目次

改訂情報

変更年月日	変更内容
2012-10-01	初版
2012-11-30	第2版　下記を追加しました ■ 「モジュール開発の基本機能」のページに「プログラム開発」を追加
2013-04-01	第3版　下記を追加・変更しました ■ 「*e Builder での開発の流れ*」にコラム追加しました。 ■ 「*プロジェクトの設定*」にコラム追加しました。 ■ 目次に「*module.xml*」を追加しました。 ■ 「*デバッグ（スクリプト開発）*」のデバッグ方法の記述を修正しました。 ■ 「*デバッグ（Java開発）*」のデバッグ方法の記述を修正しました。 ■ 「*デバッグ（Java開発）*」のデバッグ方法の記述を修正しました。 ■ 目次に「*業務スケルトンで利用できる機能*」を追加しました。
2013-06-20	第4版　下記を追加・変更しました ■ 「*業務スケルトンの実行*」にコラム追加しました。 ■ 目次に「*利用できる業務スケルトンテンプレート一覧*」を追加しました。 ■ 「*業務スケルトン 変更履歴*」を追加しました。
2013-08-01	第5版　下記を追加・変更しました ■ 「*業務スケルトンテンプレート利用ガイド*」を追加しました。 ■ 「*利用できる業務スケルトンテンプレート一覧*」に記述を追加しました。
2013-10-01	第6版　下記を追加・変更しました ■ 「*TERASOLUNA Server Framework for Java (5.x) for Accel Platform 開発で利用できる機能*」を追加しました。 ■ 「*利用できる業務スケルトンテンプレート一覧*」に TERASOLUNA Global Framework 汎用画面テンプレート の項を追加しました。
2014-01-01	第7版　下記を追加・変更しました。 ■ 「*利用できる業務スケルトンテンプレート一覧*」の TERASOLUNA Global Framework 汎用画面テンプレート の項を修正しました。 ■ 「*利用できる業務スケルトンテンプレート一覧*」に TERASOLUNA Global Framework ポートレットテンプレート の項を追加しました。 ■ 「*プロジェクトの設定*」にプロジェクトとデバッグサーバの紐付けの項を追加しました。 ■ 「*e Builder での開発の流れ*」の「.ebuilder-export.xml」に表示される.ebuilder-export.xmlの内容を「2013 Winter」の内容に更新しました。
2014-04-01	第8版　下記を追加・変更しました。 ■ 「*module.xml*」に注意を追加しました。
2014-08-01	第9版　下記を追加・変更しました ■ 「*モジュール開発の基本機能*」ー「*module.xml*」にショートモジュールIDの確認方法へのリンクを追加 ■ 「*利用できる業務スケルトンテンプレート一覧*」に TERASOLUNA Global Framework リポジトリテンプレート の項を追加しました。
2014-12-01	第10版　下記を追加・変更しました。 ■ 「*利用できる業務スケルトンテンプレート一覧*」の Seasar SAStruts 汎用画面テンプレートにCSRF対策の設定について追記しました。 ■ 「*利用できる業務スケルトンテンプレート一覧*」の TERASOLUNA Global Framework 汎用画面テンプレートにCSRF対策の設定について追記しました。

変更年月日	変更内容
2015-04-01	第11版　下記を追加・変更しました。 ■ 「*利用できる業務スケルトンテンプレート一覧*」の Seasar S2JDBCテンプレートにDtoとService生成について追記しました。 ■ 「*Seasar S2JDBC DtoとService生成*」を追記しました。 ■ 「*e Builder での開発の流れ*」にwarファイル展開時のリソースの配置先に関する追記を行いました。 ■ 「*プロジェクトの設定*」にstorageフォルダのデプロイ先のパスの変更方法について追記しました。
2015-12-01	第12版 下記を追加・変更しました。 ■ TERASOLUNA Global FrameworkをTERASOLUNA Server Framework for Java (5.x)に変更しました。 ■ 「*TERASOLUNA Server Framework for Java (5.x) 汎用画面テンプレート*」を「TERASOLUNA Global Framework 汎用画面テンプレート」から変更しました。 ■ 「*TERASOLUNA Server Framework for Java (5.x) ポートレットテンプレート*」を「TERASOLUNA Global Framework ポートレットテンプレート」から変更しました。 ■ 「*TERASOLUNA Server Framework for Java (5.x) for Accel Platform 開発で利用できる機能*」の TERASOLUNA Server Framework for Java (5.x)を TERASOLUNA Global Framework から変更しました。 ■ 「*デバッグ（Java開発）*」のTERASOLUNA Server Framework for Java (5.x)を TERASOLUNA Global Framework から変更しました。 ■ 「*利用できる業務スケルトンテンプレート一覧*」にTERASOLUNA Server Framework for Java (5.x)-リポジトリ-MyBatis3を追記しました。 ■ 「*利用できる業務スケルトンテンプレート一覧*」にスマートフォン画面のベースにしているjQuery Mobileのバージョンについて追記しました。 ■ 各開発フレームワークのデバッグ機能に、デバッグサーバ実行時に Server プロジェクトを閉じないように注意書きを行いました。 ■ 「*CUIでのモジュール作成方法について*」を追加しました。
2016-12-01	第13版 下記を追加・変更しました。 ■ 「*e Builder での開発の流れ*」の「.ebuilder-export.xml」に表示される.ebuilder-export.xmlの内容を「2016 Winter」の内容に更新しました。
2018-08-01	第14版 下記を追加・変更しました。 ■ 「*module.xml*」の「*モジュール・エディタ*」に表示される概要タブの注意事項の記述を修正しました。
2019-12-01	第15版 下記を追加・変更しました。 ■ 以下の設定ファイルの文書型定義に記述されているURLスキームを http から https に変更 　■ 「*利用できる業務スケルトンテンプレート一覧*」の 「*CSRF対策を有効化させるために（Ver 2.0.2以降）*」 の app.dicon, customizer.dicon
2023-10-01	第16版 下記を追加・変更しました。 ■ 基盤となる Eclipse のバージョン変更に伴い、各手順の画像の差し替えを行いました。 ■ 「*モジュール・プロジェクト作成*」の IM-Jugglingのプラグインに関するコラムを削除しました。 ■ 「*実行構成の設定*」を追加しました。 ■ 「*モジュール・プロジェクト作成*」の「*手順*」に、新規プロジェクト作成に関するコラムを追記しました。

intra-mart e Builder for Accel Platform とは

ここでは e Builder の概要や基本的な知識について紹介します。

概要

e Builder は、intra-mart Accel Platform 上で動く業務アプリケーションの開発を支援するためのツールです。

e Builder では、利用者が独自でアプリケーションを開発する際のプロジェクトの作成から、最終的な成果物の生成までを支援する機能を提供します。

対象読者

- 本書は e Builder を利用して、アプリケーションの開発を行おうとする開発者の方を対象にしています。

e Builder 提供機能

- *モジュール開発の基本機能*
- *スクリプト開発で利用できる機能*
- *im-JavaEE フレームワーク開発で利用できる機能*
- *SAStruts + S2JDBC フレームワーク開発で利用できる機能*
- *TERASOLUNA Server Framework for Java (5.x) for Accel Platform 開発で利用できる機能*

e Builder での開発の流れ

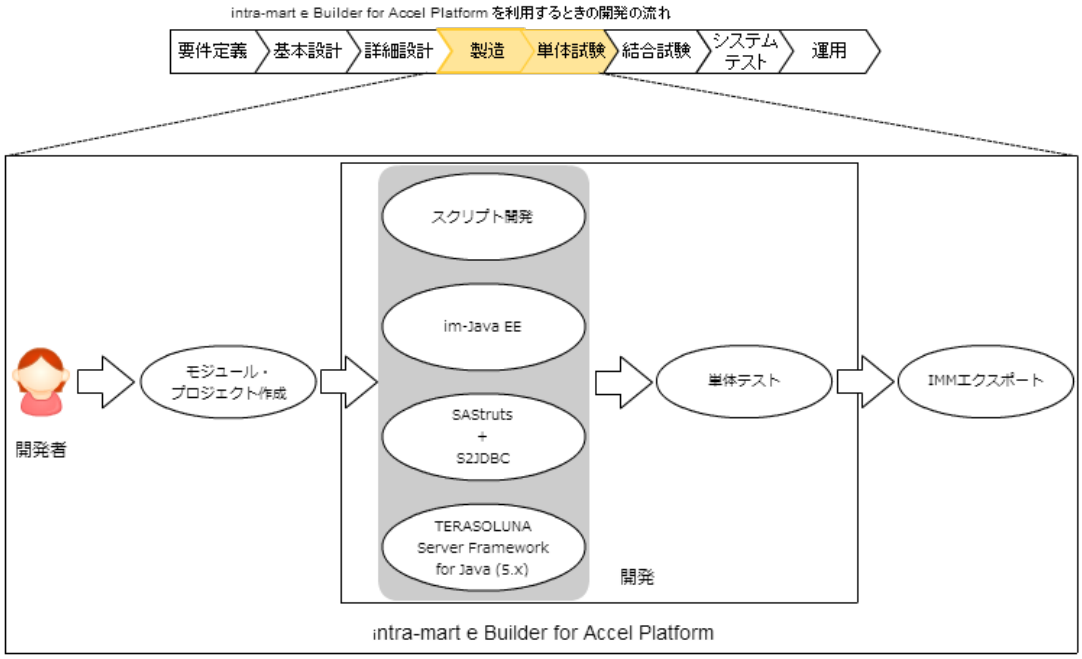
概要

本項では e Builder を利用した際に、アプリケーションを開発する流れについて説明します。

開発の流れ

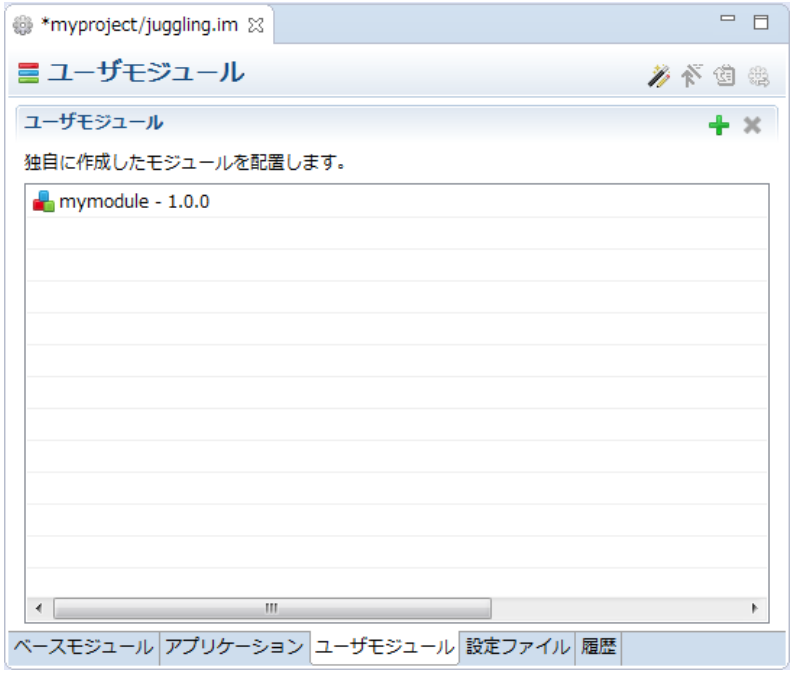
e Builder が提供する機能は、アプリケーション開発における「製造」から「ユニットテスト」工程を支援します。
具体的には、intra-mart Accel Platform に組込みができる「ユーザモジュール」を作成する機能から構成されます。
開発者が e Builder を利用する一連の流れは以下のとおりです。

1. ユーザモジュールを作成するための「モジュール・プロジェクト」を作成します。
2. モジュール・プロジェクトに対して各開発モデルを選択してプログラムを開発します。
3. 開発の完了後、ユーザモジュールとしてIMMエクスポート（immファイルの出力）を行います。



作成されたユーザモジュールは、結合試験のフェーズにおいてIM-Jugglingでwarを作成する際に設定を行います。

1. Jugglingプロジェクト内にあるjuggling.imをIM-Juggling Editorで開きます。
2. 「ユーザモジュール」タブを選択し、右上にある「+」のアイコンをクリックし、出力したimmファイルを選択します。
3. ユーザモジュールを設定後、必要に応じてユーザモジュールに対する依存モジュールを設定し、エラーメッセージが表示されないようにします。

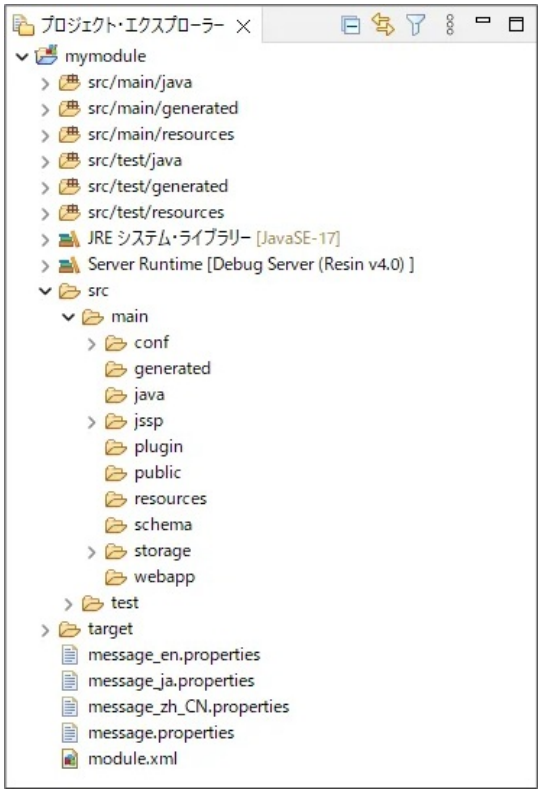


> **コラム**
>
> ユーザモジュールがSAStruts + S2JDBC フレームワーク開発を利用している場合、Jugglingプロジェクト内のclasses/convention.dicon等各diconファイルを適切に設定しておいてください。

上記の操作でwarにユーザモジュールを含めるための設定は完了です。
最終的にwarファイルを作成する方法に関しては「intra-mart Accel Platform / セットアップガイド」の「WARファイルの作成」を参照してください。

モジュール・プロジェクトの構成

e Builder で作成するモジュール・プロジェクトの構成は以下のとおりです。
下の画像は作成後のモジュール・プロジェクトをパッケージ・エクスプローラーで表示した場合の図です。



開発者は作成したアプリケーションのリソースを以下の場所に配置して、モジュール・プロジェクトを開発していきます。
モジュール・プロジェクトでは各用途に対して、パーティショニングの利用を以下の想定しています。

パーティショニング	説明
main	そのモジュールの主たるコードやリソースを配置します。
test	テストを行うためのコード・リソースを配置します。

また、モジュール・プロジェクトでは各パーティショニングに対して、それぞれコードやリソースを配置するためのディレクトリを用意してあります。
各ディレクトリに対して、細かいプログラムコードやリソースを以下の表に沿って配置を行っていきます。

ディレクトリ	想定用途	対象例	war展開後の配置場所
generated	自動生成されたJavaファイルを格納します。	*.java	WEB-INF/libのjarファイル内
java	Javaファイルを格納します。	*.java	WEB-INF/libのjarファイル内
resources	Javaファイル以外でプログラム中に利用するファイルを格納します。	*.properties、*.xml	WEB-INF/libのjarファイル内
conf	モジュールの動作を制御する設定ファイルを格納します。	*.properties、*.xml	WEB-INF/conf配下
jssp	スクリプト開発用のソースを格納します。	*.html、*.js	WEB-INF/jssp配下
plugin	プラグインとして利用する設定ファイルを格納します。	*.properties、*.xml	WEB-INF/plugin配下
public	Webコンテンツで静的コンテンツを格納します。 **（補足）**	*.html、*.css、*.swf 等	WARを展開したフォルダ配下
schema	設定ファイルに関するXMLスキーマファイルを格納します。	*.xsd	WEB-INF/schema配下
storage/public	StorageのPublicStorageにて管理されるファイルを格納します。	*.* storageに格納する全ファイル	%ストレージのパス%/storage/public配下
storage/system	StorageのSystemStorageにて管理されるファイルを格納します。	*.* storageに格納する全ファイル	%ストレージのパス%/storage/system配下
webapp	Webコンテンツで動的コンテンツを格納します。	*.jsp、WEB-INF/に配置するファイル diconファイル等	WARを展開したフォルダ配下

（補足1） Juglingビルドウィザードで、「静的ファイル出力」を行った場合に出力対象になるディレクトリです。

コラム

- サードパーティのライブラリを利用したい場合、src/main/webappの配下に
  「WEB-INF/lib」という階層でフォルダを作成し、そこにライブラリを配置してください。
  その後、プロジェクトのビルドを行った後、プロジェクトを一度クローズして開きなおすことによって
  配置したライブラリがクラスパスに通されます。
- e Builder プロジェクトでは、Storageのフォルダは、PublicStorageとSystemStorageをサポートします。
  Storageの仕様については、各フレームワークのプログラミングガイドの「Storage」を参照してください。
- e Builder で作成されたプロジェクトには、ユーザ定義モジュールを作成するために必要な2つのファイルを配置しています。
  もしこれらのファイルがなくなった場合、e Builderで支援する機能の動作に影響を及ぼします。
  下記のリンクにはそれぞれのファイルがなくなった場合の対処方法について記述しています。
  module.xml
  .ebuilder-export.xml

.ebuilder-export.xml

プロジェクトから .ebuilder-export.xml を削除してしまった場合、以下の手順で復旧してください。
また、このファイルの有無の確認はナビゲーター・ビューで表示することによって確認できます。

1. プロジェクト直下に「 .ebuilder-export.xml 」を作成してください。
2. 以下のフォーマットの記述を .ebuilder-export.xml にコピーしてください。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
```

```
<groupId>${group_id}</groupId>
<artifactId>${artifact_id}</artifactId>
<packaging>im_module</packaging>
<version>${version}</version>

<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<build>
 <plugins>
  <!-- configure for lifecycle -->
  <plugin>
   <groupId>jp.co.intra_mart.maven</groupId>
   <artifactId>lifecycle-plugin</artifactId>
   <version>1.0.0</version>
   <extensions>true</extensions>
  </plugin>
  <!-- configure for adding generated source folder -->
  <plugin>
   <groupId>org.codehaus.mojo</groupId>
   <artifactId>build-helper-maven-plugin</artifactId>
   <version>1.7</version>
   <executions>
    <execution>
     <id>add-source</id>
     <phase>generate-sources</phase>
     <goals>
      <goal>add-source</goal>
     </goals>
     <configuration>
      <sources>
       <source>${basedir}/src/main/generated</source>
      </sources>
     </configuration>
    </execution>
    <execution>
     <id>add-test-source</id>
     <phase>generate-sources</phase>
     <goals>
      <goal>add-test-source</goal>
     </goals>
     <configuration>
      <sources>
       <source>${basedir}/src/test/generated</source>
      </sources>
     </configuration>
    </execution>
   </executions>
  </plugin>
  <!-- configure for compiler -->
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <version>3.1</version>
   <configuration>
    <source>1.8</source>
    <target>1.8</target>
    <encoding>${project.reporting.outputEncoding}</encoding>
    <useIncrementalCompilation>false</useIncrementalCompilation>
   </configuration>
  </plugin>
  <!-- configure maven-jar-plugin version for mac -->
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-jar-plugin</artifactId>
   <version>2.4</version>
  </plugin>
  <plugin>
   <groupId>jp.co.intra_mart.maven</groupId>
   <artifactId>sample-build-plugin</artifactId>
   <version>1.0.0</version>
   <dependencies>
    <dependency>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-jar-plugin</artifactId>
```

```
      <version>2.4</version>
     </dependency>
    </dependencies>
   </plugin>
   <plugin>
    <groupId>jp.co.intra_mart.maven</groupId>
    <artifactId>archiver-plugin</artifactId>
    <version>1.0.0</version>
    <dependencies>
     <dependency>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-jar-plugin</artifactId>
     <version>2.4</version>
    </dependency>
    </dependencies>
   </plugin>
  </plugins>
 </build>

