面へ遷移させます。

```
/**
 * 処理画面表示処理
 * @param model モデル
 * @return 遷移先
 * @throws AccessSecurityException
 */
@RequestMapping(value = "approve")
public String approve(final Model model, final PurchaseForm purchaseForm) throws AccessSecurityException {
    SampleImwTPurchase matterData = repository.select(purchaseForm.getImwUserDataId());

    if (matterData == null) {
        final Message message = new Message();
        message.setTitle(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        message.setMessage(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        final String[] details = { MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.005") };
        message.setDetails(details);
        model.addAttribute(TransferView.MESSAGE, message);
        return TransferView.VIEW_NAME_ERROR;
    }

    purchaseForm.setItem_name(matterData.getItemName());
    purchaseForm.setItem_amount(matterData.getItemAmount());
    purchaseForm.setItem_price(matterData.getItemPrice());
    purchaseForm.setItem_total(matterData.getItemTotal());
    purchaseForm.setItem_comment(matterData.getItemComment());

    model.addAttribute(purchaseForm);
    return "sample/im_workflow/tsfw/app/purchase/approve.jsp";
}
```

- 確認画面表示処理
  - 確認画面

    ユーザデータIDをキーにデータベースから入力済みのレコードを取得し、画面にバインドします。
    レコードが取得できない場合は、エラー画面へ遷移させます。

```
/**
 * 確認画面表示処理
 * @param model モデル
 * @return 遷移先
 * @throws AccessSecurityException
 */
@RequestMapping(value = "confirm")
public String confirm(final Model model, final PurchaseForm purchaseForm) throws AccessSecurityException {
    SampleImwTPurchase matterData = repository.select(purchaseForm.getImwUserDataId());

    if (matterData == null) {
        final Message message = new Message();
        message.setTitle(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        message.setMessage(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        final String[] details = { MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.005") };
        message.setDetails(details);
        model.addAttribute(TransferView.MESSAGE, message);
        return TransferView.VIEW_NAME_ERROR;
    }

    purchaseForm.setItem_name(matterData.getItemName());
    purchaseForm.setItem_amount(matterData.getItemAmount());
    purchaseForm.setItem_price(matterData.getItemPrice());
    purchaseForm.setItem_total(matterData.getItemTotal());
    purchaseForm.setItem_comment(matterData.getItemComment());

    model.addAttribute(purchaseForm);
    return "sample/im_workflow/tsfw/app/purchase/confirm.jsp";
}
```

- 詳細画面表示処理
  - 処理画面

    ユーザデータIDをキーにデータベースから入力済みのレコードを取得し、画面にバインドします。
    レコードが取得できない場合は、エラー画面へ遷移させます。

クライアントタイプがスマートフォンの場合は、PCに切り替えます。

```
/**
 * 詳細画面表示処理
 * @param model モデル
 * @return 遷移先
 * @throws AccessSecurityException
 */
@RequestMapping(value = "detail")
public final String detail(final Model model, final PurchaseForm purchaseForm) throws AccessSecurityException {
    SampleImwTPurchase matterData = repository.select(purchaseForm.getImwUserDataId());

    if (matterData == null) {
        final Message message = new Message();
        message.setTitle(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        message.setMessage(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        final String[] details = { MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.005") };
        message.setDetails(details);
        model.addAttribute(TransferView.MESSAGE, message);
        return TransferView.VIEW_NAME_ERROR;
    }

    purchaseForm.setItem_name(matterData.getItemName());
    purchaseForm.setItem_amount(matterData.getItemAmount());
    purchaseForm.setItem_price(matterData.getItemPrice());
    purchaseForm.setItem_total(matterData.getItemTotal());
    purchaseForm.setItem_comment(matterData.getItemComment());

    model.addAttribute(purchaseForm);

    // ショートカットURLからのアクセスかどうかを判定します。
    model.addAttribute("imwShortCutAccess", "1".equals(purchaseForm.getImwShortCutFlag()));

    // クライアントタイプがスマートフォンの場合は、PCに切り替えます。
    final ClientContext context = Contexts.get(ClientContext.class);
    if ("sp".equals(context.getClientTypeId())) {
        ClientTypeSwitcher.oneTimeSwitchTo("pc");
        model.addAttribute("isSmartPhone", true);
    }

    return "sample/im_workflow/tsfw/app/purchase/detail.jsp";
}
```

- エラーハンドリング処理
  - 例外ハンドラ

    例外クラスごとに@ExceptionHandlerを指定したメソッドにてエラーハンドリング処理を記述します。
    サンプルでは、例外が発生した場合にシステムエラー画面へ遷移させています。

```
/**
 * エラーハンドリング処理
 * @param 例外クラス e
 * @return ModelAndView
 * @throws AccessSecurityException
 */
@ExceptionHandler(Exception.class)
public ModelAndView ExceptionHandler(Exception e) throws AccessSecurityException {
    final Message message = new Message();
    message.setTitle(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
    message.setMessage(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
    final String[] details = { MessageManager.getInstance().getMessage("SAMPLE.IMW.TSFW.ERR.001") };
    message.setDetails(details);
    return new ModelAndView(TransferView.VIEW_NAME_ERROR).addObject(TransferView.MESSAGE, message);
}
```

Controllerクラスの実装例（クライアントタイプ「スマートフォン」）

- ルートURLの設定

  @RequestMappingアノテーションによってControllerクラス全体のルートURLをセットします。

```
@Controller
@RequestMapping("sample/im_workflow/tsfw/app/smartphone/purchase")
public class PurchaseSmartphoneController {
```

- 申請画面（スマートフォン）表示処理
  - 申請画面（スマートフォン）

    ユーザデータIDを採番して画面にバインドします。

  - 申請（起票）画面（スマートフォン）

    案件が作成されているため、ユーザデータIDは採番しません。

  - 一時保存画面（スマートフォン）

    ユーザデータIDをキーにデータベースから入力済みのレコードを取得し、画面にバインドします。
    レコードが取得できない場合は、エラー画面へ遷移させます。

  - 再申請画面（スマートフォン）

    ユーザデータIDをキーにデータベースから入力済みのレコードを取得し、画面にバインドします。
    レコードが取得できない場合は、エラー画面へ遷移させます。

```
/**
 * 申請画面（スマートフォン）表示処理
 * @param model モデル
 * @return 遷移先
 * @throws AccessSecurityException
 * @throws IOException
 */
@RequestMapping(value = "apply")
public String apply(HttpServletRequest request, final Model model, final PurchaseForm purchaseForm) throws IOException,
AccessSecurityException {
    if (PageType.pageTyp_App_Sp.toString().equals(purchaseForm.getImwPageType())) {
        // 申請
        final Identifier identifier = new Identifier();
        String userDataId = identifier.get();

        purchaseForm.setImwUserDataId(userDataId);
    } else if (PageType.pageTyp_UnApp_Sp.toString().equals(purchaseForm.getImwPageType())) {
        // 起票
    } else {
        // 一時保存 or 再申請
        SampleImwTPurchase matterData = repository.select(purchaseForm.getImwUserDataId());
        if (matterData == null) {
            final Message message = new Message();
            message.setTitle(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
            message.setMessage(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
            final String[] details = { MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.005") };
            message.setDetails(details);
            model.addAttribute(TransferView.MESSAGE, message);
            return TransferView.VIEW_NAME_ERROR;
        }

        purchaseForm.setItem_name(matterData.getItemName());
        purchaseForm.setItem_amount(matterData.getItemAmount());
        purchaseForm.setItem_price(matterData.getItemPrice());
        purchaseForm.setItem_comment(matterData.getItemComment());

    }

    model.addAttribute(purchaseForm);
    model.addAttribute("homeUrl", getHomeUrl(request));

    return "sample/im_workflow/tsfw/app/purchase/apply_sp.jsp";
}
```

