IM-MessageHub プログラミングガイド

改訂情報

変更年月日	変更内容
2015-04-01	初版
2015-12-01	第2版 下記を追加・変更しました ■ 「*DeliveryMediaResolver 一覧*」のIMBoxStandardDeliveryMediaResolverに関する記載を修正
2021-04-01	第3版 下記を追加・変更しました ■ 「*配信先メディア一覧*」にSlackのメッセージ通知を追加

はじめに

本書の目的

本書では IM-MessageHub を利用した開発手順について説明します。

説明範囲は以下の通りです。

- IM-MessageHub の概要・基本仕様
- IM-MessageHub を利用したメッセージの配信方法
- IM-MessageHub からのメッセージを受け取る方法（配信先メディアを追加する方法）

対象読者

本書では次の開発者を対象としています。

- IM-MessageHub を利用してメッセージの配信を行いたい。
- IM-MessageHub から配信されるメッセージを受け取りたい（配信先メディアを追加したい）。

また、本書を読み進める上で次の内容を理解していることが必須となります。

- JavaEE を利用したWebアプリケーションの基礎

対象開発モデル

本書では以下の開発モデルを対象としています。

- JavaEE開発モデル

サンプルコードについて

本書に掲載されているサンプルコードは可読性を重視しており、性能面や保守性といった観点において必ずしも適切な実装ではありません。
サンプルコードを参考に開発する場合は、上記について十分に注意してください。

なお、ドキュメントの流れに合わせてサンプルを実装し、実際に動作確認を行う場合の前提条件は以下の通りです。

- 以下のモジュールをインストールした環境を構築してください。
  - IM-MessageHub （標準機能/基盤機能）
  - 汎用テンプレートエンジン （標準機能/基盤機能）
  - IMBox （標準アプリケーション）
- intra-mart Accel Platform のサンプルをインストールしてください。

本書の構成

- *IM-MessageHub 概要*

  IM-MessageHub の概要と、 IM-MessageHub を構成する要素に関して説明します。

- *IM-MessageHub を利用してメッセージを配信する*

  IM-MessageHub を利用したメッセージ配信方法を説明します。 IMBox、IM-Notice、メールなど、IM-MessageHub に登録された様々な配信先メディアに対して、メッセージを一括で配信する機能を開発する場合に参照してください。

- *IM-MessageHub に配信先を追加する*

  IM-MessageHub から配信されるメッセージを受け取る方法（配信先メディアを追加する方法）を説明します。 intra-mart Accel Platform 上で動作する様々なアプリケーションから配信されるメッセージを受け取り、活用する機能を開発する場合に参照してください。

- *付録*

  IM-MessageHub が提供する標準実装の一覧や仕様などを説明します。

IM-MessageHub 概要

ここでは IM-MessageHub の概要と構成要素について説明します。

項目

IM-MessageHub とは

IM-MessageHub とは、メッセージ配信の共通機構です。
intra-mart Accel Platform 上のアプリケーションや各機能は、共通の受け口にメッセージの配信を依頼することで、メッセージを一斉配信することができます。

IM-MessageHub が提供する主な機能は以下の通りです。

- **メッセージ配信の共通的な受け口**

  アプリケーションや各機能は、 IM-MessageHub へ配信を依頼するだけで、 IM-MessageHub に登録されている全ての配信先メディアにメッセージを配信することができます。