</project>
```

3. 以下の文字列を、プロジェクト作成時に入力した内容に合わせて記述を行ってください。

${group_id}	プロジェクトのグループID
${artifact_id}	プロジェクトのショートモジュールID（プロジェクト名）
${version}	プロジェクトのバージョン番号

モジュール開発の基本機能

ここでは e Builder におけるモジュール・プロジェクト作成の共通機能に関して説明します。
モジュール開発の基本機能とは、モジュールを開発していく上で、開発に利用するフレームワークを限定せずに利用できる機能の総称です。

モジュール・プロジェクト作成

概要

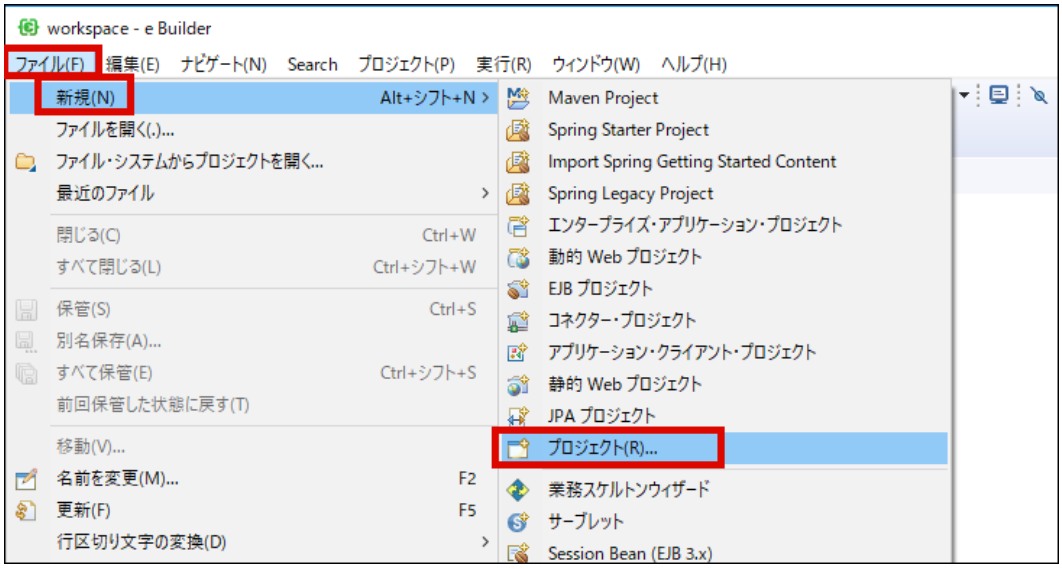
- 本項では、e Builder でモジュール・プロジェクトを作成する方法に関して説明します。

前提条件

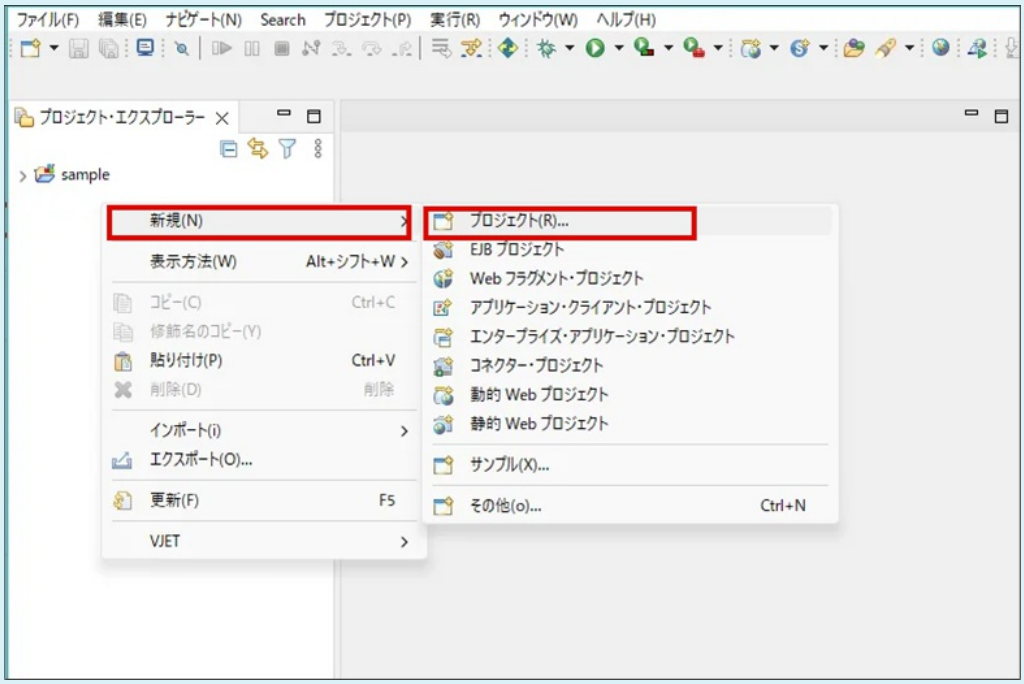
- e Builder をセットアップガイドに基づいてセットアップが完了していること
- e Builderを起動していること

手順

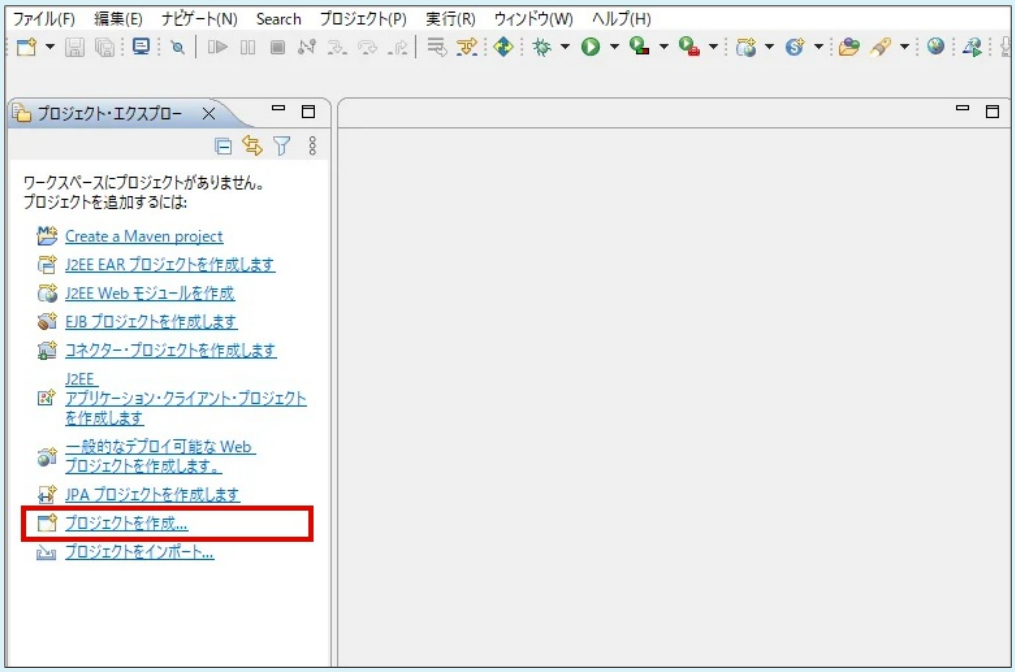
1. e Builder のメニュー・バーから、「ファイル」→「新規」→「プロジェクト」を選択します。



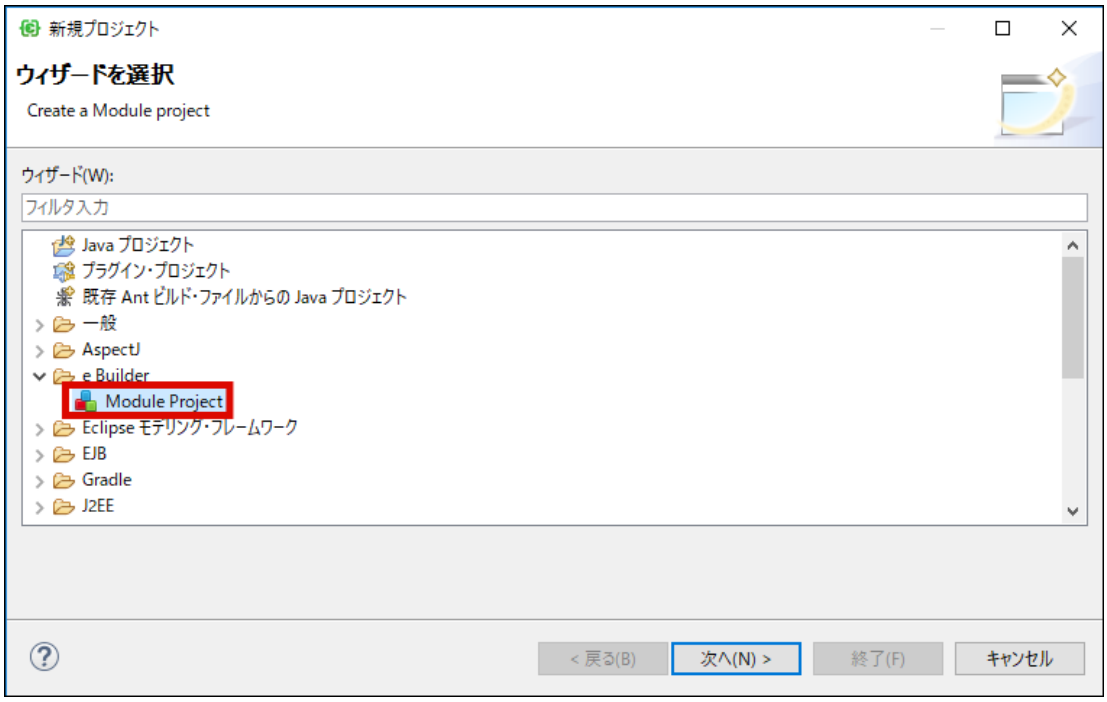
> **コラム**
>
> 「プロジェクト・エクスプローラー」の任意の場所を右クリックし、「新規」 → 「プロジェクト」を選択することでも、新規プロジェクト作成ウィザードを呼び出せます。



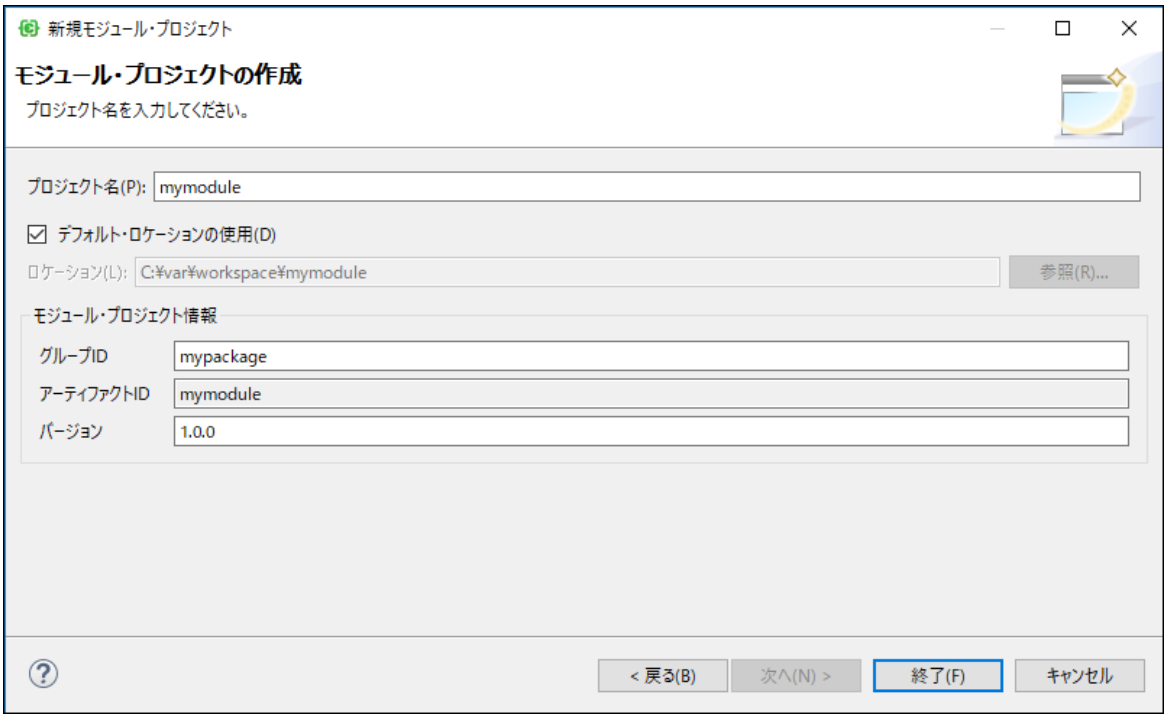
プロジェクトが 1 件もない場合は、「プロジェクトを作成」をクリックすることで新規プロジェクト作成ウィザードを呼び出せます。



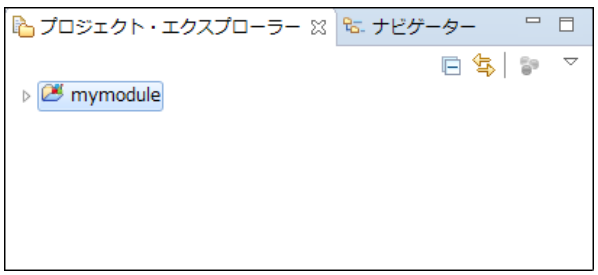
2. 新規プロジェクト作成ウィザードから「 e Builder 」→「 Module Project 」を選択し、「次へ」を押下します。



3. モジュール・プロジェクト作成ウィザードにて、以下の情報を入力し、完了を押下します。
   - プロジェクト名：プロジェクト名を入力します。プロジェクト名は「アーティファクトID」と自動的に同期をとります。
   - グループID（※1）：モジュール・プロジェクトに設定するグループIDを設定します。
   - アーティファクトID（※2）：モジュール・プロジェクトに設定するショートモジュールIDを設定します。入力不可。
   - バージョン（※3）：モジュール・プロジェクトのバージョン番号を設定します。



4. ワークスペース上にプロジェクトが作成されます。



> **コラム**
>
> ※1：グループIDで入力できる文字は半角英数字、.（ピリオド）、_（アンダースコア）で、以下のような命名規則である必要があります。
>
> ex. foo.bar_xxx.yyy1
>
> ※2： アーティファクトIDで入力できる文字は半角英数字、_（アンダースコア）です。また、アーティファクトIDはプロジェクト名と統一しないといけません。
>
> ※3：バージョンに入力できる文字は数字、.（ピリオド）で、以下のような命名規則である必要があります。
>
> ex. 1.0.0

プロジェクトの設定

項目

概要

- 本項では e Builder で作成したプロジェクトに対して、開発環境と紐づける方法について説明します。
  具体的には、以下の作業を実施します。

  1. プロジェクトとデバッグサーバの紐付け
  2. プロジェクトにサーバの情報を設定
  3. 開発で利用するライブラリの設定方法
     3.1 サーバ・ランタイムの追加
     3.2 Accel Platform Library の設定
     3.3 サードパーティライブラリ の設定

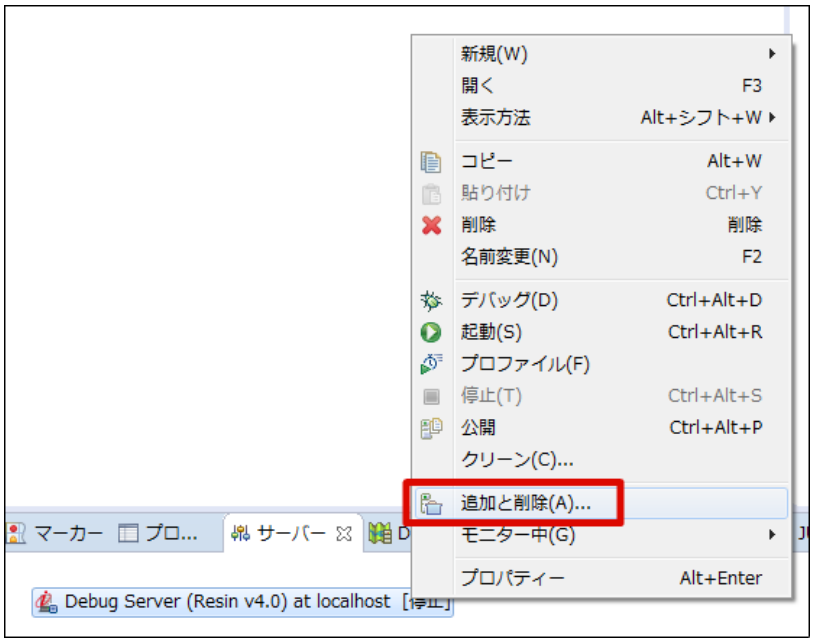
前提条件

- e Builder でプロジェクトの作成が完了していること。
- 本項では Resin 上に imart.war を展開したという前提で説明を行います。

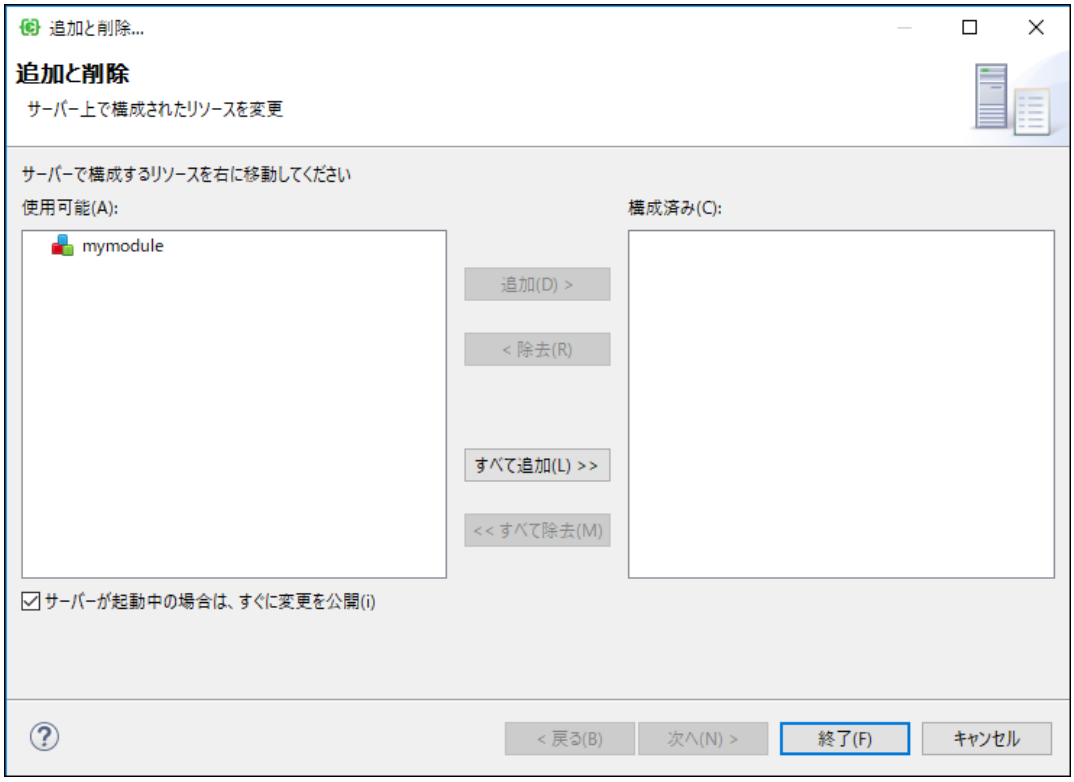
プロジェクトとデバッグサーバの紐付け

スクリプト開発やSAStrutsのデバッグを行う際、ユーザは利用するデバッグサーバに対してプロジェクトを紐付ける必要があります。

1. 「サーバ」ビューで設定したサーバにカーソルを合わせて右クリックし、「追加と削除」を選択します。

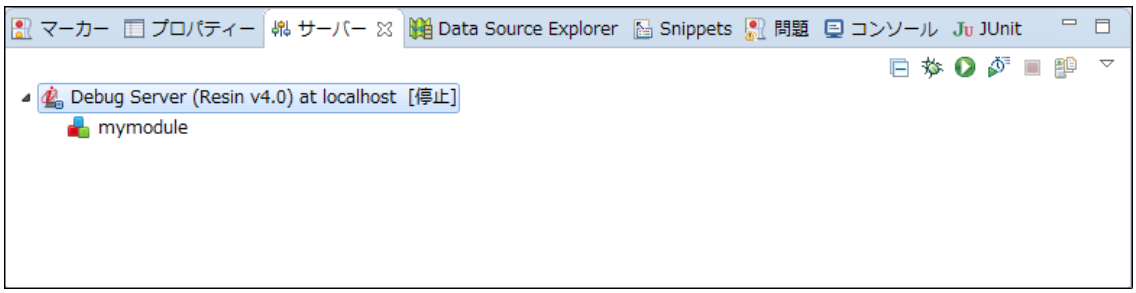


2. 「追加と削除」の画面で、サーバに紐付けたいプロジェクトを選択し、「追加」ボタンを押下して「使用可能」から「構成済み」へと移動させます。



3. プロジェクトを移動させた後、「終了」ボタンを押下します。

4. 「サーバ」ビューで「追加と削除」を行ったサーバをクリックし、下にプロジェクト名が出てくれば紐付けは完了です。



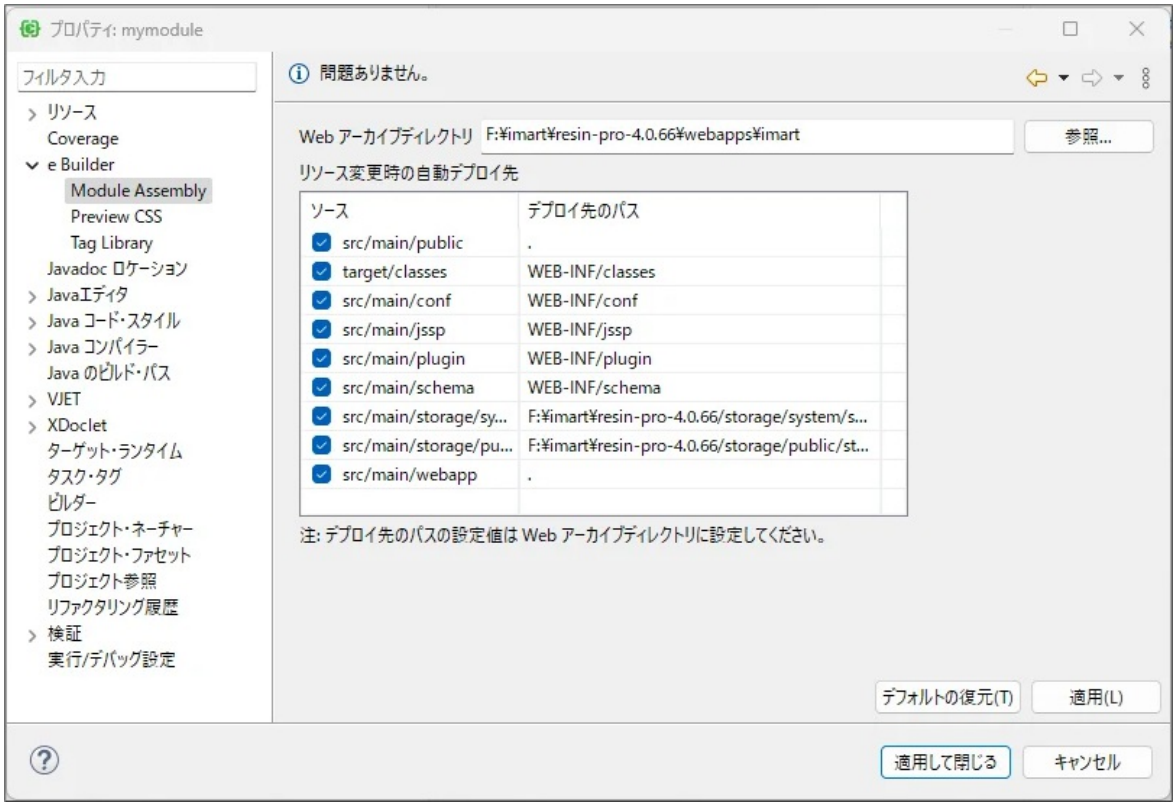
プロジェクトにサーバの情報を設定

- プロジェクト中のソースのデバッグを行ったり、

intra-mart Accel Platform の API を利用する場合、プロジェクトに対して開発環境用のサーバを紐づける必要があります。
その手順は以下のとおりです。

1. 対象となるプロジェクトに対して右クリックし「プロパティ」を選択します。

2. プロジェクトの設定ウィザードから「 e Builder 」→「 Module Assembly 」を選択します。
3. 「 Web アーカイブ ディレクトリ」に、開発環境用の Resin で war を展開してできたフォルダを指定します。
   - 下記の画像は、 imart.war を展開した場合にできた imart フォルダを選択しています。



4. プロジェクトのソースでデプロイの対象とするフォルダを選択します。
5. 「適用」を押下します。

> **コラム**
>
> e Builder で作成されたプロジェクトには、ユーザ定義モジュールを作成するために必要な <module.xml> ファイルを配置しています。
> もしこのファイルがない状態でWebアーカイブディレクトリを設定しようとした場合、 <module.xml> がないとエラーが発生します。
> 下記のリンクには <module.xml>ファイルがなくなった場合の対処方法について記述しています。
> *module.xml*

- サーバとの紐づけを行うことにより、プロジェクトで作成したソースを開発環境用のサーバに自動的にデプロイします。
  デプロイ元のフォルダとデプロイ先のソースの関連は、「リソース変更時の自動デプロイ先」を参照してください。

> **コラム**
>
> intra-mart e Builder for Accel Platform 2015 Spring よりstorageのパスの変更ができるようになりました。
> 変更方法
> 1. パスを変更したいstorageの「ソース」カラムを選択し、フォーカスを当てます。
> 2. フォーカスの当たったstorageの「デプロイ先のパス」をクリックします。
> 3. デプロイ先のパスの変更をします。
> 4. 変更終了後、「適用」ボタンを押下し、設定を反映させます。

開発で利用するライブラリの設定方法

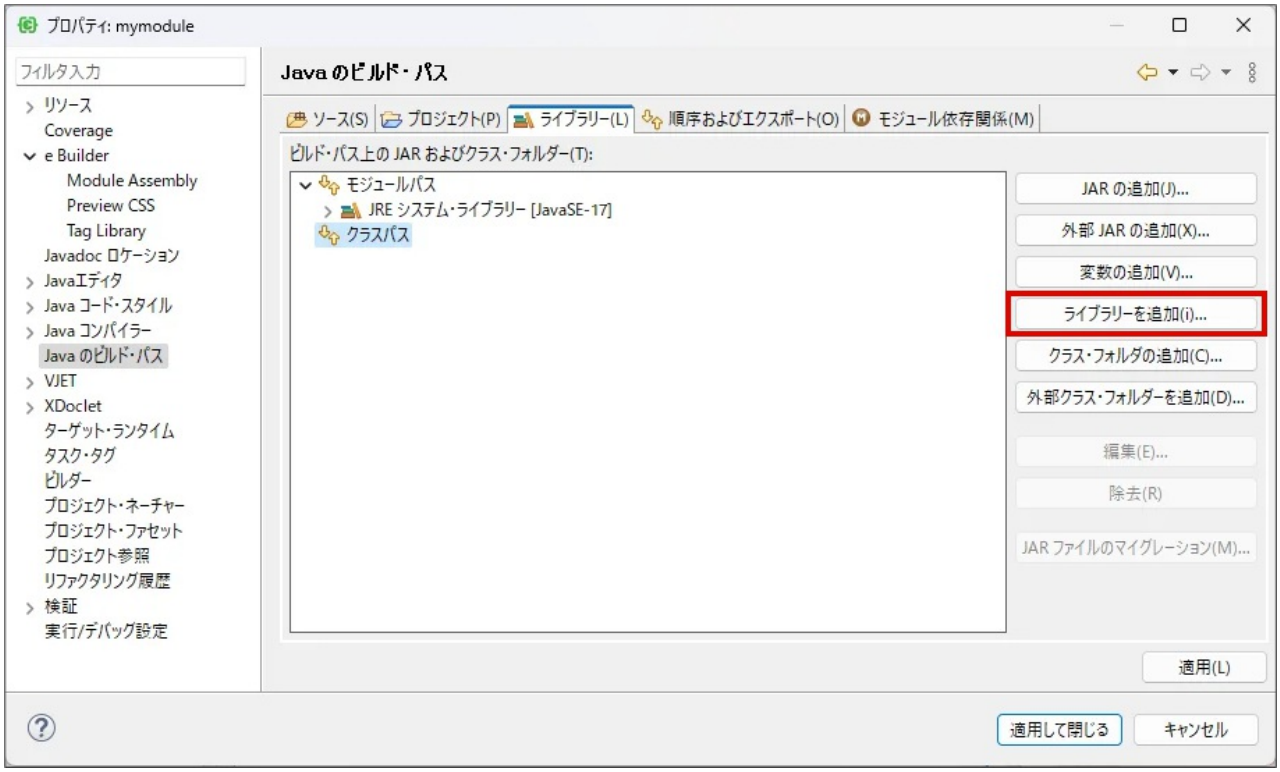
- Javaの開発を行う際に利用するライブラリの設定を行います。
  ライブラリを設定する方法は3つあり、使用したいライブラリに合わせてプロジェクトの設定を行います。
  1. サーバ・ランタイムの追加
     JavaEE などで開発する場合、セッションに関するクラスなどを利用する場合
  2. Accel Platform Library の設定
     intra-mart で提供するAPIを各フレームワークで利用する場合
  3. サードパーティライブラリ の設定
     任意のサードパーティ製ライブラリを利用したい場合

サーバ・ランタイムの追加

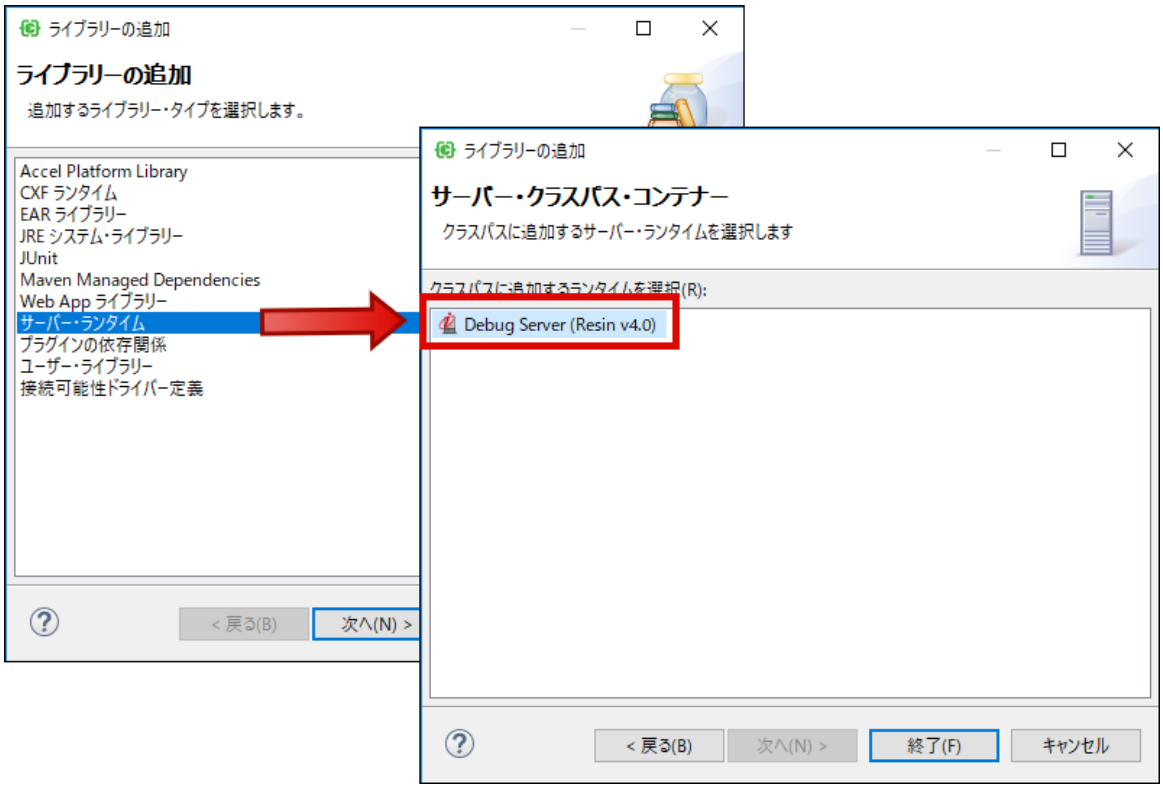
- JavaEE などで開発する場合、セッションに関するクラスなどを利用するためには、
  サーバのライブラリが登録されている必要があります。
  本項では個別にサーバのライブラリを登録する方法について説明します。
  なお、この操作を行う際にあらかじめサーバ・ビューに開発で利用するデバッグ サーバを登録しておく必要があります。
  その方法は「 intra-mart Accel Platform / セットアップガイド 」を参照してください。

> **コラム**
>
> intra-mart e Builder for Accel Platform 2013 Spring 以降、モジュールプロジェクト作成時にサーバ・ランタイムおよび Accel Platform Library が
> モジュールプロジェクトに自動的に追加されるようになりました。
> 個別にライブラリを設定する必要がある場合のみ、以下手順を参考に設定してください。

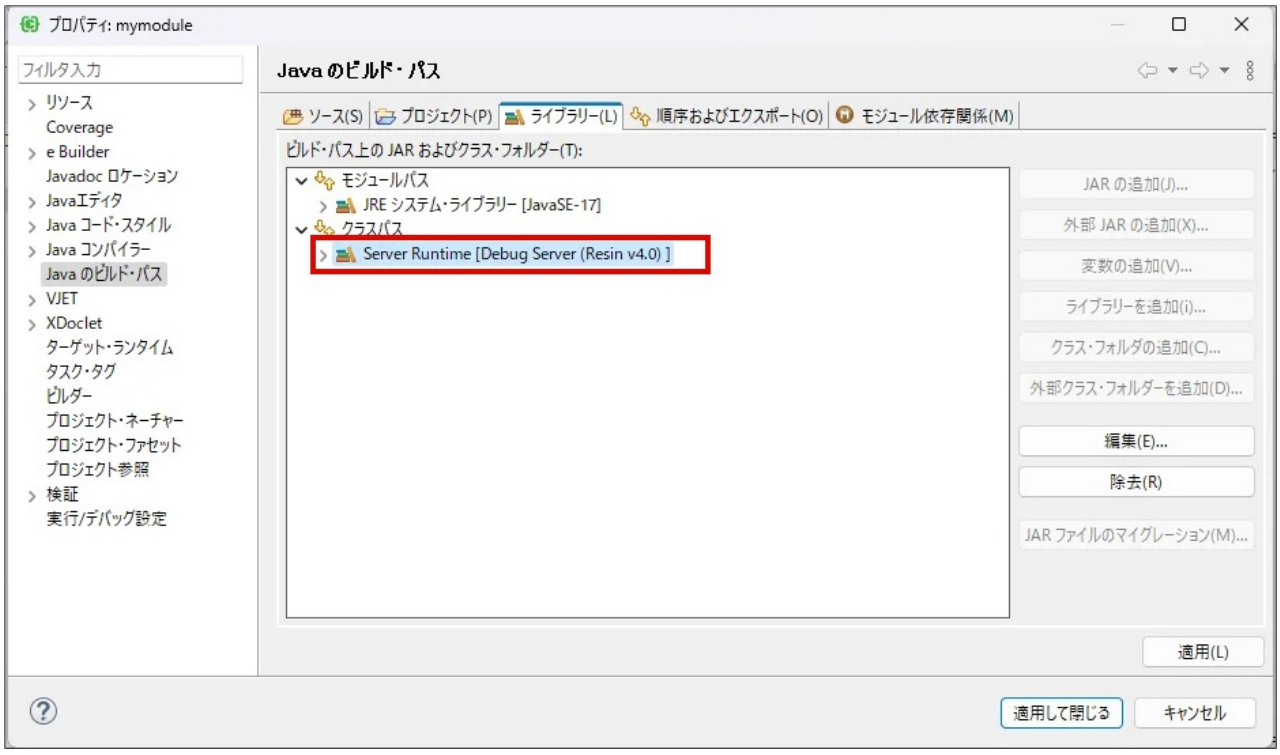
1. プロジェクトの設定ウィザードから、「 Javaのビルド・パス」を選択します。
2. 「ライブラリ」タグを選択し、「ライブラリの追加」ボタンを押下します。



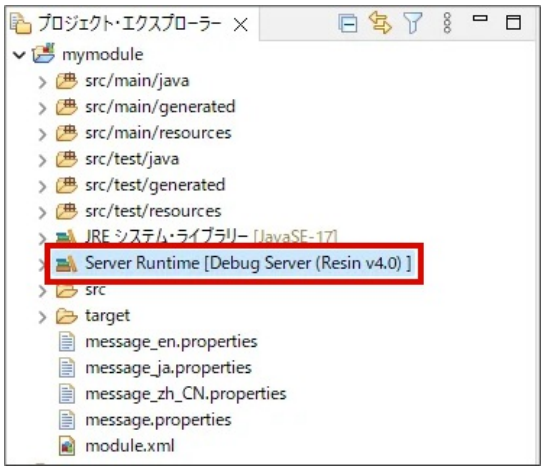
3. ライブラリの選択で「サーバ・ランタイム」→「Debug Server(Resin v4.0) 」を選択し、「終了」を押下します。



4. ライブラリの一覧にDebug Serverが追加されますので、「 OK 」ボタンを押下して、プロジェクトにライブラリを反映させます。



5. プロジェクト配下に Server Runtime のリンクができます。

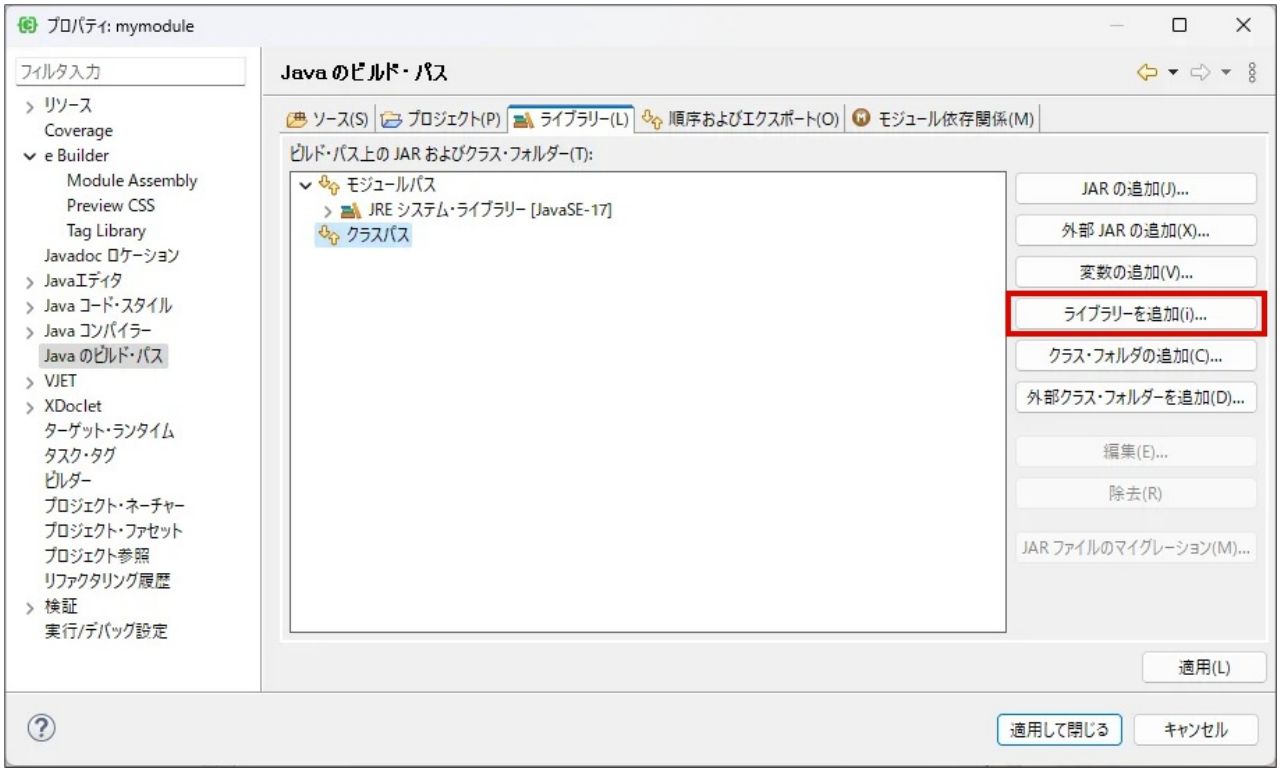


Accel Platform Library の設定

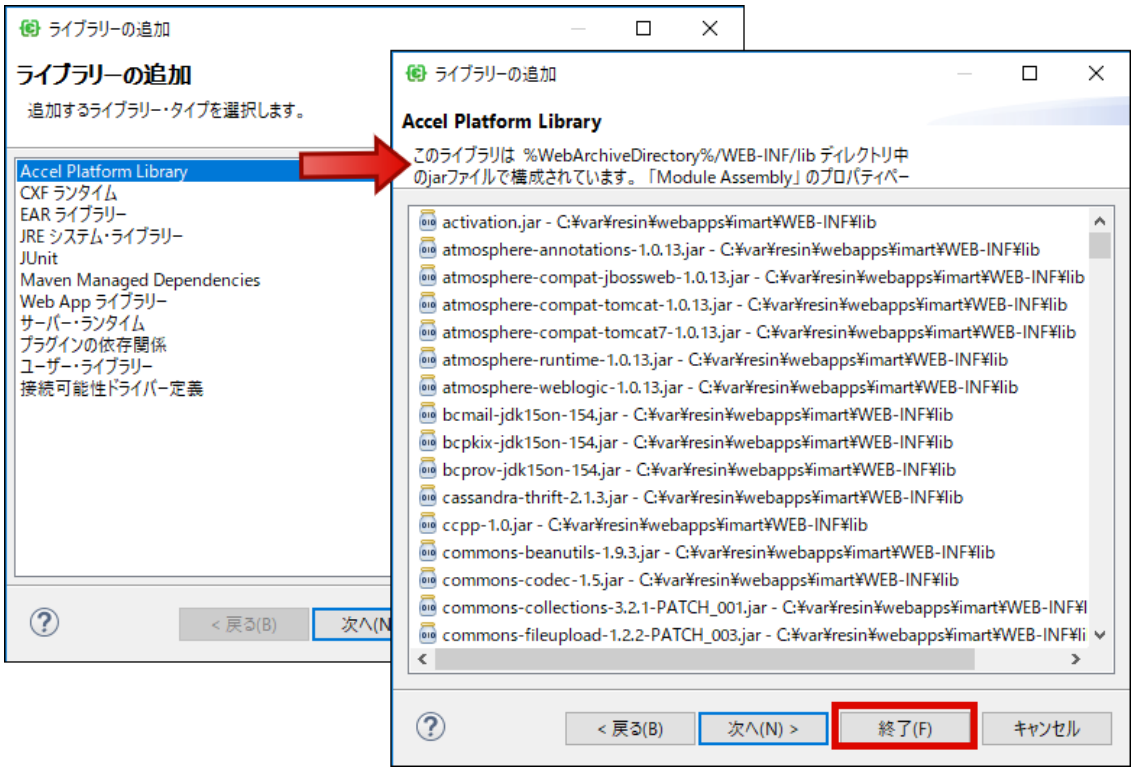
- intra-mart で提供するAPIを各フレームワークで利用するためには、 Accel Platform Library を設定する必要があります。
  その手順は以下のとおりです。

> **コラム**
>
> intra-mart e Builder for Accel Platform 2013 Spring 以降、モジュールプロジェクト作成時にサーバ・ランタイムおよび Accel Platform Library が
> モジュールプロジェクトに自動的に追加されるようになりました。
> 個別にライブラリを設定する必要がある場合のみ、以下手順を参考に設定してください。

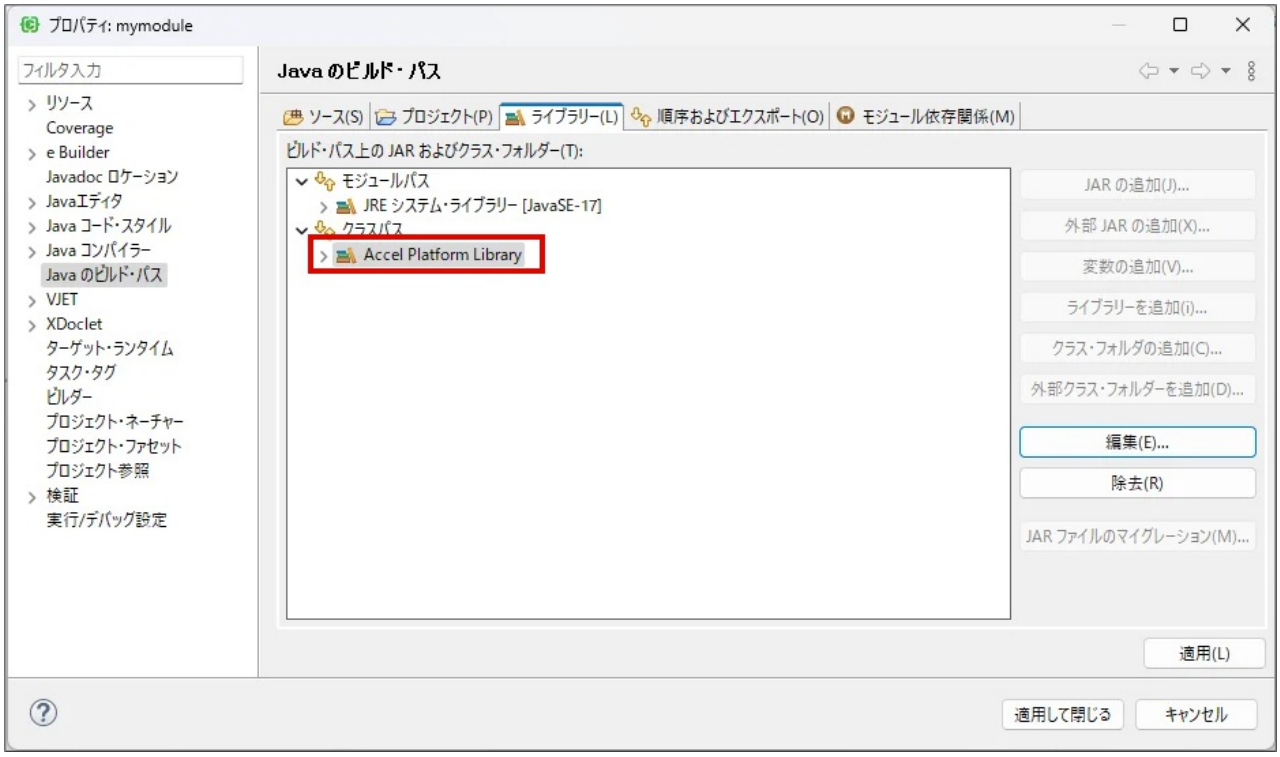
1. プロジェクトの設定ウィザードから、「 Javaのビルド・パス」を選択します。
2. 「ライブラリ」タグを選択し、「ライブラリの追加」ボタンを押下します。



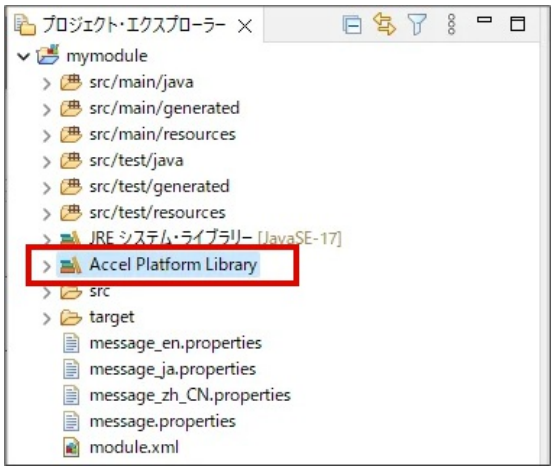
3. ライブラリの選択で「 Accel Platform Library 」を選択し、追加される jar の一覧を確認して、「 終了 」ボタンを押下します。



4. 「 OK 」ボタンを押下して、プロジェクトにライブラリを反映させます。

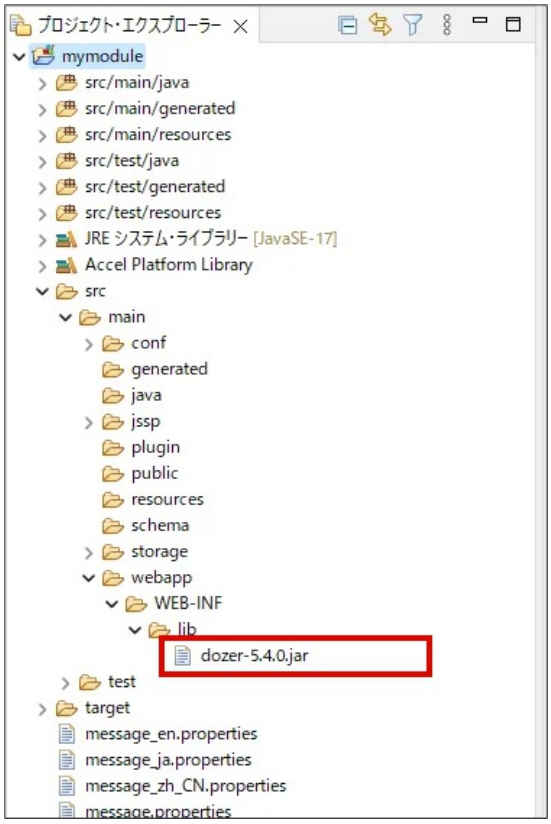


5. プロジェクト配下に Accel Platform Library のリンクができます。

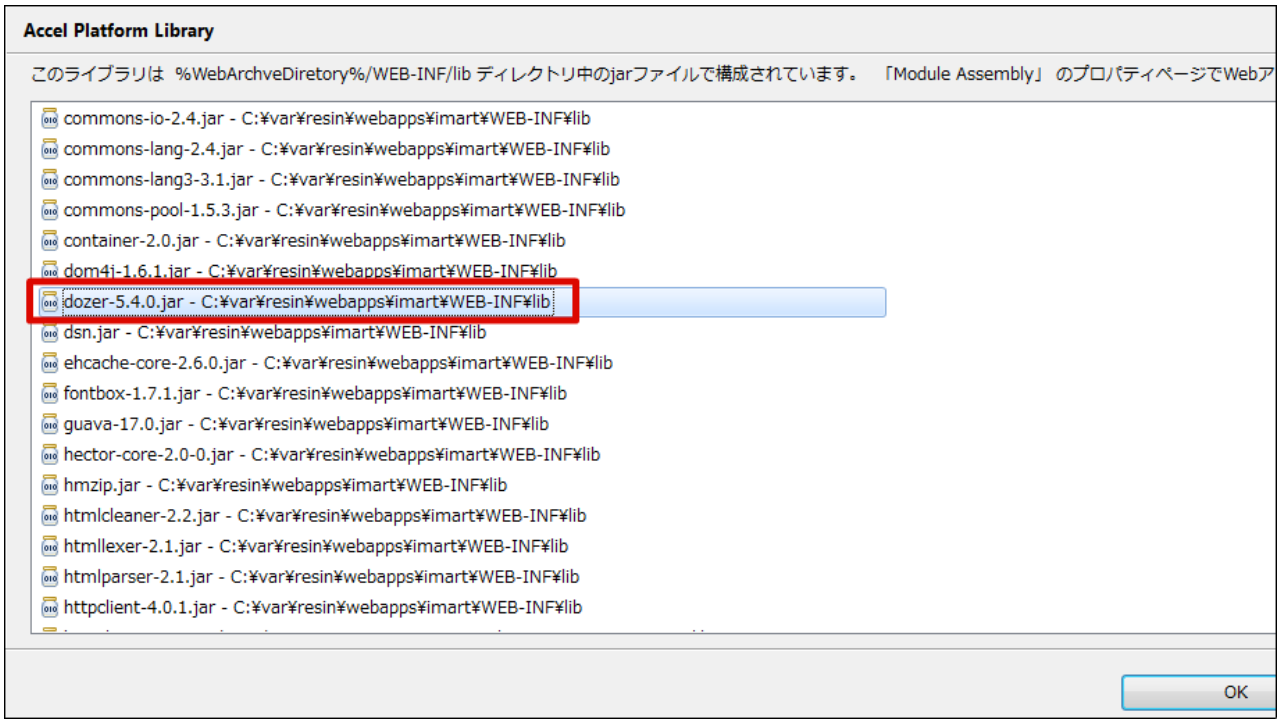


サードパーティライブラリ の設定

- 任意のサードパーティ製ライブラリにクラスパスを通したい場合、
  モジュールプロジェクトのルートパスから、src/main/WEB-INF/libフォルダを作成し、対象のライブラリを配置します。



- 配置後、 Accel Platform Library を展開して対象ライブラリがあるか確認します。
  反映されていない場合、 Accel Platform Library を右クリック→「プロパティ」から一覧を表示し、対象ライブラリの有無を確認してください。



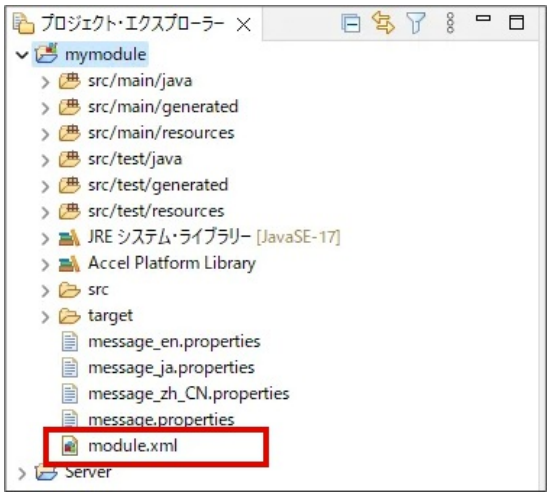
module.xml

概要

- 本項では、モジュールメタデータファイル(module.xml)について説明します。

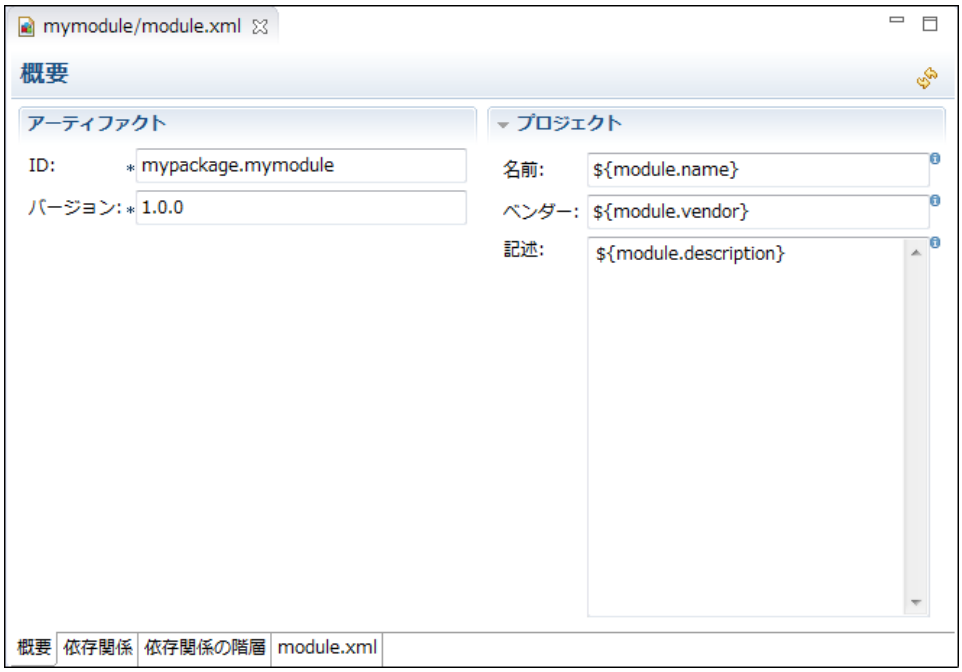
module.xml

- module.xmlは、ユーザ定義モジュールを作成するために必要なファイルです。
  module.xmlは通常モジュールプロジェクトのルートディレクトリに配置されます。



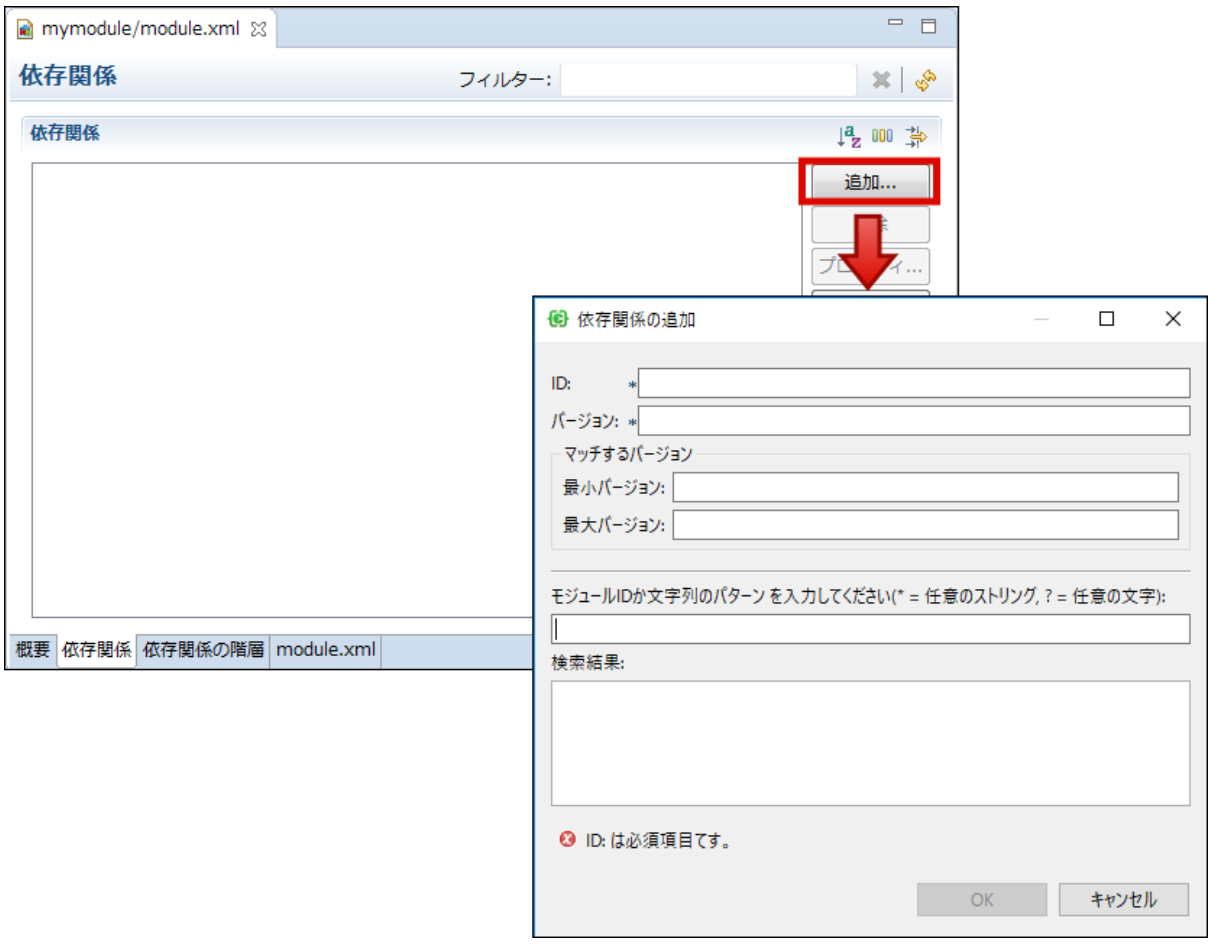
モジュール・エディタ

- module.xmlを選択すると、module.xml専用のエディタ「モジュールエディタ」ビューが表示されます。
  各タブの機能について以下に示します。
  - 「概要」タブ
    モジュールのモジュールID・バージョンなどの基本情報を表示します。
    名前・ベンダ・記述はモジュールプロジェクト直下の各ロケールごとのメッセージプロパティファイル上に定義します。



> **注意**
>
> ID（モジュールID）およびショートモジュールIDは、既存モジュールと重複しないようにしてください。
> 既存モジュールのモジュール情報を確認するには、「WARファイルに含まれるモジュール情報・ショートモジュールIDの一覧を確認する方法」を参照してください。

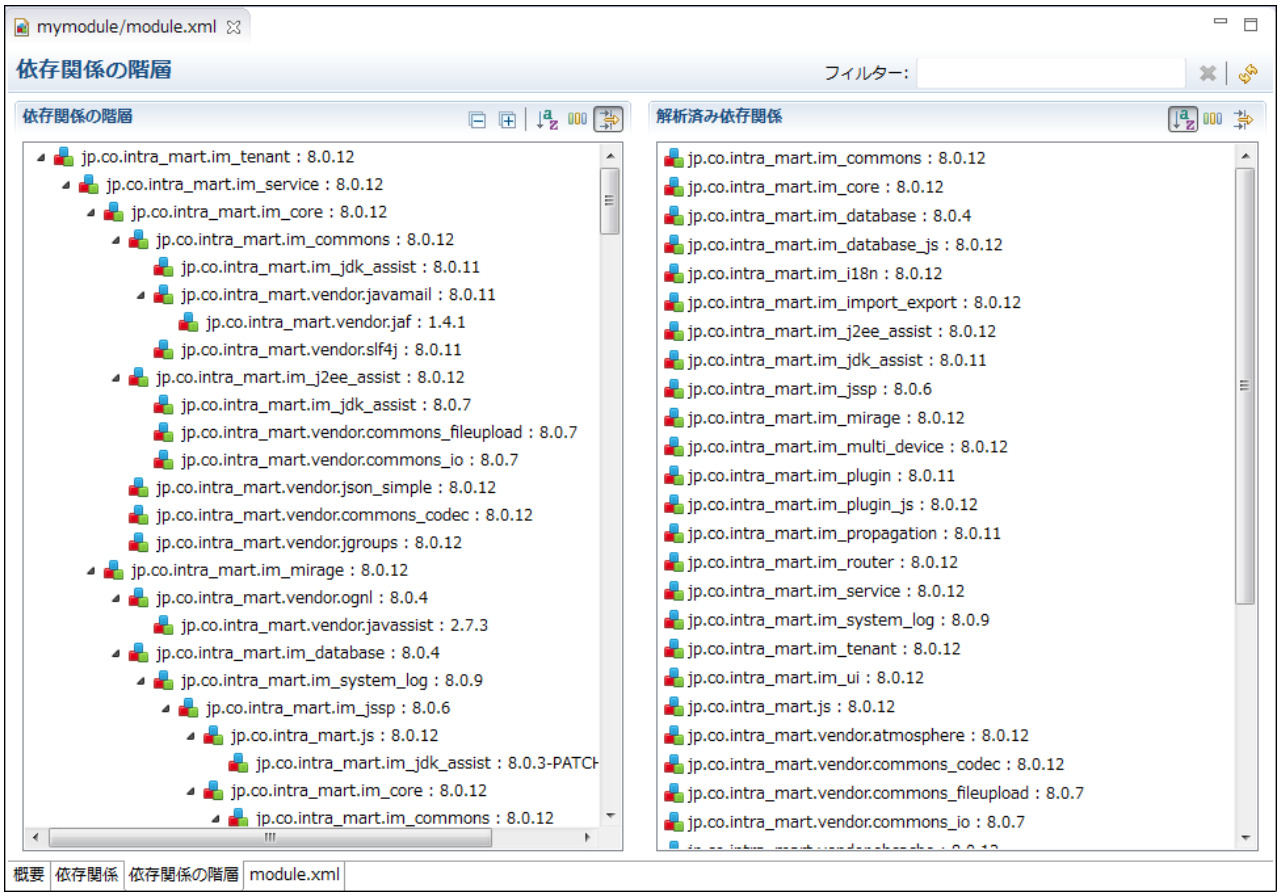
- 「依存関係」タブ
  モジュールプロジェクトが依存するモジュールを設定します。



> **注意**
>
> 依存関係には必ず自分自身を除いたモジュールを依存関係として設定してください。

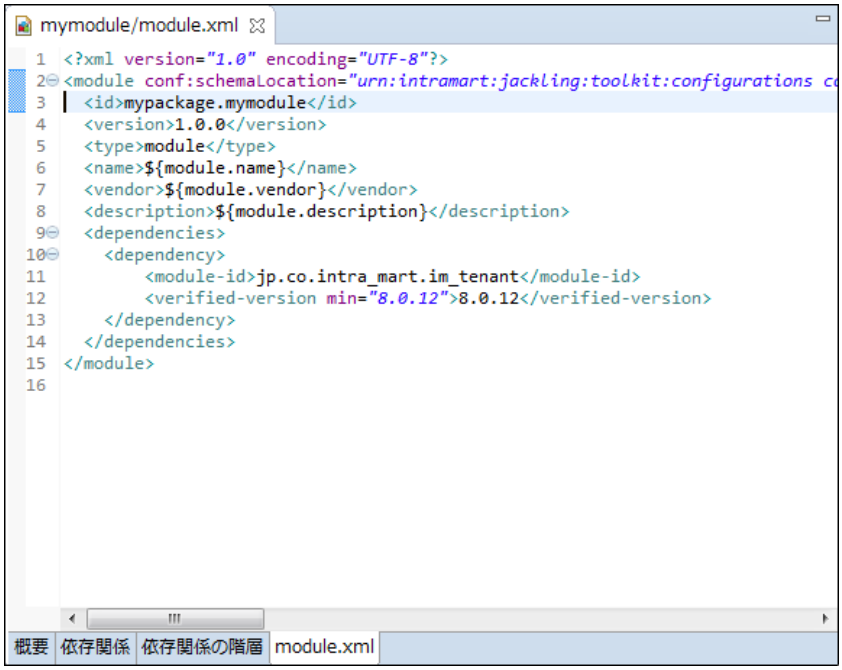
- 「依存関係の階層」タブ

「依存関係」タブで設定したモジュールについて、各モジュールの依存関係を階層構造で表示します。



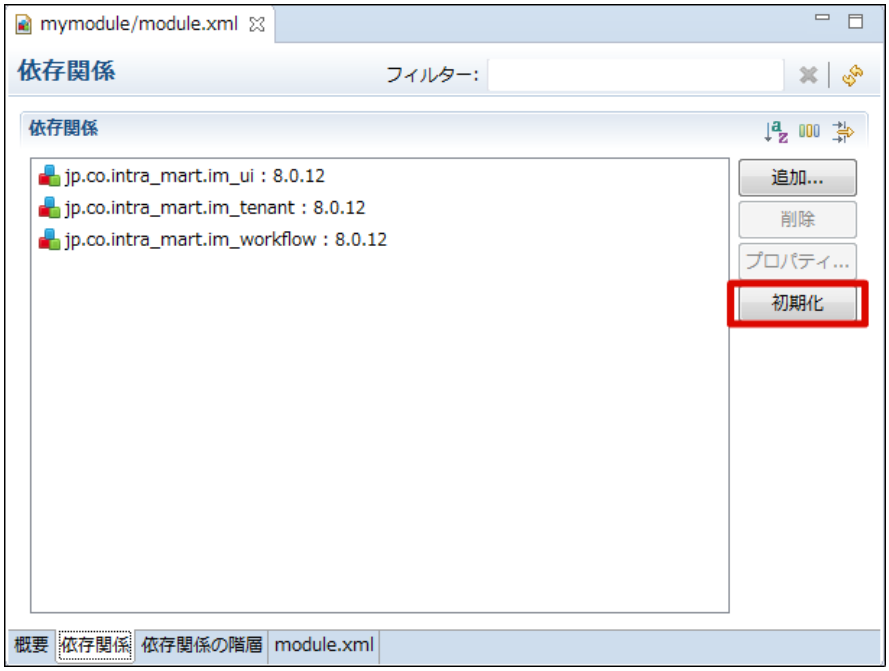
- 「module.xml」タブ

  <module.xml>ファイルをXMLエディタで表示します。



依存するモジュールを最新に更新する場合

Jugglingよりwarファイルを再出力するなどでWebアーカイブディレクトリ上のプロジェクトが変更されたとき、依存するモジュールが更新されている場合は初期化を行う必要があります。
「依存関係」タブより「初期化」ボタンを押下し、再度依存関係を設定してください。



モジュールのパッチの番号を更新する場合

モジュール・プロジェクトのバージョンは、以下の2つのファイルで管理されています。
モジュール・プロジェクトのバージョンを更新したい場合は、以下の2つのファイルの各version属性を更新する必要があります。

- プロジェクト直下のmodule.xmlの4行目に記述されている<version>の値
- プロジェクト直下の.ebuilder-export.xmlの7行目に記述されている<version>の値

module.xml を削除してしまった場合

プロジェクトから module.xml を削除してしまった場合、以下の手順で復旧してください。

1. プロジェクト直下に「 module.xml 」を作成してください。
2. module.xmlを開き、「module.xml」タブを表示します。その後、以下のフォーマットの記述を module.xml にコピーしてください。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<module xmlns="urn:intramart:jackling:module" xmlns:conf="urn:intramart:jackling:toolkit:configurations"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" conf:schemaLocation="urn:intramart:jackling:toolkit:configurations
configurations.xsd" xsi:schemeLocation="urn:intramart:jackling:module module.xsd">

 <id>${group_id}.${artifact_id}</id>
 <version>${version}</version>
 <type>module</type>

 <name>${module.name}</name>
 <vendor>${module.vendor}</vendor>
 <description>${module.description}</description>

 <!-- 変更を不可とする場合やサードパーティモジュールの場合のみ以下を指定する -->
 <tags>
  <tag>immutable</tag>
  <tag>3rd-party</tag>
 </tags>

 <dependencies>
  <dependency>
  </dependency>
 </dependencies>

</module>
```