- 処理画面（スマートフォン）表示処理
  - 処理画面（スマートフォン）

    ユーザデータIDをキーにデータベースから入力済みのレコードを取得し、画面にバインドします。
    レコードが取得できない場合は、エラー画面へ遷移させます。

```
/**
 * 処理画面（スマートフォン）表示処理
 * @param model モデル
 * @return 遷移先
 * @throws AccessSecurityException
 */
@RequestMapping(value = "approve")
public String approve(HttpServletRequest request, final Model model, final PurchaseForm purchaseForm) throws
AccessSecurityException {
    SampleImwTPurchase matterData = repository.select(purchaseForm.getImwUserDataId());

    if (matterData == null) {
        final Message message = new Message();
        message.setTitle(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        message.setMessage(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        final String[] details = { MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.005") };
        message.setDetails(details);
        model.addAttribute(TransferView.MESSAGE, message);
        return TransferView.VIEW_NAME_ERROR;
    }

    purchaseForm.setItem_name(matterData.getItemName());
    purchaseForm.setItem_amount(matterData.getItemAmount());
    purchaseForm.setItem_price(matterData.getItemPrice());
    purchaseForm.setItem_total(matterData.getItemTotal());
    purchaseForm.setItem_comment(matterData.getItemComment());

    model.addAttribute(purchaseForm);
    model.addAttribute("homeUrl", getHomeUrl(request));

    return "sample/im_workflow/tsfw/app/purchase/approve_sp.jsp";
}
```

- 確認画面（スマートフォン）表示処理
  - 確認画面（スマートフォン）

    ユーザデータIDをキーにデータベースから入力済みのレコードを取得し、画面にバインドします。
    レコードが取得できない場合は、エラー画面へ遷移させます。

```
/**
 * 確認画面（スマートフォン）表示処理
 * @param model モデル
 * @return 遷移先
 * @throws AccessSecurityException
 */
@RequestMapping(value = "confirm")
public String confirm(HttpServletRequest request, final Model model, final PurchaseForm purchaseForm) throws
AccessSecurityException {
    SampleImwTPurchase matterData = repository.select(purchaseForm.getImwUserDataId());

    if (matterData == null) {
        final Message message = new Message();
        message.setTitle(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        message.setMessage(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
        final String[] details = { MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.005") };
        message.setDetails(details);
        model.addAttribute(TransferView.MESSAGE, message);
        return TransferView.VIEW_NAME_ERROR;
    }

    purchaseForm.setItem_name(matterData.getItemName());
    purchaseForm.setItem_amount(matterData.getItemAmount());
    purchaseForm.setItem_price(matterData.getItemPrice());
    purchaseForm.setItem_total(matterData.getItemTotal());
    purchaseForm.setItem_comment(matterData.getItemComment());

    model.addAttribute(purchaseForm);
    model.addAttribute("homeUrl", getHomeUrl(request));

    return "sample/im_workflow/tsfw/app/purchase/confirm_sp.jsp";
}
```

- エラーハンドリング処理
  - 例外ハンドラ

    例外クラスごとに@ExceptionHandlerを指定したメソッドにてエラーハンドリング処理を記述します。
    サンプルでは、例外が発生した場合にシステムエラー画面へ遷移させています。

```
/**
 * エラーハンドリング処理
 * @param 例外クラス e
 * @return ModelAndView
 * @throws AccessSecurityException
 */
@ExceptionHandler(Exception.class)
public ModelAndView ExceptionHandler(Exception e) throws AccessSecurityException {
    final Message message = new Message();
    message.setTitle(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
    message.setMessage(MessageManager.getInstance().getMessage("SAMPLE.IMW.ERR.004"));
    final String[] details = { MessageManager.getInstance().getMessage("SAMPLE.IMW.TSFW.ERR.001") };
    message.setDetails(details);
    return new ModelAndView(TransferView.VIEW_NAME_ERROR).addObject(TransferView.MESSAGE, message);
}
```

画面の作成

- Formクラスから受け取った情報に基づいてユーザコンテンツ画面を表示させるためのJSPファイルを作成します。

項目

- タグライブラリの指定
- ユーザコンテンツ画面の表示権限制御
- 標準申請・処理画面の呼び出し（クライアントタイプ「PC」）
- 標準申請・処理画面の呼び出し（クライアントタイプ「SP」）

タグライブラリの指定

- JSPファイルのtaglibディレクティブにてタグライブラリの読み込みの指定を行います。
  - クライアントタイプ「PC」：IM-Workflow タグライブラリ

```
<%@ taglib prefix="workflow" uri="http://www.intra-mart.co.jp/taglib/imw/workflow" %>
```

  - クライアントタイプ「SP」：IM-Workflow スマートフォン タグライブラリ

```
<%@ taglib prefix="workflowSmartphone" uri="http://www.intra-mart.co.jp/taglib/imw/workflow-smartphone" %>
```

ユーザコンテンツ画面の表示権限制御

- ワークフロー上の権限が有効なユーザのみがユーザコンテンツ画面を表示できるように、workflowUserContentsAuthタグを配置します。
  - workflowUserContentsAuthタグ を配置することで、ログインユーザがユーザコンテンツ画面の表示権限があるかどうかを確認します。

```
<workflow:workflowUserContentsAuth imwApplyBaseDate='${f:h(purchaseForm.imwApplyBaseDate)}'
imwAuthUserCode = '${f:h(purchaseForm.imwAuthUserCode)}'
imwFlowId='${f:h(purchaseForm.imwFlowId)}'
imwNodeId ='${f:h(purchaseForm.imwNodeId)}'
imwPageType = '${f:h(purchaseForm.imwPageType)}'
imwSystemMatterId='${f:h(purchaseForm.imwSystemMatterId)}'
imwUserDataId='${f:h(purchaseForm.imwUserDataId)}'/>
```

  - クライアントタイプ「SP」の場合は、spWorkflowUserContentsAuthタグ を利用します。

```
<workflowSmartphone:spWorkflowUserContentsAuth imwApplyBaseDate='${f:h(purchaseForm.imwApplyBaseDate)}'
imwAuthUserCode = '${f:h(purchaseForm.imwAuthUserCode)}'
imwFlowId='${f:h(purchaseForm.imwFlowId)}'
imwNodeId ='${f:h(purchaseForm.imwNodeId)}'
imwPageType = '${f:h(purchaseForm.imwPageType)}'
imwSystemMatterId='${f:h(purchaseForm.imwSystemMatterId)}'
imwUserDataId='${f:h(purchaseForm.imwUserDataId)}'/>
```

> **コラム**
>
> ユーザコンテンツ画面の表示権限制御の仕様については、「IM-Workflow プログラミングガイド」-「ユーザコンテンツ画面への不正な直接アクセスを抑止する」を参照してください。

標準申請・処理画面の呼び出し（クライアントタイプ「PC」）

- ユーザコンテンツ画面での入力完了後に、標準申請・処理画面を呼び出す処理を記述します。
  - workflowOpenPageCsjsタグ を設置し、workflowOpenPageタグを利用するためのデザインスタイルシートの宣言・Javascriptを読み込みます。

```
<workflow:workflowOpenPageCsjs />
```