- **配信先をプラガブルに追加可能**

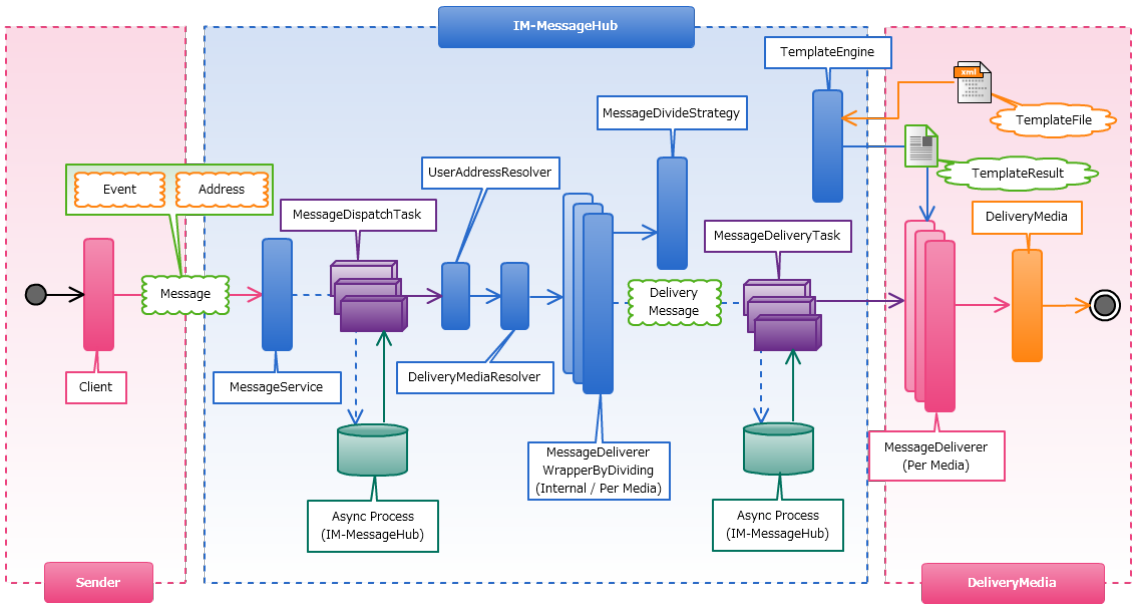
  新たに配信先メディアやデバイスが増えた場合でも、送信元のアプリケーションや各機能のソースコードへの修正は不要です。

- **メッセージの宛先指定方法を拡張可能**

  IM-MessageHub では指定された宛先をユーザ単位に自動解決した上でメッセージ配信する仕組みが備わっています。
  これにより、例えば「ある組織に所属するメンバー全員」というようなグループ単位で宛先を指定するといった拡張が可能です。

IM-MessageHub の全体の流れ

IM-MessageHub を利用したメッセージ配信は、下図のように行われます。



図：メッセージの流れ

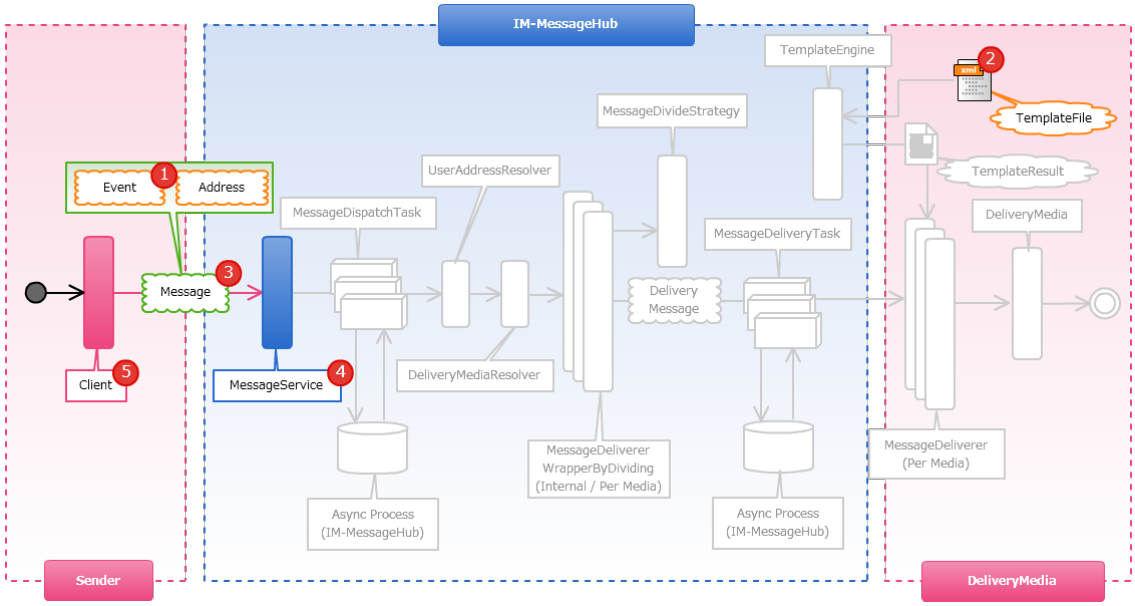
メッセージ配信は、大きく3つの部分に分けられます。

1. *メッセージ配信アプリケーション*（図中の Sender ）
2. *配信先メディア*（図中の DeliveryMedia ）
3. 上記２つの間でメッセージ配信の制御を行う「 IM-MessageHub 」（図中の IM-MessageHub ）