3. 以下の文字列を、プロジェクト作成時に入力した内容に合わせて記述を行ってください。

${group_id}	プロジェクトのグループID

${artifact_id}	プロジェクトのショートモジュールID（プロジェクト名）
${version}	プロジェクトのバージョン番号

プログラム開発

概要

- 本項では、モジュール・プロジェクト内でプログラムを作成する方法について説明します。

プロジェクト構成と配置するファイル

- プロジェクトの構成とその構成に対して配置するファイルの説明は同ガイドの *e Builder での開発の流れ* を参照してください

開発モデル

- intra-mart Accel Platformにて利用できる開発モデルは以下の通りです。
  詳しくは各種プログラミングガイドを参照してください。
  - スクリプト開発モデル
  - im-JavaEE フレームワーク
  - SAStruts+S2JDBC フレームワーク
- また各開発モデルのe Builderの開発支援機能については下記ページをご覧ください。
  - *スクリプト開発で利用できる機能*
  - *im-JavaEE フレームワーク開発で利用できる機能*
  - *SAStruts + S2JDBC フレームワーク開発で利用できる機能*
  - *TERASOLUNA Server Framework for Java (5.x) for Accel Platform 開発で利用できる機能*

immファイルのエクスポート

概要

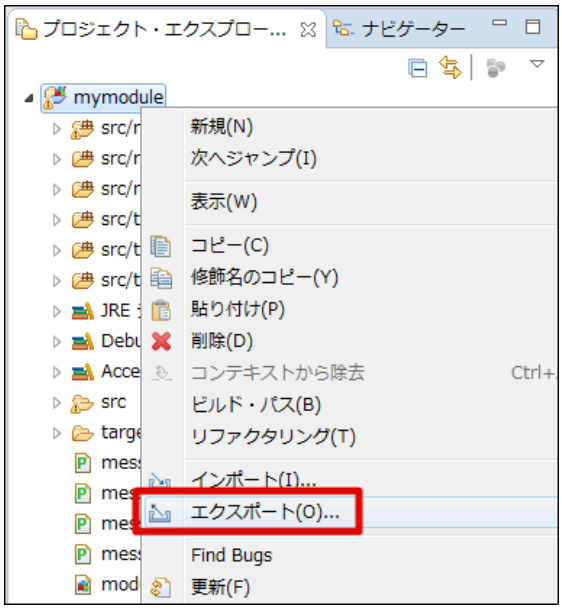
- 本項では、e Builder上で開発したモジュール・プロジェクトからユーザ定義モジュールとなるimmファイルをエクスポートする方法に関して説明します。

前提条件

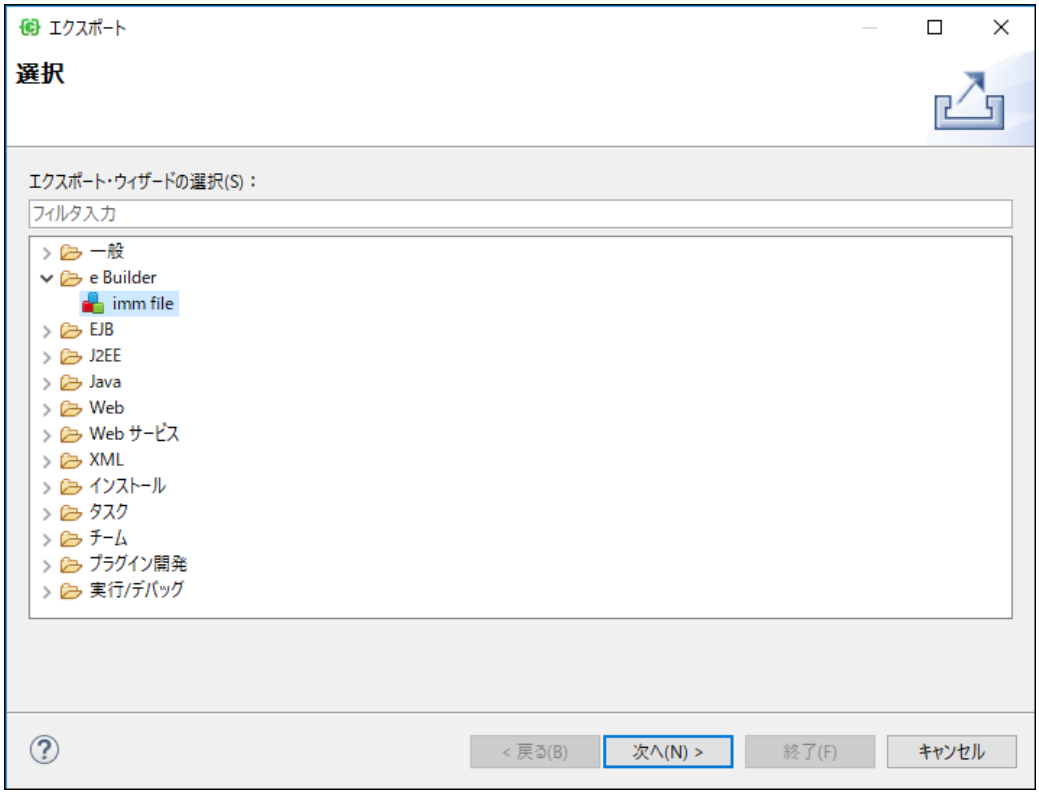
- e Builder をセットアップガイドに基づいてセットアップが完了していること
- e Builder でモジュールの開発を行っていること

手順

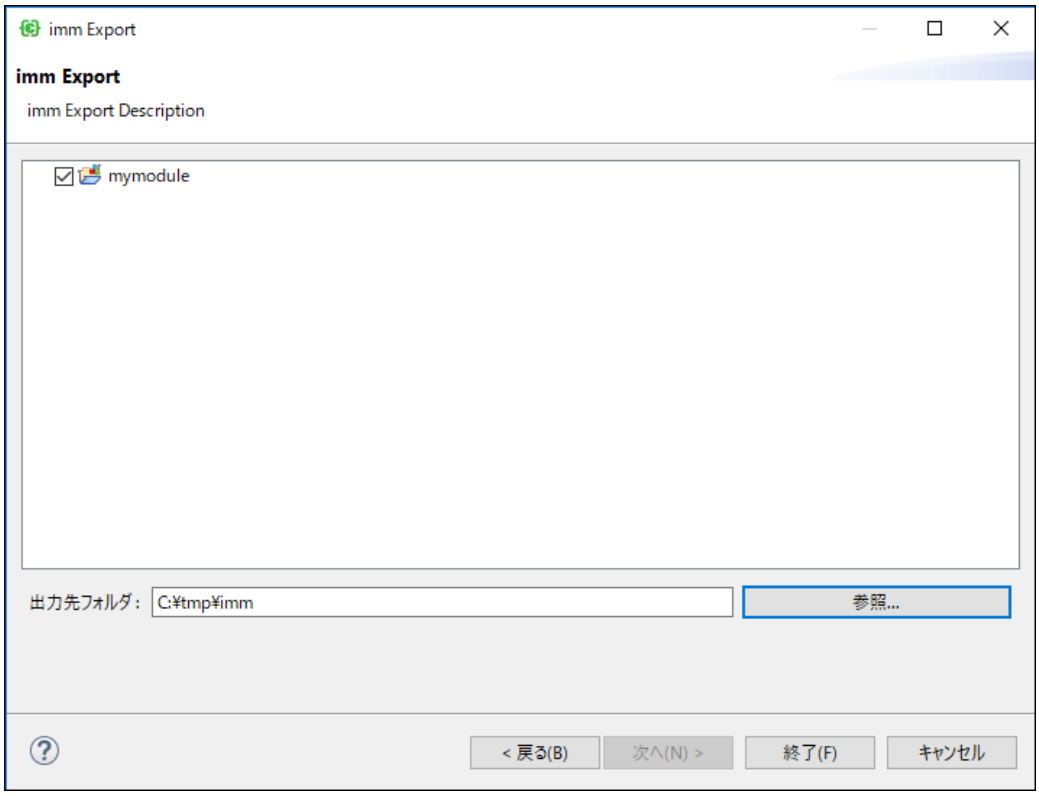
1. エクスポートを行いたいモジュール・プロジェクトにフォーカスを合わせて右クリックし、「エクスポート」を選択します。



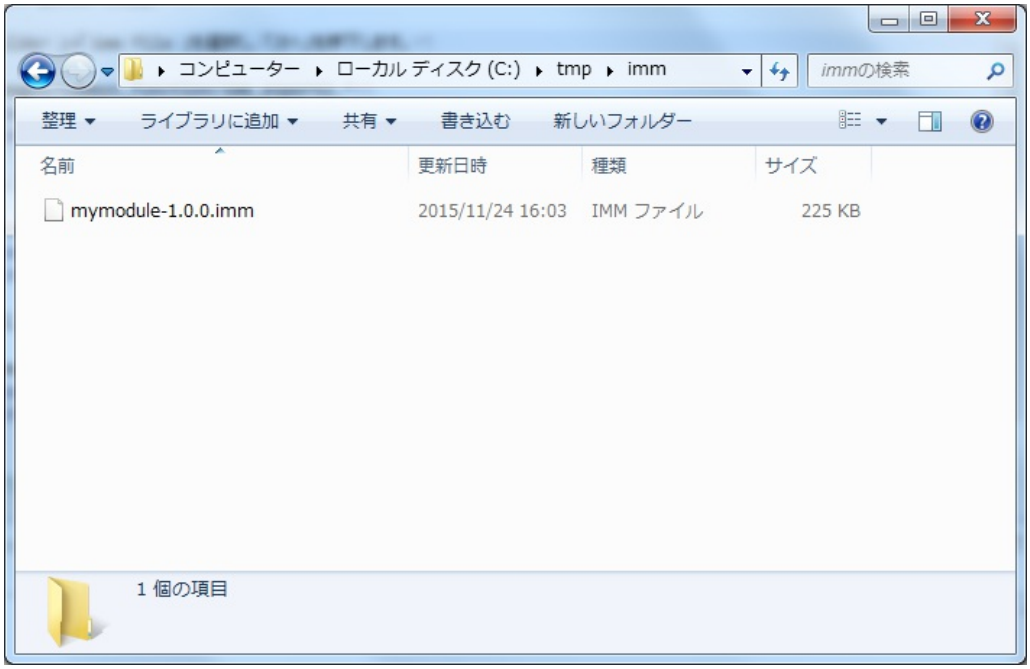
2. エクスポートウィザードから「 e Builder 」→「 imm file 」を選択し、「次へ」を押下します。



3. imm ファイルエクスポート画面から、エクスポートするモジュール・プロジェクトとファイルの出力先フォルダを設定し「完了」を押下します。



4. 出力先フォルダに指定された場所に、imm ファイルが作成されます。



> **コラム**
>
> e Builderで作成されたプロジェクトには、ユーザ定義モジュールを作成するために必要な <.ebuilder-export.xml> ファイルを配置しています。
> もしこのファイルがなくなった場合、エクスポート処理中にエラーが発生します。
> 下記のリンクには <.ebuilder-export.xml> ファイルがなくなった場合の対処方法について記述しています。
> *.ebuilder-export.xml*

CUIでのモジュール作成方法について

- モジュール・プロジェクトを作成する際、 Maven の機能を利用して CUI でモジュールを作成できます。
  e Builder で作成される imm ファイルは mvn package コマンドを実行後 target フォルダ配下に生成された、<アーティファクト名>-<バージョン>.zip というファイルを imm ファイルに拡張子をリネームして作成されています。
  Maven コマンドを実行する際に必要となる pom.xml と ローカル・リポジトリ は標準で下記のものを利用しています。
  - pom.xml - <モジュール・プロジェクトのパス>/.ebuilder-export.xml
  - ローカル・リポジトリ - <e Builderの展開パス

>/plugins/jp.co.intra_mart.module.developer.maven_8.0.X.yyyyMMddHHmm/repository

- settings.xml - <ワークスペース>/.metadata/.plugins/jp.co.intra_mart.module.developer.maven/.m2/settings.xml

ご利用になる際は、必要に応じてファイルの再作成や書き換えを行い、Mavenコマンドにオプションをつけるなどしてご利用ください。

コラム

この機能を利用する際、コマンドを実行するOSに対してMavenのセットアップが完了している必要があります。

スクリプト開発で利用できる機能

ここでは、 e Builder で利用できるスクリプト開発用の基本機能を紹介します。

HTMLエディタ

項目

概要

- 本項では、スクリプト開発を行う際に利用できる HTML エディタの機能について説明します。
  e Builder で提供されている HTML エディタは、 Eclipse WTP で提供されている Web ページ・エディタの機能を拡張しています。

前提条件

- モジュール・プロジェクト内に編集可能な HTML ファイルが存在すること

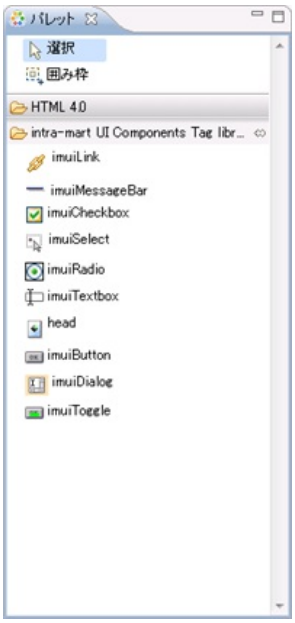
Web ページ・エディタの編集機能

HTML エディタは編集したい HTML ファイルに対して、開くエディタに「 Web ページ・エディタ」を選択して開きます。
Web ページ・エディタを拡張した HTML エディタは、以下の3つの機能で構成されています。

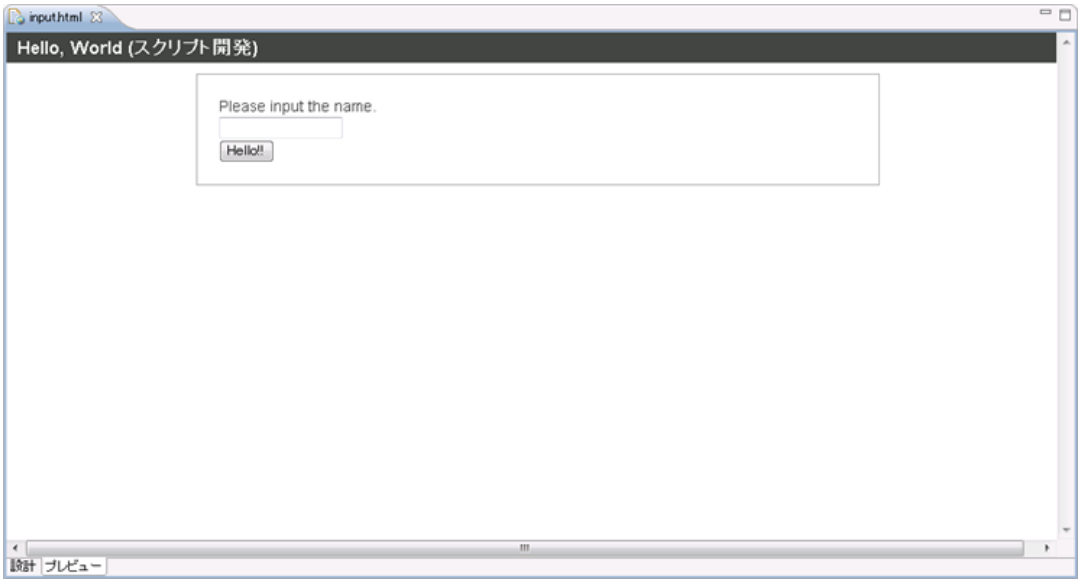
- 設計機能

  HTML エディタの左下に表示されている「設計」タブを選択することにより利用できます。
  HTML の画面設計をデザインベース、およびテキストベースで編集します。
  この機能は、 Eclipse WTP で提供されているものと同等のものです。

- パレット機能
  「パレット」ビューを表示することによって利用できます。
  intra-mart Accel Platform で利用可能な HTML タグをパレットから提供します。
  パレットから利用したいアイテムを選択し、ドラッグ&ドロップでアイテムを配置できます。



- プレビュー機能
  HTML エディタの左下に表示されている「プレビュー」タブを選択することにより利用できます。
  編集時点での HTML ファイルのソースから、実際のブラウザ上で表示される画面を表示します。
  プレビューでは、スタイルシートが適用された状態で表示されます。



> **コラム**
>
> なお、 HTML ファイル内で message タグを利用した場合、プレビュー画面には OS のロケーションの言語からプロパティメッセージを読み込みます。
> 別のロケーションの言語を利用したい場合は、 eBuilder8.ini に以下の引数を指定してください。
> ※国をアメリカ、言語を英語に設定したい場合
> -Duser.language=en
> -Duser.country=US

デバッグ（スクリプト開発）

概要

- 本項では、 e Builder で開発したモジュール・プロジェクトのソースを
  デバッグするために必要となる Resin の設定方法に関する説明をします。

前提条件

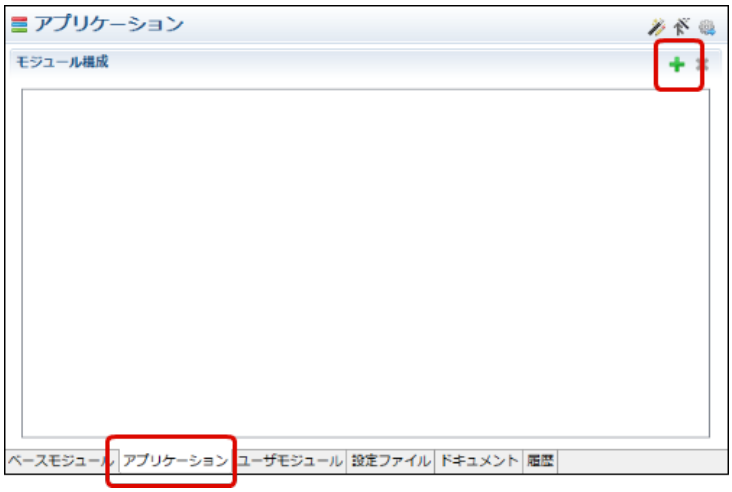
- intra-mart Accel Platform の設定が完了している デバッグ サーバがあること

> **コラム**
>
> 本項では、 Eclipse のサーバ・ビュー上に設定した デバッグ サーバを動かすことを前提で説明を行います。
> デバッグサーバを起動する場合は、ワークスペース上にできた Server プロジェクトを開いた状態にしておく必要があります。

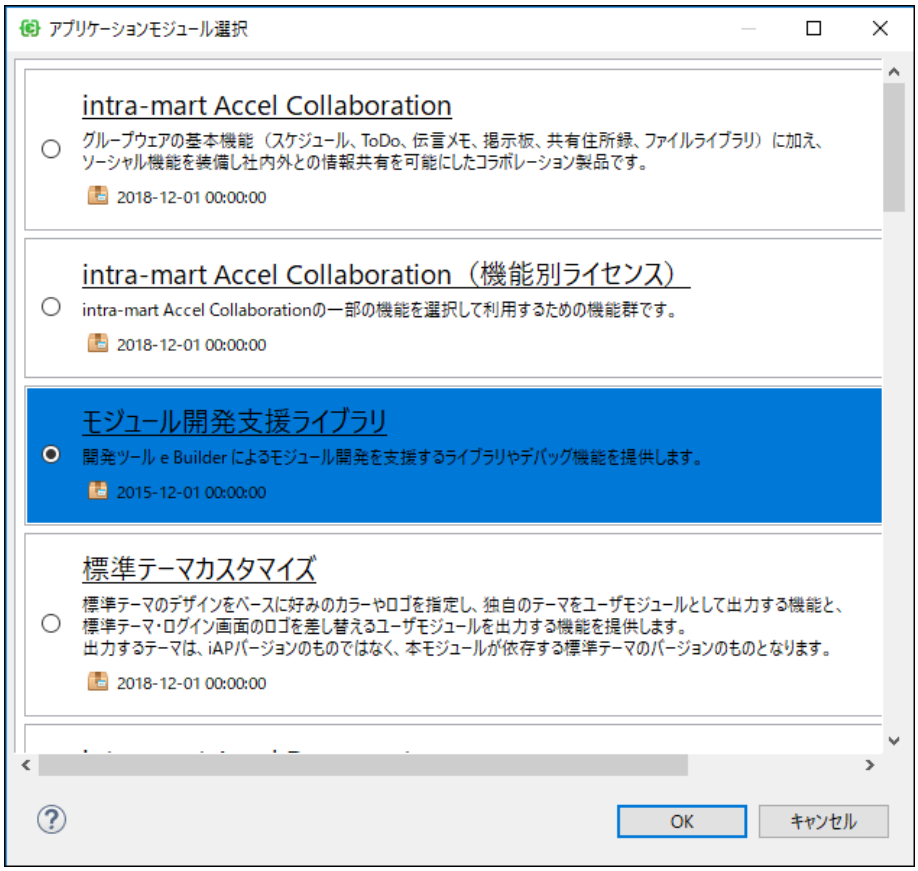
プロジェクトの設定

デバッグを行うためのプロジェクト設定を行います。

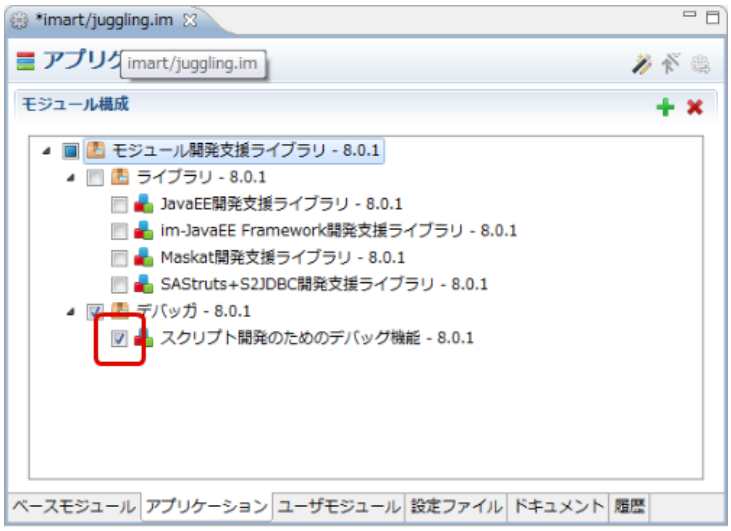
- IM-Jugglingから、任意の作成済プロジェクトを開き、
  Juggling.imの「アプリケーション」タブを選択し画面右上部の「+」ボタンを押下します。



- 「アプリケーション選択」ウィザードから「モジュール支援ライブラリ」を選択し「OK」ボタンを選択します。

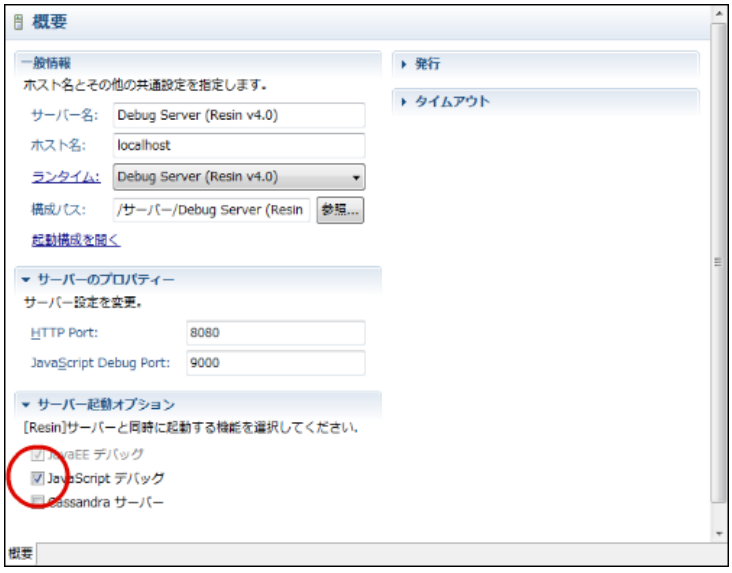


- 「モジュール開発支援ライブラリ」→「デバッガ」→「スクリプト開発のためのデバッグ機能」にチェックを入れ、保存します。
  その後warファイルを出力し、Resinサーバにデプロイしてください。



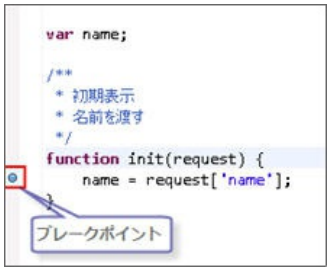
デバッグの設定

- 「サーバ」ビューから、登録済のデバッグサーバを選択し、起動構成ビューを表示します。
- 「サーバ起動オプション」の「JavaScript デバッグ」にチェックを入れ、保存します。



デバッグの実行

- デバッグサーバをデバッグ起動します。
- デバッグを行うプロジェクト内の JS ファイルを開いてブレークポイントを設定します。



- ブラウザでデバッグ対象の処理を実行します。
- Eclipse のデバッグモードが起動します。

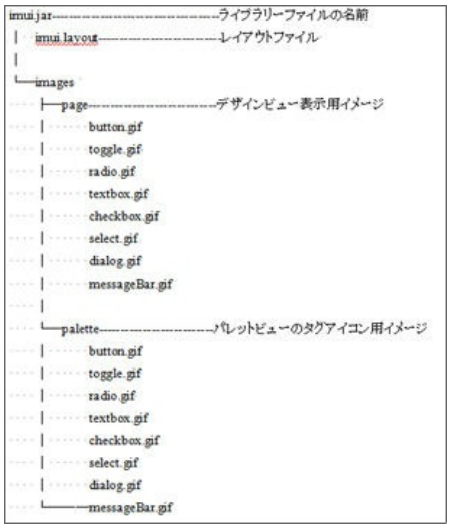
独自で作成したタグの利用方法

概要

- 本項ではユーザが独自で作成したタグを、 palette に追加し、利用する方法について説明します。

タグライブラリの構成

e Builder でユーザ独自のタグライブラリを追加する場合、以下のようなフォルダ構成のファイルを作成する必要があります。
タグの構成は以下のとおりです。



タグライブラリの追加方法

ここではタグライブラリをzipファイル形式で作成した場合の追加方法の説明をします。

1. タグライブラリの構成で作成した zip ファイルをプロジェクト上に配置します。
2. 独自で作成したタグライブラリを利用したいプロジェクトにフォーカスを合わせて右クリックし、「設定」を選択します。

3. 「 e Builder 」→「 Tag Library 」を選択します。

4. 「 Tag Library 」ウィザードで「 Zip の追加」を選択します。
5. 追加したいタグライブラリの Zip ファイルを選択し、「 OK 」ボタンをクリックします。
6. 「 Tag Library 」のウィザードで「適用」、または「 OK 」ボタンをクリックすると完了します。

im-JavaEEフレームワーク開発で利用できる機能

ここでは、 e Builder で利用できる im-JavaEE フレームワーク開発用の基本機能を紹介します。

JSP エディタ

項目

概要

- 本項では、 JSP の編集を行うための JSP エディタの機能について説明します。
  e Builder で提供されている JSP エディタは、 Eclipse WTP で提供されている Web ページ・エディタの機能を拡張しています。

前提条件

- e Builder を利用し、モジュール・プロジェクト内に編集可能なJSPファイルが存在すること

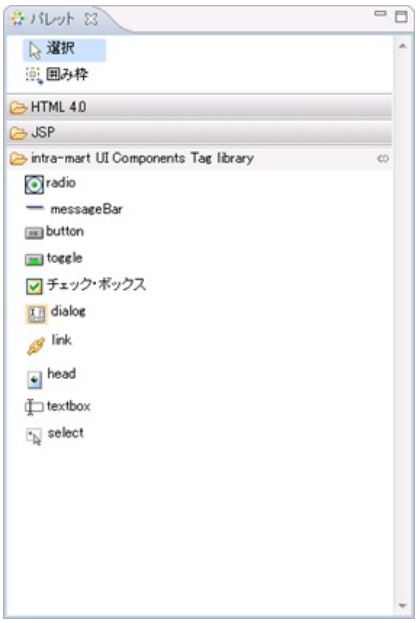
JSPエディタの編集機能

JSP エディタは編集したい JSP ファイルに対して、開くエディタに「Web ページ・エディタ」を選択して開きます。
Web ページ・エディタを拡張した JSP エディタは、以下の3つの機能で構成されています。

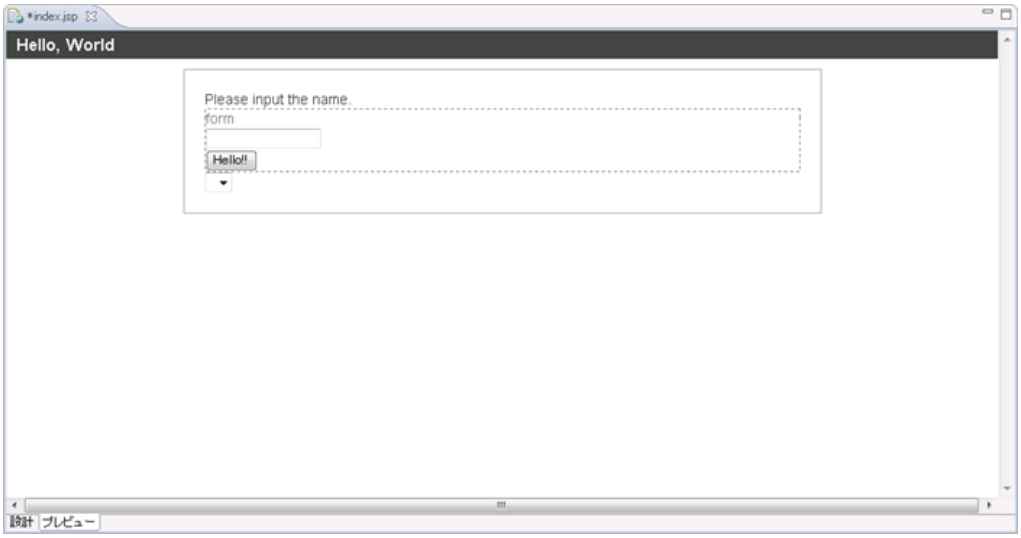
- 設計機能

  JSP エディタの左下に表示されている「設計」タブを選択することにより利用できます。
  JSP の画面設計をデザインベース、およびテキストベースで編集します。
  この機能は、 Eclipse WTP で提供されているものと同等のものです。

- パレット機能
  「パレット」ビューを表示することによって利用できます。
  intra-mart Accel Platform で利用可能な JSP タグをパレットから提供します。
  パレットから利用したいアイテムを選択し、ドラッグ&ドロップでアイテムを配置できます。



- プレビュー機能
  JSP エディタの左下に表示されている「プレビュー」タブを選択することにより利用できます。
  編集時点での JSP ファイルのソースから、実際のブラウザ上で表示される画面を表示します。
  プレビューでは、スタイルシートが適用された状態で表示されます。



> **コラム**
>
> なお、 JSP ファイル内で message タグを利用した場合、プレビュー画面には OS のロケーションの言語からプロパティメッセージを読み込みます。
> 別のロケーションの言語を利用したい場合は、 eBuilder8.ini に以下の引数を指定してください。
> ※国をアメリカ、言語を英語に設定したい場合
> -Duser.language=en
> -Duser.country=US

プレビュー機能

Webページ・エディタでjspファイルを開くと、実際のブラウザ上でどのように表示されるかを確認できる「プレビュー」という機能があります。
プレビューでは、スタイルシートが適用された状態で表示されます。

<画像>

なお、JSPファイル内でmessageタグを利用した場合、プレビュー画面にはOSのロケーションの言語からプロパティメッセージを読み込みます。
別のロケーションの言語を利用したい場合は、eBuilder8.iniに以下の引数を指定してください。
※国をアメリカ、言語を英語に設定したい場合
-Duser.language=en
-Duser.country=US

本項では、 e Builder で利用できる im-JavaEE フレームワークエディタの基本機能に関して紹介します。

アプリケーション

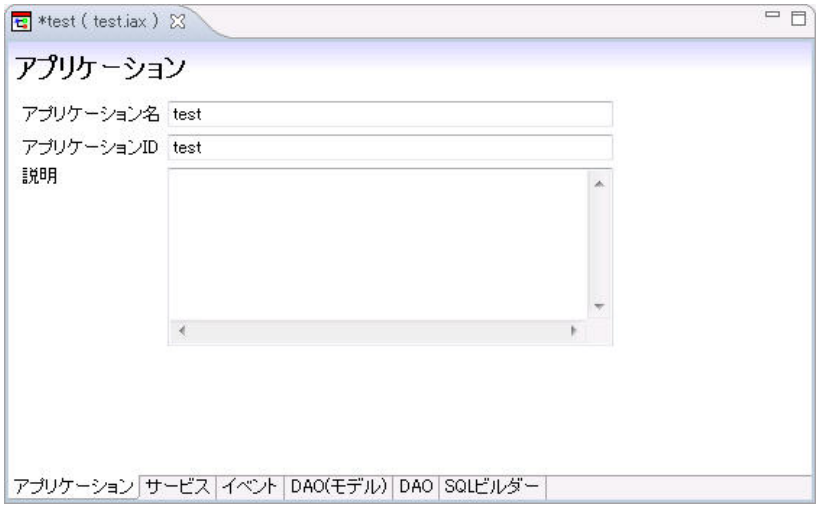
項目

概要

- 本項では、 im-JavaEE フレームワークエディタにおける、「アプリケーション」タブの設定内容について説明します。

設定項目

項目名	説明
アプリケーション名	開発するアプリケーションの名前を設定します。
アプリケーションID	開発するアプリケーションのIDを設定します。
説明	開発するアプリケーションの説明を記述します。

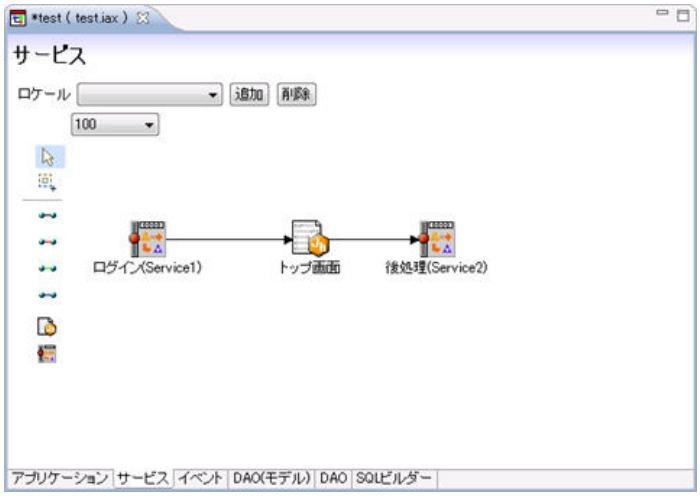


サービス

項目

概要

- 本項では、 im-JavaEE フレームワークエディタにおける、「サービス」タブの設定内容について説明します。



操作方法

1. GUI 画面上にサービスと jsp ページを接続ツールで結んでいき、サービスと jsp ページのマッピングを行っていきます。
2. サービスや jsp ページのアイコンにフォーカスを合わせて、プロパティ・ビューにて必要な情報を設定していきます。（設定内容は下記参照）
3. 保存すると自動的にサービスフレームワークに関連するコンフィグファイル、および Java ファイルを生成します。

設定メニュー

- GUI 画面上で設定できる内容は以下の通りです

項目	概要
選択ツール	カーソルを通常のカーソルに戻します。
範囲選択ツール	範囲指定で含まれた要素すべてにフォーカスを当てます。
接続ツール	正常系の動作に対する遷移先を指定します。
サービスエラー接続	サービスエラー発生時の遷移先を指定します。
システムエラー接続	システムエラー発生時の遷移先を指定します。
入力エラー接続	入力エラー発生時の遷移先を指定します。
遷移ページ	遷移先となる JSP ページを指定します。
サービス	実行するサービスを指定します。

各項目のプロパティ設定

設定したアイコンにフォーカスを当てた場合、プロパティ・ビューで設定できる項目は以下の通りです。

- 接続ツール

項目	概要
キー	この接続を実行するためのキーを指定します。
説明	この接続の説明を記述します。

- サービスエラー接続

項目	概要
キー	この接続を実行するためのキーを指定します。
説明	この接続の説明を記述します。

- システムエラー接続

項目	概要
キー	この接続を実行するためのキーを指定します。
説明	この接続の説明を記述します。

- 入力エラー接続

項目	概要
キー	この接続を実行するためのキーを指定します。
説明	この接続の説明を記述します。

- 遷移ページ

項目	説明
ページパス	JSP ページのパスを指定します。 JSP ページは src/main/webapp フォルダ配下にある JSP ファイルが対象です。
ページ名	JSP のページ名を指定します。
位置	フォーカスを当てているアイコンの GUI 画面上での位置情報を指定します。（編集不可）

- サービス

項目	説明
コントローラクラス名	サービスに紐づくコントローラクラスを指定します。
コントローラ コンバーター	コントローラ コンバーターとなるクラスを指定します。
サービスID	このサービスを利用するためのキーとなる ID を指定します。
サービス処理結果クラス名	サービスで実行した処理の結果を格納するクラスを指定します。
サービス名	サービス名を指定します。
トランジションクラス名	このサービスに紐づくトランジションクラスを指定します。
バリデータ	バリデータの情報を記述します。
位置	フォーカスを当てているアイコンの GUI 画面上での位置情報を指定します。（編集不可）
説明	このサービスの説明を記述します。

サービス関連のクラス作成ウィザード

サービスに関するクラスを作成する際に、ウィザードを利用したクラスの作成が可能です。
ここでは各サービスの設定情報で利用できるウィザードの説明を行います。

- コントローラクラス名（サービスコントローラクラスの作成ウィザード）

項目	説明
既存のクラスから作成	すでに作成されているクラスを指定します。
新規のクラスを作成	新しくサービスクラスを作成します。 作成するサービスクラスは jp.co.intra_mart.framework.base.service.ServiceControllerAdapter を継承したクラスです。
スケルトンから作成	e Builder で利用できるサービスコントローラクラスの雛型を元にサービスクラスを作成します。

- コントローラコンバータークラス名（コントローラ コンバーター ダイアログ）

項目	説明
コントローラオブジェクトクラス名	コントローラオブジェクトのクラスを指定します。 指定するクラスはインタフェース jp.co.intra_mart.framework.base.service.controller.ControllerObject を実装している必要があります。

項目	説明
コントローラコンバーター クラス名	コントローラコンバーターのクラスを指定します。 指定するクラスはインタフェース jp.co.intra_mart.framework.base.service.controller.ControllerConverter を実装している必要があります。
パラメータ定義リスト	受け渡すパラメータを定義します。

- サービス処理結果クラス名（サービス処理結果クラスの作成）

項目	説明
既存のクラスから作成	すでに作成されているクラスを指定します。
新規のクラスを作成	新しくサービスクラスを作成します。 作成するサービスクラスはインタフェース jp.co.intra_mart.framework.base.service.ServiceResult を実装したクラスです。
スケルトンから作成	e Builder で利用できるサービス処理結果クラスの雛型を元にサービス処理結果クラスを作成します。

- トランジションクラス名（トランジションクラスの作成）

項目	説明
既存のクラスから作成	すでに作成されているクラスを指定します。
新規のクラスを作成	新しくサービスクラスを作成します。 作成するサービスクラスは jp.co.intra_mart.framework.base.service.DefaultTransition を継承したクラスです。
スケルトンから作成	e Builder で利用できるトランジションクラスの雛型を元にトランジションクラスを作成します。