  - workflowOpenPageタグ 設置します。
    - workflowOpenPageタグは標準申請・処理画面へ遷移するためのHTMLのフォームタグ（<FORM>）を生成するため、ユーザアプリケーションデータのINPUTタグが内包されるように配置します。

```
<workflow:workflowOpenPage name="workflowOpenPageForm"
id="workflowOpenPageForm"
method="POST"
target="_top"
imwUserDataId="${f:h(purchaseForm.imwUserDataId)}"
imwSystemMatterId="${f:h(purchaseForm.imwSystemMatterId)}"
imwAuthUserCode="${f:h(purchaseForm.imwAuthUserCode)}"
imwApplyBaseDate="${f:h(purchaseForm.imwApplyBaseDate)}"
imwNodeId="${f:h(purchaseForm.imwNodeId)}"
imwFlowId="${f:h(purchaseForm.imwFlowId)}"
imwCallOriginalParams="${f:h(purchaseForm.imwCallOriginalParams)}"
imwNextScriptPath="${f:h(purchaseForm.imwCallOriginalPagePath)}">

<div class="imui-title-small-window">
  <h1><spring:message code="SAMPLE.IMW.TSFW.CAP.001" /></h1>
</div>
<div class="imui-toolbar-wrap">
  <div class="imui-toolbar-inner">
    <ul class="imui-list-toolbar">
      <li>
        <a href="javascript:void(0);" id="back">
          <span class="im-ui-icon-common-16-back"></span>
        </a>
      </li>
    </ul>
  </div>
</div>

<div class="imui-form-container">
  <header class="imui-chapter-title">
    <h2><spring:message code="SAMPLE.IMW.TSFW.CAP.001" /></h2>
  </header>

  <table class="imui-form">
    <tbody>
      <tr>
        <th><label class="imui-required"><spring:message code="SAMPLE.IMW.CAP.020" /></label></th>
        <td><input type="text" value="${f:h(purchaseForm.item_name)}" id="item_name" name="item_name"
size="50"></td>
      </tr>
      <tr>
        <th><label class="imui-required"><spring:message code="SAMPLE.IMW.CAP.021" /></label></th>
        <td><input type="text" value="${f:h(purchaseForm.item_amount)}" id="item_amount"
name="item_amount" size="50"></td>
```

```
          </tr>
          <tr>
            <th><label class="imui-required"><spring:message code="SAMPLE.IMW.CAP.022" /></label></th>
            <td><input type="text" value="${f:h(purchaseForm.item_price)}" id="item_price" name="item_price"
size="50"></td>
          </tr>
          <tr>
            <th><label><spring:message code="SAMPLE.IMW.CAP.024" /></label></th>
            <td>
              <textarea id="item_comment" name="item_comment" rows="3"
cols="40">${f:h(purchaseForm.item_comment)}</textarea>
            </td>
          </tr>
        </tbody>
      </table>

      <div class="imui-operation-parts">

        <imart:decision case="0" value="${f:h(purchaseForm.imwPageType)}">
          <input type="button" value='<spring:message code="SAMPLE.IMW.CAP.003" />' id="openPage0"
name="openPage0" class="imui-large-button" escapeXml="true" escapeJs="false" />
          <input type="button" value='<spring:message code="SAMPLE.IMW.CAP.002" />' id="openPage1"
name="openPage1" class="imui-large-button" escapeXml="true" escapeJs="false" />
        </imart:decision>
        <imart:decision case="1" value="${f:h(purchaseForm.imwPageType)}">
          <input type="button" value='<spring:message code="SAMPLE.IMW.CAP.003" />' id="openPage0"
name="openPage0" class="imui-large-button" escapeXml="true" escapeJs="false" />
          <input type="button" value='<spring:message code="SAMPLE.IMW.CAP.002" />' id="openPage1"
name="openPage1" class="imui-large-button" escapeXml="true" escapeJs="false" />
        </imart:decision>
        <imart:decision case="2" value="${f:h(purchaseForm.imwPageType)}">
          <input type="button" value='<spring:message code="SAMPLE.IMW.CAP.003" />' id="openPage2"
name="openPage2" class="imui-large-button" escapeXml="true" escapeJs="false" />
        </imart:decision>
        <imart:decision case="3" value="${f:h(purchaseForm.imwPageType)}">
          <input type="button" value='<spring:message code="SAMPLE.IMW.CAP.004" />' id="openPage3"
name="openPage3" class="imui-large-button" escapeXml="true" escapeJs="false" />
        </imart:decision>
      </div>
    </div>
</workflow:workflowOpenPage>
```

- ユーザコンテンツ画面での入力完了後に、csjsファンクション workflowOpenPage(String pageType, String callback)を呼び出します。

```
$('#openPage0').click(function(){
  if (!imuiValidate('#workflowOpenPageForm', rules, messages)) return;
  workflowOpenPage('0');
  return false;
});
```

標準申請・処理画面の呼び出し（クライアントタイプ「SP」）

- ユーザコンテンツ画面での入力完了後に、標準申請・処理画面を呼び出す処理を記述します。
  - spWorkflowOpenPageCsjsタグ を設置し、spWorkflowOpenPageタグを利用するためのデザインスタイルシートの宣言・Javascriptを読み込みます。

```
<workflowSmartphone:spWorkflowOpenPageCsjs />
```

  - spWorkflowOpenPageタグ 設置します。
    - spWorkflowOpenPageタグは標準申請・処理画面へ遷移するためのHTMLのフォームタグ（<FORM>）を生成するため、ユーザアプリケーションデータのINPUTタグが内包されるように配置します。

```
<workflowSmartphone:spWorkflowOpenPage name="workflowOpenPageForm"
      id="workflowOpenPageForm"
      method="POST"
      target="_top"
      imwUserDataId='${f:h(purchaseForm.imwUserDataId)}'
      imwSystemMatterId='${f:h(purchaseForm.imwSystemMatterId)}'
      imwAuthUserCode='${f:h(purchaseForm.imwAuthUserCode)}'
```

```
        imwApplyBaseDate='${f:h(purchaseForm.imwApplyBaseDate)}'
        imwNodeId='${f:h(purchaseForm.imwNodeId)}'
        imwFlowId='${f:h(purchaseForm.imwFlowId)}'
        imwCallOriginalParams='${f:h(purchaseForm.imwCallOriginalParams)}'
        imwNextScriptPath='${f:h(purchaseForm.imwCallOriginalPagePath)}'>
  <ul data-role="listview" class="wrap">
    <li data-role="fieldcontain">
     <label for="item_name" class="imui-smart-ui-required"><spring:message code="SAMPLE.IMW.CAP.020" />
</label>
     <input type="text" id="item_name" name="item_name" value='${f:h(purchaseForm.item_name)}' data-theme="d"
/>
    </li>
    <li data-role="fieldcontain">
     <label for="item_amount" class="imui-smart-ui-required"><spring:message code="SAMPLE.IMW.CAP.021" />
</label>
     <input type="number" id="item_amount" name="item_amount" value='${f:h(purchaseForm.item_amount)}' data-
theme="d" />
    </li>
    <li data-role="fieldcontain">
     <label for="item_price" class="imui-smart-ui-required"><spring:message code="SAMPLE.IMW.CAP.022" />
</label>
     <input type="number" id="item_price" name="item_price" value='${f:h(purchaseForm.item_price)}' data-
theme="d" />
    </li>
    <li data-role="fieldcontain">
     <label for="item_comment"><spring:message code="SAMPLE.IMW.CAP.024" /></label>
     <textarea id="item_comment" name="item_comment" data-
theme="d">${f:h(purchaseForm.item_comment)}</textarea>
    </li>
    <imart:decision case="10" value='${f:h(purchaseForm.imwPageType)}'>
    <li data-role="fieldcontain">
     <fieldset class="ui-grid-a">
      <div class="ui-block-a"><button type="button" id="openPage0" data-theme="b"><spring:message
code="SAMPLE.IMW.CAP.003" /></button></div>
      <div class="ui-block-b"><button type="button" id="openPage1" data-theme="b"><spring:message
code="SAMPLE.IMW.CAP.002" /></button></div>
     </fieldset>
    </li>
    </imart:decision>
    <imart:decision case="11" value='${f:h(purchaseForm.imwPageType)}'>
    <li data-role="fieldcontain">
     <fieldset class="ui-grid-a">
      <div class="ui-block-a"><button type="button" id="openPage0" data-theme="b"><spring:message
code="SAMPLE.IMW.CAP.003" /></button></div>
      <div class="ui-block-b"><button type="button" id="openPage1" data-theme="b"><spring:message
code="SAMPLE.IMW.CAP.002" /></button></div>
     </fieldset>
    </li>
    </imart:decision>
    <imart:decision case="12" value='${f:h(purchaseForm.imwPageType)}'>
    <li data-role="fieldcontain">
     <fieldset>
      <div><button type="button" id="openPage2" data-theme="b"><spring:message
code="SAMPLE.IMW.CAP.003" /></button></div>
     </fieldset>
    </li>
    </imart:decision>
    <imart:decision case="13" value='${f:h(purchaseForm.imwPageType)}'>
    <li data-role="fieldcontain">
     <fieldset>
      <div><button type="button" id="openPage3" data-theme="b"><spring:message
code="SAMPLE.IMW.CAP.004" /></button></div>
     </fieldset>
    </li>
    </imart:decision>
  </ul>
</workflowSmartphone:spWorkflowOpenPage>
```