そのうち、本書では「 *メッセージ配信アプリケーション* 」側 と「 *配信先メディア* 」側に分けて、実装の流れを説明します。

IM-MessageHub を利用してメッセージを配信する

IM-MessageHub を利用してメッセージを配信する 「 *メッセージ配信アプリケーション* 」側の構成は、下図の通りです。



図： *IM-MessageHub を利用してメッセージを配信する* 「 *メッセージ配信アプリケーション* 」側の構成図

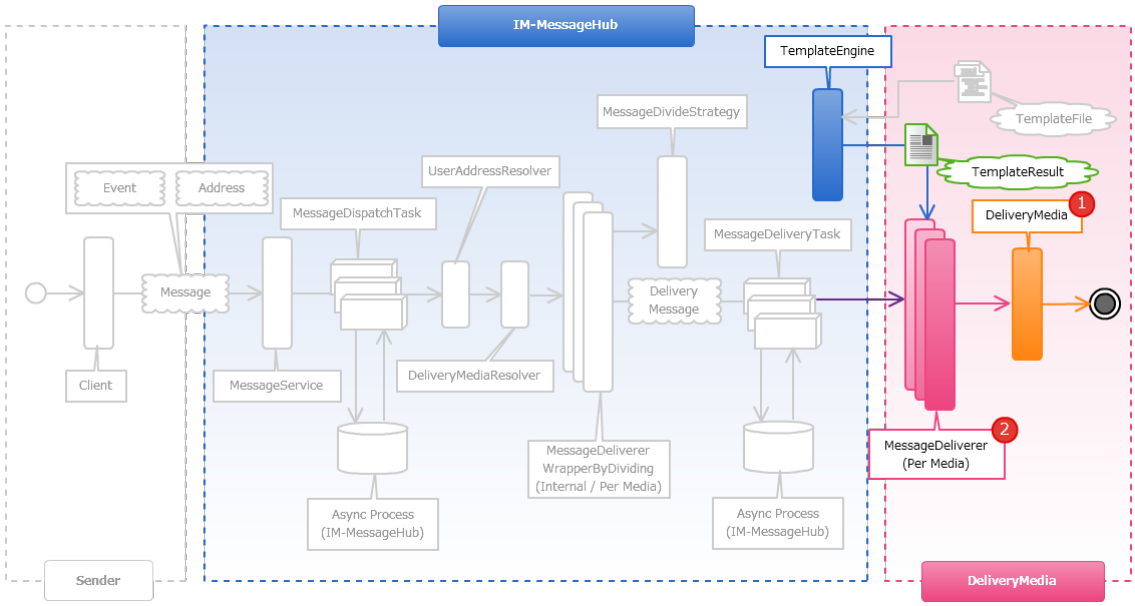
IM-MessageHub を利用して、メッセージを配信するために必要な資材は以下の通りです。

1. 配信の契機を表すイベントクラス（図中の Event ）
2. 配信するメッセージのテンプレート（図中の TemplateFile ）
   - 処理フローとして「 *配信先メディア* 」側にありますが、実際には配信する側が用意すべき資材になります。
3. 配信するメッセージ（図中の Message ）
4. 配信処理を依頼するサービスクラス（図中の MessageService ）
5. （1～4の資材・クラスを用いて）配信処理を実際に行うクライアント（図中の Client ）

各資材の準備方法についての詳細は「 *IM-MessageHub を利用してメッセージを配信する* 」を参照してください。

IM-MessageHub に配信先を追加する

IM-MessageHub に配信先を追加する 「 *配信先メディア* 」側の構成は、下図の通りです。



図： *IM-MessageHub に配信先を追加する* 「 *配信先メディア* 」側の構成図

IM-MessageHub を利用して、配信されるメッセージを受け取る（配信先メディアを追加する）ために必要な資材は以下の通りです。

1. 配信先メディアの定義（図中の DeliveryMedia ）
2. 配信先メディアへメッセージを配信するクラス（図中の MessageDeliverer(Per Media) ）