- バリデータ（バリデータリストダイアログ）

  バリデータリストを追加するには、リストのテーブル上で右クリックをして、「追加」を選択する必要があります。

項目	説明
バリデータ名	設定しているバリデータを識別するためのラベルです。
パラメータリスト	設定されているバリデータの結果に応じたメッセージ、プロパティ情報です。

イベント

項目

概要

- 本項では、 im-JavaEE フレームワークエディタにおける、「イベント」タブの設定内容について説明します。

操作方法

1. イベントグループリストにイベントを新規に追加します。
2. 「表示名」、「イベントキー」などといった項目に必要な情報を記述していきます。
3. 「イベントファクトリ」 画面で、イベントファクトリの設定を行います。
4. 必要に応じて「イベントトリガ」画面で前処理となるトリガと、「後処理」となるトリガを設定します。
5. 保存すると自動的にイベントフレームワークに関するコンフィグファイル、および Java ファイルが生成されます。

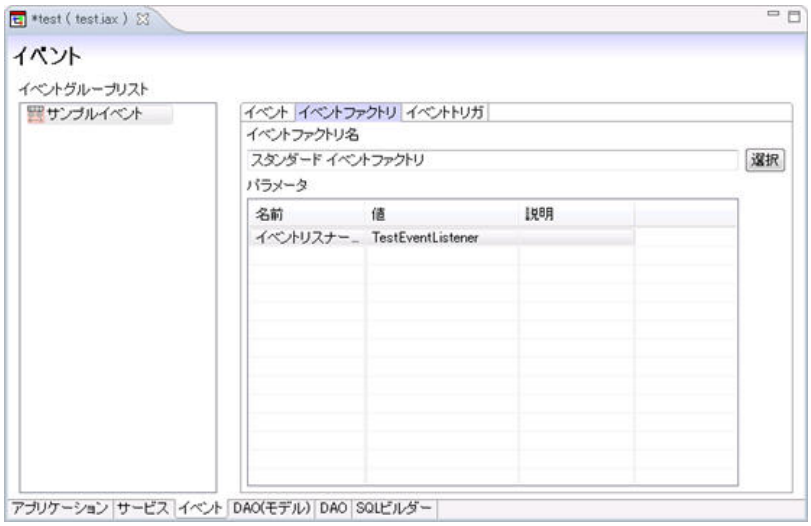
設定項目

各設定項目はイベントグループリストにイベントを登録し、対象のイベントにフォーカスを当てたうえで設定してください。

- イベント画面

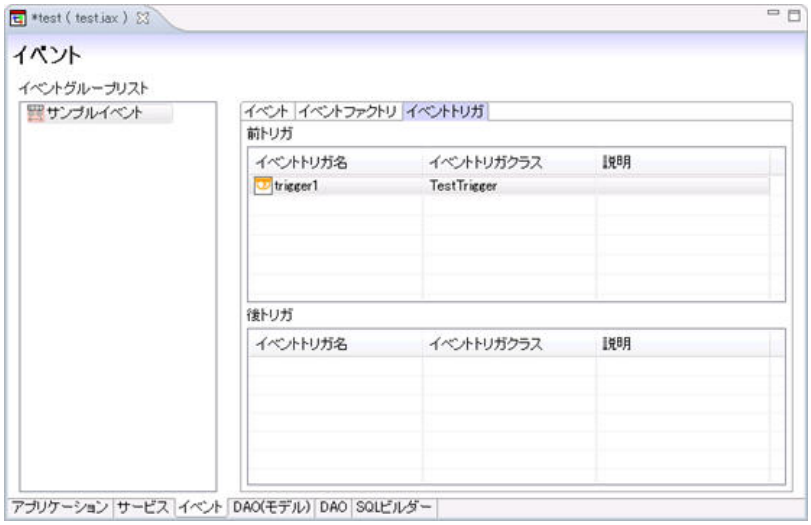
項目	説明
表示名	イベントを表示するためのラベルです。
イベントキー	イベントを呼び出すためのキーを設定します。
イベントクラス名	このイベントに紐づくイベントクラスを設定します。
イベント処理結果名	このイベントに紐づくイベント処理結果クラスを設定します。
説明	このイベント対する説明を記述します。

- イベントファクトリ画面



項目	説明
イベントファクトリ名	イベントで利用するイベントファクトリを指定します。
パラメータ	イベントファクトリ作成時のパラメータを指定します。

- イベントトリガ画面



項目	説明
前トリガ	イベント処理実行前のトリガを指定します。
後トリガ	イベント処理実行後のトリガを指定します。

イベント関連のクラス作成ウィザード

イベントに関するクラスを作成する際に、ウィザードを利用したクラスの作成が可能です。

- イベントクラス名（クラスの作成）

項目	説明
既存のクラスから作成	すでに作成されているクラスを指定します。
新規のクラスを作成	新しくイベントクラスを作成します。 作成するイベントクラスは jp.co.intra_mart.framework.base.event.Event を継承したクラスです。
スケルトンから作成	e Builder で利用できるイベントクラスの雛型を元にイベントクラスを作成します。

- イベント処理結果名（クラスの作成）

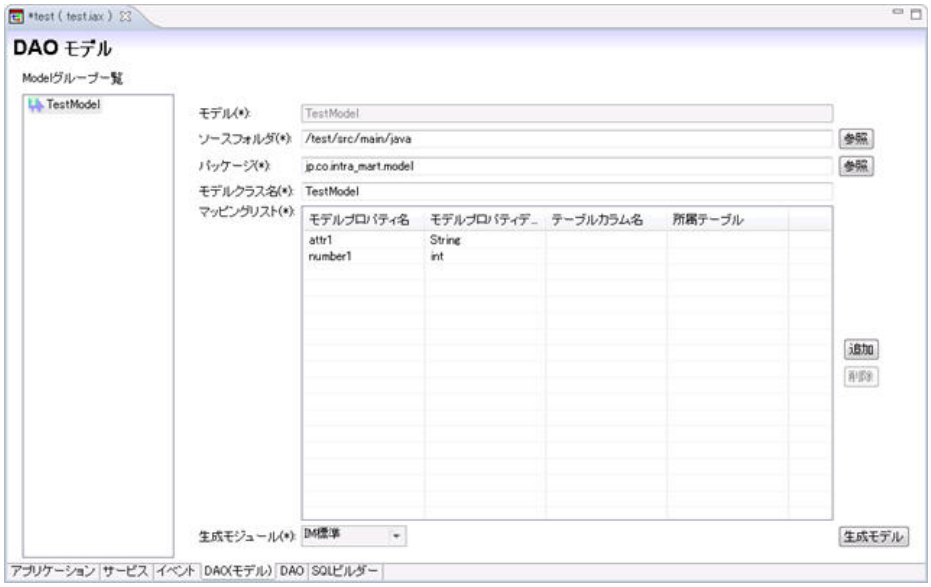
項目	説明
既存のクラスから作成	すでに作成されているクラスを指定します。
新規のクラスを作成	新しくイベント処理結果クラスを作成します。 作成するイベント処理結果クラスはインタフェース jp.co.intra_mart.framework.base.event.EventResult を実装したクラスです。
スケルトンから作成	e Builder で利用できるイベント処理結果クラスの雛型を元にイベント処理結果クラスを作成します。

DAO（モデル）

項目

概要

- 本項では、im-JavaEE フレームワークエディタにおける、「 DAO（モデル）」タブの設定内容について説明します。



操作方法

1. Model グループ一覧にモデルグループを追加します。
2. 生成するモデルの各種情報を入力します。
3. 入力完了後、エディタを保存して「生成モデル」ボタンを押下し、Modelクラスを自動生成します。

DAO（モデル）の設定項目

1つのモデルグループに対して設定できる項目は以下のとおりです。

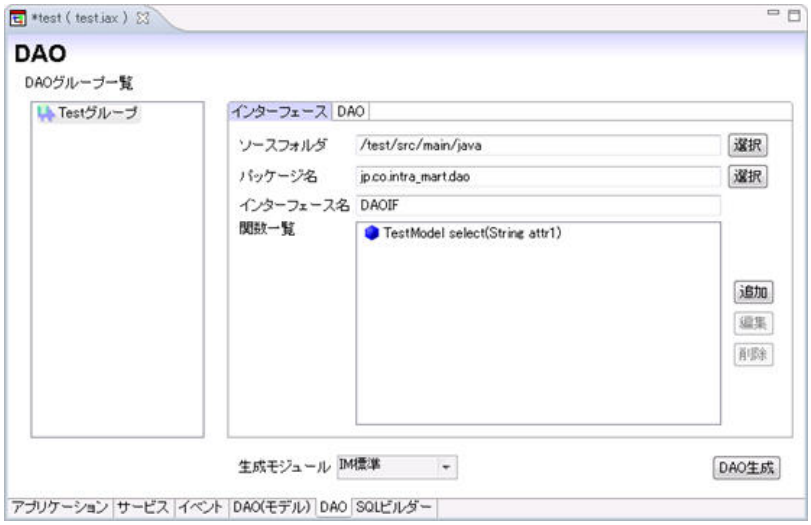
項目	説明
モデル	選択されているモデルグループを表示します。（編集不可）
ソースフォルダ	モデルクラスを生成するソースフォルダを選択、設定します。
パッケージ	モデルクラスのパッケージを選択、設定します。
モデルクラス名	モデルクラスの名前を設定します。
マッピングリスト	モデルクラスのインスタンスを設定します。 また、既存の DB テーブルを利用したい場合、データ・ソース・エクスプローラーからドラッグ&ドロップで指定することもできます。
生成モジュール	生成するフレームワークを指定します。 im-JavaEE フレームワークの場合、生成モジュールは「 IM標準」を指定します。

DAO

項目

概要

- 本項では、 im-JavaEE フレームワークエディタにおける、「 DAO 」タブの設定内容について説明します。



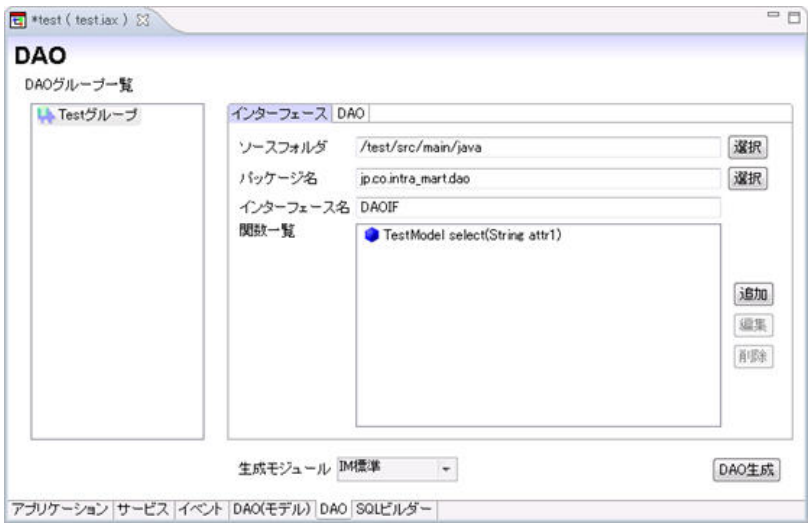
操作方法

1. DAO グループを作成します。
2. DAO のインタフェースクラスを作成するため、インタフェースタブ内に必要な情報を設定します。
3. DAO タグを選択し、 DAO リストに実装クラスの情報を入力していきます。
4. DAO リストから、実装クラスの情報にフォーカスを当ててメソッドリストにメソッドを作成します。
5. 作成したメソッドにフォーカスを当て、 SQL ビルダで SQL と入出力のマッピングを行います。
6. 4. と5. を何度か行い DAO に必要なメソッドを揃えた後、エディタを保存します。
7. 「 DAO生成」のボタンを押下し、 DAO のインタフェース、および実装クラスを生成します。

設定項目

各設定項目は DAO グループを作成、フォーカスを当てた上で作成します。

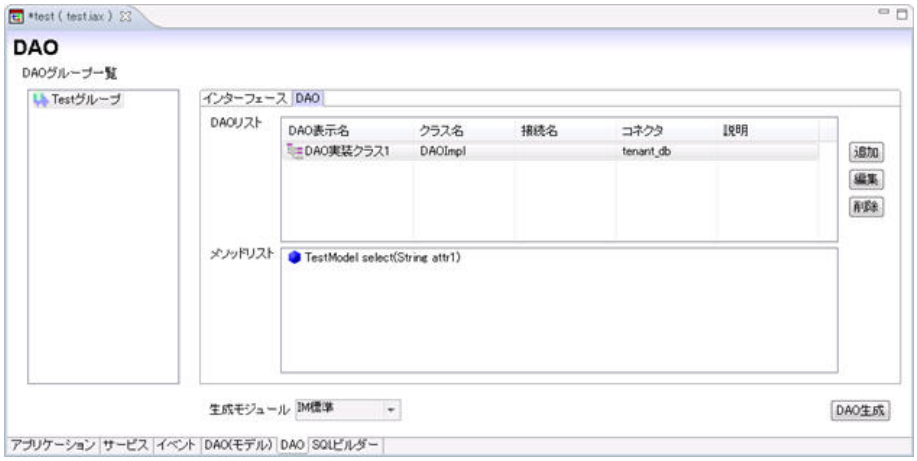
- インタフェース



項目	説明
ソースフォルダ	インタフェースクラスを生成するソースフォルダを設定します。

項目	説明
パッケージ名	インタフェースクラスのパッケージ名を指定します。
インタフェース名 | インタフェースクラスのクラス名を指定します。	
関数一覧	このインタフェースクラスに定義する関数を設定します。 この関数一覧では1つの関数に対し、以下の3つの情報を設定します。 * 関数名 * 関数返却タイプ * 関数引数一覧
生成モジュール	生成するフレームワークを指定します。 im-JavaEE フレームワークの場合、生成モジュールは「 IM標準」を指定します。

- DAO



DAO リストでは以下の情報を設定する必要があります。

項目	説明
DAO表示名	この DAO リストの表示ラベルを設定します。
ソースフォルダ	DAO の実装クラスを生成するソースフォルダを指定します。
パッケージ名	DAO の実装クラスを生成するパッケージを指定します。
DAOクラス名	DAO の実装クラスのクラス名を指定します。
コネクタ	この実装クラスで利用するコネクタを指定します。 このコネクタはプロジェクトの設定で、 Web アーカイブディレクトリを指定する必要があります。 ここで選択出来るコネクター覧は「 <%Webアーカイブディレクトリのパス%>/WEB-INF/classes/data-config.xml 」で設定されている情報です。
接続名	この実装クラスのコネクタに対する接続ラベルを指定します。
説明	この DAO の説明を記述します。

DAO リストにフォーカスを当てると、メソッドリストにインタフェースで設定したメソッド一覧が表示されます。
メソッドリストから、1つのメソッドにフォーカスを当てると、 SQL ビルダで SQL の設定ができます。

SQLビルダ

項目

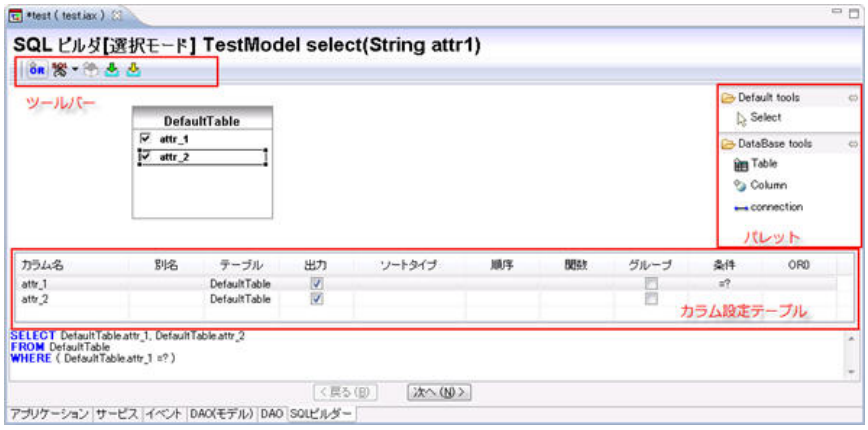
概要

- 本項では、 im-JavaEE フレームワークエディタにおける、「 SQLビルダ」タブの設定内容について説明します。

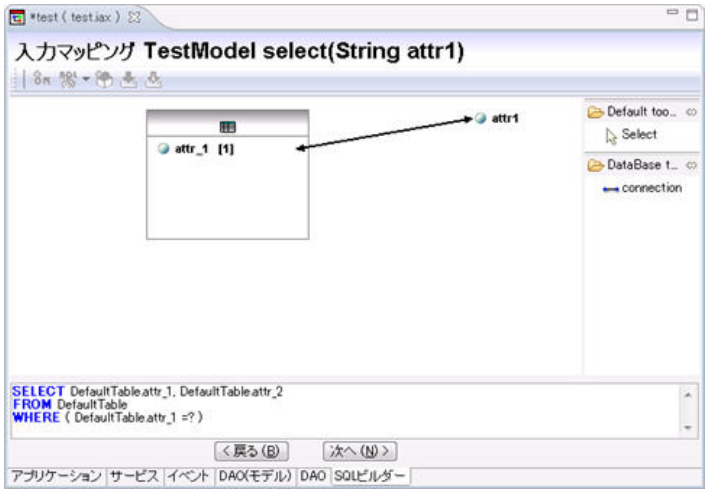
操作手順

1. DAO タブのメソッドリストから SQL の設定を行いたいメソッドを選択し、

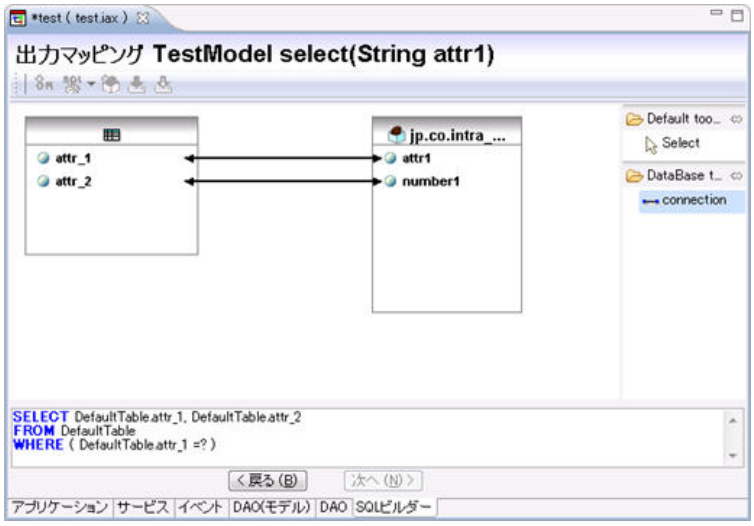
2. SQL ビルダ[選択モード]で SQL の設計を行います。
   - SQL ビルダでは、 GUI モードとテキストモードでの SQL の設計が可能です。
   - GUI モードでは、データ・ソース・エクスプローラから既存のテーブルを設定可能です。
   - GUI モードからテキストモードへ変更する場合、テキストモードの個所で右クリックして、「直接編集」を選択してください。
     テキストモードの編集を実行すると、 GUI で設定した情報は利用できなくなります。
   - テキストモードから GUI モードへ変更する場合、テキストモードの個所で右クリックして、「 GUI編集」を選択してください。
     GUI モードの編集を実行すると、テキストで設定した情報は利用できなくなります。



3. 入力マッピング画面で、メソッドに定義した引数と SQL のパラメータをマッピングします。



4. 出力マッピング画面で、メソッドに定義した返り値と SQL の実行結果となるカラムをマッピングします。



5. エディタを保存すると、 DAO のメソッドに対する SQL の設定が完了です。

SQLビルダ[選択モード] GUI画面のメニュー

SQL ビルダの GUI モードで利用できるメニューの説明をします。
なお、この画面では各画面を以下のように名前付けをして説明していきます。

- ツールバー

項目	説明
カラム追加	カラム設定テーブルに条件句となるORの項目を追加します。
SQLタイプ変更	GUIで編集するSQLのCRUDタイプを変更します。
DAOモデル追加	モデルクラスを画面内に挿入します。
CSVインポート	CSVで作成されたデータを挿入します。
SQLインポート	SQL文をインポートします。

- パレット

項目	説明
Table	GUI画面上にテーブルを配置します。
Column	GUI画面のテーブル内にカラムを配置します。
Connection	異なるテーブル内にあるカラムとカラムを結びつけます。

- カラム設定テーブル

項目	説明
カラム名	カラム名を表示します。カラム設定テーブル上ではカラム名の編集はできません。 カラム名の変更はGUI画面上で操作できます。
別名	カラムのエイリアスを指定します。
テーブル	テーブル名を表示します。カラム設定テーブル上ではテーブル名の編集はできません。 テーブル名の変更はGUI画面上で操作できます。
出力	このカラムをSQLの実行結果として取得するかどうかを指定します。
ソートタイプ	このカラムを出力することを選択した場合、ソート順を設定します。
順序	このカラムをソートするときのソートの優先順位を選択します。
関数	このカラムの出力結果となる関数を設定します。
グループ	このカラムをGROUP BY句で指定します。
条件	WHERE句の条件を設定します。
OR	WHEREの条件句をORで追加します。

デバッグ（Java開発）

概要

- 本項では im-JavaEE フレームワークで開発したアプリケーションのデバッグの方法について説明します。
- デバッグの方法として2通りの方法があります。
  1. サーバ・ビューから起動する方法
  2. resin.exe からサーバを起動し、Javaリモートデバッグを利用する方法

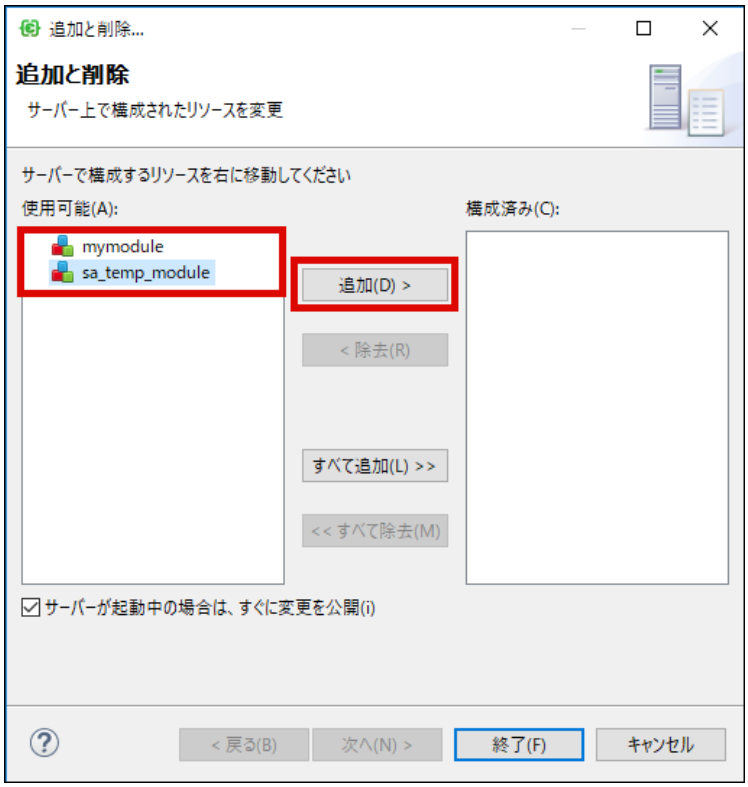
> **コラム**
>
> デバッグサーバを起動する場合は、ワークスペース上にできた Server プロジェクトを開いた状態にしておく必要があります。

サーバ・ビューから起動する方法

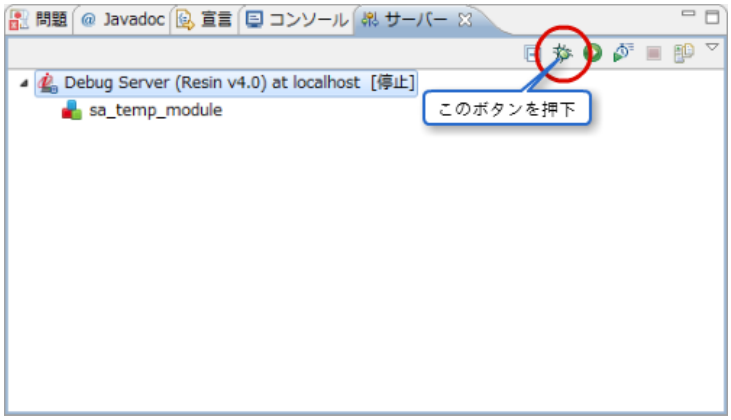
- デバッグ対象のモジュールプロジェクトがデバッグサーバに関連付いてない場合、以下の作業を行います。

1. サーバ・ビューからデバッグサーバを右クリックし、「追加と削除」を選択します。

2. 「追加と削除」ウィザードで、対象のモジュールプロジェクトを追加し、終了ボタンを押下します。

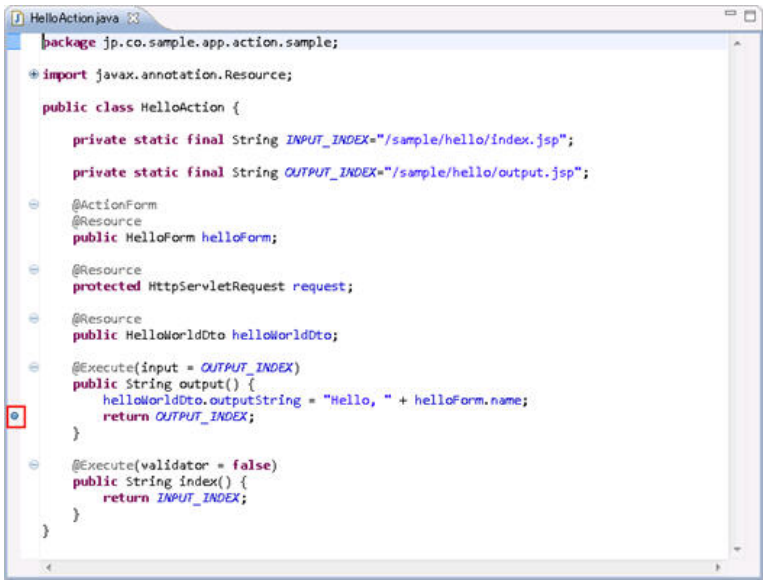


3. デバッグサーバにモジュールプロジェクトが追加されますので、デバッグサーバをデバッグ起動します。



> **コラム**
>
> サーバ・ビューへの デバッグ サーバへの設定の仕方は、 e Builder セットアップガイドの「開発環境用のResin設定」を参照してください。

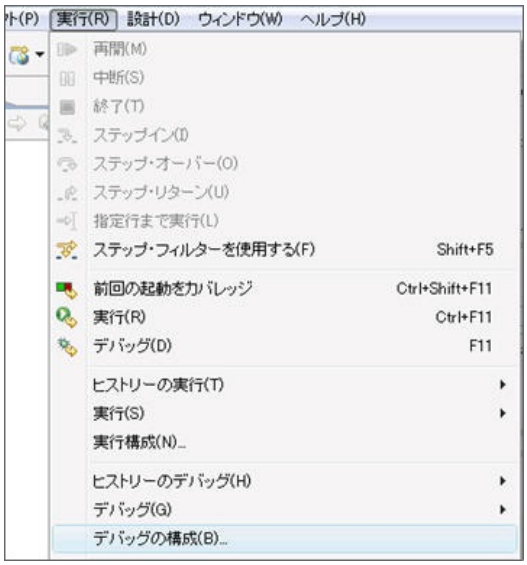
4. デバッグを行うプロジェクト内の Java ファイルを開いてブレークポイントを設定します。



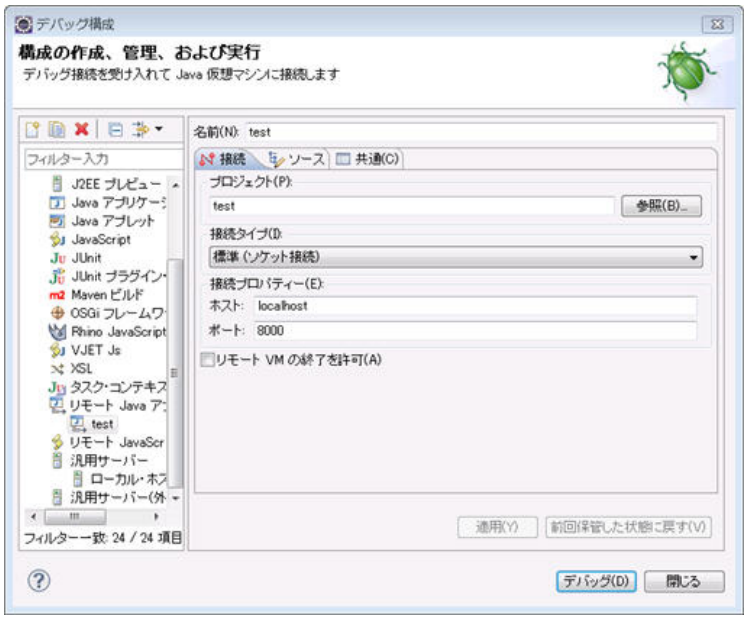
5. ブラウザでデバッグ対象の処理を実行します。
6. Eclipse のデバッグモードが起動します。

resin.exe からサーバを起動し、Javaリモートデバッグを利用する方法

1. <% resin_home %>/conf フォルダにある resin.properties にある jvm_args に以下の引数を追加します。
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=<ポート番号>
2. Resin サーバを起動します。
3. Eclipse のツール・バーから「実行」→「デバッグの構成」を選択します。



4. メニューから 「リモート Java アプリケーション」 から「新規」を選択し、新しい起動構成を作成します。



5. 接続タブで以下のように設定します。
   - 名前欄に、起動構成のラベルを入力します。（任意）
   - ソースにデバッグ対象のプロジェクトを選択します。
   - 接続タイプは「標準（ソケット接続）」を設定します。
   - 接続プロパティのホスト名は、サーバ起動時のホスト名を、ポート番号は jvm_args に指定したポート番号を記入します。
6. ソースタブでデバッグの対象となるプロジェクトを選択し、追加します。
7. デバッグの構成を適用し、実行します。
8. デバッグを行うプロジェクト内の Java ファイルを開いてブレークポイントを設定します。
9. ブラウザでデバッグ対象の処理を実行します。
10. Eclipse のデバッグモードが起動します。

SAStruts + S2JDBC フレームワーク開発で利用できる機能

ここでは、 e Builder で利用できる SAStruts + S2JDBC フレームワーク開発用の基本機能を紹介します。

JSP エディタ

項目

概要

- 本項では、 JSP の編集を行うための JSP エディタの機能について説明します。
  e Builder で提供されている JSP エディタは、 Eclipse WTP で提供されている Web ページ・エディタの機能を拡張しています。

前提条件

- e Builder を利用し、モジュール・プロジェクト内に編集可能なJSPファイルが存在すること

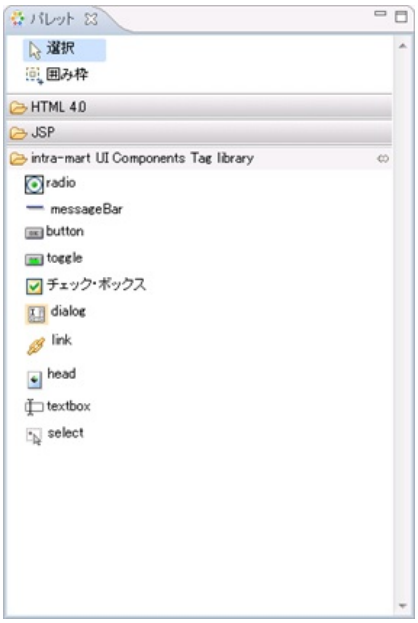
JSPエディタの編集機能

JSP エディタは編集したい JSP ファイルに対して、開くエディタに「Web ページ・エディタ」を選択して開きます。
Web ページ・エディタを拡張した JSP エディタは、以下の3つの機能で構成されています。

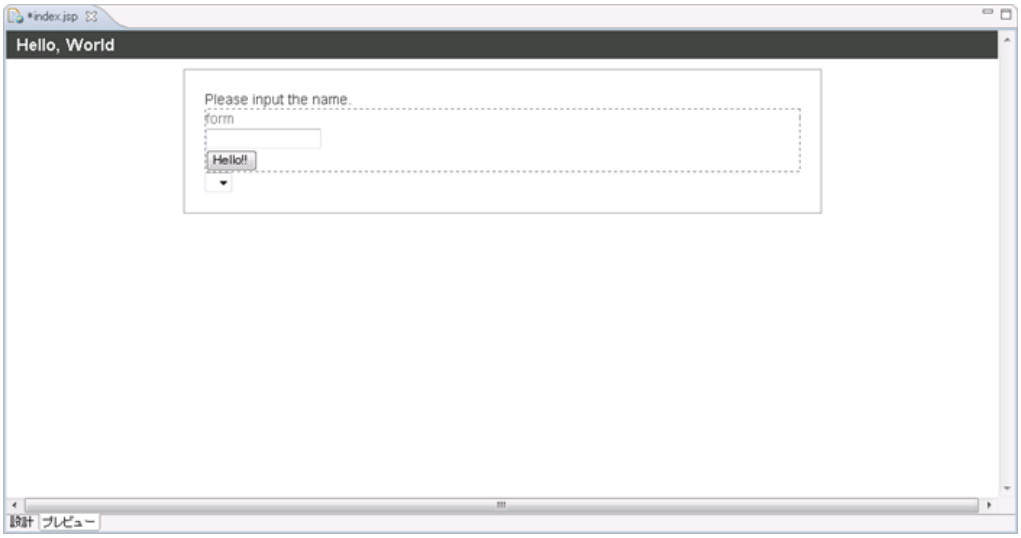
- 設計機能

  JSP エディタの左下に表示されている「設計」タブを選択することにより利用できます。
  JSP の画面設計をデザインベース、およびテキストベースで編集します。
  この機能は、 Eclipse WTP で提供されているものと同等です。

- パレット機能
  「パレット」ビューを表示することによって利用できます。
  intra-mart Accel Platform で利用可能な JSP タグをパレットから提供します。
  パレットから利用したいアイテムを選択し、ドラッグ&ドロップでアイテムを配置できます。



- プレビュー機能
  JSP エディタの左下に表示されている「プレビュー」タブを選択することにより利用できます。
  編集時点での JSP ファイルのソースから、実際のブラウザ上で表示される画面を表示します。
  プレビューでは、スタイルシートが適用された状態で表示されます。



> **コラム**
>
> なお、 JSP ファイル内で message タグを利用した場合、プレビュー画面には OS のロケーションの言語からプロパティメッセージを読み込みます。
> 別のロケーションの言語を利用したい場合は、 eBuilder8.ini に以下の引数を指定してください。
> ※国をアメリカ、言語を英語に設定したい場合
> -Duser.language=en
> -Duser.country=US

デバッグ（Java開発）

概要

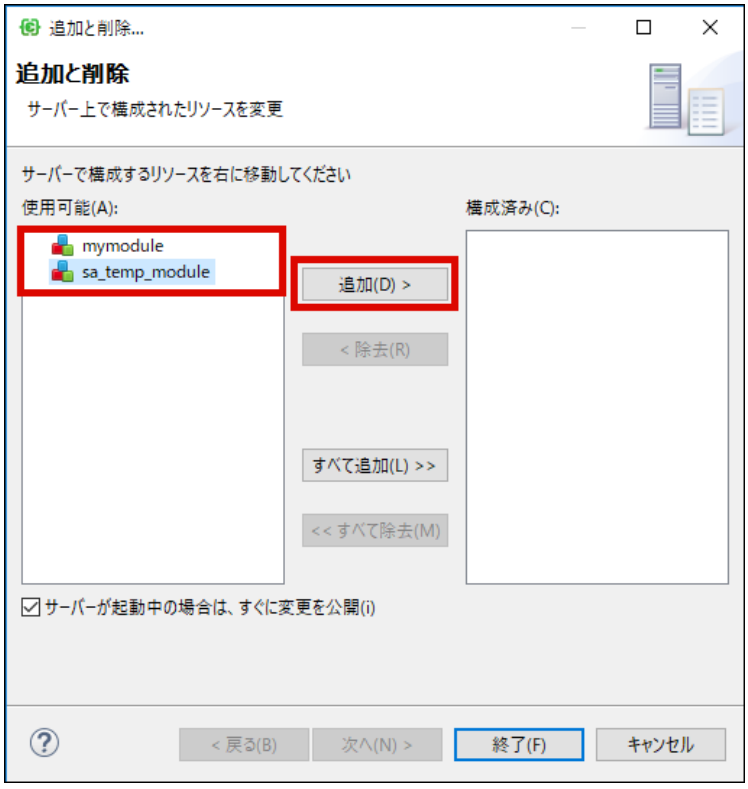
- 本項では SAStruts + S2JDBC フレームワークで開発したアプリケーションのデバッグの方法について説明します。
- デバッグの方法として2通りの方法があります。
  1. サーバ・ビューから起動する方法
  2. resin.exe からサーバを起動し、Javaリモートデバッグを利用する方法

> **コラム**
>
> デバッグサーバを起動する場合は、ワークスペース上にできた Server プロジェクトを開いた状態にしておく必要があります。

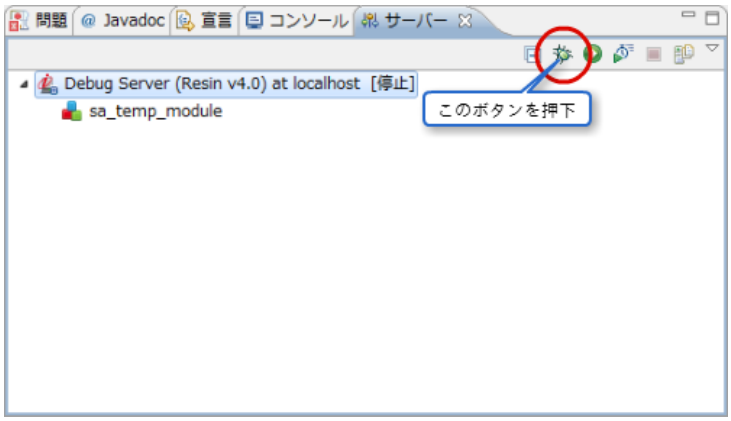
- デバッグ対象のモジュールプロジェクトがデバッグサーバに関連付いてない場合、以下の作業を行います。

1. サーバ・ビューからデバッグサーバを右クリックし、「追加と削除」を選択します。

2. 「追加と削除」ウィザードで、対象のモジュールプロジェクトを追加し、終了ボタンを押下します。

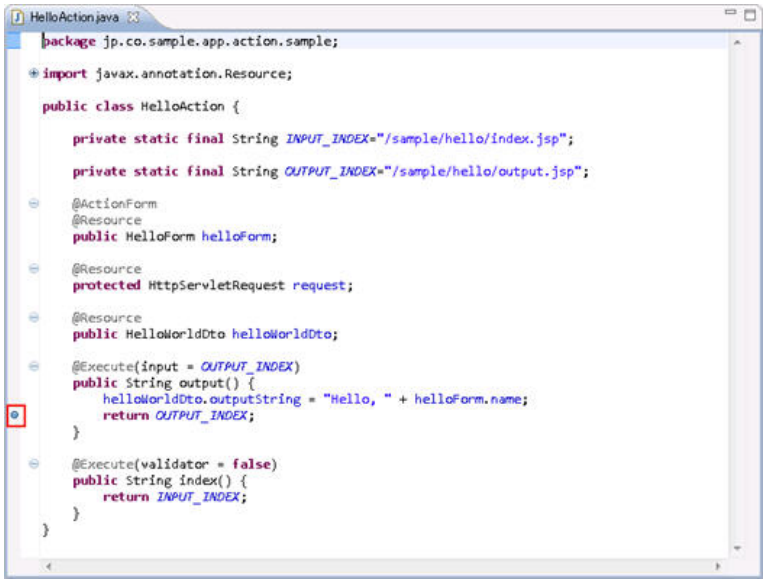


3. デバッグサーバにモジュールプロジェクトが追加されますので、デバッグサーバをデバッグ起動します。



> **コラム**
>
> サーバ・ビューへの デバッグ サーバへの設定の仕方は、 e Builder セットアップガイドの
> 「開発環境用のResin設定」を参照してください。

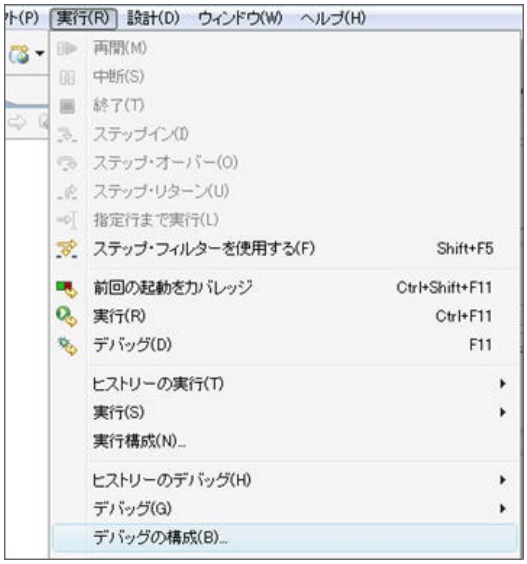
4. デバッグを行うプロジェクト内の Java ファイルを開いてブレークポイントを設定します。



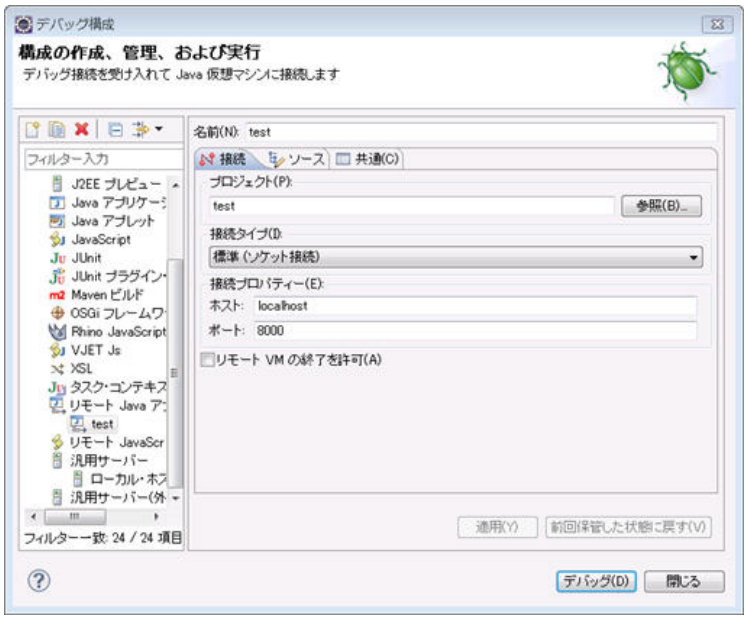
5. ブラウザでデバッグ対象の処理を実行します。
6. Eclipse のデバッグモードが起動します。

resin.exe からサーバを起動し、Javaリモートデバッグを利用する方法

1. <% resin_home %>/conf フォルダにある resin.properties にある jvm_args に以下の引数を追加します。
   -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=<ポート番号>
2. Resin サーバを起動します。
3. Eclipse のツール・バーから「実行」→「デバッグの構成」を選択します。