- ユーザコンテンツ画面での入力完了後に、csjsファンクション workflowOpenPage4Sp(String pageType, String callback)を呼び出します。

```
$('#openPage0').click(function(){
  if (!imspValidate('#workflowOpenPageForm', rules, messages)) return;
  workflowOpenPage4Sp('10');
  return false;
});
```

作成した画面の設定方法

以下の手順で、作成した画面のURLをコンテンツ定義に登録します。

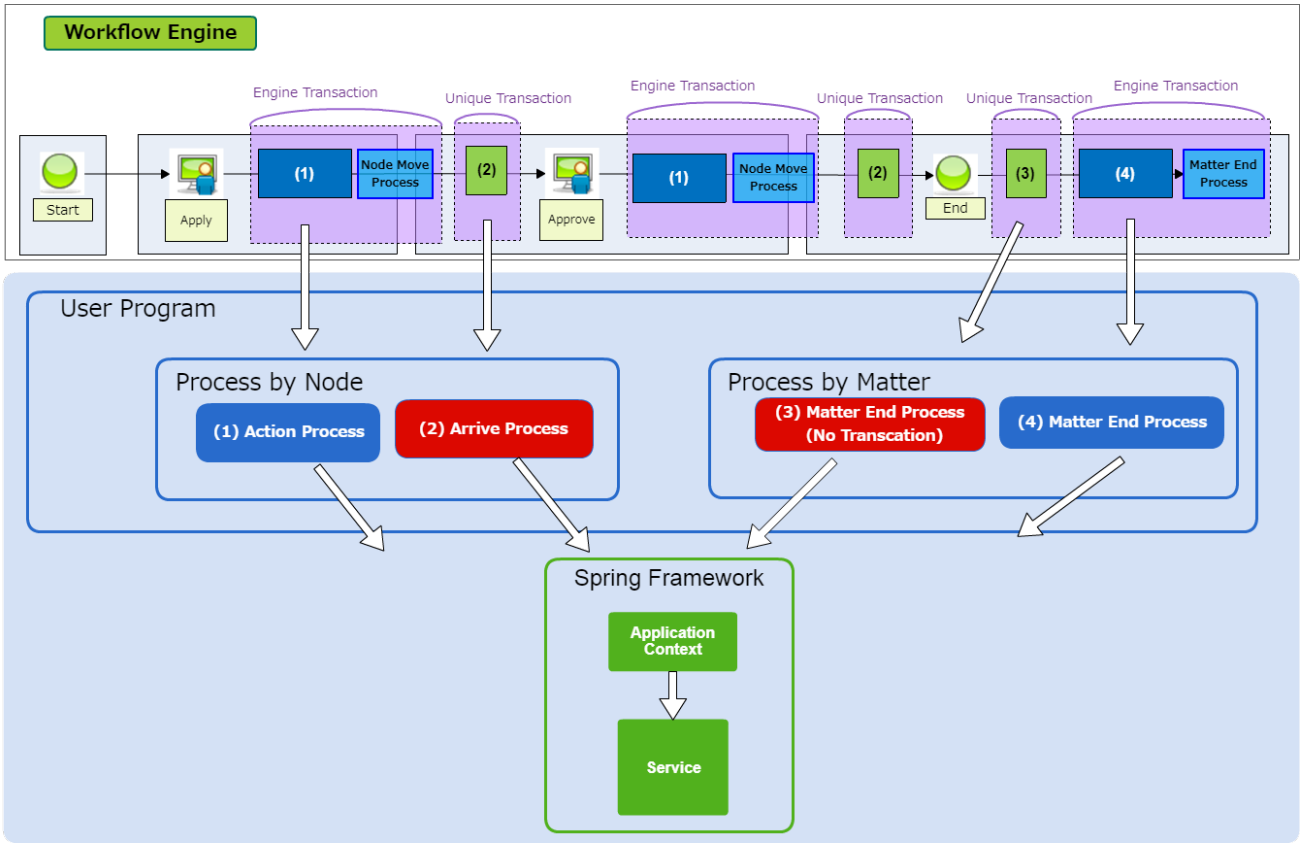
- コンテンツ定義の編集画面を表示し、画面タブを選択します。
- 「画面定義 - パス種別URL(旧名称:JSP or Servlet)」を選択します。
- 「ページパス」にControllerクラスに記述した各ユーザコンテンツ画面のURLを設定します。

ユーザプログラムの開発

IM-Workflow では、案件処理の様々なタイミングにて設定されたユーザプログラムを呼び出す機構を提供しています。
ユーザプログラムを利用して、独自の業務ロジックの実行やワークフローエンジンの制御を行うことが可能です。

本書では、ユーザプログラムのうち、よく利用される以下の4つに限定して説明を行います。

- 「アクション処理」
  - 申請・承認など操作時に実行される処理です。画面操作のスレッドにて処理が実行されます。
- 「到達処理」
  - ノードに到達した場合に実行される処理です。画面操作と別のスレッドにて処理が実行されます。
- 「案件終了処理/案件終了処理（トランザクションなし）」
  - 案件が完了する直前に実行される処理です。画面操作と別のスレッドにて処理が実行されます。



他のユーザプログラムについては、「 IM-Workflow プログラミングガイド 」-「 ユーザプログラムの作成 」 を参照してください。

項目

- DIコンテナの呼び出し
  - 明示的な自動バインディング呼び出しのコード例
- アクション処理
  - トリガーとなるユーザ操作
  - 同期処理
  - トランザクション制御
  - エラー処理
- 到達処理
  - 非同期処理
  - トランザクション制御
- 案件終了処理・案件終了処理（トランザクションなし）
  - 非同期処理
  - トランザクション制御
  - エラー処理

DIコンテナの呼び出し

- ユーザコンテンツ画面とは異なり、ユーザプログラムの呼び出し時にはDIコンテナによる自動バインディング（@Inject アノテーションを付与したフィールドへの自動設定）が有効ではありません。

  コンポーネントを使いたい場合には、ApplicationContextを使用して明示的に取得してください。
  ApplicationContext経由で取得したコンポーネント内の空間では、自動バインディングが有効です。