各資材の準備方法についての詳細は「 *IM-MessageHub に配信先を追加する* 」を参照してください。

IM-MessageHub を構成する要素

IM-MessageHub を構成する主要な要素と、その詳細について説明します。

メッセージ

IM-MessageHub が一回の処理で取り扱う情報です。

IM-MessageHub の扱うメッセージの構成内容は以下の通りです。

- メッセージID（ユニーク）
- メッセージが発生した *イベント*
- メッセージの送信者
- メッセージの宛先
- メッセージの付帯情報（属性）

メッセージ配信受付API

IM-MessageHub が提供する、メッセージ配信の受け口となるAPIです。

IM-MessageHub を利用してメッセージ配信を行うすべてのアプリケーションは、配信依頼を行う際にこのAPIを利用します。

メッセージ配信アプリケーション

IM-MessageHub を用いてメッセージの配信を行っているアプリケーションを指します。
intra-mart Accel Platform が提供するアプリケーションでは「 IM-Workflow 」「 IMBox 」「 intra-mart Accel Collaboration 」などが挙げられます。

イベント

IM-MessageHub に対してメッセージ配信依頼を行う契機となった情報です。
イベントを構成する具体的な要素は以下の通りです。

- メッセージの配信依頼をしたアプリケーション
- メッセージの配信依頼の契機となった処理

IM-MessageHub はイベントの情報からテンプレートの決定などを行います。

配信先メディア

IM-MessageHub が依頼されたメッセージを実際に配信する先となる媒体やアプリケーションなどの総称を指します。
intra-mart Accel Platform が提供している配信先メディアには 「メール」「 IMBox 」「 IM-Notice 」などが挙げられます。

配信先メディア解決

IM-MessageHub が配信するメッセージの配信先メディアを決定する機能です。

通常、IM-MessageHub は、登録されている全ての配信先メディアへ配信処理を行います。
配信先メディア解決では、配信を依頼されたメッセージのイベント情報などから、登録されている配信先メディアを絞り込みます。
IM-MessageHub では配信先メディア解決を利用した場合、絞り込まれた配信先メディアにのみ配信処理を行います。

配信クラス

IM-MessageHub に配信依頼されたメッセージを、登録されている配信先メディアへ実際に配信するクラスです。

IM-MessageHub が依頼されたメッセージを配信先メディアへ配信する場合、内部的にはそれぞれの配信先メディアに紐づく配信クラスへ、配信処理を依頼する形となります。
開発者が新しい配信先メディアを追加しようとした場合、新しい配信先メディアへ配信処理を行う配信クラスを作成することとなります。

テンプレート

IM-MessageHub がメッセージを配信する際に利用するテンプレートを指します。
IM-MessageHub を利用してメッセージを配信する際、メッセージ配信アプリケーションは、利用するテンプレートを作成することが可能です。
その他、 IM-MessageHub では、従来 intra-mart Accel Platform で利用してきたメールテンプレートについてもサポートしています。