4. メニューから 「リモート Java アプリケーション」 から「新規」を選択し、新しい起動構成を作成します。



5. 接続タブで以下のように設定します。
   - 名前欄に、起動構成のラベルを入力します。（任意）
   - ソースにデバッグ対象のプロジェクトを選択します。
   - 接続タイプは「標準（ソケット接続）」を設定します。
   - 接続プロパティのホスト名は、サーバ起動時のホスト名を、ポート番号は jvm_args に指定したポート番号を記入します。
6. ソースタブでデバッグの対象となるプロジェクトを選択し、追加します。
7. デバッグの構成を適用し、実行します。
8. デバッグを行うプロジェクト内の Java ファイルを開いてブレークポイントを設定します。
9. ブラウザでデバッグ対象の処理を実行します。
10. Eclipse のデバッグモードが起動します。

TERASOLUNA Server Framework for Java (5.x) for Accel Platform 開発で利用できる機能

ここでは、 e Builder で利用できる TERASOLUNA Server Framework for Java (5.x) for Accel Platform 開発用の基本機能を紹介します。

JSP エディタ

項目

概要

- 本項では、 JSP の編集を行うための JSP エディタの機能について説明します。
  e Builder で提供されている JSP エディタは、 Eclipse WTP で提供されている Web ページ・エディタの機能を拡張しています。

前提条件

- e Builder を利用し、モジュール・プロジェクト内に編集可能なJSPファイルが存在すること

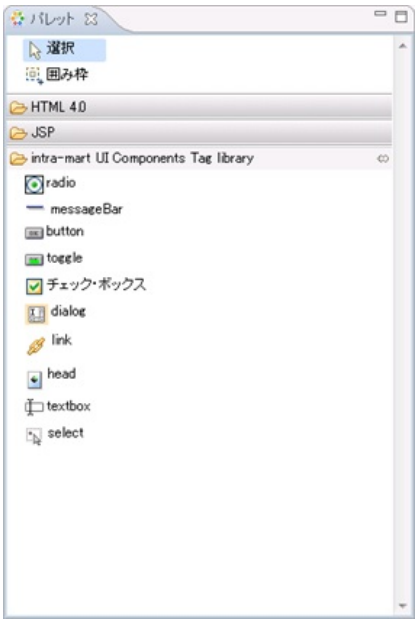
JSPエディタの編集機能

JSP エディタは編集したい JSP ファイルに対して、開くエディタに「Web ページ・エディタ」を選択して開きます。
Web ページ・エディタを拡張した JSP エディタは、以下の3つの機能で構成されています。

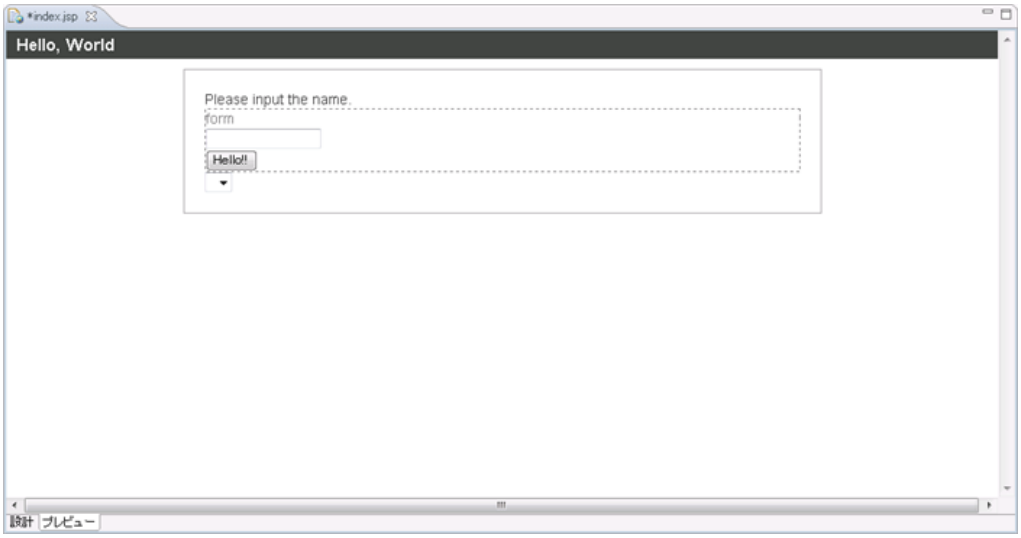
- 設計機能

  JSP エディタの左下に表示されている「設計」タブを選択することにより利用できます。
  JSP の画面設計をデザインベース、およびテキストベースで編集します。
  この機能は、 Eclipse WTP で提供されているものと同等のものです。

- パレット機能
  「パレット」ビューを表示することによって利用できます。
  intra-mart Accel Platform で利用可能な JSP タグをパレットから提供します。
  パレットから利用したいアイテムを選択し、ドラッグ&ドロップでアイテムを配置できます。



- プレビュー機能
  JSP エディタの左下に表示されている「プレビュー」タブを選択することにより利用できます。
  編集時点での JSP ファイルのソースから、実際のブラウザ上で表示される画面を表示します。
  プレビューでは、スタイルシートが適用された状態で表示されます。



> **コラム**
>
> なお、 JSP ファイル内で message タグを利用した場合、プレビュー画面には OS のロケーションの言語からプロパティメッセージを読み込みます。
> 別のロケーションの言語を利用したい場合は、 eBuilder8.ini に以下の引数を指定してください。
> ※国をアメリカ、言語を英語に設定したい場合
> -Duser.language=en
> -Duser.country=US

デバッグ（Java開発）

概要

- 本項では TERASOLUNA Server Framework for Java (5.x) for Accel Platform で開発したアプリケーションのデバッグの方法について説明します。
- デバッグの方法として2通りの方法があります。
  1. サーバ・ビューから起動する方法
  2. resin.exe からサーバを起動し、Javaリモートデバッグを利用する方法

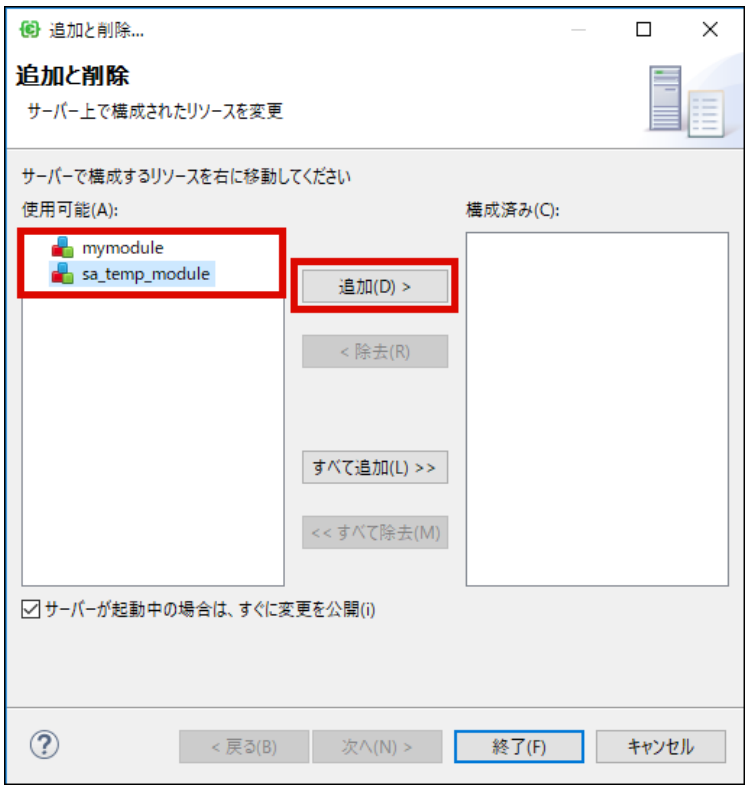
> **コラム**
>
> デバッグサーバを起動する場合は、ワークスペース上にできた Server プロジェクトを開いた状態にしておく必要があります。

サーバ・ビューから起動する方法

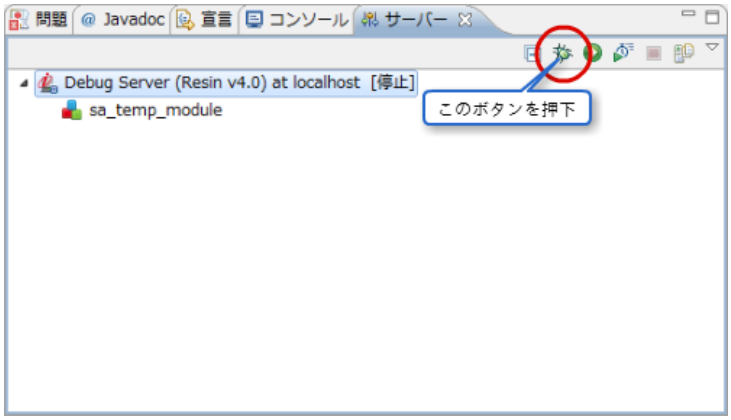
- デバッグ対象のモジュールプロジェクトがデバッグサーバに関連付いてない場合、以下の作業を行います。

1. サーバ・ビューからデバッグサーバを右クリックし、「追加と削除」を選択します。

2. 「追加と削除」ウィザードで、対象のモジュールプロジェクトを追加し、終了ボタンを押下します。

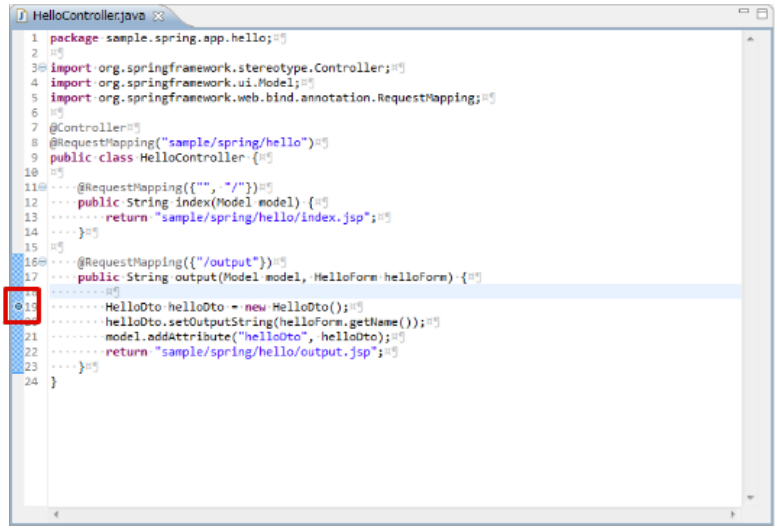


3. デバッグサーバにモジュールプロジェクトが追加されますので、デバッグサーバをデバッグ起動します。



> **コラム**
>
> サーバ・ビューへの デバッグ サーバへの設定の仕方は、 e Builder セットアップガイドの
> 「開発環境用のResin設定」を参照してください。

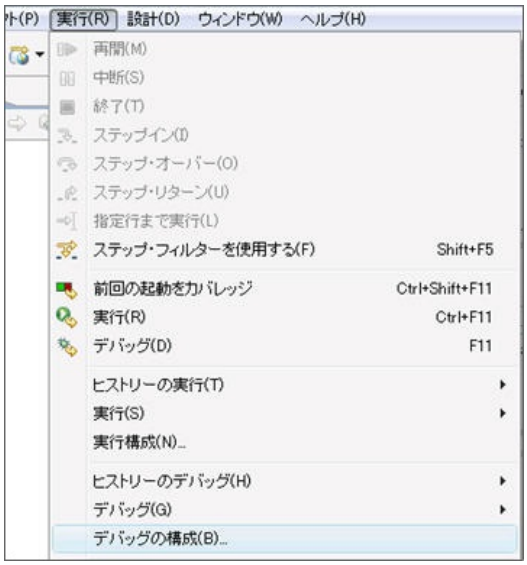
4. デバッグを行うプロジェクト内の Java ファイルを開いてブレークポイントを設定します。



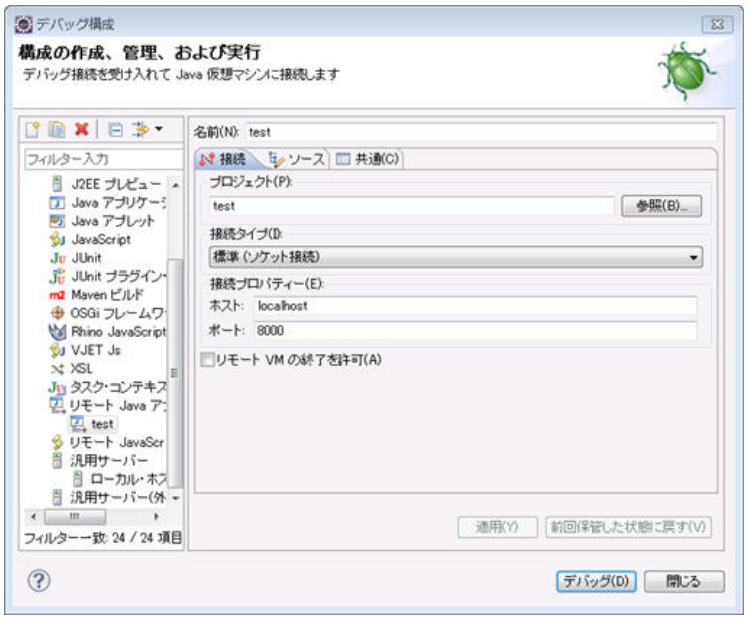
5. ブラウザでデバッグ対象の処理を実行します。
6. Eclipse のデバッグモードが起動します。

resin.exe からサーバを起動し、Javaリモートデバッグを利用する方法

1. <% resin_home %>/conf フォルダにある resin.properties にある jvm_args に以下の引数を追加します。
   -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=<ポート番号>
2. Resin サーバを起動します。
3. Eclipse のツール・バーから「実行」→「デバッグの構成」を選択します。



4. メニューから 「リモート Java アプリケーション」 から「新規」を選択し、新しい起動構成を作成します。



5. 接続タブで以下のように設定します。
   - 名前欄に、起動構成のラベルを入力します。（任意）
   - ソースにデバッグ対象のプロジェクトを選択します。
   - 接続タイプは「標準（ソケット接続）」を設定します。
   - 接続プロパティのホスト名は、サーバ起動時のホスト名を、ポート番号は jvm_args に指定したポート番号を記入します。
6. ソースタブでデバッグの対象となるプロジェクトを選択し、追加します。
7. デバッグの構成を適用し、実行します。
8. デバッグを行うプロジェクト内の Java ファイルを開いてブレークポイントを設定します。
9. ブラウザでデバッグ対象の処理を実行します。
10. Eclipse のデバッグモードが起動します。

業務スケルトンで利用できる機能

ここでは e Builder における業務スケルトンの利用方法に関して説明します。

業務スケルトンとは

項目

概要

業務スケルトンはe Builderを利用したintra-mart向けアプリケーションの開発生産性の向上を目的とした開発補助ツールです。

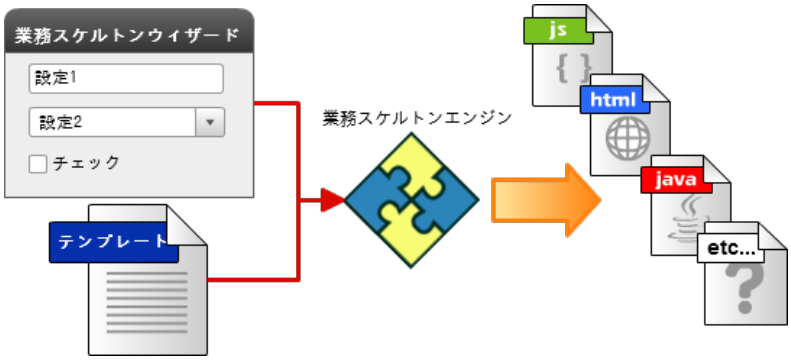
コンセプト

業務スケルトンは下記のコンセプトを元に作られてます。

- 初回の開発生産性の向上
- 品質の均一化
- 開発方式の統一
- 開発工数の削減

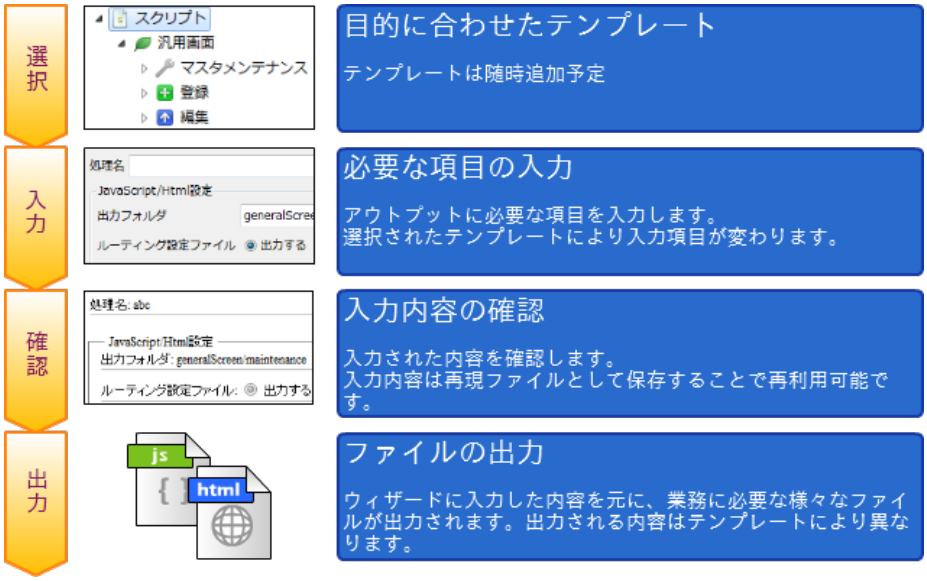
機能

業務スケルトンはe Builder上で利用可能な”拡張可能な”ウィザードです。
ウィザードより、様々なアプリケーション開発向けの雛形(業務テンプレート)を利用し、アプリケーション開発における定型的なコード、設定等を自動的に出力する為の仕組みです。
業務テンプレートを元とした定型的なコード、設定による開発方式の統一、品質の均一化が可能となります。
また、独自の業務テンプレートを作成する事により、既存のアプリケーション開発で利用されてきたコード、設定等の再利用を促進する事が可能となります。



業務スケルトンの流れ

ここでは、業務スケルトンを利用した開発の流れについて説明します。
業務スケルトンウィザードの実際の実行方法に関しては、 *業務スケルトンの実行* を参照して下さい。
業務スケルトンを利用した開発の流れでは、初回に業務スケルトンウィザードを利用し、アプリケーション開発における雛形を生成します。
業務スケルトンウィザードを利用するに当たり必要な作業、手順は下記の図を参照下さい。



その後、生成された雛形をカスタマイズし、開発目的に沿った形のアプリケーションへと開発を行います。アプリケーション開発中、開発後において雛形(テンプレート)化が可能であるコード、設定が存在する場合は、業務テンプレートを作成し、以降の開発に有用なテンプレートを作成する事をお勧めします。
業務テンプレートの作成方法に関しては、 *テンプレート作成チュートリアル* を参照してください。

業務スケルトンの実行

項目

概要

本項では、業務スケルトンを実行する為の手順について説明します。

プロジェクトの作成

業務スケルトンの実行には必ずプロジェクトが必要となります。
プロジェクトが未作成の場合は、プロジェクトを作成してください。

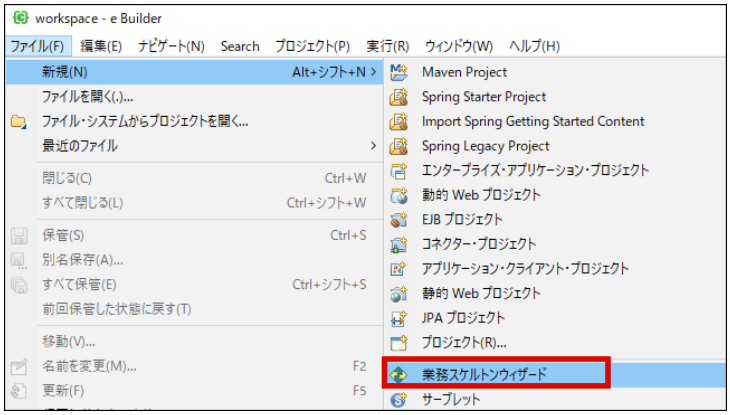
尚、業務スケルトンでサポートされているプロジェクトは下記の通りです。

- Module Project

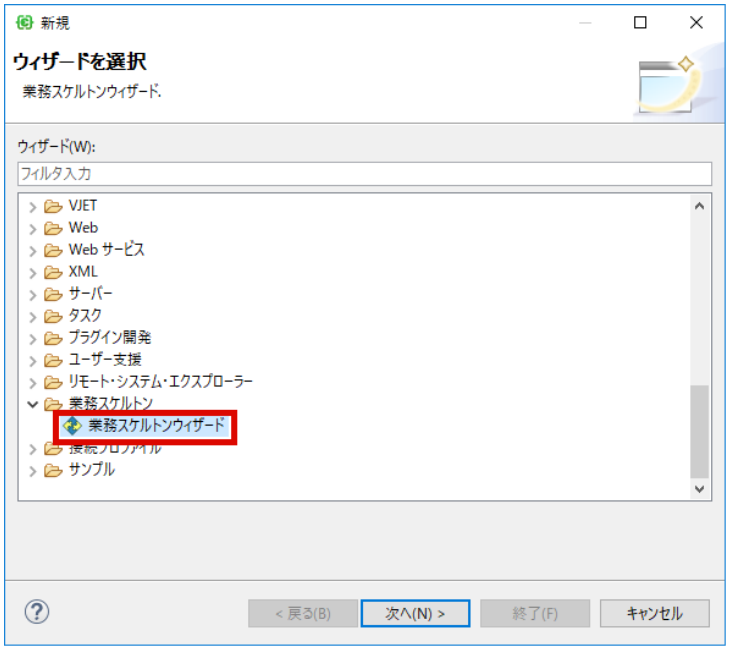
サポート外のプロジェクトから業務スケルトンウィザードを実行した場合、対応するテンプレートが存在しない為、業務スケルトンウィザードの実行は行えませんのでご注意下さい。

業務スケルトンウィザードの開始

業務スケルトンウィザードを開始するには、e Builder上部メニューより、
「ファイル」 → 「新規」 → 「業務スケルトンウィザード」を選択します。



「業務スケルトンウィザード」が新規作成メニューに存在しない場合は、パースペクティブをJavaに変更するか、
または「ファイル」 → 「新規作成」 → 「その他」を選択し、ウィザードの選択画面内より業務スケルトンウィザードを選択してください。



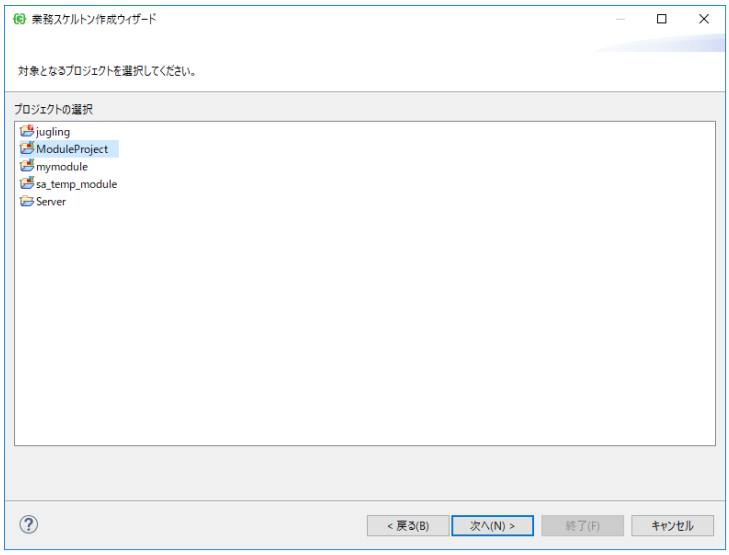
また、メニューバーより、業務スケルトンウィザード起動用のアイコンを押下することにより起動することも可能です。

業務スケルトンウィザード

業務スケルトンウィザードが表示されます。
ここでは、「次へ」を押下します。

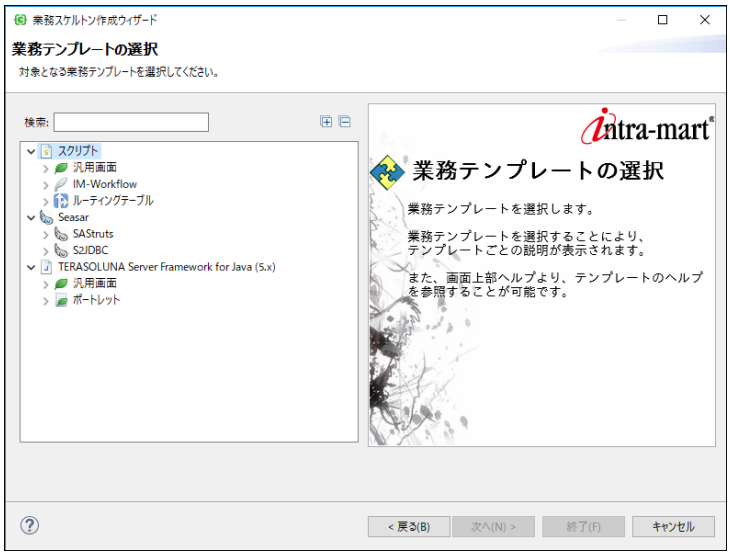
プロジェクトの選択

業務スケルトンウィザードを開始する際に、プロジェクトが選択されていなかった場合にのみこの画面が表示されます。
雛形の作成対象となるプロジェクトを選択し、「次へ」を押下します。



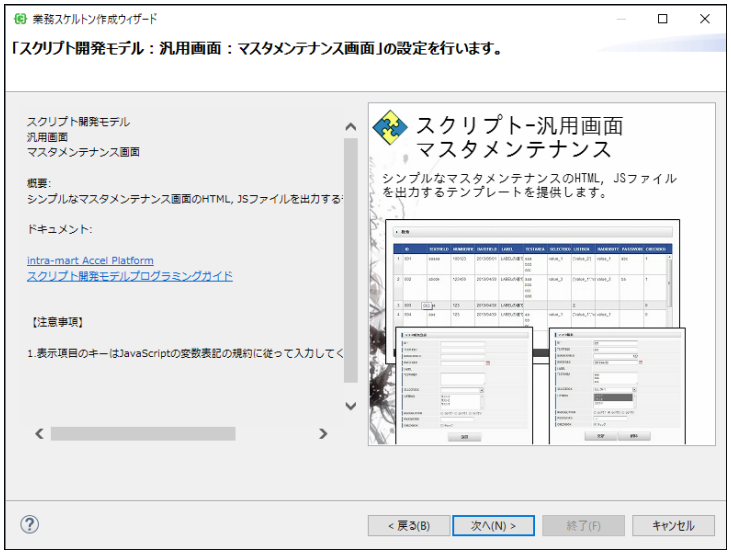
テンプレートの選択

選択されたプロジェクトに対し、生成可能な業務テンプレートの一覧が画面左部に表示されます。
一覧に表示されているテンプレートを選択する事により、テンプレートの説明等が画面右部に表示されます。
テンプレートの絞込みを行う場合は、画面左上部のテキストボックスにキーワードを入力する事により絞込みが可能です。
開発目的に沿ったテンプレートを選択し、「次へ」を押下します。



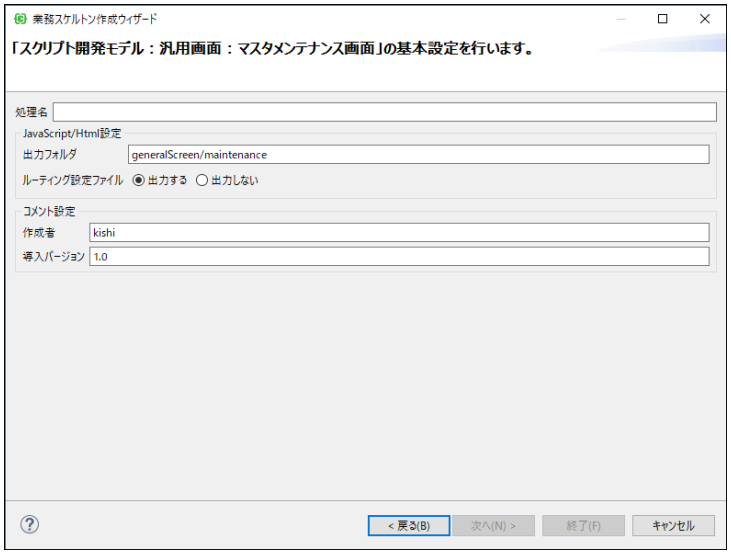
テンプレートの説明

テンプレートの説明が表示されます。
テンプレートにより制約、制限事項等が表記されている場合があります。
詳細な内容に関してはそれぞれのテンプレートの説明を参照下さい。
テンプレートの説明を確認後、「次へ」を押下します。



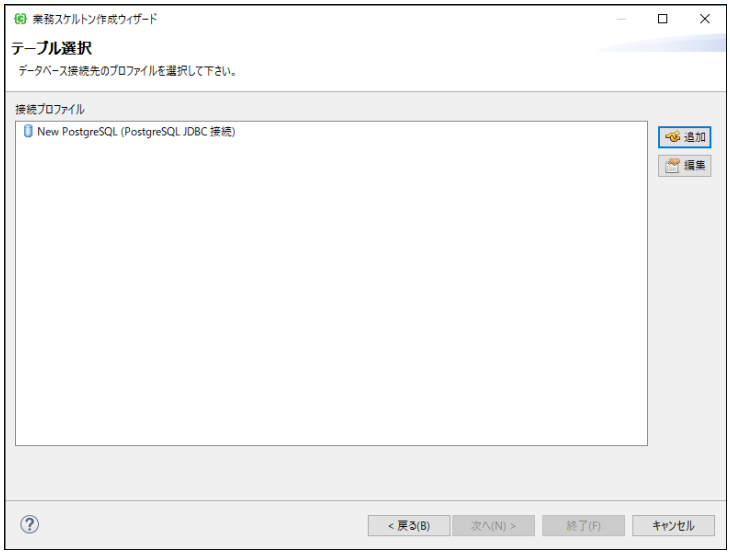
ウィザード項目の入力

選択された業務テンプレート毎に異なる入力画面が表示されます。
それぞれ業務テンプレートの利用に必要な入力項目となりますので、目的に沿った形での内容を入力してください。
入力画面は業務テンプレート毎に複数、また入力内容により可変で存在します。
それぞれの入力項目に対し、マウスを近づける事により、入力項目に関する説明(ツールチップ)が表示されます。
必要な内容を入力し、「次へ」を押下します。



データベース接続の選択

データベース接続が必要となるテンプレートの場合、データベース接続プロファイルの選択画面が表示されます。
データベース接続プロファイルが存在しない場合は *接続プロファイルの新規作成* が必要となります。



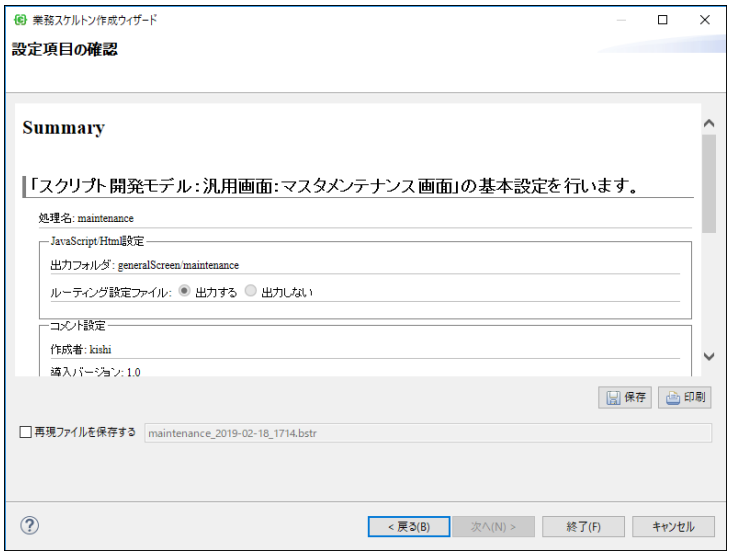
入力内容の確認とウィザードの完了

業務テンプレートの利用に必要な項目が全て入力された場合、入力内容の確認画面が表示されます。
入力内容に間違いが無いか確認を行ってください。

保存ボタンを押下する事により、入力内容の確認画面に表示されている内容をhtmlファイルで保存する事が可能です。
同様に印刷ボタンを押下する事により、入力内容の確認画面に表示されている内容を印刷する事が可能です。

再現ファイルを保存するチェックボックスを選択する事により、このウィザード入力項目を元に、業務テンプレートを利用するための再現ファイルがプロジェクト直下に保存されます。
必要に応じてチェックボックス右部にある再現用ファイルの名称を変更してください。

入力内容に不備が無ければ、「完了」ボタンを押下し業務スケルトンウィザードを完了します。



完了ボタンを押下する事により、選択された業務テンプレート、入力内容を元に最適化されたコード、設定がプロジェクトに対し出力されます。

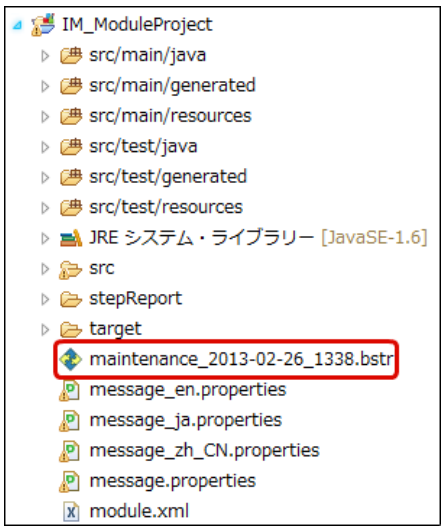
> **コラム**
>
> 出力されたJavaソースは、プロジェクトのコードフォーマッターの設定に依存します。
> コードフォーマッターの設定については
> プロジェクトを右クリック -> プロパティ -> Javaコード・スタイル -> フォーマッター で確認できます。

生成ファイル、設定の確認

業務テンプレートを共に、業務スケルトンウィザードが生成したファイルはコンソールビューに表示されます。
また、生成されたコード、設定を元に変更が必要な部分(実際の業務ロジックの記述が必要になる部分)は、タスクビュー、またはマーカービューを参照する事により確認する事が可能です。

再現ファイルの利用

業務スケルトンウィザードにおいて、再現ファイルを出力した場合、そのファイルを利用することにより再度同一の入力項目を元にウィザードの再現を行う事が可能です。
再現ファイルを利用する場合は、プロジェクト内に配備された再現ファイル(*.bstr)をダブルクリックし、再現用ウィザードを表示します。
再現用ウィザードを表示後、「完了」ボタンを押下する事により業務テンプレートの適用を再度行う事が可能です。



業務テンプレートのカスタマイズを行った場合、再現ファイル内容を変更した場合の動作に関しては保障されませんのでご注意下さい。

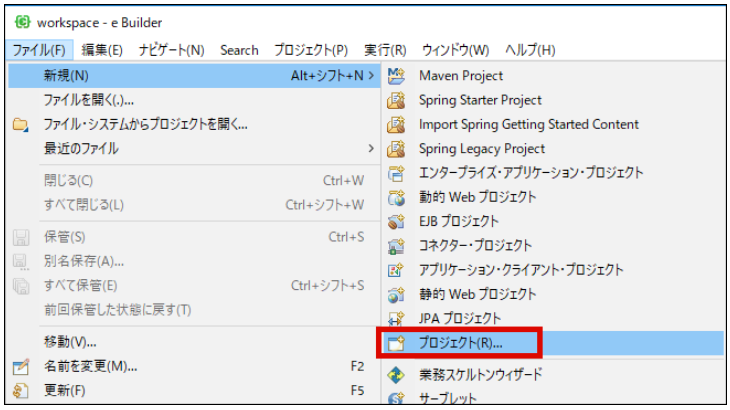
テンプレート作成チュートリアル

業務スケルトンはユーザが任意のテンプレートを作成し、拡張できます。
この項では、スクリプト開発用の簡単なhtmlファイルおよびjsファイルを出力する
業務スケルトンテンプレートを作成し、業務スケルトンの拡張方法について体験します。

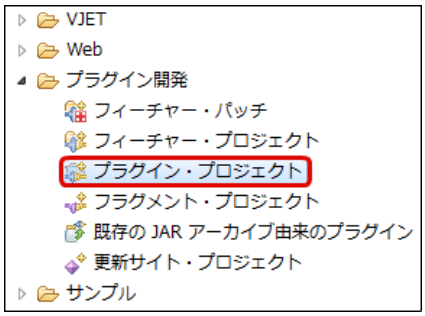
チュートリアルの流れ

プラグインプロジェクトの作成

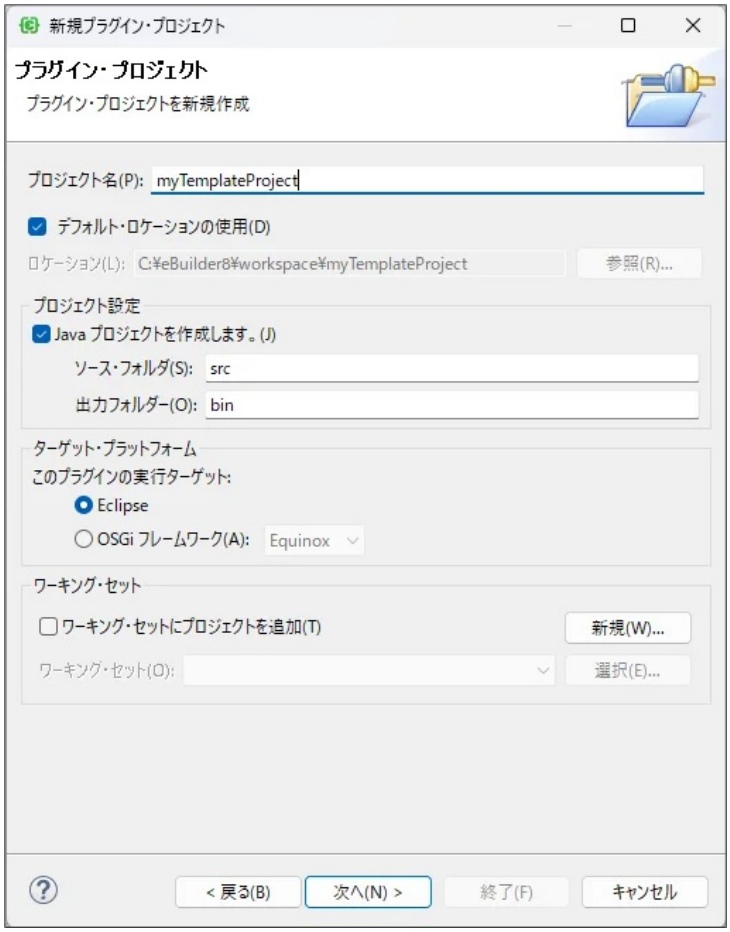
まず、e Builder上部ツールバーより「ファイル」→「新規作成」→「プロジェクト」を選択します。



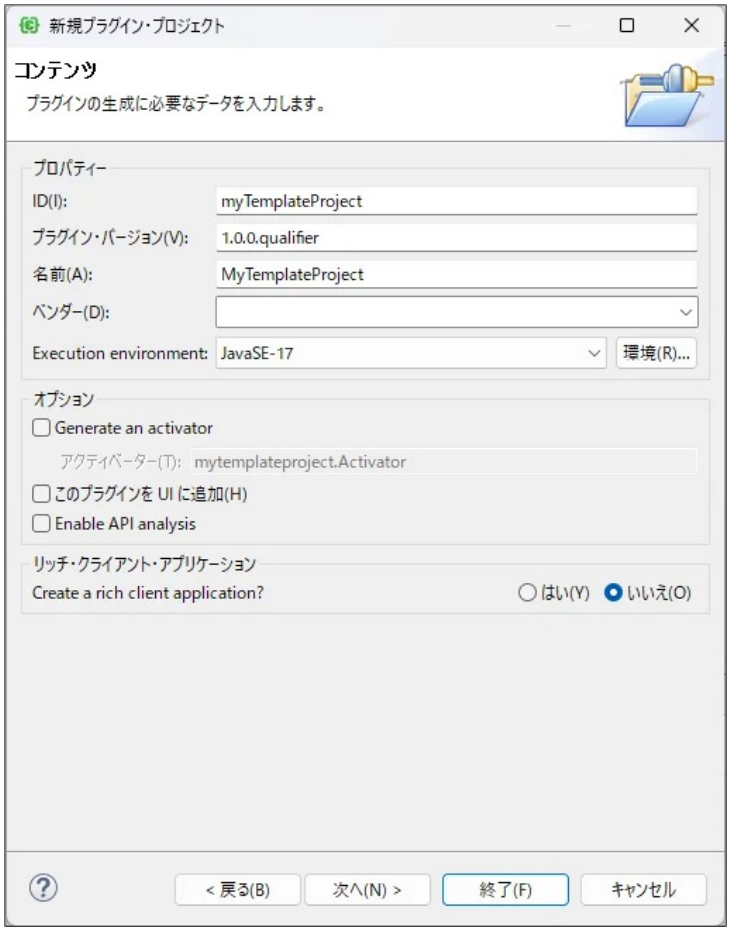
一覧より、「プラグイン開発」→「プラグイン・プロジェクト」を選択します。



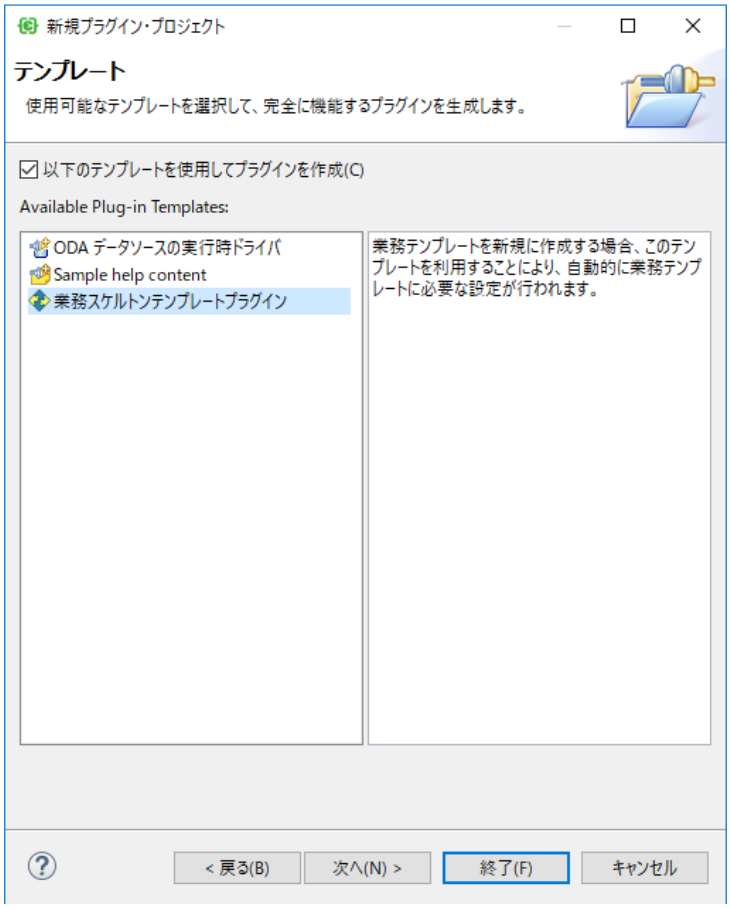
プロジェクトの設定をします。ここではプロジェクト名を「myTemplateProject」とし、その他項目はデフォルトとし、次へ進みます。