  > **コラム**
  >
  > ユーザコンテンツ画面はSpring MVC経由で呼び出されるため、自動バインディングが有効です。

明示的な自動バインディング呼び出しのコード例

- ApplicationContextは、 intra-mart Accel Platform で提供されているAPIApplicationContextProvider にて取得可能です。

```
public class ActionProcess extends ActionProcessEventListener {

  // 申請
  @Override
  public final String apply(final ActionProcessParameter parameter,
    final Map<String, Object> userParameter) throws Exception {

    // アクション処理用のサービスを取得します。
    final ActionProcessService service = ApplicationContextProvider.getApplicationContext().
                                getBean(ActionProcessService.class);
    return service.apply(parameter, userParameter);
  }
```

```
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional(propagation = Propagation.MANDATORY)
public class ActionProcessServiceImpl implements ActionProcessService {

    @Inject
    SampleImwTPurchaseRepository repository;

    private final String PURCHASE_MATTER_PROPERTY_KEY = "item_total";

    // 申請
    public final String apply(final ActionProcessParameter parameter,
      final Map<String, Object> userParameter) throws Exception {

        System.out.println("----- ActionProcessParameter - apply -----");
        outputLog(parameter);
        System.out.println("----- ActionProcessParameter - apply -----");

        // サーバサイドバリデーションの実行
        validate(parameter, userParameter);

        // 案件番号の採番
        String number = null;
        try {
            number = WorkflowNumberingManager.getNumber();
        } catch (final WorkflowException e) {
            throw new WorkflowExternalException(
              MessageManager.getInstance().getMessage(
                LocaleUtil.toLocale(parameter.getLocaleId()), "SAMPLE.IMW.ERR.003"));
        }

        // 入力データのテーブルへの登録
        SampleImwTPurchase entity = getEntity(parameter, userParameter);
        repository.insert(entity);

        // 入力データの案件プロパティへの登録
        createMatterProperty(
          parameter.getUserDataId(), PURCHASE_MATTER_PROPERTY_KEY, entity.getItemTotal());
        return number;
    }
```

アクション処理

標準申請・処理画面にて申請・処理ボタンを押下するなど、ユーザが案件に関する処理を行った際に呼ばれる処理です。

- ワークフローデータに加えて、申請・処理画面で入力されたユーザアプリケーションデータをUserParameterとして受け取ることが可能です。
- ユーザアプリケーションデータに対してサーバサイド・バリデーションとデータベースへの永続化などを行います。

トリガーとなるユーザ操作

- 具体的には、以下のユーザ操作がアクション処理のトリガーです。
- トリガーとなるユーザ操作ごとに呼び出されるメソッドが異なります。

アクション処理 メソッド一覧

処理種別	メソッド	説明
申請	apply	標準申請画面にて「申請」ボタンを押下した際に呼び出されるメソッドです。
再申請	reapply	標準処理画面にて、処理種別「再申請」を選択し、「処理」ボタンを押下した際に呼び出されるメソッドです。
申請（一時保存）	applyFromTempSave	一時保存の状態で、標準申請画面にて「申請」ボタンを押下した際に呼び出されるメソッドです。
申請（未処理）	applyFromUnapply	未申請の状態で、標準申請画面にて「申請」ボタンを押下した際に呼び出されるメソッドです。
取止め	discontinue	標準処理画面にて、処理種別「取止め」を選択し、「処理」ボタンを押下した際に呼び出されるメソッドです。
引戻し	pullBack	処理済み一覧で「引戻し」ボタンを押下した際に表示される標準画面にて、「引戻し」ボタンを押下した際に呼び出されるメソッドです。 ユーザコンテンツ画面での操作を経由しないため、UserParameterはnullです。
差戻し後引戻し	sendBackToPullBack	差戻しされた状態にて、処理済み一覧で「引戻し」ボタンを押下した際に表示される標準画面にて、「引戻し」ボタンを押下した際に呼び出されるメソッドです。 ユーザコンテンツ画面での操作を経由しないため、UserParameterはnullです。
承認	approve	標準処理画面にて、処理種別「承認」を選択し、「処理」ボタンを押下した際に呼び出されるメソッドです。 自動処理による承認の場合、ユーザコンテンツ画面での操作を経由しないため、UserParameterはnullです。 一括処理による承認の場合、ユーザコンテンツ画面での操作を経由しないため、UserParameterは空のMapです。
承認終了	approveEnd	標準処理画面にて、処理種別「承認終了」を選択し、「処理」ボタンを押下した際に呼び出されるメソッドです。
否認	deny	標準処理画面にて、処理種別「否認」を選択し、「処理」ボタンを押下した際に呼び出されるメソッドです。 自動処理による否認の場合、ユーザコンテンツ画面での操作を経由しないため、UserParameterはnullです。
差戻し	sendBack	標準処理画面にて、処理種別「差戻し」を選択し、「処理」ボタンを押下した際に呼び出されるメソッドです。 自動処理による差戻しの場合、ユーザコンテンツ画面での操作を経由しないため、UserParameterはnullです。
保留	reserve	標準処理画面にて、処理種別「保留」を選択し、「処理」ボタンを押下した際に呼び出されるメソッドです。
保留解除	reserveCancel	標準処理画面にて、処理種別「保留解除」を選択し、「処理」ボタンを押下した際に呼び出されるメソッドです。

処理種別	メソッド	説明
案件操作	matterHandle	案件操作画面にてノード移動を行った際に呼び出されるメソッドです。 ユーザコンテンツ画面での操作を経由しないため、UserParameterはnullです。
一時保存（新規登録）	tempSaveCreate	標準画面にて「一時保存」ボタンを押下した際に呼び出されるメソッドです。
一時保存（更新）	tempSaveUpdate	一時保存の状態で、標準画面にて「一時保存」ボタンを押下した際に呼び出されるメソッドです。
一時保存（削除）	tempSaveDelete	一時保存一覧画面にて、一時保存データに対する「削除」ボタンを押下した際に呼び出されるメソッドです。 ユーザコンテンツ画面での操作を経由しないため、UserParameterはnullです。

同期処理

- アクション処理は標準申請・処理画面からのHTTPリクエストと同期的に実行されます。処理結果のメッセージを標準申請・処理画面に表示することが可能です。

  ただし、Controllerクラスとして呼び出される訳ではないので、Spring MVCの機能は利用できません。

- 標準画面の処理の非同期

  ワークフローパラメータのstandard-exec-jssp-asyncをtrueに変更した場合は、アクション処理はHTTPリクエストと非同期で実行されます。

> **コラム**
>
> standard-exec-jssp-asyncの仕様については、「IM-Workflow 仕様書」-「3.22 標準画面の処理の非同期」を参照してください。

トランザクション制御

- アクション処理は IM-Workflow エンジン側が張るトランザクションの内部で呼び出されるため、データの一貫性を保証する必要がある処理です。 @Transactionalアノテーション を付与してトランザクションの管理を行う必要があります。
- アクション処理内でシェアードデータベースへの更新処理を行うことはできません。

  IM-Workflow エンジンがテナントデータベースへトランザクションを張っている、かつ intra-mart Accel Platform ではXAトランザクションが利用できないため。

エラー処理

- WorkflowExternalException をスローすることで、指定したメッセージを画面に表示することが可能です。

  メッセージIDではなく、メッセージ本文にて指定します。

```
throw new WorkflowExternalException(
    MessageManager.getInstance().getMessage(
      LocaleUtil.toLocale(parameter.getLocaleId()), "SAMPLE.IMW.ERR.003"));
```

到達処理

ノードに到達したタイミングで実行される処理です。

- 下記のような場合に実行されます。
  - 前ノードの処理者が、「申請」または「承認」を行って到達した場合
  - 他のノードから、「差戻し」され到達した場合
  - 「引戻し」を行って到達した場合
  - 案件操作で到達した場合
- 外部データベース上のユーザアプリケーションデータの更新やワークフローエンジンの制御などを行います。
- ワークフローデータのみパラメータとして、受け取ることが可能です。

非同期処理

- 標準申請・処理画面からのHTTPリクエストとは非同期で実行されます。

  到達処理では、HTTPリクエスト・セッションの情報を参照することはできません。

- 同期／非同期制御

  ワークフローパラメータのarrive-process-asyncをfalseに変更した場合は、到達処理はHTTPリクエストと同期的に実行されます。

> **コラム**
>
> arrive-process-asyncの仕様については、 「 IM-Workflow 仕様書 」-「5.1.1.1.1 案件終了処理、到達処理、メール送信処理、IMBox 送信処理の同期／非同期制御の設定」を参照してください。