メッセージ配信アプリケーションから渡ってきた情報 と テンプレートを元に、配信先メディアに配信される実際のメッセージを生成する機能です。

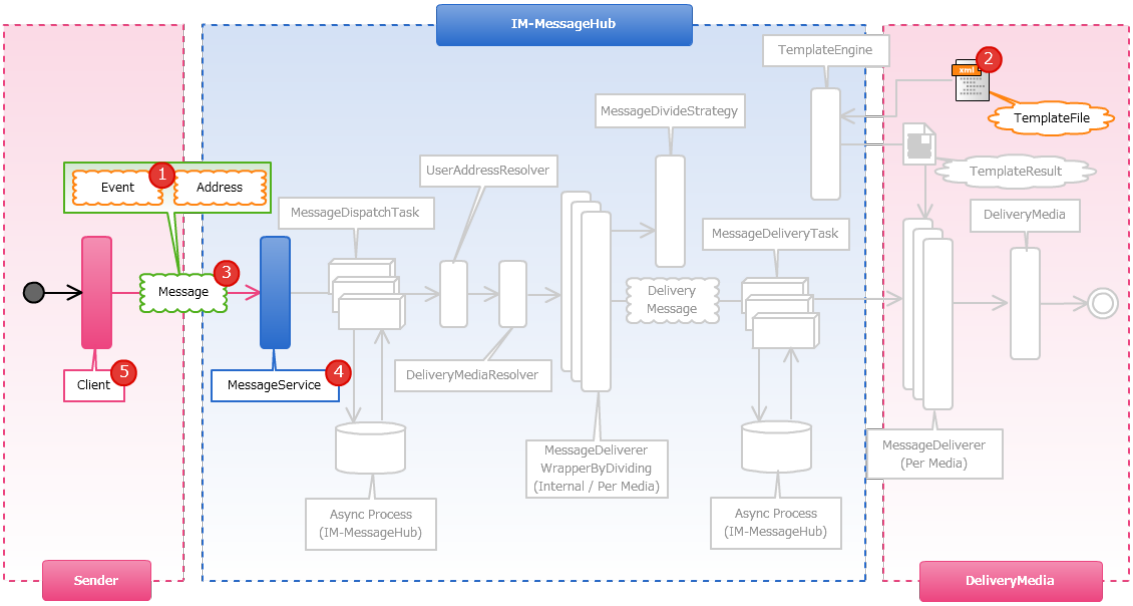
全体の作業の流れ

ここでは IM-MessageHub を利用してメッセージを配信する方法を説明します。

この章を通して以下のことを学びます。

- IM-MessageHub へのメッセージ配信依頼方法
  - イベントの定義方法
  - テンプレートの定義方法
  - メッセージの作成方法
  - メッセージ配信受付 API の利用方法

IM-MessageHub を利用して、メッセージを配信するための流れは以下の通りです。



図：メッセージを配信するための流れ

まず、IM-MessageHub を利用してメッセージを配信処理を実装する前に、以下の３つを定義してください。

1. **どのようなイベントが発生した時に、**
2. **どのようなメディアを通じて、**
3. **どのようなメッセージを送りたいか？**

例えば、 IMBox では、(1) Direct Messageを受信した時に、(2) 「IM-Notice」と「メール」を通じて、(3) Direct Messageを受信したことを伝えるメッセージを配信しています。

今回のサンプルでは、以下のようなメッセージを配信することとします。

1. SampleMessageEvent が発生した時に、
2. IMBox の ApplicationBox を通じて、
3. SampleMessageEvent が発生したことを伝えるメッセージ

全体の作業手順は以下の通りです。

表： IM-MessageHub を利用してメッセージを配信するための作業手順

手順	作成する資材	作業解説
1	配信の契機を表すイベントクラス（図中の `Event` ） ▪ イベントクラス	*イベントを定義する*
2	配信するメッセージのテンプレート（図中の `TemplateFile` ） ▪ テンプレートファイル（日本語、英語、中国語）	*テンプレートを定義する*
3	配信するメッセージ（図中の `Message` ） ▪ メッセージを作成する処理	*メッセージを作成する*

手順	作成する資材	作業解説
4	配信処理を依頼するサービスクラス（図中の MessageService ） ▪ メッセージの配信を IM-MessageHub へ依頼する処理	*メッセージを配信する*
5	配信処理を実際に行うクライアント（図中の Client ） ▪ 配信処理を行うクライアントアプリケーション	*サンプルの動作を確認する*

なお、流れに合わせてサンプルを実装し、実際に動作確認を行う際の前提条件は「 *サンプルコードについて* 」を参照してください。

イベントを定義する

最初に、イベント情報を定義するため、 Event クラスを継承したクラスを作成してください。

- jp.co.intra_mart.foundation.message_hub.model.Event

イベントクラスの実装例は以下の通りです。

```
1  package sample.message_hub.event;
2
3  import jp.co.intra_mart.foundation.message_hub.model.Event;
4
5  /**
6   * メッセージ送信のサンプルイベントクラスです。
7   */
8  public class SampleMessageEvent extends Event {
9
10  private static final long serialVersionUID = -221439640194597296L;
11
12 }
```

> **コラム**
> Event クラスを継承したクラスに、実装が必要なメソッドはありません。

テンプレートを定義する

次に、配信するメッセージのテンプレートを定義します。
ここでは、テンプレートの仕様や規約、および、テンプレートのフォーマットについて説明します。

テンプレートファイル

テンプレートファイルとは、 IM-MessageHub で利用するテンプレート情報を定義した XML ファイルです。
テンプレートファイルを作成する上での規約、および、仕様は以下の通りです。

ファイル名

テンプレートファイル名の命名規約は以下の通りです。

＜配信先メディアID＞-＜イベントID＞_＜ロケール＞.xml

テンプレートファイル名は、配信先メディアやイベントによって決定されます。
動的に決定が行われる箇所（＜～＞によって囲まれている部分）についての詳細は以下の通りです。