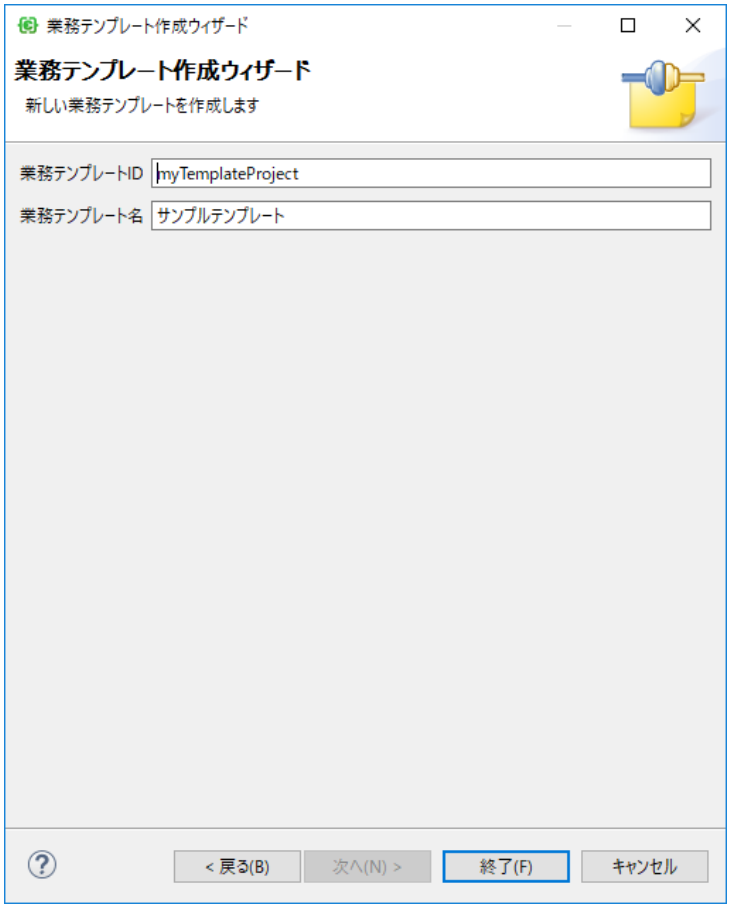
プラグインの設定をします。オプション項目の「Generate an activator」および「このプラグインをUIに追加」のチェックを外し、次へ進みます。



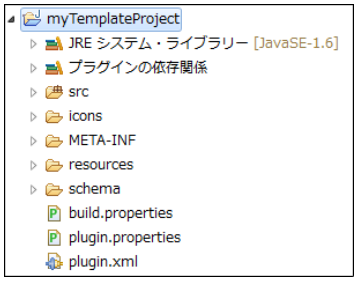
プラグインのテンプレート一覧が表示されますので、「業務スケルトンテンプレートプラグイン」を選択して次へ進みます。



業務スケルトンの設定をします。任意のID・名称を入力して終了ボタンをクリックします。

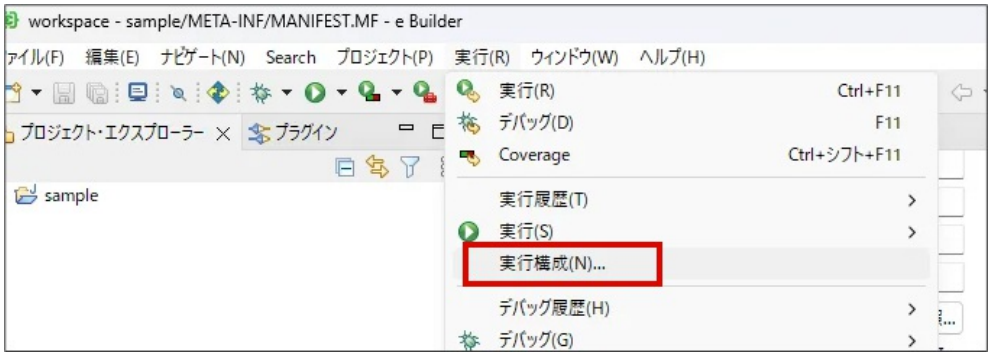


業務スケルトンテンプレート用のプラグインプロジェクトが出力されました。

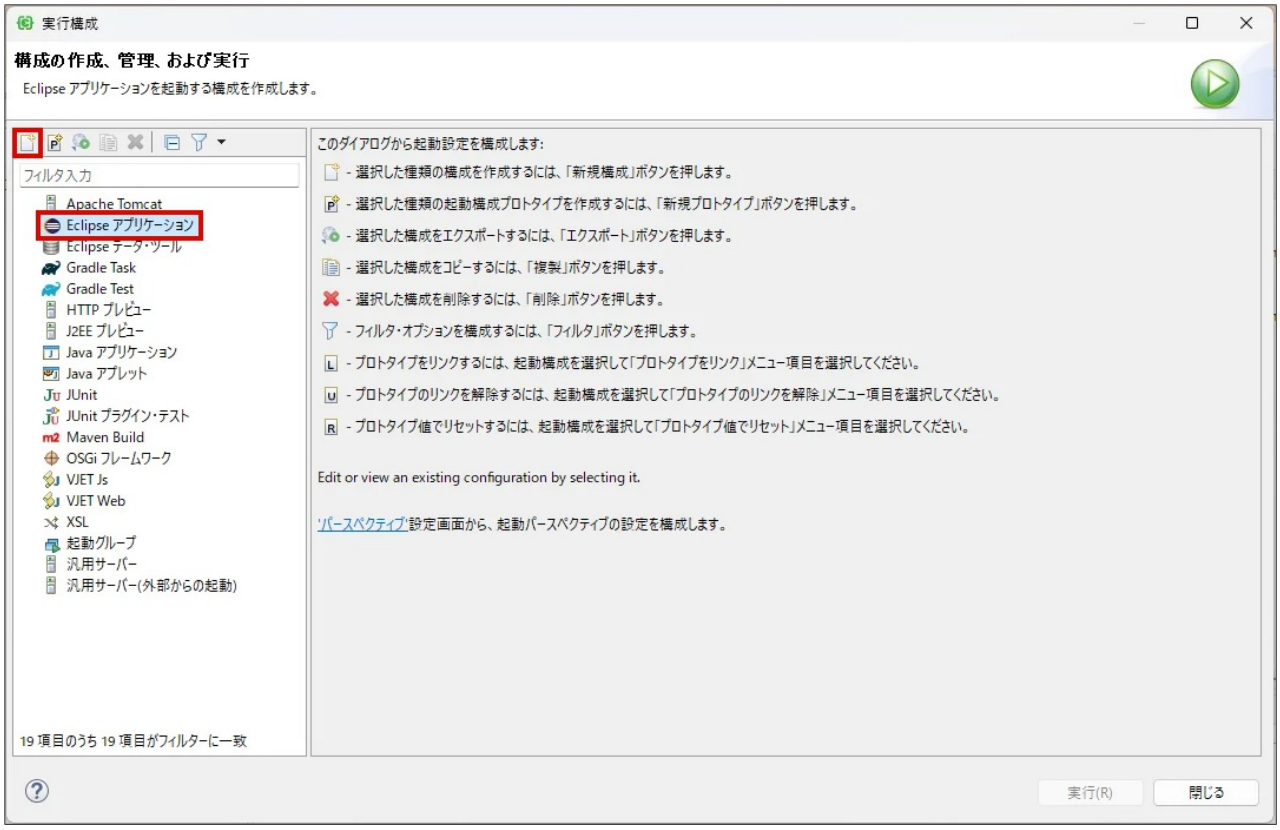


実行構成の設定

ツールバーより「実行」→「実行構成」を選択します。

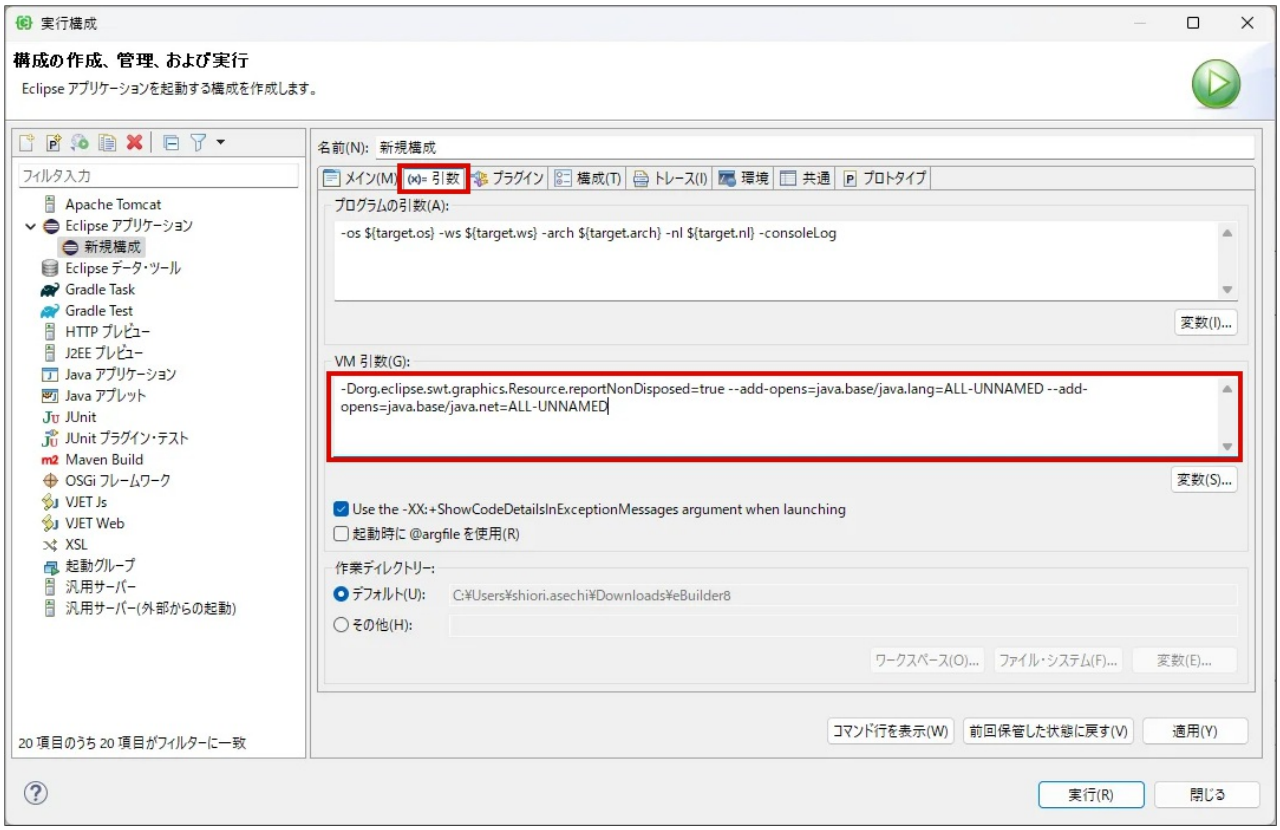


「Eclipse アプリケーション」を選択して、「新規の起動構成」アイコンをクリックします。



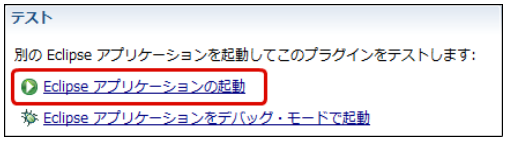
「引数」タブを開き、「VM引数」の入力欄に以下の引数を追加します。

```
--add-opens=java.base/java.lang=ALL-UNNAMED
--add-opens=java.base/java.net=ALL-UNNAMED
```

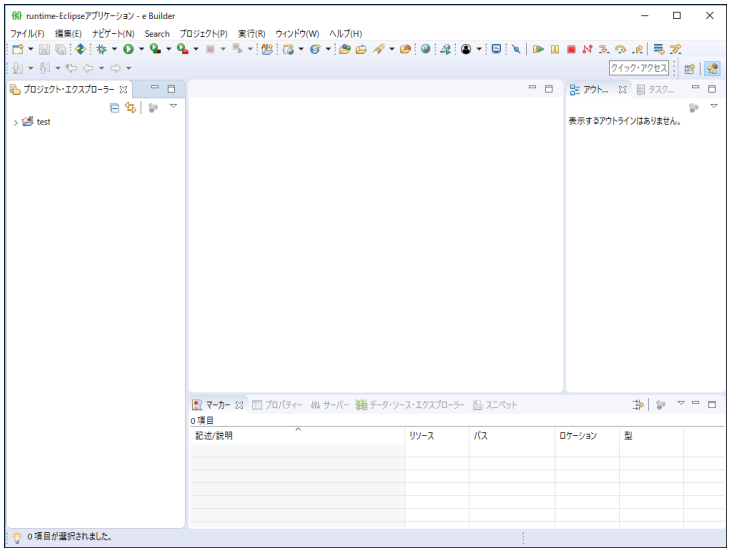


プラグインプロジェクトの実行

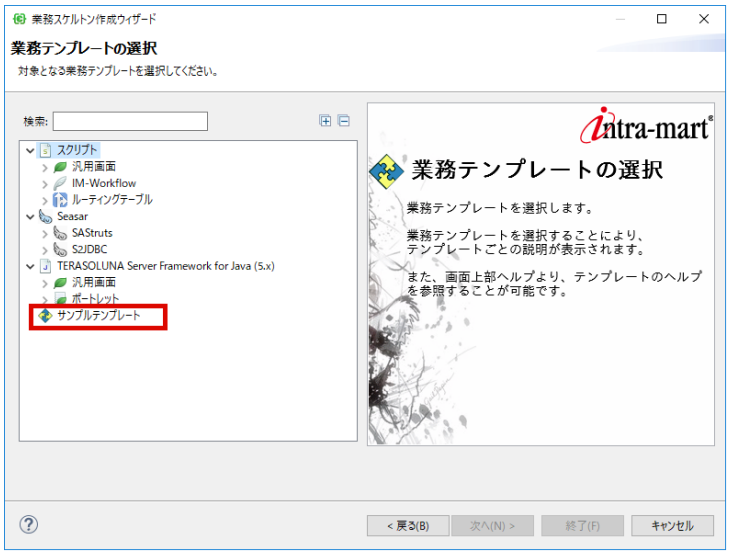
では、出力されたプラグインプロジェクトを実行してみましょう。
plugin.xmlを表示し、「概要」タブを選択し、テストの項目の「Eclipseアプリケーションの起動」を選択します。



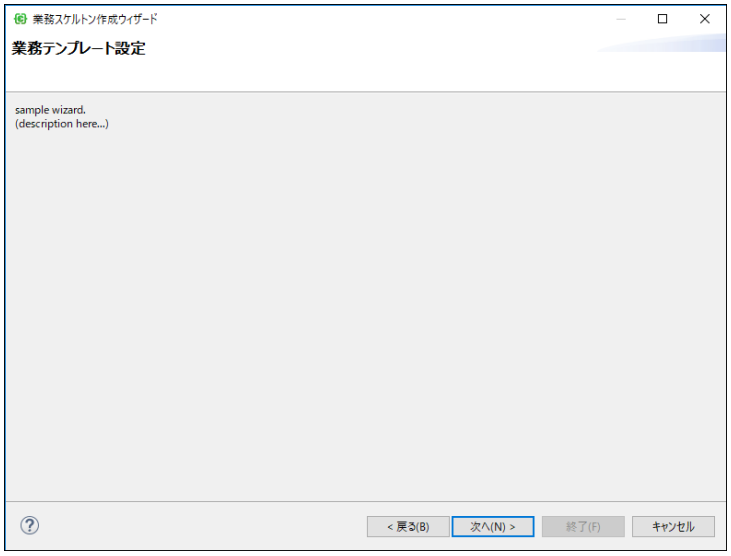
新規でe Builderが起動します。初回起動時はプロジェクトがまだ何もない状態のため、対象となるモジュールプロジェクトを新規作成します。（手順は割愛）



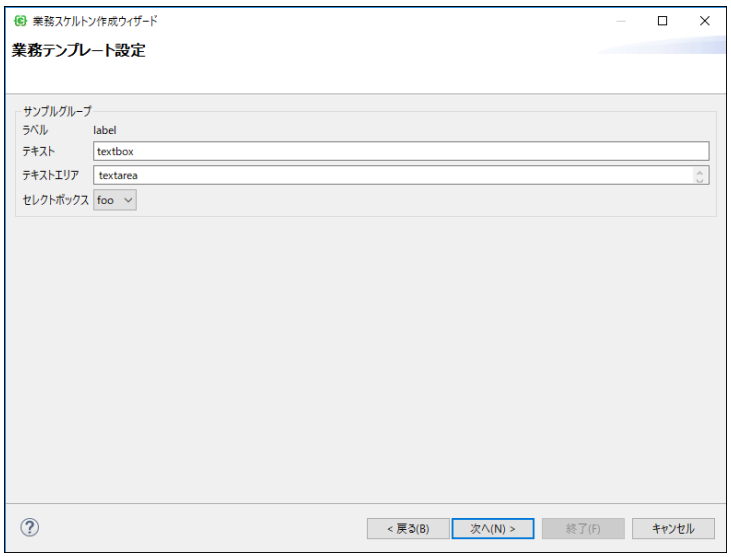
業務スケルトンウィザードを実行します。
「テンプレート選択」画面に作成したテンプレートが表示されているのが確認できます。
作成されたテンプレートを選択して次へ進みます。



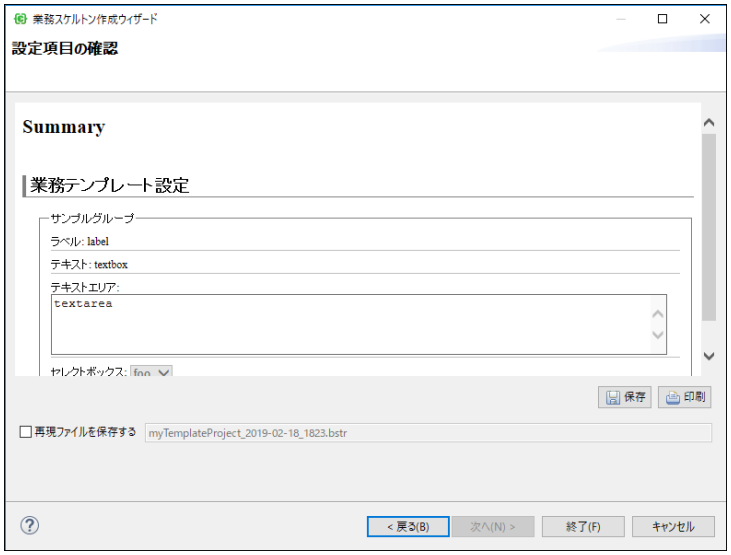
概要ページが表示されます。
そのまま次へ進みます。



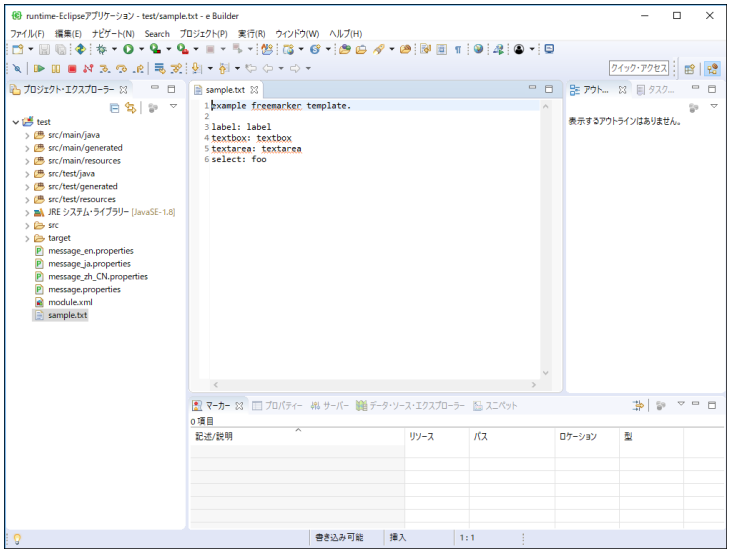
業務テンプレート設定ページが表示されます。
各項目に適当な値を設定して次へ進みます。



設定項目の確認ページが表示されますので
終了ボタンを押下してウィザードを終了します。



プラグインプロジェクト直下にsample.txtが生成され、内容にウィザードの入力内容が反映されていることが確認できます。



テンプレートのカスタマイズ

それでは前項で作成したプロジェクトを元に、簡単なスクリプト開発用の画面(html)とサーバサイド処理(js)を作成するテンプレートにカスタマイズしてみましょう。

ウィザードページの編集

plugin.xmlを開き「plugin.xml」のタブを選択すると、plugin.xmlがxmlエディタで表示されます。
このxml内の<plugin>→<wizard>要素を参照すると、デフォルトでは下記のようになっているはずです。

```
<wizard>
  <page
      class="jp.co.intra_mart.business.skeleton.ui.wizard.page.OutlineWizardPageProvider"
      resource="resources/wizardPages/page0.txt"
      id="page0">
  </page>
  <page
      resource="resources/wizardPages/page1.xml"
      id="page1">
  </page>
</wizard>
```

wizardタグ内はウィザード内で表示するページを設定します。
pageタグは表示するページの設定を記述しており、resource属性に使用するテンプレートファイルを指定します。
ここでは編集しないでそのまま利用することにします。

<resources/wizardPages/page0.txt>ファイルを開き、このウィザードに関する説明を記述します。
例として下記のように記述してみます。

```
チュートリアルで作成したテンプレートウィザードです。
```

次に、resources/wizardPages/page1.xmlを表示します。
デフォルトでは下記のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<wizardPage xmlns="http://business.skeleton.intra-mart.jp/xsd/CompositeWizardDefinition"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://business.skeleton.intra-
mart.jp/xsd/CompositeWizardDefinition ../../schema/CompositeWizardPageDefinition.xsd ">
    <group id="group" name="group">
      <label id="label" name="label">label</label>
      <text id="text" name="text">textbox</text>
      <textarea id="textarea" name="textarea">textarea</textarea>
      <select id="select" name="select" selected="foo">{"foo", "bar", "baz"}</select>
    </group>
  </wizardPage>
```

上記のように、業務スケルトンウィザードの入力項目はXMLで記述して設定します。
デフォルトのままでは目的に沿わないので、例として下記のように修正します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wizardPage xmlns="http://business.skeleton.intra-mart.jp/xsd/CompositeWizardDefinition"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://business.skeleton.intra-
mart.jp/xsd/CompositeWizardDefinition ../../schema/CompositeWizardPageDefinition.xsd ">
 <group id="group" name="出力するファイル設定">
  <text id="fileName" name="ファイル名">tutorial</text>
  <text id="displayName" name="画面名">チュートリアル</text>
 </group>
</wizardPage>
```

ビルダーの編集

次にファイルを生成する処理（以下ビルダーと呼びます）のカスタマイズをしていきます。
plugin.xmlを開き <plugin>→<builders>を参照します。
デフォルトでは以下のようになっているはずです。

```
<builders>
 <builder
    name="%builderName"
    resource="resources/builders/ant/build.xml"
    id="build">
 </builder>
 <templateBuilder
    templateEncoding="UTF-8"
    name="%template BuilderName"
    template="resources/builders/templates/sample.ftl"
    id="template">
  <output>
    &amp;quot;sample.txt&amp;quot;
  </output>
 </templateBuilder>
</builders>
```

上記ではantの実行とテンプレートファイルの出力の2つの記述がされていますが
今回はantの実行についての説明は割愛します。
チュートリアルではhtmlファイルとjsファイルの２つを出力しますので、下記のように修正します。

```
<builders>
  <templateBuilder
      templateEncoding="UTF-8"
      name="チュートリアルのhtml"
      template="resources/builders/templates/tutorial_html.ftl"
      id="template">
    <output>
      "src/main/jssp/src/" + page1.group.fileName + ".html"
    </output>
  </templateBuilder>
  <templateBuilder
      templateEncoding="UTF-8"
      name="チュートリアルのjs"
      template="resources/builders/templates/tutorial_js.ftl"
      id="template">
    <output>
      "src/main/jssp/src/" + page1.group.fileName + ".js"
    </output>
  </templateBuilder>
</builders>
```

templateBuilder のtemplate属性に使用するテンプレートファイル名を指定します。
outputタグには出力するファイルのプロジェクトのルートからの絶対パスを入力します。

このとき、ウィザードページで入力した値を変数として利用できます。
例では変数「page1.group.fileName」を使用していますが、idが「page1」であるウィザードページの「id=group」内の「id=fileName」の値、つまり「ファイル名」の入力値を示します。

テンプレートファイルの編集

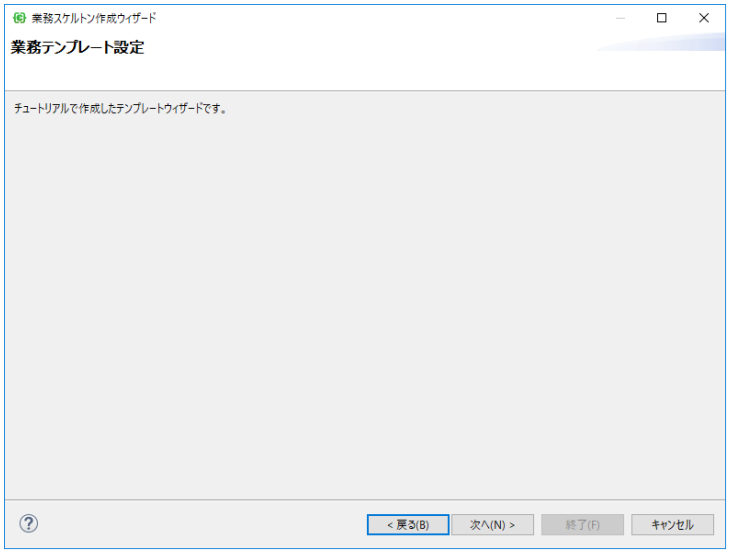
それでは、上記のtemplate属性で指定したそれぞれのファイルのテンプレートを作成していきます。
まず、新規ファイル resources/builders/templates/tutorial/tutorial_html.ftlを作成し、以下のように編集します。

```
<#ftl encoding="UTF8" strip_whitespace=false strip_text=false strict_syntax=true /><#t>
 <!-- HEADタグ -->
 <imart type="head">
  <title>${page1.group.displayName}</title>
 </imart>
 <!-- 画面上に表示されるタイトル -->
 <div class="imui-title">
  <h1>${page1.group.displayName}</h1>
 </div>
        チュートリアルで作成されたページです。
```

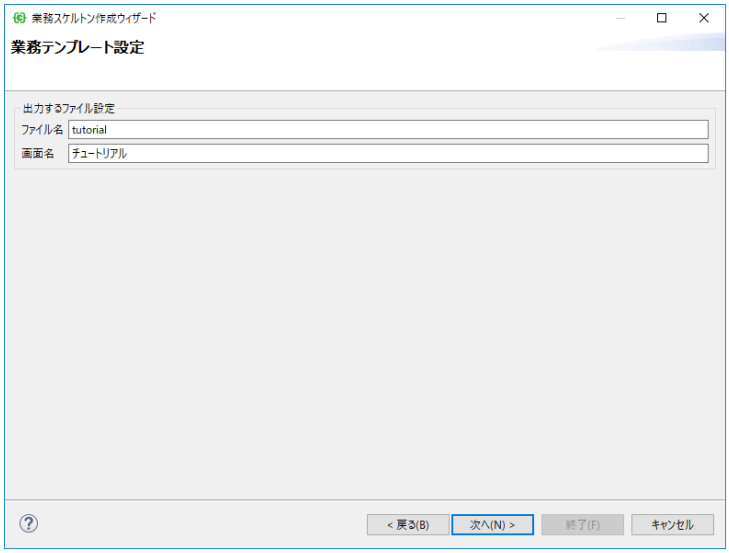
テンプレートファイルでは、生成するファイルの内容を記述します。
変数の参照は${}で囲んで使用します。
例では${page1.group.displayName}を指定していますが、これはウィザードページの画面名の値を参照しています。
同様に、新規ファイル resources/builders/templates/tutorial/tutorial_js.ftlを作成し、以下のように編集します。

```
<#ftl encoding="UTF8" strip_whitespace=false strip_text=false strict_syntax=true /><#t>

/**
 * @fileoverview ${page1.group.displayName}画面。
 */

/**
 * 初期化処理。
 *
 * 「登録」画面を表示するための初期化処理を行います。<br />
 *
 * @param {Request} request リクエスト情報
 * @type void
 */
function init(request) {
}
```

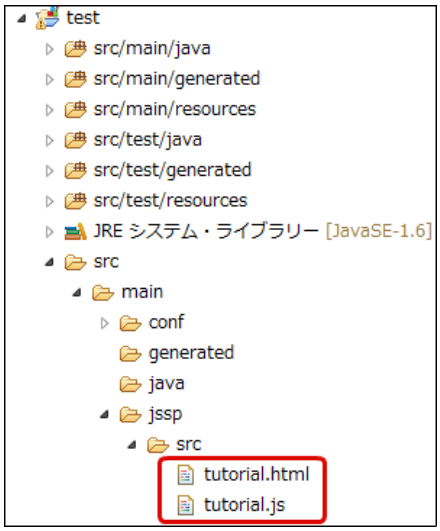
これでひとまず動作する状態になりましたので実際に実行してみましょう。
先ほどと同様にプラグインプロジェクトを実行し、別途起動したe Builder上で業務スケルトンを起動します。
その後、作成したテンプレートを選択すると、テンプレートの説明ページにpage0.txtで設定した文言が表示されます。



テンプレートの設定ページではpage1.xmlで設定した要素が配置されているのが分かります。
ファイル名と画面名を任意の文字列に変更してウィザードを進めて出力を実行します。



出力結果。src/main/jssp/src配下に設定したファイル名でhtmlとjsファイルが出力されました。



htmlファイルの出力結果です。
変数を使用した箇所がウィザードの入力値で置換されているのが分かります。

```
<!-- HEADタグ -->
<imart type="head">
  <title>チュートリアルの画面</title>
</imart>
<!-- 画面上に表示されるタイトル -->
<div class="imui-title">
  <h1>チュートリアルの画面</h1>
</div>
  チュートリアルで作成されたページです。
```

jsファイルの出力結果です。
変数が同様に置換されています。

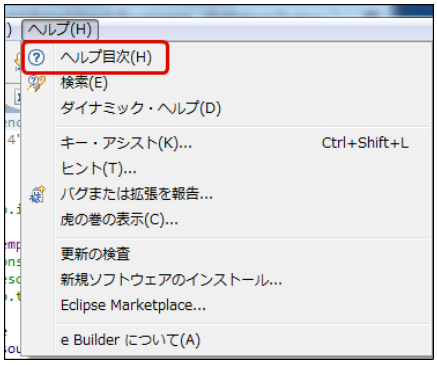
```
/**
 * @fileoverview チュートリアルの画面画面。
 */

/**
 * 初期化処理。
 *
 * 「登録」画面を表示するための初期化処理を行います。<br />
 *
 * @param {Request} request リクエスト情報
 * @type void
 */
function init(request) {
}
```

> **コラム**
>
> このチュートリアルでは、下記のポイントを確認しました。
> スクリプト開発用の簡単なhtmlおよびjsファイルを出力するテンプレート作成を通して業務スケルトン拡張の基本的な流れを把握しました。

業務スケルトンガイドの参照

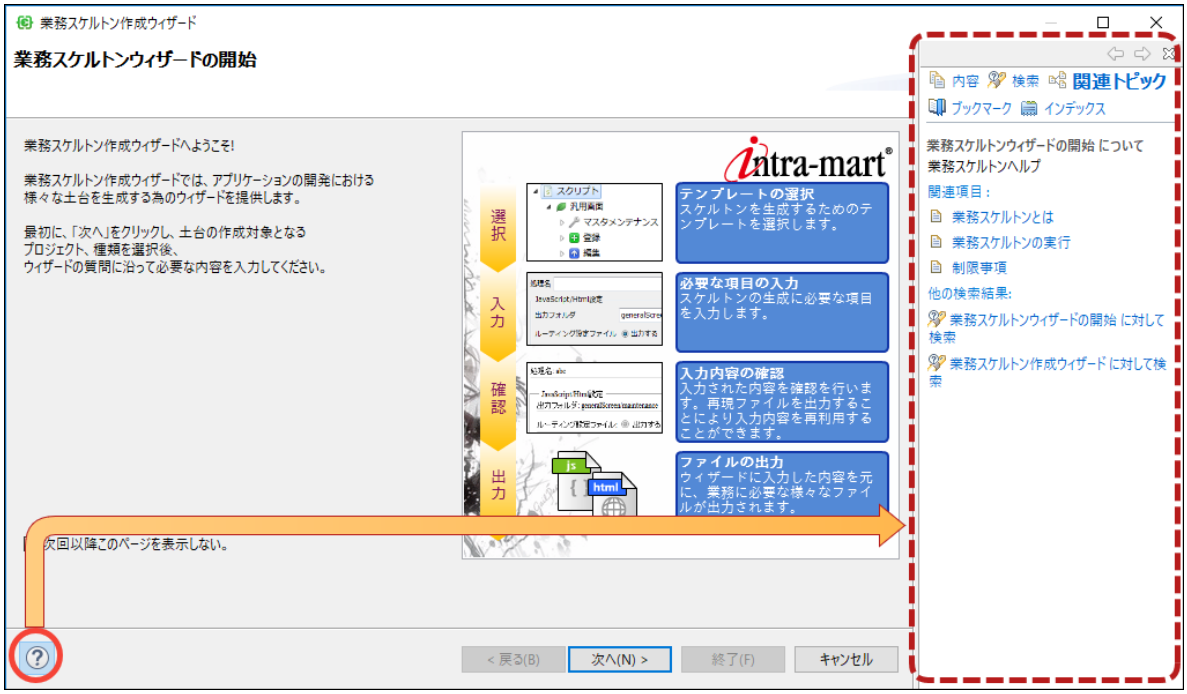
ここまでで業務スケルトンテンプレートの簡単な作成方法について説明しましたが、より実用的なテンプレートを作成したい場合、業務スケルトンガイドを一読しておくことをお勧めします。

e Builder上部メニューより、「ヘルプ」→「ヘルプ目次」を選択します。



ヘルプ画面が表示されますので、インデックスより「業務スケルトンガイド」を選択します。

また、業務スケルトン作成ウィザードの左下「？」ボタンを押下しても参照できます。



利用できる業務スケルトンテンプレート一覧

項目

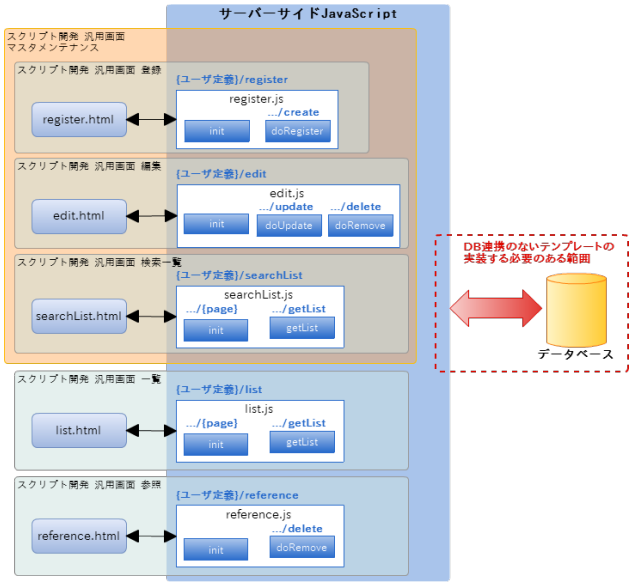
概要

本項では、各業務スケルトンテンプレートで利用可能な機能について説明します。
本項では業務スケルトン Ver.2.0.1で提供される機能について説明します。
最新のバージョン情報または変更履歴については *業務スケルトン 変更履歴* を参照してください。
また、業務スケルトンのアップデート方法については e Builder のアップデート を参照してください。

スクリプト開発 汎用画面テンプレート

スクリプト開発ベースのシンプルな登録/更新/一覧等の画面を出力します。
DB連携の場合、データベース接続を利用したソースを出力します。
また、スマートフォン画面を出力する設定を行うことで、PC/スマートフォンの双方のソースを一括出力できます（Ver 2.0.1以降）。
Ver 2.0.3以降では、CSRF対策したソースを出力しています。
プログラミング方法の習熟、または新規画面開発の際のソースのひな形等にご利用頂けます。

- スクリプト開発 汎用画面テンプレートで出力される成果物の概要図。
  マスタメンテナンスの場合、登録・編集・検索一覧および各々の画面遷移をサポートします。
  DB連携のないテンプレートの場合、データベース接続関連の処理は未実装の状態で出力されますので
  目的・用途に応じて追加実装してください。



コラム

DB連携テンプレートは単一テーブルのCRUDのみサポートします。
複数テーブルの結合条件が必要な場合や、複雑な業務ロジックを実装する必要がある場合は
DB連携なしのテンプレートを使用することをお勧めします。

注意

スマートフォン画面は、jQuery Mobile 1.3 をベースに作成しています。
jQuery Mobileのバージョンをベースにする場合には、適宜修正をおこなってください。

テンプレート一覧

NO	テンプレート名	説明
1	スクリプト開発 汎用画面 登録	登録画面のHTML, サーバサイドJSを出力します。 ※一部処理は追加実装する必要があります。
2	スクリプト開発 汎用画面 登録（DB連携）	データベースアクセスを伴う登録画面のHTML, サーバサイドJSを出力します。
3	スクリプト開発 汎用画面 編集	編集画面のHTML, サーバサイドJSを出力します。 ※一部処理は追加実装する必要があります。
4	スクリプト開発 汎用画面 編集（DB連携）	データベースアクセスを伴う編集画面のHTML, サーバサイドJSを出力します。
5	スクリプト開発 汎用画面 参照	参照画面のHTML, サーバサイドJSを出力します。 ※一部処理は追加実装する必要があります。
6	スクリプト開発 汎用画面 参照（DB連携）	データベースアクセスを伴う参照画面のHTML, サーバサイドJSを出力します。
7	スクリプト開発 汎用画面 一覧	一覧画面のHTML, サーバサイドJSを出力します。 ※一部処理は追加実装する必要があります。
8	スクリプト開発 汎用画面 一覧（DB連携）	データベースアクセスを伴う一覧画面のHTML, サーバサイドJSを出力します。
9	スクリプト開発 汎用画面 検索一覧	検索一覧画面のHTML, サーバサイドJSを出力します。 ※一部処理は追加実装する必要があります。
10	スクリプト開発 汎用画面 検索一覧（DB連携）	データベースアクセスを伴うる検索一覧画面のHTML, サーバサイドJSを出力します。
11	スクリプト開発 汎用画面 マスタメンテナンス	登録画面/編集画面/検索一覧画面のHTML, サーバサイドJSを出力します。 ※一部処理は追加実装する必要があります。
12	スクリプト開発 汎用画面 マスタメンテナンス（DB連携）	データベースアクセスを伴う登録画面/編集画面/検索一覧画面のHTML, サーバサイドJSを出力します。

スクリプト開発 IM-Workflowテンプレート

IM-Workflowのコンテンツ定義に使用できる各種スクリプト開発ベースのソースを出力します。
DB連携の場合、データベース接続を利用したソースを出力します。

また、コンテンツのインポートXMLの出力、項目定義からテーブル定義の生成、スマートフォン画面を出力するなどの機能が利用できます（Ver 2.0.1以降）。
プログラミング方法の習熟、または新規ワークフローのスタートアップ等にご利用頂けます。

> **注意**
>
> スマートフォン画面は、jQuery Mobile 1.3 をベースに作成しています。
> jQuery Mobileのバージョンをベースにする場合には、適宜修正をおこなってください。

テンプレート一覧

NO	テンプレート名	説明
1	スクリプト開発 IM-Workflow 画面 各種画面/アクション処理出力	各ユーザコンテンツ画面および連携するアクション処理を出力します。 ※一部処理は追加実装する必要があります。
2	スクリプト開発 IM-Workflow 画面 各種画面/アクション処理出（DB連携）	データベースアクセスを伴う各ユーザコンテンツ画面および連携するアクション処理を出力します。
3	スクリプト開発 IM-Workflow ユーザプログラム 各種ユーザプログラム出力	コンテンツに定義可能な各種ユーザプログラムのテンプレートを出力します。 ※一部処理は追加実装する必要があります。
4	スクリプト開発 IM-Workflow ユーザプログラム 各種ユーザプログラム出力	コンテンツに定義可能な各種リスナのテンプレートを出力します。 ※一部処理は追加実装する必要があります。

スクリプト開発 ルーティングテーブル

スクリプト開発でルーティングテーブルxmlを出力する機能です。
jsファイルのJsDocにアノテーションを記述することにより、それを解釈してルーティングテーブルxmlを出力します。
詳細については、「*スクリプト開発 ルーティングテーブル設定の仕様*」を参照してください。
ルーティングについては、「*ルーティング*」を参照してください。

テンプレート一覧

NO	テンプレート名	説明
1	スクリプト開発 ルーティングテーブル	スクリプト開発モデルにおいて、ルーティングテーブルxmlを出力します。

Seasar S2JDBCテンプレート

- エンティティ生成
  S2JDBC-Genを利用し、エンティティクラス等各種S2JDBC関連クラスを生成します（Ver 2.0.1以降）。
  S2JDBC-Genについては下記を参照してください。
  http://s2container.seasar.org/2.4/ja/s2jdbc_gen/
- DtoとService生成
  SQLファイルを元にデータベースに接続しテーブル情報を取得して、DtoとServiceクラスを生成します。
  生成したServiceクラスでは、SQLファイルによってデータベースからレコードを取得します。
  エンティティ生成に対して、このテンプレートはSQLファイルのSQL文を実行するServiceを生成します。
  エンティティ生成では複雑なSQLを実装するのは大変なので、このテンプレートと使い分けてください。
  詳細については、「*Seasar S2JDBC DtoとService生成*」を参照してください。