トランザクション制御

- 到達処理が実行されるタイミングでは IM-Workflow エンジンによるトランザクションが確定しているため、独自にトランザクション制御を行う必要があります。

  通常は、データベースへの更新処理を行うメソッドに対して、@Transactionalアノテーション を付加します。

```
public class ArriveProcess extends ArriveProcessEventListener {

  public ArriveProcess() {
    super();
  }

  public boolean execute(final ArriveProcessParameter parameter) throws Exception {
    // 到達処理用のサービスを取得します。
    final ArriveProcessService service = ApplicationContextProvider.getApplicationContext().
                                getBean(ArriveProcessService.class);
    service.execute(parameter);
  }
```

```
import org.springframework.transaction.annotation.Transactional;

@Service
public class ArriveProcessServiceImpl implements ArriveProcessService {

    @Inject
    ExternalDatabaseRepository repository;

    @Override
    @Transactional(rollbackFor=java.lang.Exception.class)
    public boolean execute(
        final ArriveProcessParameter parameter) throws Exception {
        SampleImwTPurchase entity = repository.insert(parameter.getUserDataId());
        return true;
    }
}
```

> **注意**
>
> 到達処理では独自にトランザクション制御を行うことが可能ですが、 intra-mart Accel Platform にてXAトランザクションが利用できないため、テナントデータベースとシェアードデータベースに対して一括でコミットを行うことはできません。

案件終了処理・案件終了処理（トランザクションなし）

案件終了処理・案件終了処理（トランザクションなし）は、案件が終了する時に一度実行される処理です。

- 下記の場合に実行されます。
  - 最後の承認者が「承認」を行った場合
  - 承認者が「承認終了」を行った場合
  - 承認者が「否認」を行った場合
  - 申請者が「取止め」を行った場合

    - 案件操作で終了ノードに到達した場合
- ワークフローデータのみパラメータとして、受け取ることが可能です。

非同期処理

- 標準申請・処理画面からのHTTPリクエストとは非同期で実行されます。

  案件終了処理・案件終了処理（トランザクションなし）では、HTTPリクエスト・セッションの情報を参照することはできません。

トランザクション制御

- 直前のアクション処理や到達処理とは独立した処理となるため、案件終了処理・案件終了処理（トランザクションなし）でエラーが発生した場合、直前の処理を戻す（ロールバック）することはできません。
- 案件終了処理（トランザクションなし）は、案件終了処理の前に実行され、独自にトランザクション制御を行う必要があります。
- 案件終了処理は IM-Workflow エンジン側が張るトランザクションの内部で呼び出されるあめ、データの一貫性を保証する必要がある処理です。 @Transactionalアノテーション を付与してトランザクションの管理を行う必要があります。

エラー処理

- Exceptionクラスをスローすることで、案件終了処理・案件終了処理（トランザクションなし）をエラーとすることが可能です。
    - 案件終了処理（トランザクションなし）

      ユーザプログラム側でトランザクション制御を行うため、エラー時にはユーザプログラム側でロールバックを行ってください。

    - 案件終了処理

      IM-Workflow エンジンの内部処理にてトランザクションを管理しているため、エラー時にユーザプログラム側でロールバックを行わないでください。
- 案件終了処理・案件終了処理（トランザクションなし）がエラーとなった場合は、案件が完了せず、終了ノードで停止した状態に変化します。

案件処理系APIの実行について

案件処理系APIを実行することで、ユーザによる画面操作を経由することなく、プログラム上から案件の処理を行うことが可能です。

- ジョブプログラム、ユーザプログラム「到達処理」などからシステム的に案件の処理を行うことが可能です。
- 標準申請・処理画面を経由して申請・承認を行う場合でも、内部の動作としてはHTTPリクエストを受け取った同期的な処理内で画面から送信されたパラメータを基に案件処理系APIが実行されています。
  - そのため、案件処理系APIを実行する処理を作成することで、ユーザコンテンツ画面から標準申請・処理画面を経由せず、直接申請・承認を行うことも可能です。

案件処理系API

- ユーザコンテンツ画面から標準申請・処理画面を呼び出す際と同様に、ワークフローデータ・ユーザアプリケーションデータをパラメータとしてAPIを呼び出します。
  - 案件処理系APIの詳細については、各APIのAPIドキュメントを参照してください。

案件処理系API 一覧

API	説明
ApplyManager	申請マネージャ。 申請・起票処理を扱います。
ProcessManager	処理マネージャ。 承認・承認終了・再申請・否認・取止め・差戻し・保留・保留削除処理を扱います。
PullBackManager	引戻しマネージャ。 引戻し処理を扱います。
TempSaveManager	一時保存マネージャ。 一時保存案件の新規作成・更新・削除処理を扱います。

トランザクション制御

- 案件処理系APIでは内部で独自にトランザクションを管理しているため、案件処理系APIの呼び出し側の処理にて親のトランザクションを張ることはできません。
  - @Transactionalアノテーション を付与したメソッド内では案件処理系APIを実行しないでください。

非同期処理

- 標準申請・処理画面から申請・承認を行った際は、HTTPリクエストと同期的にユーザプログラム「アクション処理」が実行されるため、アクション処理内でHTTPリクエスト・セッションを取得できます。
- 案件処理系APIを直接実行して申請・処理を行う場合は、アクション処理内でHTTPリクエスト・セッションを取得できるかどうかは実行時の環境に依存します。
  - 標準申請・処理画面と同様に、画面操作から同期的に呼び出された処理内で案件処理系APIを実行した場合は、アクション処理内でHTTPリクエスト・セッションを取得可能です。
  - ジョブプログラムや到達処理など非同期処理内で案件処理系APIを実行した場合は、アクション処理内でHTTPリクエスト・セッションを取得できません。

サンプルプログラム

サンプルフロー「TERASOLUNA Server Framework開発モデル」に設定されているサンプルプログラムについて説明します。

項目

- 共通
- 共通（テナント環境セットアップ）
- 共通（データベース）
- ユーザコンテンツ画面
- ユーザプログラム

共通

プロジェクト全般で利用する設定ファイルです。

- Bean定義ファイル

  Spring Frameworkのコンポーネントスキャン対象となるルートパッケージを指定しています。
  Mybatis3のリポジトリインタフェースが格納されるパッケージを指定しています。

  - <%サンプルプログラムディレクトリ%/src/main/resources/META-INF/spring/applicationContext-sample_im_workflow_tsfw.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd
        http://mybatis.org/schema/mybatis-spring
        http://mybatis.org/schema/mybatis-spring.xsd">

   <!-- DIコンポーネントの対象とする要素のルートパッケージ -->
   <context:annotation-config />
   <context:component-scan base-package="jp.co.intra_mart.sample.im_workflow.tsfw" />

    <!-- Mybatis3のリポジトリ クラスのルートパッケージ -->
    <mybatis:scan base-package="jp.co.intra_mart.sample.im_workflow.tsfw.domain" />

</beans>
```

- テーマ設定ファイル

  グローバルナビが表示されないモードでテーマが表示されるように、詳細画面のURLに対してHeadOnlyThemeBuilderを適用します。

  - <%サンプルプログラムディレクトリ%/src/main/conf/theme-head-only-path-config/im_workflow_tsfw_sample.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<theme-head-only-path-config xmlns="http://www.intra-mart.jp/theme/theme-head-only-path-config"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.intra-mart.jp/theme/theme-head-only-path-config theme-head-only-path-config.xsd">
  <path>/sample/im_workflow/tsfw/app/purchase/detail</path>
</theme-head-only-path-config>
```

- メッセージファイル
  - <%サンプルプログラムディレクトリ%/src/main/conf/message/sample/im_workflow_tgfw_sample/>