テンプレート一覧

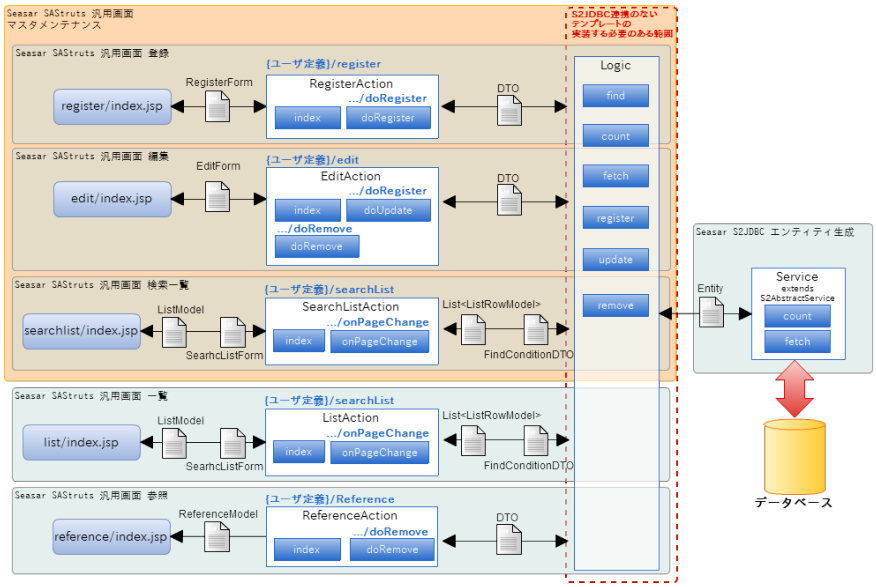
NO	テンプレート名	説明
1	Seasar S2JDBC エンティティ生成	S2JDBCのエンティティのJAVAファイルを出力します。サービスクラスおよびNameクラス、コンディションクラスも同時出力します（オプション）
2	Seasar S2JDBC DtoとService生成	SQLファイルを元にデータベースに接続しテーブル情報を取得し、DtoとServiceクラスを生成します。 SELECT文のみ対応しています。

Seasar SAStruts 汎用画面テンプレート

SAStrutsベースのシンプルな登録/更新/一覧等の画面を出力します。
S2JDBC連携の場合、サービスクラスを利用したソースを出力します。
また、スマートフォン画面を出力する設定を行うことで、PC/スマートフォンの双方のソースを一括出力できます（Ver 2.0.1以降）。

プログラミング方法の習熟、または新規画面開発の際のソースのひな形等にご利用頂けます。

- Seasar SAStruts 汎用画面テンプレートで出力される成果物の概要図。
  マスタメンテナンスの場合、登録・編集・検索一覧および各々の画面遷移をサポートします。
  エンティティおよびサービスクラスの生成は別途 Seasar S2JDBCテンプレート でサポートします。
  S2JDBC連携のないテンプレートの場合、ロジッククラスは未実装の状態で出力されますので
  目的・用途に応じて追加実装してください。



> **注意**
>
> S2JDBC連携で出力されるクラスの処理は、Seasar-S2JDBC-エンティティ生成テンプレートで
> 出力したサービスクラスを利用することを前提としています。
>
> スマートフォン画面は、jQuery Mobile 1.3 をベースに作成しています。
> jQuery Mobileのバージョンをベースにする場合には、適宜修正をおこなってください。

> **コラム**
>
> S2JDBC連携テンプレートは単一エンティティのCRUDのみサポートします。
> 複数テーブルの結合条件が必要な場合や、複雑な業務ロジックを実装する必要がある場合は
> S2JDBC連携なしのテンプレートを使用することをお勧めします。

CSRF対策を有効化させるために（Ver 2.0.2以降）

SAStruts汎用画面テンプレートプラグイン Ver 2.0.2以降では、出力するソースコードにCSRF対策をしています。 CSRF対策を有効化するために、app.diconとcustomizer.diconにインターセプターの設定をする必要があります。 設定内容は以下の通りです。

- app.dicon
  secureTokenValidationInterceptorコンポーネントを定義します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.4//EN"
   "https://www.seasar.org/dtd/components24.dtd">
<components>
   <include path="convention.dicon"/>
   <include path="aop.dicon"/>
   <include path="j2ee.dicon"/>
   <!--
   <include path="s2jdbc.dicon"/>
   -->
   <component name="actionMessagesThrowsInterceptor"
class="org.seasar.struts.interceptor.ActionMessagesThrowsInterceptor"/>

   <component name="secureTokenValidationInterceptor"
         class="jp.co.intra_mart.framework.extension.seasar.struts.interceptor.SecureTokenValidationInterceptor"
         instance="prototype">
      <aspect pointcut="invoke">
         <component class="org.seasar.framework.aop.interceptors.InterceptorLifecycleAdapter" />
      </aspect>
   </component>

</components>
```

- customizer.dicon
  actionCustomizerにsecureTokenValidationInterceptorを追加します。

```
   <component name="actionCustomizer"
      class="org.seasar.framework.container.customizer.CustomizerChain">
      <initMethod name="addAspectCustomizer">
         <arg>"aop.traceInterceptor"</arg>
      </initMethod>
      <initMethod name="addAspectCustomizer">
         <arg>"actionMessagesThrowsInterceptor"</arg>
      </initMethod>
      <initMethod name="addAspectCustomizer">
         <arg>"secureTokenValidationInterceptor"</arg>
         <arg>"doRegister, doUpdate, doRemove"</arg>
      </initMethod>
      <initMethod name="addCustomizer">
         <arg>
            <component
               class="org.seasar.framework.container.customizer.TxAttributeCustomizer"/>
         </arg>
      </initMethod>
      <initMethod name="addCustomizer">
         <arg>
            <component
               class="org.seasar.struts.customizer.ActionCustomizer"/>
         </arg>
      </initMethod>
   </component>
```

> **注意**
>
> SecureTokenValidationInterceptorはintra-mart Accel Platform 2014 Winter(Iceberg)から追加されたクラスです。
> このCSRF対策を有効にするためには、intra-mart Accel Platformのバージョンが2014 Winter(Iceberg)以降である必要があります。

テンプレート一覧

NO	テンプレート名	説明
1	Seasar SAStruts 汎用画面 登録	登録画面のJSP, アクションクラス, Logicクラスを出力します。 ※一部処理は追加実装する必要があります。
2	Seasar SAStruts 汎用画面 登録（S2JDBC連携）	S2JDBCのサービスクラスと連携する登録画面のJSP, アクションクラス, Logicクラスを出力します。
3	Seasar SAStruts 汎用画面 編集	編集画面のJSP, アクションクラス, Logicクラスを出力します。 ※一部処理は追加実装する必要があります。
4	Seasar SAStruts 汎用画面 編集（S2JDBC連携）	S2JDBCのサービスクラスと連携する編集画面のJSP, アクションクラス, Logicクラスを出力します。
5	Seasar SAStruts 汎用画面 参照	参照画面のJSP, アクションクラス, Logicクラスを出力します。 ※一部処理は追加実装する必要があります。
6	Seasar SAStruts 汎用画面 参照（S2JDBC連携）	S2JDBCのサービスクラスと連携する参照画面のJSP, アクションクラス, Logicクラスを出力します。
7	Seasar SAStruts 汎用画面 一覧	一覧画面のJSP, アクションクラス, Logicクラスを出力します。 ※一部処理は追加実装する必要があります。
8	Seasar SAStruts 汎用画面 一覧（S2JDBC連携）	S2JDBCのサービスクラスと連携する一覧画面のJSP, アクションクラス, Logicクラスを出力します。
9	Seasar SAStruts 汎用画面 検索一覧	検索一覧画面のJSP, アクションクラス, Logicクラスを出力します。 ※一部処理は追加実装する必要があります。
10	Seasar SAStruts 汎用画面 検索一覧（S2JDBC連携）	S2JDBCのサービスクラスと連携する検索一覧画面のJSP, アクションクラス, Logicクラスを出力します。
11	Seasar SAStruts 汎用画面 マスタメンテナンス	登録画面/編集画面/検索一覧画面のJSP, アクションクラス, Logicクラスを出力します。 ※一部処理は追加実装する必要があります。
12	Seasar SAStruts 汎用画面 マスタメンテナンス（S2JDBC連携）	S2JDBCのサービスクラスと連携する登録画面/編集画面/検索一覧画面のJSP, アクションクラス, Logicクラスを出力します。

Seasar SAStruts IM-Workflowテンプレート

IM-Workflowのコンテンツ定義に使用できる各種SAStrutsベースのソースを出力します。
S2JDBC連携の場合、サービスクラスを利用したソースを出力します。
また、コンテンツのインポートXMLの出力、項目定義からテーブル定義の生成、スマートフォン画面を出力するなどの機能が利用できます（Ver 2.0.1以降）。
プログラミング方法の習熟、または新規ワークフローのスタートアップ等にご利用頂けます。

> **注意**
>
> スマートフォン画面は、jQuery Mobile 1.3 をベースに作成しています。
> jQuery Mobileのバージョンをベースにする場合には、適宜修正をおこなってください。

テンプレート一覧

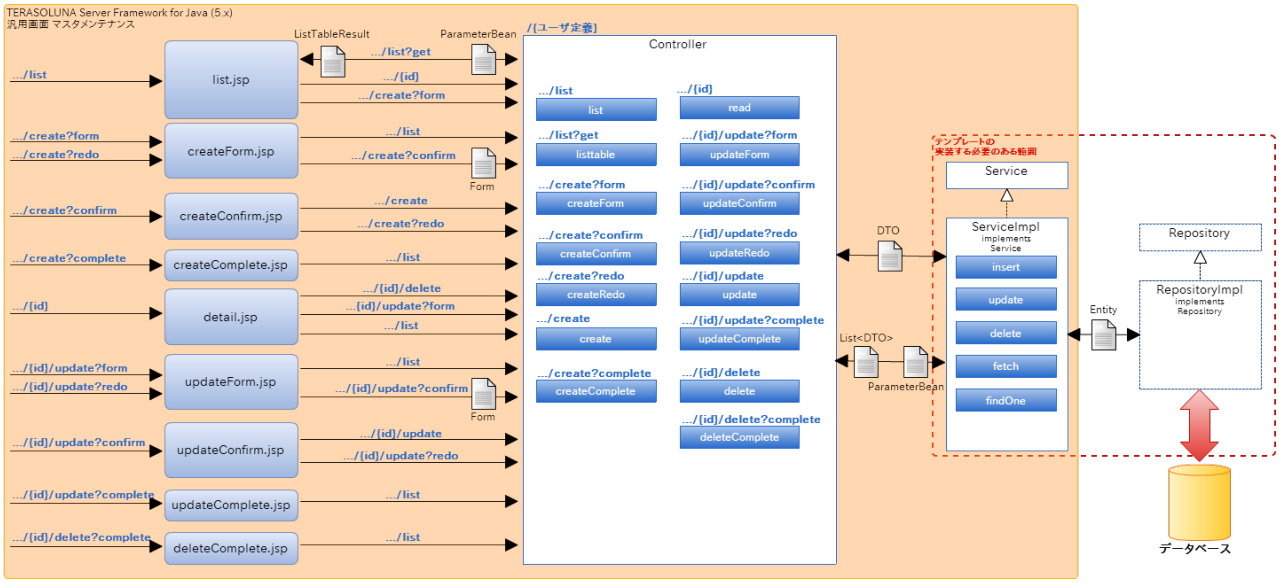
NO	テンプレート名	説明
1	Seasar SAStruts IM-Workflow 画面 各種画面/アクション処理出力	各ユーザコンテンツ画面および連携するアクション処理を出力します。
2	Seasar SAStruts IM-Workflow 画面 各種画面/アクション処理出（DB連携）	データベースアクセスを伴う各ユーザコンテンツ画面および連携するアクション処理を出力します。
3	Seasar SAStruts IM-Workflow ユーザプログラム 各種ユーザプログラム出力	コンテンツに定義可能な各種ユーザプログラムのテンプレートを出力します。

NO	テンプレート名	説明
4	Seasar SAStruts IM-Workflow ユーザプログラム 各種ユーザプログラム出力	コンテンツに定義可能な各種リスナのテンプレートを出力します。

TERASOLUNA Server Framework for Java (5.x) 汎用画面テンプレート

TERASOLUNA Server Framework for Java (5.x)ベースのシンプルな登録/更新/一覧等の画面を出力します。
また、スマートフォン画面を出力する設定を行うことで、PC/スマートフォンの双方のソースを一括出力できます。
プログラミング方法の習熟、または新規画面開発の際のソースのひな形等にご利用頂けます。

- TERASOLUNA Server Framework for Java (5.x) 汎用画面テンプレートの概要図。
  TERASOLUNA Server Framework for Java (5.x)のガイドラインに従い登録・編集・検索一覧および各々の画面遷移をサポートします。
  サービスクラスは未実装の状態で出力されますので、目的・用途に応じて追加実装してください。



> **注意**
>
> スマートフォン画面は、jQuery Mobile 1.3 をベースに作成しています。
> jQuery Mobileのバージョンをベースにする場合には、適宜修正をおこなってください。

CSRF対策を有効化させるために（Ver 1.0.2以降）

TERASOLUNA Server Framework for Java (5.x) 汎用画面テンプレートプラグイン Ver 1.0.2以降では、出力するソースコードにCSRF対策をしています。 CSRF対策を有効化するためには、intra-mart Accel Platform 2014 Winter(Iceberg)で変更された applicationContext-im_tgfw_common.xmlとapplicationContext-im_tgfw_web.xmlの設定内容が反映されている必要があります。 以下の設定が無い場合は追記、変更してください。

- applicationContext-im_tgfw_common.xml
  InvalidSecureTokenExceptionの設定があることを確認してください。設定が無い場合は追記してください。

```
<!-- Exception Code Resolver. -->
<bean id="exceptionCodeResolver"
    class="org.terasoluna.gfw.common.exception.SimpleMappingExceptionCodeResolver">
    <!-- Setting and Customization by project. -->
    <property name="exceptionMappings">
        <map>
            <entry key="ResourceNotFoundException" value="w.im.fw.0001" />
            <entry key="InvalidTransactionTokenException" value="w.im.fw.0004" />
            <entry key="InvalidSecureTokenException" value="w.im.fw.0005" />
            <entry key="BusinessException" value="w.im.fw.0002" />
        </map>
    </property>
    <property name="defaultExceptionCode" value="e.im.fw.0001" />
</bean>
```

- applicationContext-im_tgfw_web.xml - secureTokenValidatorのbean定義
  secureTokenValidatorのbean定義を確認します。設定が無い場合は追記してください。

```
<!-- secure token validator -->
<bean id="secureTokenValidator" class="jp.co.intra_mart.framework.extension.spring.web.csrf.SecureTokenValidator" />
```

- applicationContext-im_tgfw_web.xml - SystemExceptionResolverのbean定義
  SystemExceptionResolverのbean定義を確認します。設定が無い場合は追記してください。

```
<!-- Setting Exception Handling. -->
<!-- Exception Resolver. -->
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
  <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
  <!-- Setting and Customization by project. -->
  <property name="order" value="3" />
  <property name="exceptionMappings">
    <map>
      <entry key="ResourceNotFoundException" value="im_tgfw/common/error/resourceNotFoundError.jsp" />
      <entry key="BusinessException" value="im_tgfw/common/error/businessError.jsp" />
      <entry key="InvalidTransactionTokenException" value="im_tgfw/common/error/transactionTokenError.jsp" />
      <entry key="InvalidSecureTokenException" value="im_tgfw/common/error/secureTokenError.jsp" />
    </map>
  </property>
  <property name="statusCodes">
    <map>
      <entry key="im_tgfw/common/error/resourceNotFoundError" value="404" />
      <entry key="im_tgfw/common/error/businessError" value="200" />
      <entry key="im_tgfw/common/error/transactionTokenError" value="409" />
      <entry key="im_tgfw/common/error/secureTokenError" value="403" />
    </map>
  </property>
  <property name="defaultErrorView" value="im_tgfw/common/error/systemError.jsp" />
  <property name="defaultStatusCode" value="500" />
</bean>
```


注意

SecureTokenValidatorとInvalidSecureTokenExceptionはintra-mart Accel Platform 2014 Winter(Iceberg)から追加されたクラスです。
このCSRF対策を有効にするためには、intra-mart Accel Platformのバージョンが2014 Winter(Iceberg)以降である必要があります。

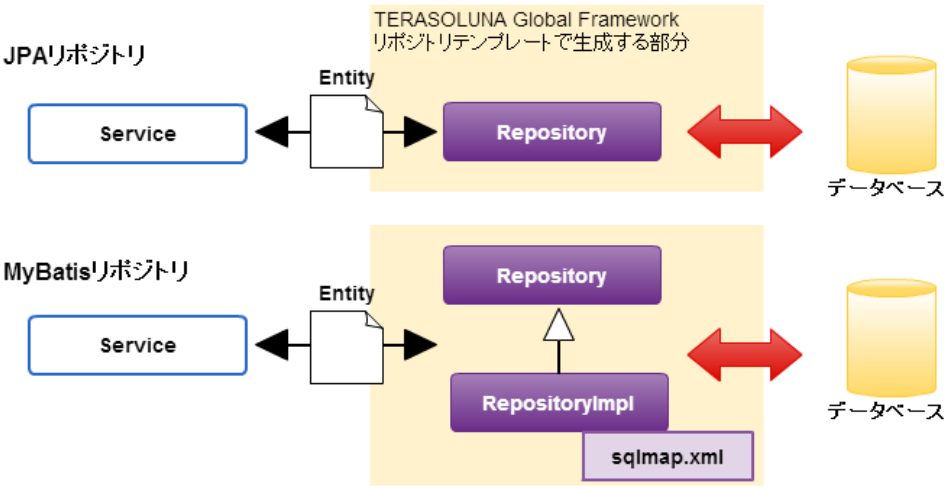
テンプレート一覧

NO	テンプレート名	説明
1	TERASOLUNA Server Framework for Java (5.x) 汎用画面 マスタメンテナンス	登録画面/編集画面/検索一覧画面のJSP, Controllerクラス, Serviceクラスを出力します。

TERASOLUNA Global Framework リポジトリテンプレート （iAP 2014 Winter(Iceberg) まで）

JPA, MyBatis2用のリポジトリクラスとそのテストクラスを出力します。
プログラミング方法の習熟、または新規開発の際のソースのひな形等にご利用頂けます。

- TERASOLUNA Global Framework リポジトリテンプレートの概要図。
  TERASOLUNA Global Frameworkのガイドラインに従いデータベースへのアクセスをサポートします。
  目的・用途に応じて追加実装してください。



> **注意**
>
> TERASOLUNA Global Framework は、intra-mart Accel Platform 2014 Winter(Iceberg)までご利用頂けます。
> intra-mart Accel Platform 2015 Spring(Juno)以降では、ご利用頂けません。

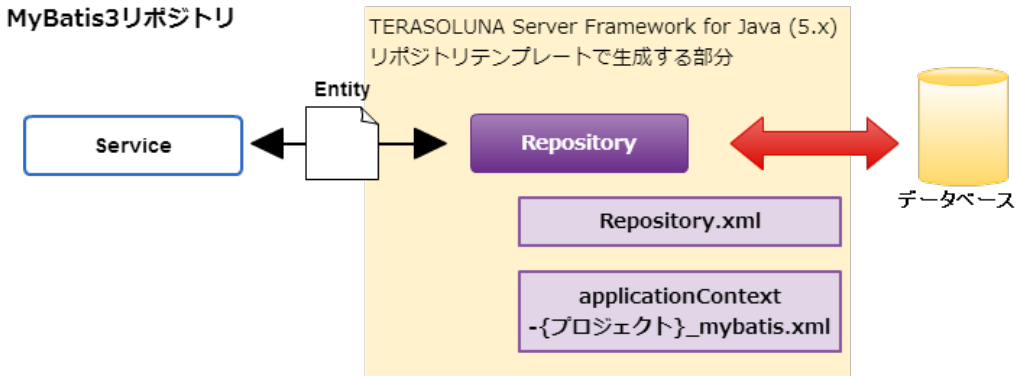
テンプレート一覧

NO	テンプレート名	説明
1	TERASOLUNA Global Framework JPAリポジトリ	Spring Data JPAのJPA Repositoryインタフェースを出力します。 テストクラスは目的・用途に応じて追加実装してください。
2	TERASOLUNA Global Framework MyBatis2リポジトリ	MyBatis2を利用したRepositoryインタフェース, RepositoryImplクラス, *-sqlmap.xmlを出力します。 *-sqlmap.xmlのresultMap, insert, updateは追加実装する必要がある場合があります。 テストクラスは目的・用途に応じて追加実装してください。

TERASOLUNA Server Framework for Java (5.x) リポジトリテンプレート （iAP 2015 Spring(Juno) から）

MyBatis3用のリポジトリクラスとそのテストクラスを出力します。
プログラミング方法の習熟、または新規開発の際のソースのひな形等にご利用頂けます。

- TERASOLUNA Server Framework for Java (5.x) リポジトリテンプレートの概要図。
  TERASOLUNA Server Framework for Java (5.x)のガイドラインに従いデータベースへのアクセスをサポートします。
  目的・用途に応じて追加実装してください。



> **注意**
>
> intra-mart Accel Platform TERASOLUNA Server Framework for Java (5.x) は、intra-mart Accel Platform 2015 Spring(Juno)からご利用頂けます。
> intra-mart Accel Platform 2014 Winter(Iceberg)以前では、ご利用頂けません。

テンプレート一覧

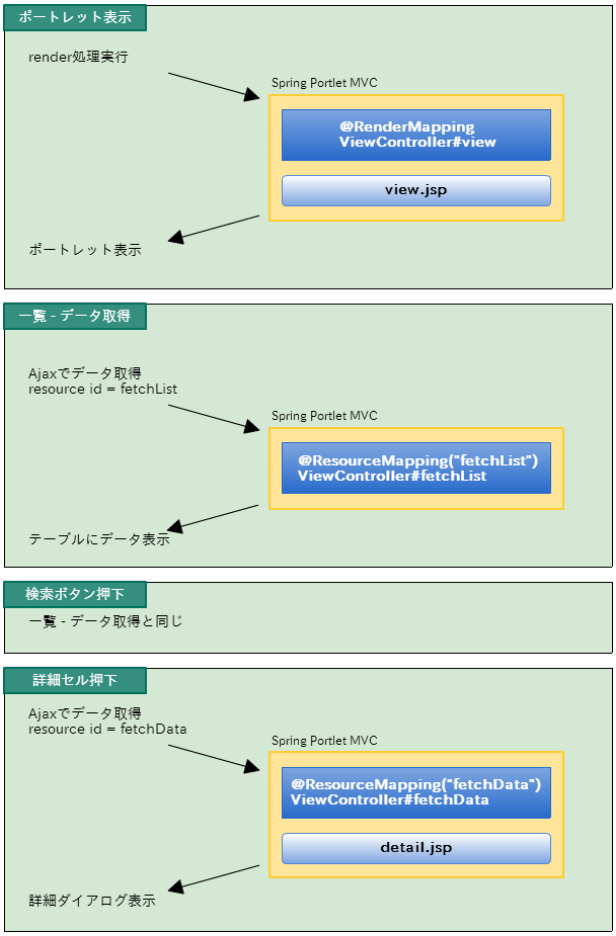
NO	テンプレート名	説明

NO	テンプレート名	説明
1	TERASOLUNA Server Framework for Java (5.x) MyBatis3リポジトリ	MyBatis3を利用した Repositoryインタフェース, Repository.xml, applicationContext-{プロジェクト}_mybatis.xml を出力します。 META-INF/mybatis/mybatis-config.xml の type alias に使用する entity クラスを設定する必要があります。 テストクラスは目的・用途に応じて追加実装してください。

TERASOLUNA Server Framework for Java (5.x) ポートレットテンプレート

TERASOLUNA Server Framework for Java (5.x)ベースのシンプルなポートレット用の一覧画面などを出力します。
プログラミング方法の習熟、または新規画面開発の際のソースのひな形等にご利用頂けます。

- TERASOLUNA Server Framework for Java (5.x) ポートレットテンプレートの概要図
  ポートレット上の一覧表示、一覧のデータ取得（検索）、詳細ダイアログ表示の機能をサポートします。
  サービスクラスは未実装の状態で出力されますので、目的・用途に応じて追加実装してください。



テンプレート一覧

NO	テンプレート名	説明
1	TERASOLUNA Server Framework for Java (5.x) ポートレット 一覧画面	ポートレット用の一覧画面のJSP, Controllerクラス, Serviceクラスを出力します。

業務スケルトンテンプレート利用ガイド

概要

本項では、各業務スケルトンテンプレートの利用方法や活用方法について説明します。

接続プロファイルの新規作成

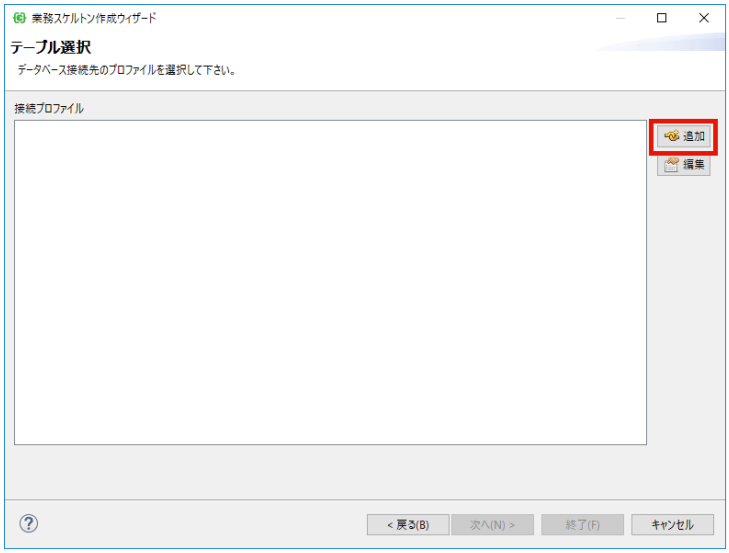
項目

概要

本項では、接続プロファイルの新規作成手順について説明します。

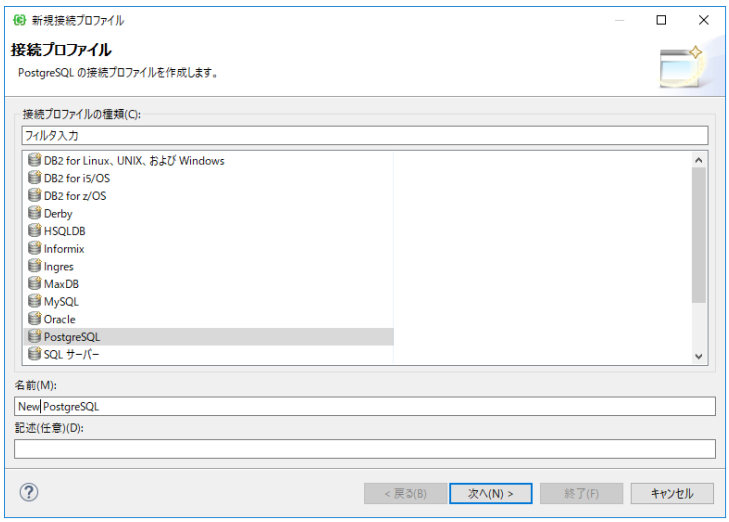
接続プロファイルの新規作成ダイアログを開く

データベース接続先のプロファイルの選択画面に置いて、「追加」ボタンを押下ます。

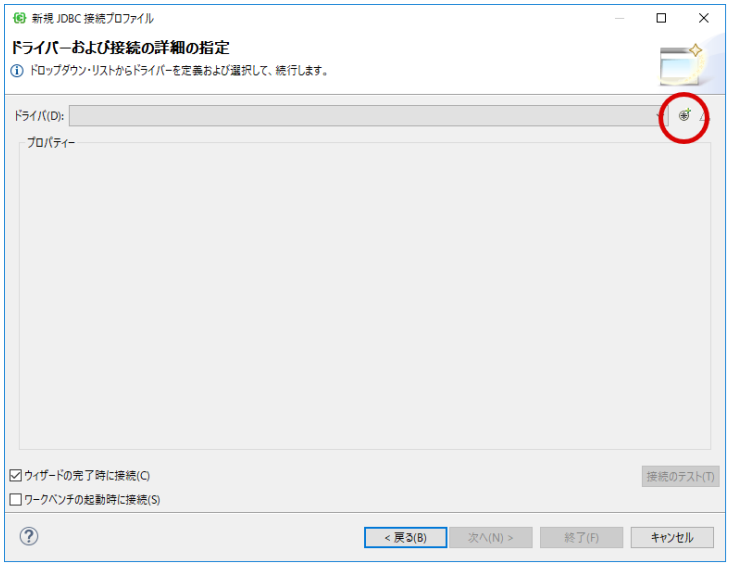


接続プロファイルの作成

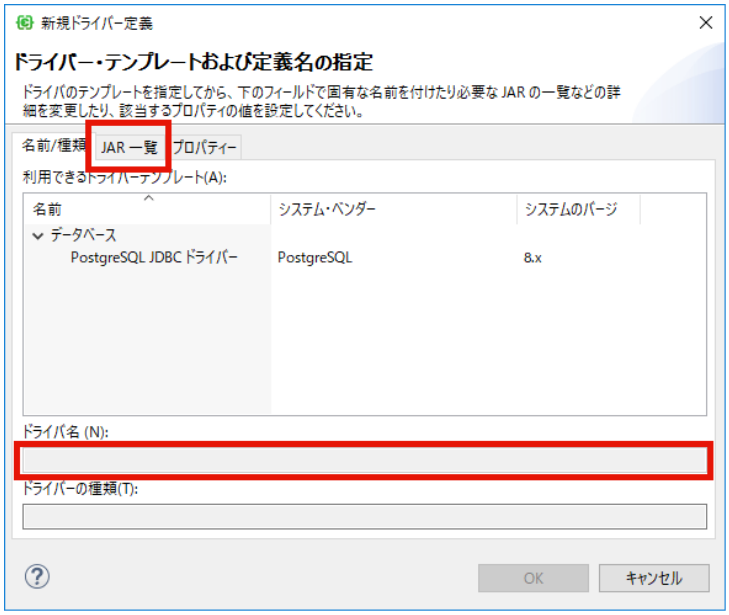
「接続プロファイル・タイプ」より、接続を行うデータベースの種類を選択してください。
「名前」項目に、接続プロファイル名を入力してください。
記述項目には、任意の内容が入力可能です。
入力が完了したら、「次へ」を押下します。



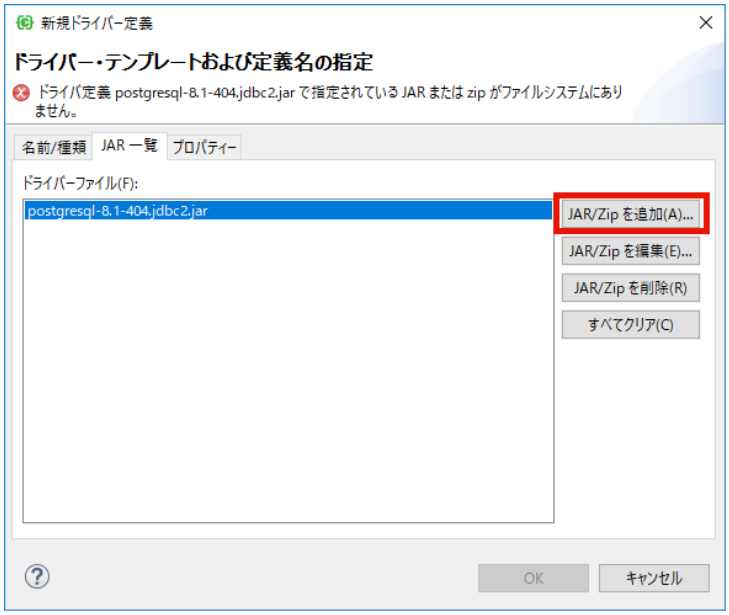
ドライバおよび接続の詳細の設定画面が開かれたら、JDBCドライバの選択を行います。
JDBCドライバの登録が行われていない場合は、ドライバ選択用のセレクトボックス右部にあるドライバの追加ボタンを押下します。



新規ドライバ定義ダイアログが表示されます。
「Driver Name」に、JDBCドライバの名前を入力し、「Jar List」タブを選択してください。
「Driver Name」に関しては、任意の名前が指定可能です。

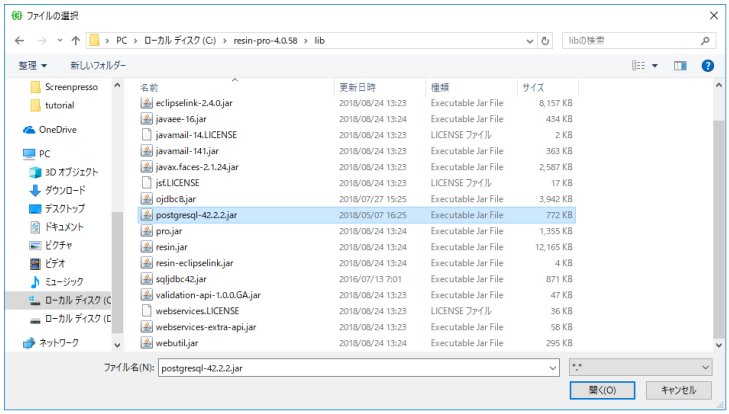


　この画面では、データベース接続に利用するJDBCドライバ（jarファイル）の登録を行います。
既に設定済みのjarファイルは不要ですので選択し、「Remove JAR/Zip」ボタンを押下し削除してください。
その後、「Add JAR/Zip」ボタンを押下し、ファイル選択ダイアログを開きます。

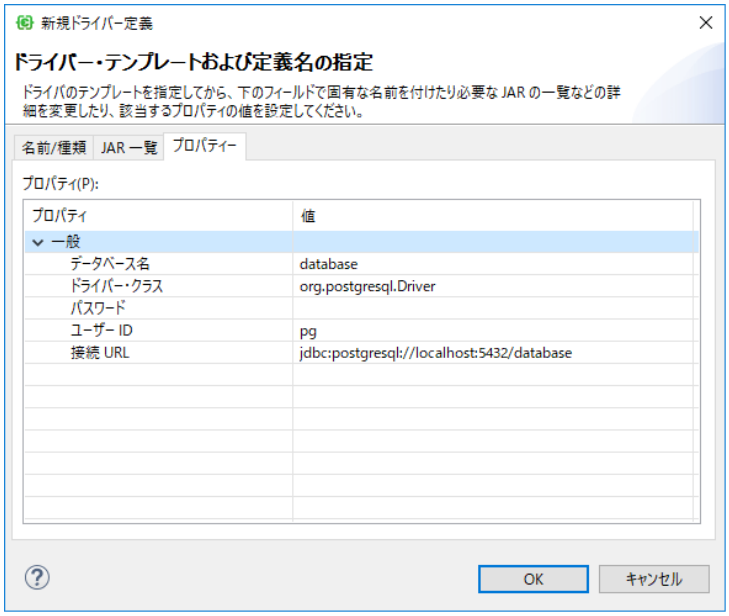


使用するJDBCドライバが含まれたjarファイルまたは、zipファイルを選択し、「開く」ボタンを押下します。

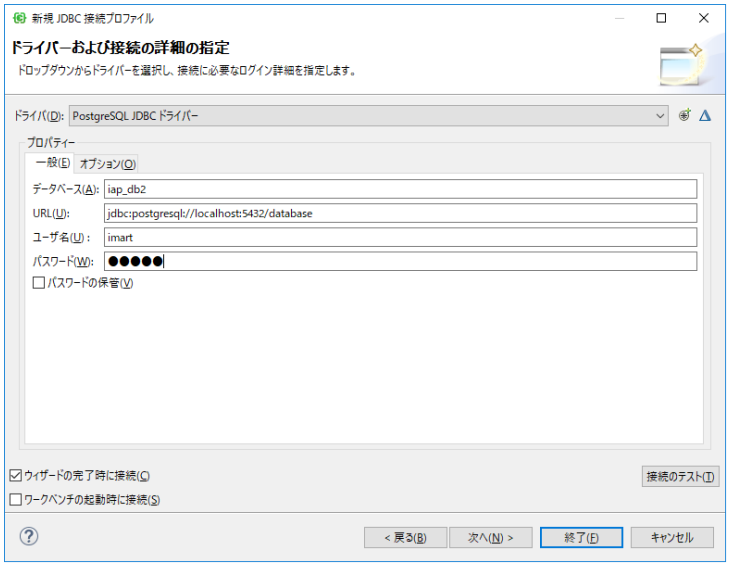
JDBCドライバが含まれたjarファイル、またはzipファイルが複数必要な場合はこの手順を複数回繰り返してください。



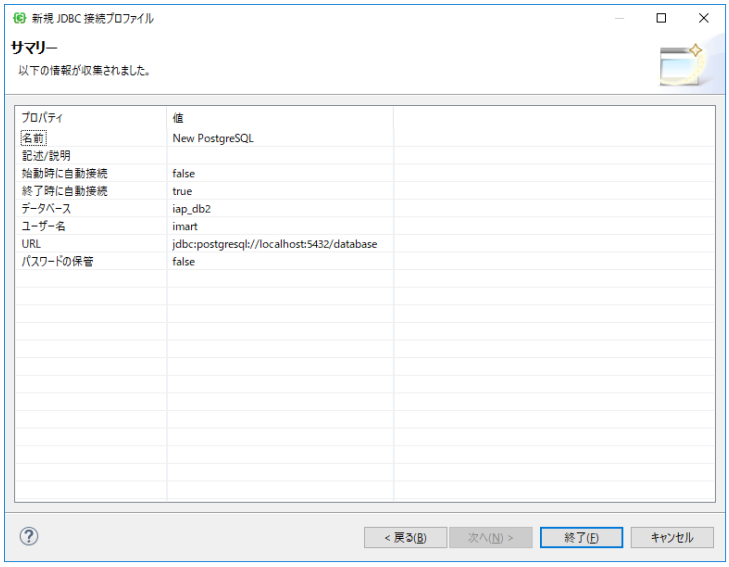
jarファイルの登録が完了したら、「Properties」タブを押下し、JDBCドライバが利用する為の接続情報を入力します。
この画面は、JDBCドライバ自体の設定画面の為、接続情報の設定のテンプレートとして利用される項目です。
実際にデータベース接続プロファイルとして利用する接続情報は後述する別画面にて設定を行います。
入力が完了したら、「OK」ボタンを押下し新規ドライバ定義ダイアログを閉じます。



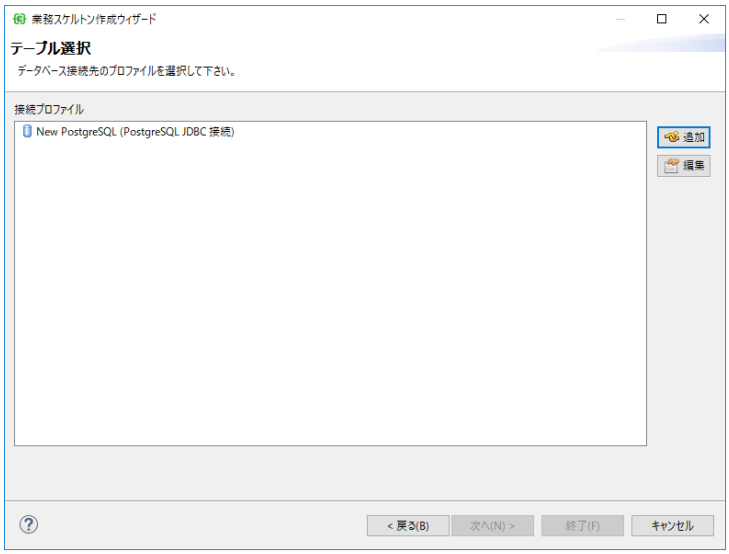
ドライバの定義が完了したら、実際にそのドライバを利用して作成する接続プロファイルで利用する為の接続設定を入力してください。
以下にPostgreSQLに接続する際の入力例を示します。



入力が完了したら、「次へ」ボタンを押下し、確認画面を表示します。
入力内容に問題が無いことを確認し、「終了」ボタンを押下し接続プロファイルの追加を完了してください。



追加した接続プロファイルが一覧に追加されます、追加したプロファイルを選択し、「次へ」を押下し、ウィザードを進めてください。



業務スケルトンテンプレート利用ガイド(IM-Workflow)

概要

本項では、業務スケルトンテンプレートのIM-Workflowの利用方法について説明します。

本テンプレートの役割について

IM-Workflowは大別して、画面ソースやユーザプログラムを管理する「コンテンツ」、ワークフローのルートを管理する「ルート」、そしてそれらをまとめてワークフロー機能として実現する「フロー」の３つの要素が存在します。

本テンプレートでは、このうち「コンテンツ」で定義可能な以下３つのプログラム作成について支援する機能を提供します。

- 画面
- ユーザプログラム
- リスナ

次項より、実際の使用方法について説明します。

画面/アクション処理

画面/アクション処理テンプレートでは、コンテンツに登録する画面およびアクション処理のソースコード出力、また出力したソースをコンテンツとしてインポートするXMLファイル出力、画面項目定義に沿ったテーブル定義の出力などの機能が利用できます。

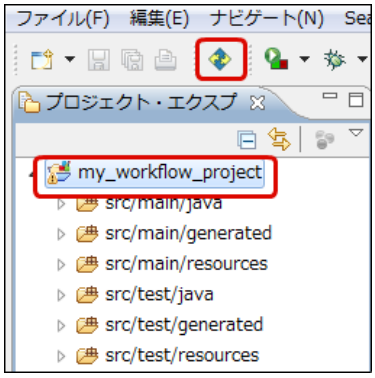
この項では、一例として「スクリプト開発 IM-Workflow 画面 各種画面/アクション処理」テンプレートを利用して
以下のような画面およびアクション処理プログラムを出力、デフォルトで提供されているルート「直線ルート」と組み合わせて
ワークフローとして利用できるまでの一連の流れについて説明します。

- PC版ワークフローの申請画面

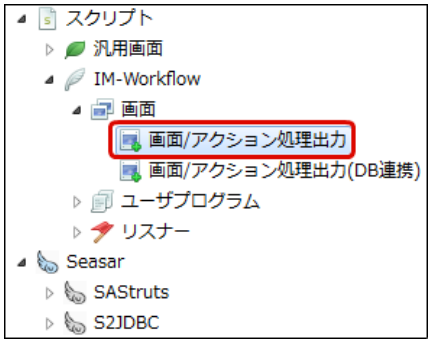
- スマートフォン版ワークフローの申請画面

ステップ1-ソースコードの出力

まず、任意のモジュールプロジェクトを選択し、業務スケルトンウィザードを起動します。

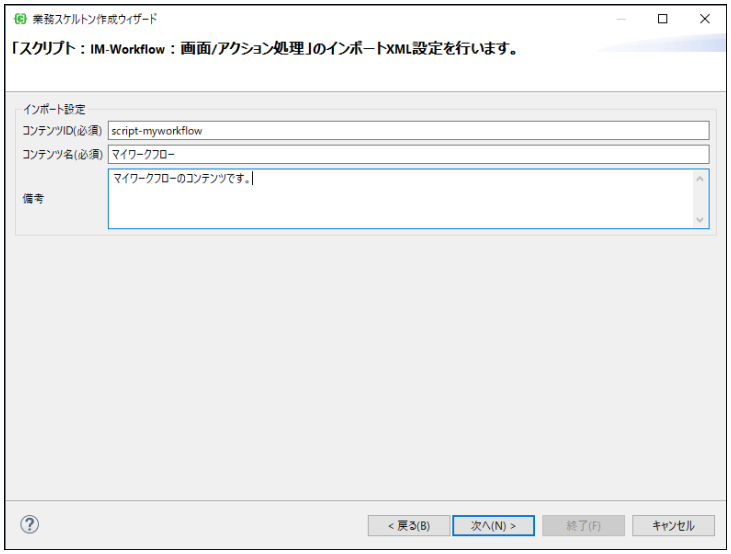


テンプレート選択画面で、「スクリプト」「IM-Workflow」「画面/アクション処理出力」を選択します。

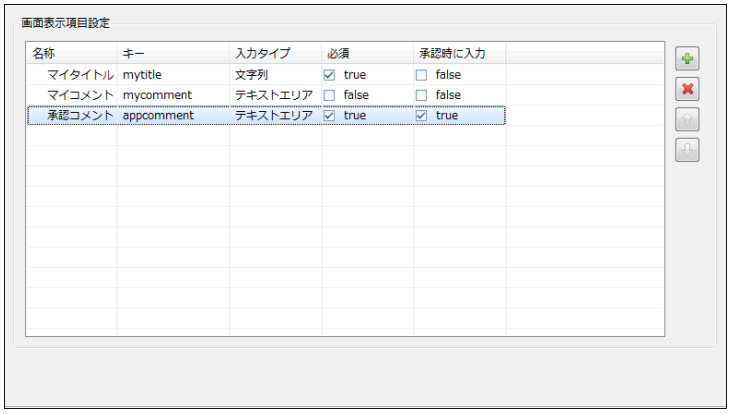


画面の基本設定を行います。
処理名には画面タイトル等に使用する文字列を指定できます。
ここでは例として、処理名に「マイワークフロー」を入力し、次へボタンを押下します。

インポートXMLの設定を行います。任意のコンテンツID/コンテンツ名を設定して次へボタンを押下してください。

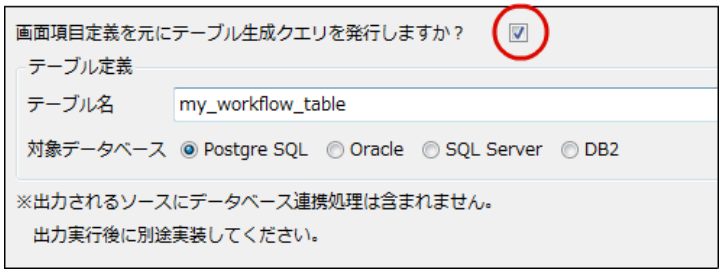


次に画面項目の設定を行います。
ここでは例として以下のような設定を行い、次へボタンを押下します。



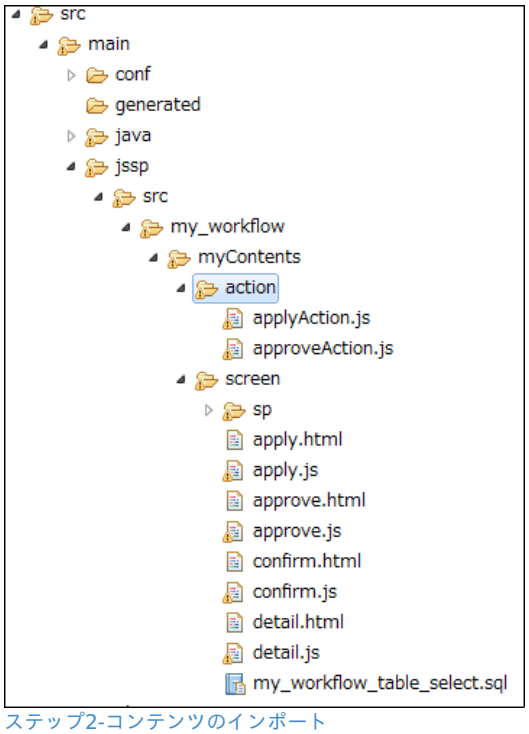
> **コラム**
>
> 「承認時に入力」にチェックした項目は、申請画面では項目表示されず、承認画面でのみ入力可能です。

「画面項目定義を元にテーブル生成クエリを発行しますか？」にチェックを入れ、テーブル名と対象データベースを選択し、次へボタンを押下します。



> **コラム**
>
> 「画面項目定義を元にテーブル生成クエリを発行しますか？」にチェックを入れた場合、画面入力項目に対応したテーブル作成クエリがモジュールプロジェクトの
> src/main/resources/ddl 配下に発行されます。
> 新規ユーザコンテンツ画面を作成するときなどにお使いください。

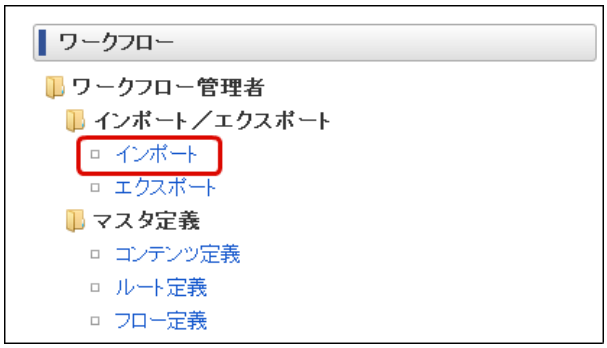
最後に、入力内容の確認をし、「終了ボタン」を押下すると、指定した画面/アクション処理のプログラムが出力されます。



ステップ2-コンテンツのインポート

前項で出力したプログラムをコンテンツ定義としてインポートします。

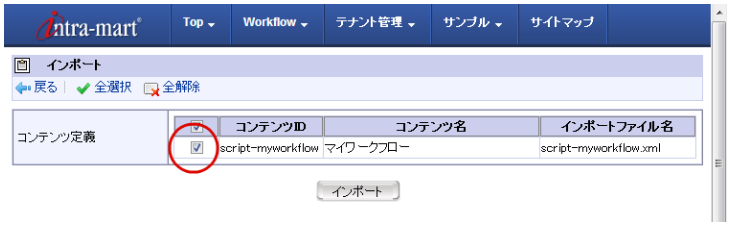
まず、デバッグサーバを起動しておき、IM-Workflow管理者ロールのあるユーザでログインします。
その後、サイトマップに移動し、「ワークフロー」「インポート/エクスポート」「インポート」を選択します。



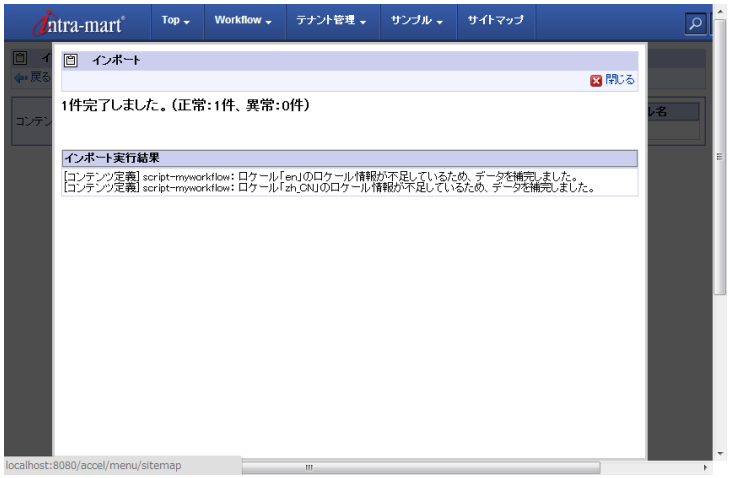
インポート画面から、出力したXMLを選択します。

> **注意**
>
> この項に対象のXMLファイルがない場合は、モジュールプロジェクトの以下のパス配下に対象ファイルがあるか確認してください。
> src/main/storage/public/im_workflow/data/default/import_export

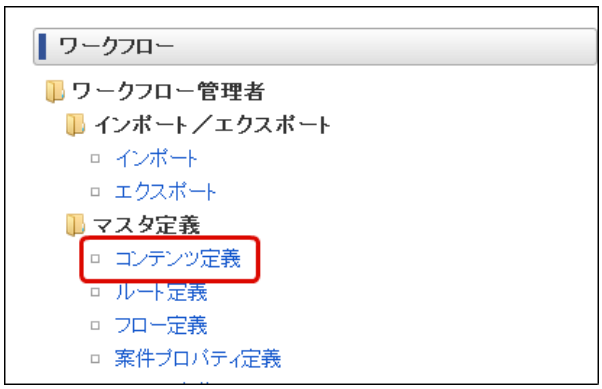
インポートする対象のコンテンツ定義を選択し、インポートを実行します。



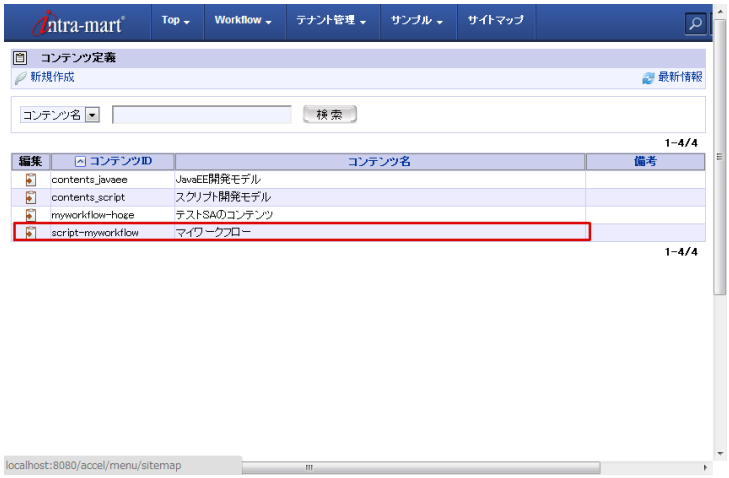
インポート実行結果が出力されます。
※インポートされるXMLには英語/中国語のロケールの定義がないため、日本語の情報がコピーされます。



再度サイトマップに遷移し、「ワークフロー管理者」「マスタ定義」「コンテンツ定義」を選択します。

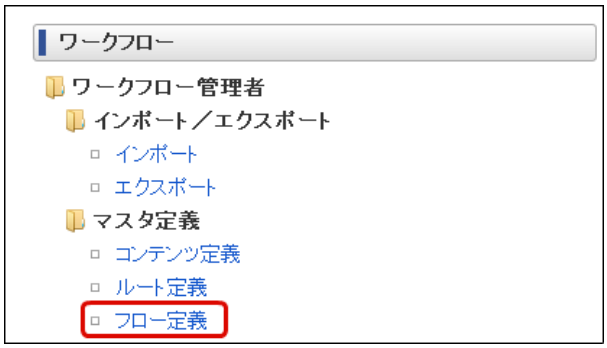


コンテンツ定義の一覧から、インポートしたコンテンツがあることを確認します。



ステップ3-対象コンテンツ定義を利用したフロー定義

前項でインポートしたコンテンツを元に、フローを定義します。
サイトマップから、「ワークフロー管理者」「マスタ定義」「フロー定義」を選択します。



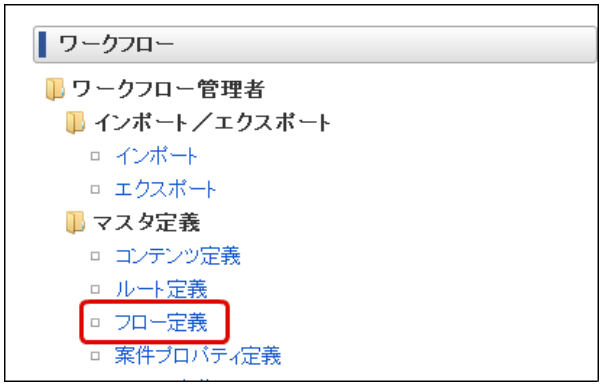
フロー定義画面の「新規作成」ボタンを押下します。

基本情報を登録します。
ここでは、フロー名に「業務スケルトンスクリプトフロー」と入力し、登録ボタンを押下します。

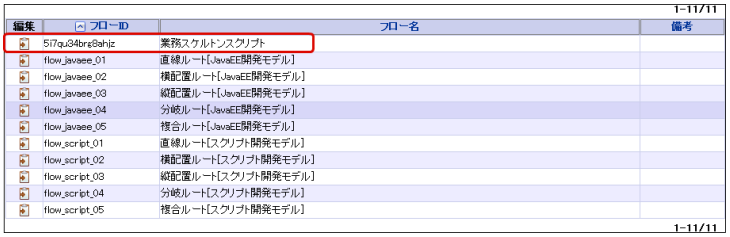
バージョン情報を作成します。
バージョンタブを選択し、「新規作成」ボタンを押下します。

バージョンの基本情報を作成します。
ここでは例として、バージョン期間を今日日付、コンテンツに前項でインポートしたコンテンツ定義を指定、ルートに「直線ルート」を指定し登録ボタンを押下します。

サイトマップより、「ワークフロー管理者」「マスタ定義」「フロー定義」を選択します。



登録されているフロー一覧が表示され、フローが登録されていることを確認します。



ここまでの作業でフローが利用できるようになりました。
メニュー「Workflow」「申請」から当フローを利用し、実際に動作確認など行ってください。

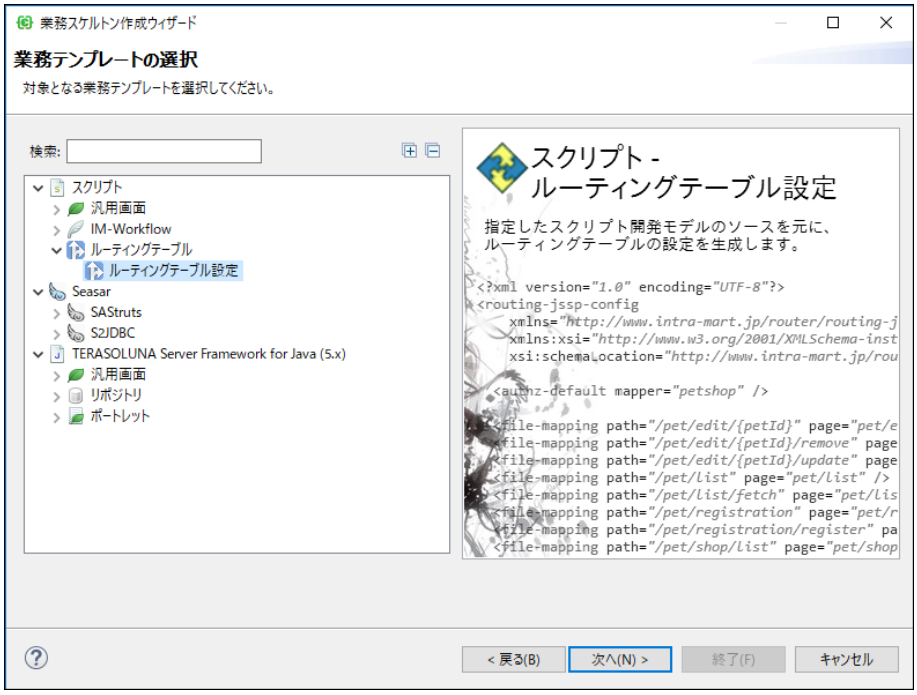
なお、当テンプレートで出力するプログラムにはデータベース接続の処理は含まれませんので、出力されたプログラムに追加実装するか、または別テンプレートの「画面/アクション処理出力（DB連携）」を利用してソースを出力してください。
また、詳しい実装方法等は、別紙「 IM-Workflowプログラミングガイド 」を参照してください。

スクリプト開発 ルーティングテーブル設定の仕様

項目

概要

2014/12月にリリースされたVer.2.0.2より、業務スケルトンにスクリプト開発用のルーティングテーブル設定ファイルを出力する機能が追加されました。
本機能はJavaScriptのメソッドのjsdocに@routeアノテーション、および設定に必要な付加情報を記述します。
この記述された情報を元に、業務スケルトンのメニューの「スクリプト開発 ルーティングテーブル設定」を実行することによってルーティングテーブル設定のxmlファイルを出力します。
本機能を利用するメリットは、jsファイルのメソッド名の修正やファイル名変更、フォルダ移動といったリファクタリングを実施した後でも、この機能を利用することによって設定ファイルの編集をすることなく自動的に出力されることで工数の削減が期待できる点にあります。
ここでは、実際に記述する@routeアノテーションの記述方法とxmlの各要素との対応について説明します。



出力先

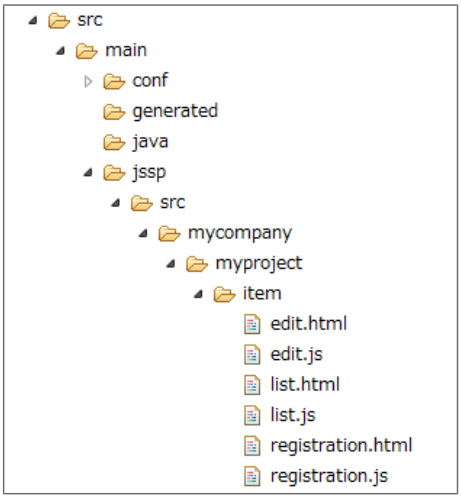
「<%モジュールプロジェクトのパス%>/src/main/conf/routing-jssp-config」にルーティングテーブル設定のxmlファイルを出力します。
ファイル名はデフォルトではプロジェクト名ですが、ウィザード上から変更することもできます。

@routeとxmlの対応

jsdocに@routeアノテーションを記述し、ルーティングの情報を書きます。
ルーティング情報を記述する方法はjson形式とパス形式の2つあります。
json形式では、@route { path : “path/to”, authzUri : *“service://myproject/myresource”* } のように記述します。
また、パス形式では、 @route path/to と記述すします。
「@route 値」形式の値が { で始まる場合はjson形式と判別し、それ以外はパス形式としています。
出力するルーティングテーブルxmlの仕様については、「スクリプト開発モデルルーティング設定」のファイルマッピング設定、認可設定を参照してください。

file-mappingタグの属性には、path, page, action, from, client-typeがあります。
共通的な項目であるpage, actionの項目について、対応する@routeの値への指定方法を説明します。

- page
  pageには、jsファイルのパスとファイル名が入ります。パスは、「src/main/jssp/src」をルートとしたパスで、ファイル名は、.jsを除いた部分です。 例えばeBuilder上で「src/main/jssp/src/mycompany/myproject/item/list.js」の場合、pageの値は「mycompany/myproject/item/list」です。



eBuilder上のファイルイメージ

- action
  init関数以外の場合、actionには@routeを記述した関数名が設定されます。@routeアノテーションで明示的に指定する必要はありません。
  init関数の場合、基本的にはactionには値が設定されません。file-mappingのfrom、actionを使う場合のみactionに値が設定されます。

from, actionを設定する場合の詳細は、「*actionを記述するパターン*」を参照してください。

記述形式のパターンの概観は以下のとおりです。

記述形式	適用する関数	詳細
json形式	*init*	*標準パターン*
		action記述パターン
	init以外	▪
パス形式	*init*	▪
	init以外	▪

json形式

json形式の記述方法について説明します。 @routeの値にjson形式でxmlのpathやclient-typeなどに対応する値を指定できます。

init関数の場合

init関数には、標準パターンとactionを記述するパターンの２つの書き方があります。actionを記述するパターンはあまり使いません。

標準パターン

以下のように<somewhere/page.js>ファイルのinit関数に@routeが記述されていると、以下のようなfile-mappingを生成します。