メッセージファイル 一覧

論理名	物理名	説明
メッセージファイル（デフォルト）	sample-message.properties	デフォルトメッセージが記述されています。
メッセージファイル（日本語）	sample-message_ja.properties	日本語メッセージが記述されています。
メッセージファイル（英語）	sample-message_en.properties	英語メッセージが記述されています。

論理名	物理名	説明
メッセージファイル（中国語）	sample-message_zh_CN.properties	中国語メッセージが記述されています。

共通（テナント環境セットアップ）

テナント環境セットアップ向けの資材です。

- テナント環境セットアップ 設定ファイル
  - <%サンプルプログラムディレクトリ%/src/main/conf/products/import/basic/im_workflow_tsfw_sample/>

テナント環境セットアップ 設定ファイル 一覧

論理名	物理名	説明
テナント環境セットアップ 設定ファイル	import-im_workflow_tsfw_sample-config-1.xml	テナント環境セットアップ時にDDL・拡張インポート処理が実行されるように指定しています。

- テナント環境セットアップ 拡張インポート処理クラス
  - <%サンプルプログラムディレクトリ%/src/main/java/jp/co/intra_mart/sample/im_workflow/tsfw/initialize/>

テナント環境セットアップ 拡張インポート処理クラス 一覧

論理名	物理名	説明
拡張インポート処理クラス	SampleFlowImporter.java	サンプルフロー「TERASOLUNA Server Framework開発モデル」をインポートする処理です。

- テナント環境セットアップ インポート資材
  - <%サンプルプログラムディレクトリ%/src/main/storage/system/products/import/basic/im_workflow_tsfw_sample/import/>

テナント環境セットアップ インポート資材 一覧

論理名	物理名	説明
インポート資材	im_workflow_tsfw.xml	サンプルフロー「TERASOLUNA Server Framework開発モデル」の定義ファイル（フロー定義・ルート定義・コンテンツ定義）です。

共通（データベース）

ユーザアプリケーションデータは、テナントデータベース上のテーブルで管理しています。 テーブルへのアクセスは、TERASOLUNA Server Frameworkが提供するMybatis3を利用しています。

Mybatis3の利用方法については、「 TERASOLUNA Server Framework for Java (5.x) プログラミングガイド 」-「 データベース 」を参照してください。

- ユーザアプリケーションデータ テーブル

テーブル sample_imw_tsfw_t_purchase

カラム名	説明
user_data_id	ユーザデータID
system_matter_id	システム案件ID
item_name	品名
item_amount	数量
item_price	金額
item_total	合計
item_comment	備考
end_flag	案件完了フラグ

カラム名	説明
archive_flag	案件アーカイブフラグ

- DDL

ユーザアプリケーションデータ テーブルを作成するためのDDLファイルです。
テナント環境セットアップ時に実行されます。

- <%サンプルプログラムディレクトリ%/src/main/storage/system/products/import/basic/im_workflow_tsfw_sample/import/>

DDL 一覧

論理名	物理名	説明
テナント環境セットアップ向けDDLファイル（デフォルト）	im_workflow_tsfw-sample-create_table-ddl.sql	テナントデータベースがPostgresqlの場合に利用されます。
テナント環境セットアップ向けDDLファイル（Oracle）	im_workflow_tsfw-sample-create_table-ddl_oracle.sql	テナントデータベースがOracleの場合に利用されます。
テナント環境セットアップ向けDDLファイル（SQLServer）	im_workflow_tsfw-sample-create_table-ddl_sqlserver.sql	テナントデータベースがSQLServerの場合に利用されます。

- エンティティクラス

テーブル sample_imw_tsfw_t_purchaseをモデル化したクラスです。
データベースから取得したレコードはリポジトリインタフェースを通して、エンティティクラスのインスタンスとして取得されます。

- <%サンプルプログラムディレクトリ%>/src/main/java/jp/co/intra_mart/sample/im_workflow/tsfw/domain/model/SampleImwTPurchase.java

```
public class SampleImwTPurchase implements Serializable {

    private static final long serialVersionUID = 5194970093421606223L;

    private String userDataId;

    private String archiveFlag;

    private String endFlag;

    private String itemAmount;

    private String itemComment;

    private String itemName;

    private String itemPrice;

    private String itemTotal;

    private String systemMatterId;

    public SampleImwTPurchase() {
    }

    public String getUserDataId() {
        return this.userDataId;
    }

    public void setUserDataId(final String userDataId) {
        this.userDataId = userDataId;
    }

    public String getArchiveFlag() {
        return this.archiveFlag;
    }

    public void setArchiveFlag(final String archiveFlag) {
        this.archiveFlag = archiveFlag;
    }

    public String getEndFlag() {
```

```
    public String getEndFlag() {
        return this.endFlag;
    }

    public void setEndFlag(final String endFlag) {
        this.endFlag = endFlag;
    }

    public String getItemAmount() {
        return this.itemAmount;
    }

    public void setItemAmount(final String itemAmount) {
        this.itemAmount = itemAmount;
    }

    public String getItemComment() {
        return this.itemComment;
    }

    public void setItemComment(final String itemComment) {
        this.itemComment = itemComment;
    }

    public String getItemName() {
        return this.itemName;
    }

    public void setItemName(final String itemName) {
        this.itemName = itemName;
    }

    public String getItemPrice() {
        return this.itemPrice;
    }

    public void setItemPrice(final String itemPrice) {
        this.itemPrice = itemPrice;
    }

    public String getItemTotal() {
        return this.itemTotal;
    }

    public void setItemTotal(final String itemTotal) {
        this.itemTotal = itemTotal;
    }

    public String getSystemMatterId() {
        return this.systemMatterId;
    }

    public void setSystemMatterId(final String systemMatterId) {
        this.systemMatterId = systemMatterId;
    }

}
```

- マッピングファイル

  実行するクエリ・クエリのパラメータ・戻り値のレコードをバインドするためのエンティティクラスを指定します。

  - <%サンプルプログラムディレクトリ%>/src/main/resources/jp/co/intra_mart/sample/im_workflow/tsfw/domain/repository/SampleImwTPurchaseRepository.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="jp.co.intra_mart.sample.im_workflow.tsfw.domain.repository.SampleImwTPurchaseRepository">

  <select id="select" parameterType="java.lang.String" resultType="SampleImwTPurchase">
    SELECT
     user_data_id as userDataId
    , system_matter_id as systemMatterId
    , item_name as itemName
    , item_amount as itemAmount
    , item_price as itemPrice
    , item_total as itemTotal
    , item_comment as itemComment
    , end_flag as endFlag
    , archive_flag as archiveFlag
    FROM sample_imw_tsfw_t_purchase
    WHERE user_data_id = #{userDataId}
  </select>

  <insert id="insert" parameterType="SampleImwTPurchase">
    INSERT
     INTO sample_imw_tsfw_t_purchase (user_data_id
       , system_matter_id
       , item_name
       , item_amount
       , item_price
       , item_total
       , item_comment
       , end_flag
       ,archive_flag)
    VALUES (#{userDataId}
       , #{systemMatterId,jdbcType=VARCHAR}
       , #{itemName,jdbcType=VARCHAR}
       , #{itemAmount,jdbcType=VARCHAR}
       , #{itemPrice,jdbcType=VARCHAR}
       , #{itemTotal,jdbcType=VARCHAR}
       , #{itemComment,jdbcType=VARCHAR}
       , #{endFlag,jdbcType=VARCHAR}
       , #{archiveFlag,jdbcType=VARCHAR})
  </insert>