```
/**
 * @route { path : 'path/to', clientType : 'pc', authzUri : 'service://myproject/myresource' }
 */
function init(request) {
 // ... initの処理
}
```

```
<file-mapping path="path/to" page="somewhere/page" client-type="pc" >
   <authz uri="service://myproject/myresource" action="execute" />
</file-mapping>
```

	xml		
json	**file-mappingタグ**	**authzタグ**	**備考**
path	path		省略時にはpageと同じ値が設定される
なし	page		jsファイルのページが設定される
なし	action		init関数の場合actionは無し
clientType	client-type		
authzUri		uri	
なし		action	固定値”execute”が設定される

actionを記述するパターン

ルーティングテーブルxmlのfile-mappingにaction, fromを記述する必要がある場合のパターンです。
以下のように、@routeアノテーションのjsonにactionを記述して、xmlにaction, fromを出力します。

	xml		
json	**file-mappingタグ**	**authzタグ**	**備考**
path	path		省略時にはpageと同じ値が設定される
なし	page		jsファイルのページが設定される
action	from		jsonのactionにはabc#xyz形式で記述する
	action		from=”abc” action=”xyz”となる
clientType	client-type		
authzUri		uri	
なし		action	固定値”execute”が設定される

init関数以外の場合

<somewhere/page.js>ファイルに、以下のようにregister関数に@routeを記述すると、以下のようなfile-mappingが生成されます。

```
/**
 * @route { path : 'path/to', clientType : 'pc', authzUri : 'service://myproject/myresource' }
 */
function register(request) {
 // ... registerの処理
}
```

```
<file-mapping path="path/to" page="somewhere/page" action="register" client-type="pc" >
   <authz uri="service://myproject/myresource" action="execute" />
</file-mapping>
```

	xml		
json	**file-mappingタグ**	**authzタグ**	**備考**
path	path		省略時にはpageと同じ値が設定される
なし	page		jsファイルのページが設定される
なし	action		@routeを記述してる関数名が設定される
clientType	client-type		
authzUri		uri	
なし		action	固定値”execute”が設定される

パス形式

「@route パス」形式の記述方法について説明します。この形式はパス以外は特に設定しない場合に有用です。 パスの部分は、ダブルクォート、シングルクォート、クォート無しで値を記述できます。例えば、 ”path/to”、’path/to’、path/to はいずれも path/to です。

xmlに出力する結果は、init関数とinit関数以外の2パターンがあります。init関数の場合はfile-mappingのactionに値を設定せず、init関数以外の場合にはactionに値を設定することが違う以外は同じです。

init関数の場合

以下のように<somewhere/page.js>ファイルのinit関数に@routeが記述されていると、以下のようなfile-mappingを生成します。

```
/**
 * @route path/to
 */
function init(request) {
 // ... initの処理
}
```

```
<file-mapping path="path/to" page="somewhere/page" />
```

また、値を省略し@routeのみ記述すると、pathにはpageと同じ値がセットされます。 上と同様に、<somewhere/page.js>ファイルのinit関数に@routeが記述されていると、以下のようなfile-mappingが生成します。

```
/**
 * @route
 */
function init(request) {
 // ... initの処理
}
```

```
<file-mapping path="somewhere/page" page="somewhere/page" />
```

init関数以外の場合

<somewhere/page.js>ファイルに、以下のようにregister関数に@routeを記述すると、以下のようなfile-mappingが生成されます。

```
/**
 * @route path/to
 */
function register(request) {
 // ... registerの処理
}
```

```
<file-mapping path="path/to" page="somewhere/page" action="register" />
```

また、pathを省略し@routeのみ記述すると、pathには「page/関数名」がセットされます。 上と同様に、<somewhere/page.js>ファイルのinit関数に@routeを記述すると、以下のようなfile-mappingが生成されます。

```
/**
 * @route
 */
function register(request) {
 // ... registerの処理
}
```

```
<file-mapping path="somewhere/page/register" page="somewhere/page" action="register" />
```

pathの一意性の判定

ルーティング上のpathは、システム全体で一意にする必要があるので、重複のないように設定してください。 client typeが設定されている場合、client typeが異なればpathの重複は可能です。

例 OK
pathが同じでも、client typeが設定されていて異なるので、一意です。

```
@route { path : "list", clientType : "pc" }
@route { path : "list", clientType : "sp" }
```

例 NG
pathが同じで、client typeが設定されていないルートもあるので、重複します。

```
@route { path : "list" }
@route { path : "list", clientType : "sp" }
```

例 NG
pathが同じなので、重複します。

```
@route { path : "list" }
@route { path : "list" }
```

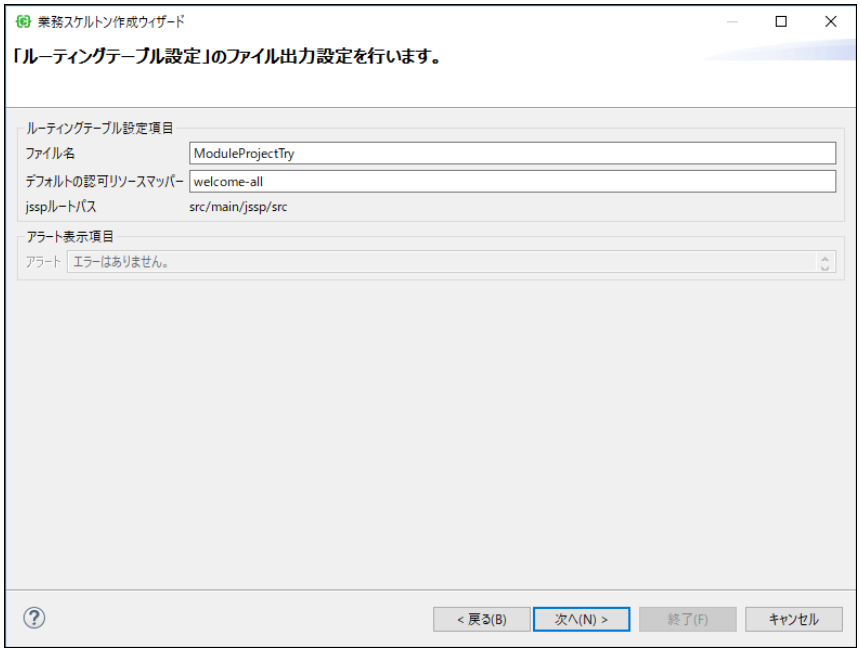
例 NG
pathが同じで、client typeも同じなので、重複します。

```
@route { path : "list", clientType : "pc" }
@route { path : "list", clientType : "pc" }
```

その他

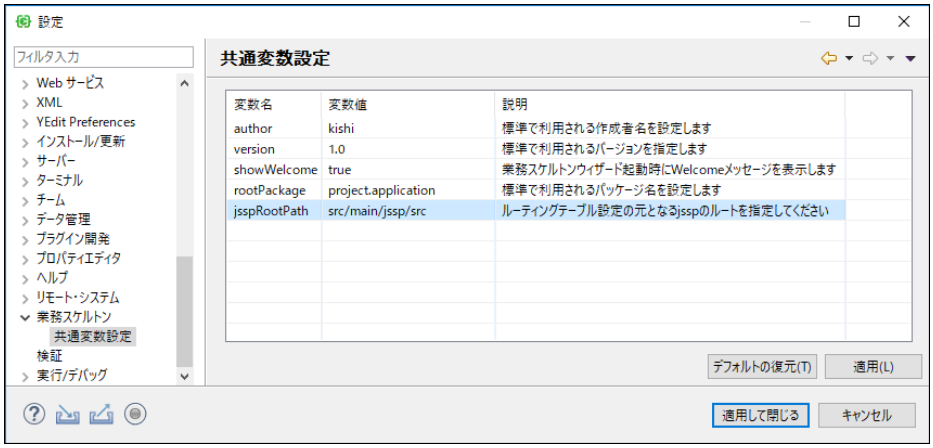
デフォルトの認可リソースマッパを変更する

デフォルトの認可リソースマッパは、デフォルトでは「welcome-all」です。 ウィザード上の「デフォルトの認可リソースマッパ」にて変更できます。



pageのrootを変更する

デフォルトでは、pageのrootは「src/main/jssp/src」です。 上記のパスの情報を元にルーティングテーブル設定ファイルを出力します。 このデフォルトのpageのrootを変更する場合、eBuilderのツールバーから「ウィンドウ - 設定」で設定画面を開き、「業務スケルトン - 共通変数設定」の「jsspRootPath」を変更します。 「src/main/jssp/product/src」のようにproductなどのプロジェクトを開発する場合を想定しています。



Seasar S2JDBC DtoとService生成

項目

概要

この業務スケルトンは、SQLファイルからDtoクラスとServiceクラスを生成します。
エンティティ生成の業務スケルトンと異なり、SQLファイルのSQLを実行するServiceを生成します。このため、複雑なSQLの場合に有効です。

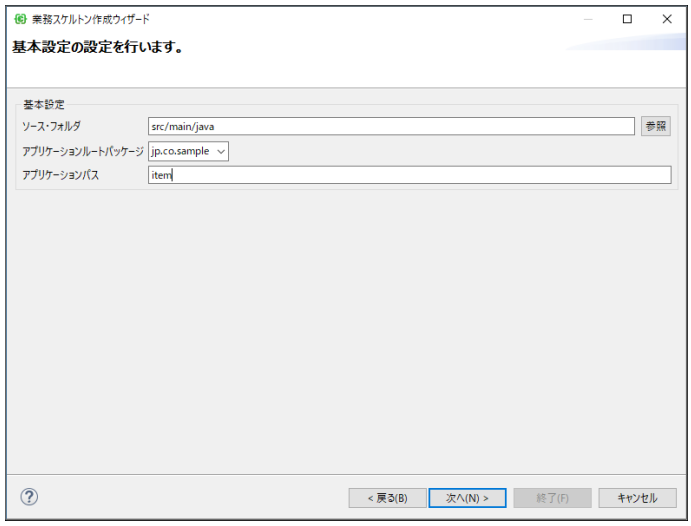
ここでは、この業務スケルトンの使用方法と制限事項について説明します。

対応データベース

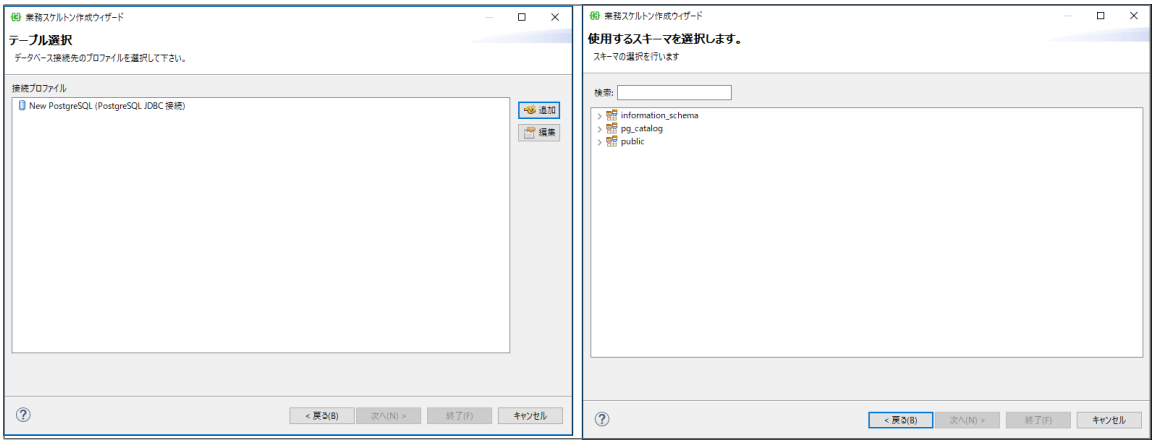
OracleとPostgreSQLに対応しています(Oracle Database 12c Release 1とPostgreSQL 9.4で動作確認しています。)

使用方法

1. アプリケーションルートパッケージ、アプリケーションパスを決定します。
2. データベースにSQLファイルで使用するテーブルをあらかじめ作成しておきます。
3. SQLファイルを配置します。
   SQLファイルの命名方法と配置場所については、「*SQLファイル*」を参照してください。
4. 業務スケルトンを実行し、「DtoとService生成」を選択します。
   1. アプリケーションルートパッケージ、アプリケーションパスを入力します。



   2. スキーマを選択します。



データベース接続先を選択してからスキーマを選択します

   3. 終了をクリックします。
      - エラーがある場合は、メッセージが示しているSQLファイルを修正してください。
   4. 入力Dto、出力Dto、Serviceクラスが出力されます。
      出力ファイルの形式については、「*出力ファイル*」を参照してください。

SQLファイル

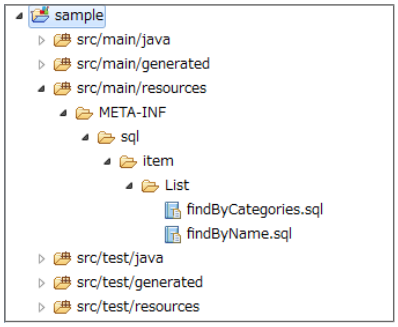
SQLファイルの配置場所やファイル名が、出力するDtoクラスやServiceクラスのクラス名、メソッド名に対応します。

- SQLファイルの配置場所
  src/main/resources配下に、以下の形式のディレクトリの下にsqlファイルを配置します。

```
META-INF/sql/アプリケーションパス/クラス名
```

  「クラス名」の部分がServiceのクラス名「クラス名Service」、Dtoのクラス名「クラス名InDto」、「クラス名OutDto」になります。
  クラス名部分は、Javaのクラス名として正しい名前を付けてください。
- SQLファイルの形式
  「メソッド名」の部分がServiceクラスの当SQLを発行するメソッドのメソッド名になります。
  メソッド名部分はアルファベットと数値のみが使用できます。
  Serviceクラスのメソッド名に対応しますので、Javaのメソッド名として正しい名前を付けてください。
  ファイル名のサフィックスは「.sql」とします。

メソッド名.sql



SQLファイル配置の例

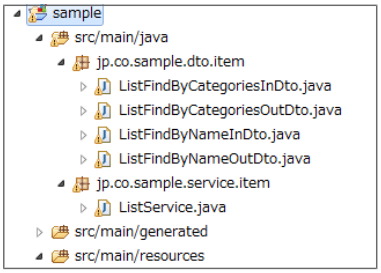
SQL文は、2way SQLで記述します。「S2JDBC SQLファイルによる操作」を参照してください。
2way SQLの例

```
SELECT
  id,
  category,
  name
FROM
  tbl_item
WHERE
  category IN /*categories*/('computer')
ORDER BY
  /*$order*/id
```

出力ファイル

出力するファイルは、入力、出力Dto、Serviceクラスです。 各クラスのパッケージ、クラス名などは以下の通りです。

- 入力、出力Dtoクラス
  - パッケージ
    「アプリケーションルートパッケージ.dto.アプリケーションパス」形式
  - クラス名
    入力Dtoクラス名: SQLファイルのクラス名部分 + メソッド名(Pascal形式) + InDto
    出力Dtoクラス名: SQLファイルのクラス名部分 + メソッド名(Pascal形式) + OutDto
  - 内容
    プロパティのアクセス修飾子はpublicです。getter、setterはありません。
- Serviceクラス
  - パッケージ
    「アプリケーションルートパッケージ.service.アプリケーションパス」形式
  - クラス名
    SQLファイルのクラス名部分 + Service



出力ファイルの例

例
以下の設定、SQLファイルがある場合

アプリケーションルートパッケージ	jp.co.sample
アプリケーションパス	item

sqlファイル	META-INF/sql/item/List/findByName.sql META-INF/sql/item/List/findByCategories.sql

出力されるクラスは以下の通りです。

Serviceクラス	jp.co.sample.service.item.ListService
findByNameのDtoクラス	jp.co.sample.dto.item.ListFindByNameInDto jp.co.sample.dto.item.ListFindByNameOutDto
findByCategoriesのDtoクラス	jp.co.sample.dto.item.ListFindByCategoriesInDto jp.co.sample.dto.item.ListFindByCategoriesOutDto

対応するデータ型

データベースのカラムの型とそれに対応するDtoクラスのプロパティの型は以下の通りです。

カラムの型			
	oracle	**postgresql**	**Javaの型**
文字列型	varchar varchar2 char	varchar char text	String
数値型		bigint integer smallint	java.math.BigInteger
	number(n,0)	numeric(n,0)	java.math.BigInteger
	number(n,m) m != 0	numeric(n,m) m != 0	java.math.BigDecimal
		real	Float
		double precision	Double
日付型	date	date	java.util.Date
		time	java.sql.Time
	timestamp	timestamp	java.sql.Timestamp

上の表にない型(論理型、通貨型、ラージオブジェクト型)は対応していません。Stringクラスとして出力されます。

制限事項

- SELECT文のみに対応しています。INSERTやUPDATEなどは対象外です。
- 条件句の左辺のコメント変数を取得できません。 右辺にコメント変数を記述してください。

```
WHERE
  id = /*id*/1                -- ok
  AND /*name*/'aoyagi' = name   -- ng
```

- サブクエリのカラムの型を解決できません。
  - 例1 SELECT句中の例

```
SELECT
  id,
  -- item_nameを型解決できない
  (SELECT name FROM item_table WHERE id = base.id) AS item_name,
  ...
```

  - 例2 WHERE句中の例

```
-- totalを型解決できない
(SELECT SUM(price) FROM item WHERE item_type = base.item_type) > /*total*/100
```

- CASE式のカラムの型を解決できません。
  - 例1 SELECT句中の例

```
SELECT
  id,
  -- item_nameの型解決できない
  CASE item_type
  WHEN /*itemTypeA*/'A' THEN name,
  WHEN /*itemTypeB*/'B' THEN detail_name,
  ELSE short_name END AS item_name
```

- 例2 WHERE句中の例

```
-- thresholdの型解決できない
CASE item_type WHEN 'A' THEN height ELSE width END > /*threshold*/100
```

- 日付型 + 数値型の型解決について
  colA + colB = /*var*/10 のようにした場合、varの型は左辺の1番目のカラムの型で解決します。

```
date1 + 7 = /*var*/'2000-01-01'  -- varは日付型になる
7 + date1 = /*var*/100          -- varは数値型になる
```

- 対応している関数について
  以下の関数に対応しています。また、その型の扱いは以下の通りです。

関数	型
SUM	パラメータと同じ型
AVG	パラメータと同じ型
CHAR	String
CEILING	java.math.BigInteger
CEIL	java.math.BigInteger
FLOOR	java.math.BigInteger
MOD	java.math.BigInteger
POWER	パラメータが整数の場合java.math.BigInteger それ以外の場合Double
ROUND	パラメータと同じ型
SIGN	パラメータが整数の場合java.math.BigInteger それ以外の場合Double
CONCAT	String
CORRELATION	Double
CORR	Double
COUNT	java.math.BigInteger
MAX	パラメータと同じ型
MIN	パラメータと同じ型
STDDEV	Double
VARIANCE	Double
ABS	パラメータと同じ型
ACOS	Double
ASCII	java.math.BigInteger
ASIN	Double
ATAN	Double
ATAN2	Double
CHR	String
COALESCE	String
COS	Double

関数	型
COSH	Double
COT	Double
DEGREES	Double
EXP	Double
LOWER	パラメータと同じ型
LENGTH	java.math.BigInteger
LN	Double
LOG	Double
LTRIM	パラメータと同じ型
NULLIF	パラメータと同じ型
RTRIM	パラメータと同じ型
SIN	Double
SINH	Double
SQRT	Double
TAN	Double
TANH	Double
TO_DATE	java.sql.Timestamp
TRUNC	パラメータと同じ型
UPPER	String

例

```
SELECT
  -- 出力される型はcol_numberに対応するJavaの型となる
  SUM(col_number1) AS total
...
HAVING
  -- 出力されるlower_limitの型はcol_number2に対応するJavaの型となる
  MAX(col_number2) > /*lower_limit*/10
```

- ネストしたコメント変数には対応していません。 以下の様なネストした変数は対応していません。

```
-- department.attach.user.user_cd のようなネストした変数には対応していません
user_cd = /*department.attach.user.user_cd*/'aoyagi'
-- paging.orderのようなネストした変数には対応していません
ORDER BY /*$paging.order*/user_cd
```

制限事項

- 業務スケルトンはeBuilder 2013 Spring(Climbing)以降において動作するよう設計されています。 eBuilder 2012 Winter(Bourbon)以前のバージョンでは動作しませんのでご注意下さい。
- intra-martが標準で提供する業務テンプレートにおいて、入力項目で指定した値によっては、出力されたコードが対象プロジェクトの制限事項に掛かる可能性があります。
- intra-martが標準で提供する業務テンプレートにおいて、クラス名、パッケージ名、フィールド名等を入力する項目に関しては一般的なJavaの命名規則に従い入力する必要があります。
  一般的なJavaの命名規則に従っていないクラス名(例: 先頭が大文字でないクラス名)、パッケージ名(例: 大文字,"-"が含まれるパッケージ名)、フィールド名(例、先頭が大文字のフィールド名)を指定した場合、意図しない動作を行うスケルトンが出力される可能性がありますのでご注意下さい。
- intra-martが標準で提供する業務テンプレートの中で、入力項目の内、モデルの定義としてフィールド名を要求するものが存在します。
  モデルの定義におけるフィールド名は必ず1つ以上のフィールド名を指定する必要があります。

業務スケルトン 変更履歴

業務スケルトン テンプレート ver3.0.0(2015/12/01リリース)

- 新規テンプレートとしてTERASOLUNA Server Framework for Java (5.x)-リポジトリ-MyBatis3を追加しました。

> **コラム**
>
> TERASOLUNA Global Framework-リポジトリ-MyBatis2、TERASOLUNA Global Framework-リポジトリ-JPAは intra-mart Accel Platform2014 Winter(Iceberg)までご利用いただけます。
> intra-mart Accel Platform2015 Spring(Juno)以降では、ご利用いただけません。
>
> TERASOLUNA Server Framework for Java (5.x)-リポジトリ-MyBatis3は intra-mart Accel Platform2015 Spring(Juno)からご利用いただけます。
> intra-mart Accel Platform2014 Winter(Iceberg)以前では、ご利用いただけません。

業務スケルトン テンプレート ver2.0.2(2014/12/01リリース)

- 新規テンプレートとしてスクリプト開発 ルーティングテーブル設定を追加しました。
- スクリプト開発-汎用画面テンプレート、Seasar-SAStruts-汎用画面、TERASOLUNA Global Framework-汎用画面にCSRF対策を適用しました。

> **コラム**
>
> Seasar-SAStruts-汎用画面、TERASOLUNA Global Framework-汎用画面でCSRF対策を有効化させる方法については、「*Seasar SAStruts 汎用画面テンプレート*」、「*TERASOLUNA Server Framework for Java (5.x) 汎用画面テンプレート*」を参照してください。

業務スケルトン テンプレート ver2.0.1(2013/10/01リリース)

- 新規テンプレートとしてTERASOLUNA Global Framework-汎用画面-マスタメンテナンスを追加しました。

業務スケルトン テンプレート ver2.0.1(2013/06/20リリース)

- 新規テンプレートとしてSeasar-S2JDBC-エンティティ作成を追加しました。
- Seasar-SAStruts-汎用画面(DB連携)を仕様変更し、Seasar-SAStruts-汎用画面(S2JDBC連携)と名称変更しました。

> **コラム**
>
> これまで Seasar-SAStruts-汎用画面(DB連携) を使用されていた方は、
> Seasar-S2JDBC-エンティティ作成テンプレートで予めエンティティ/サービスクラスを出力し、
> Seasar-SAStruts-汎用画面(S2JDBC連携)を使用することで以前同様の出力ができます。

- スクリプト開発-汎用画面テンプレートにスマートフォン版画面を出力する機能を追加しました。
- Seasar-SAStruts-汎用画面テンプレートにスマートフォン版画面を出力する機能を追加しました。
- 業務テンプレート選択ウィザード時に表示する各テンプレート画像を更新しました。

業務スケルトン テンプレート ver2.0.0(2013/04/01リリース)

- 新規テンプレートとしてSeasar SAStruts 汎用画面を追加しました。
- 新規テンプレートとしてスクリプト開発 汎用画面を追加しました。