  <update id="update" parameterType="SampleImwTPurchase">
    UPDATE
     sample_imw_tsfw_t_purchase
    SET system_matter_id = #{systemMatterId,jdbcType=VARCHAR}
    , item_name = #{itemName,jdbcType=VARCHAR}
    , item_amount = #{itemAmount,jdbcType=VARCHAR}
    , item_price = #{itemPrice,jdbcType=VARCHAR}
    , item_total = #{itemTotal,jdbcType=VARCHAR}
    , item_comment = #{itemComment,jdbcType=VARCHAR}
    , end_flag = #{endFlag,jdbcType=VARCHAR}
    , archive_flag = #{archiveFlag,jdbcType=VARCHAR}
    WHERE user_data_id = #{userDataId}
  </update>

  <delete id="delete" parameterType="java.lang.String">
    DELETE
    FROM sample_imw_tsfw_t_purchase
    WHERE user_data_id = #{userDataId}
  </delete>

</mapper>
```

- TypeAlias 設定ファイル

  マッピングファイルに指定するエンティティクラスのエイリアスを設定します。
  この設定を行うことで、マッピングファイルにエンティティクラスをエイリアスで指定することが可能です。

  - <%サンプルプログラムディレクトリ%/src/main/resources/META-INF/mybatis/mybatis-config.xml>

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
 "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

  <!-- See http://mybatis.github.io/mybatis-3/configuration.html#settings -->
  <settings>
    <setting name="mapUnderscoreToCamelCase" value="true" />
    <setting name="lazyLoadingEnabled" value="true" />
    <setting name="aggressiveLazyLoading" value="false" />
<!--
    <setting name="defaultExecutorType" value="REUSE" />
    <setting name="jdbcTypeForNull" value="NULL" />
    <setting name="proxyFactory" value="JAVASSIST" />
    <setting name="localCacheScope" value="STATEMENT" />
-->
  </settings>

  <typeAliases>
    <!-- マッピングファイルで利用するエンティティクラスのエイリアスを指定します。 -->
    <typeAlias alias="SampleImwTPurchase"
type="jp.co.intra_mart.sample.im_workflow.tsfw.domain.model.SampleImwTPurchase"/>
<!--
    <package name="xxxxxx.yyyyyy.zzzzzz.infra.mybatis.typehandler" />
-->
  </typeAliases>

  <typeHandlers>
<!--
    <package name="xxxxxx.yyyyyy.zzzzzz.infra.mybatis.typehandler" />
-->
  </typeHandlers>

</configuration>
```

> **注意**
>
> TypeAlias 設定ファイルはIM-Jugglingプロジェクトに設定ファイルとして配置します。

- リポジトリインタフェース

  マッピングファイルに指定したクエリをJavaプログラムから呼び出すためのインタフェースを定義します。
  リポジトリインタフェースのインスタンスは、MyBatis-Springによってインジェクションされます。

  - <%サンプルプログラムディレクトリ%>/src/main/java/jp/co/intra_mart/sample/im_workflow/tsfw/domain/repository/SampleImwTPurchaseRepository.java

```
package jp.co.intra_mart.sample.im_workflow.tsfw.domain.repository;

import jp.co.intra_mart.sample.im_workflow.tsfw.domain.model.SampleImwTPurchase;

public interface SampleImwTPurchaseRepository {

  public SampleImwTPurchase select(String userDataId);

  public void insert(SampleImwTPurchase entity);

  public void update(SampleImwTPurchase entity);

  public void delete(String userDataId);
}
```

ユーザコンテンツ画面

ユーザコンテンツ画面を表示するためのプログラムです。

- 画面
  - <%サンプルプログラムディレクトリ%/src/main/webapp/WEB-INF/views/sample/im_workflow/tsfw/app/purchase/>

JSPファイル 一覧

論理名	物理名	説明
申請画面	apply.jsp	クライアントタイプ「PC」の場合に、申請画面・申請（起票）画面・一時保存画面・再申請画面を表示するためのJSPファイルです。
承認画面	approve.jsp	クライアントタイプ「PC」の場合に、承認画面を表示するためのJSPファイルです。
確認画面	confirm.jsp	クライアントタイプ「PC」の場合に、確認画面を表示するためのJSPファイルです。
詳細画面	detail.jsp	クライアントタイプ「PC」「SP」の両方の場合に、詳細画面を表示するためのJSPファイルです。
申請画面（スマートフォン）	apply_sp.jsp	クライアントタイプ「SP」の場合に、申請画面・申請（起票）画面・一時保存画面・再申請画面を表示するためのJSPファイルです。
承認画面（スマートフォン）	approve_sp.jsp	クライアントタイプ「SP」の場合に、承認画面を表示するためのJSPファイルです。
確認画面（スマートフォン）	confirm_sp.jsp	クライアントタイプ「SP」の場合に、確認画面を表示するためのJSPファイルです。

- Controllerクラス
  - <%サンプルプログラムディレクトリ%/src/main/java/jp/co/intra_mart/sample/im_workflow/tsfw/app/purchase/>

Controllerクラス 一覧

論理名	物理名	説明
クライアントタイプ「PC」向けControllerクラス	PurchaseController.java	クライアントタイプ「PC」にて IM-Workflow を利用した場合に呼び出されます。
クライアントタイプ「SP」向けControllerクラス	PurchaseSmartphoneController.java	クライアントタイプ「SP」にて IM-Workflow を利用した場合に呼び出されます。

- Formクラス
  - <%サンプルプログラムディレクトリ%/src/main/java/jp/co/intra_mart/sample/im_workflow/tsfw/app/purchase/>

Formクラス 一覧

論理名	物理名	説明
Formクラス（ユーザアプリケーションデータ）	PurchaseForm.java	JSPにリクエストパラメータ（ユーザコンテンツ画面で入力された情報）をバインドします。
Formクラス（ワークフローデータ）	WorkflowForm.java	JSPにリクエストパラメータ（ワークフローのシステムパラメータ情報）をバインドします。

ユーザプログラム

ユーザプログラムを構成するServiceクラス群です。

- Serviceクラス
  - <%サンプルプログラムディレクトリ%/src/main/java/jp/co/intra_mart/sample/im_workflow/tsfw/domain/service/>

ユーザプログラム 一覧

論理名	物理名	説明
アクション処理プログラム	ActionProcess.java	アクション処理 サービスを呼び出します。

論理名	物理名	説明
アクション処理サービス	ActionProcessService.java	ユーザアプリケーションデータに対してバリデーション・データベースへの永続化・案件プロパティへの登録を行います。
案件終了処理プログラム	MatterEndProcess.java	案件終了処理 サービスを呼び出します。
案件終了処理サービス	MatterEndProcessService.java	ユーザアプリケーションデータ テーブル sample_imw_tsfw_t_purchaseの案件終了フラグを更新します。
未完了案件削除処理リスナー	WorkflowActvMatterDeleteListener.java	未完了案件削除処理リスナー サービスを呼び出します。
未完了案件削除処理リスナー サービス	WorkflowActvMatterDeleteListenerService.java	未完了案件が削除された際に、該当するユーザアプリケーションデータを削除します。
完了案件削除処理リスナー	WorkflowCplMatterDeleteListener.java	完了案件削除処理リスナー サービスを呼び出します。
完了案件削除処理リスナー サービス	WorkflowCplMatterDeleteListenerService.java	完了案件が削除された際に、該当するユーザアプリケーションデータを削除します。
過去案件削除処理リスナー	WorkflowArcMatterDeleteListener.java	過去案件削除処理リスナー サービスを呼び出します。
過去案件削除処理リスナー サービス	WorkflowArcMatterDeleteListenerService.java	過去案件が削除された際に、該当するユーザアプリケーションデータを削除します。
案件退避処理リスナー	WorkflowMatterArchiveListener.java	案件退避処理リスナー サービスを呼び出します。
案件退避処理リスナー サービス	WorkflowMatterArchiveListenerService.java	ユーザアプリケーションデータ テーブル sample_imw_tsfw_t_purchaseのアーカイブフラグを更新します。