表：動的に決定が行われる箇所の詳細

配信先メディアID	テンプレートファイルを適用する「*配信先メディア*」の ID を指定します。 intra-mart Accel Platform で提供している配信先メディアID の詳細については、「*配信先メディア一覧*」を参照してください。 今回のサンプルでは、 IMBox の ApplicationBox 向けのテンプレートを作成します。配信先メディアID は `immh.imbox.appbox` です。
イベントID	テンプレートファイルを適用するイベントの ID を指定します。 イベントIDは、適用するイベントクラスの FQCN（完全修飾クラス名）です。 今回のサンプルでは、 `sample.message_hub.event.SampleMessageEvent` がイベントID になります。
ロケール	テンプレートファイルのロケールを指定します。 言語ごとにテンプレートファイルを作成することで、多言語に対応したメッセージを配信することが可能です。 ロケールは `＜language＞_＜country＞_＜variant＞` 形式で指定します。 今回のサンプルでは、日本語（ `ja` ）、英語（ `en` ）、中国語（ `zh_CN` ）、および、ロケール指定なしを定義します。

テンプレートファイルは、これらの情報を省略して命名することも可能です。
詳しくは「*テンプレートファイルの解決順*」を参照してください。

> **コラム**
>
> メディア共通テンプレートについて
>
> テンプレートファイル名の「配信先メディアID」を省略した場合、そのテンプレートは、全ての配信先メディアで共通的に利用されるテンプレートとなります。
> これを「メディア共通テンプレート」と呼びます。
>
> 新たにテンプレートを定義する際は、メディア共通テンプレートも作成してください。
> これにより、システムに新しい配信先メディアが増えた場合でも、新たにテンプレートファイルを作成する必要がなくなります。
>
> メディア共通テンプレートを作成する際の注意点は以下の通りです。
>
> - 既存の配信先メディアで必要となるヘッダー情報を全て定義してください。
>   必須ヘッダーの詳細は「*必須ヘッダ情報*」を参照してください。
> - 利用する言語分のテンプレートファイルを作成してください。
> - どの配信先メディアに配信されても違和感の無い内容にしてください。簡潔で短い内容であることを推奨します。

配置場所

IM-MessageHub は以下のディレクトリをルートディレクトリとして、テンプレートファイルを管理します。

```
%PUBLIC_STORAGE_PATH%/im_template/
```

作成したテンプレートファイルは、このルートディレクトリ配下に配置してください。

なお、 IM-MessageHub はルートディレクトリ配下のサブディレクトリを含めて管理対象としています。
新規に作成したテンプレートファイルは、 intra-mart Accel Platform 標準で同梱されるテンプレートファイルと区別するため、専用のディレクトリを作成することを推奨します。

今回のサンプルで利用するテンプレートファイルの配置場所は以下の通りです。
いずれも、 `%PUBLIC_STORAGE_PATH%/im_template/`**`sample`**`/` ディレクトリ直下に配置しています。

- IMBox の ApplicationBox 向けテンプレートファイル
  - **immh.imbox.appbox**-sample.message_hub.event.SampleMessageEvent.xml （ロケール指定なし）
  - **immh.imbox.appbox**-sample.message_hub.event.SampleMessageEvent_**ja**.xml （日本語）
  - **immh.imbox.appbox**-sample.message_hub.event.SampleMessageEvent_**en**.xml （英語）
  - **immh.imbox.appbox**-sample.message_hub.event.SampleMessageEvent_**zh_CN**.xml （中国語）
- メディア共通テンプレート
  - sample.message_hub.event.SampleMessageEvent.xml （ロケール指定なし）
  - sample.message_hub.event.SampleMessageEvent_**ja**.xml （日本語）
  - sample.message_hub.event.SampleMessageEvent_**en**.xml （英語）
  - sample.message_hub.event.SampleMessageEvent_**zh_CN**.xml （中国語）

テンプレートファイルの解決順

IM-MessageHub は、イベント、配信先メディア、および、対象となるロケールを元に利用するテンプレートファイルを解決します。
テンプレートファイルの解決順は以下の通りです。

1. 「配信先メディアID」、「イベントID」、「ロケール」が全て一致するテンプレートファイル
2. 「配信先メディアID」、「イベントID」が一致するテンプレートファイル
3. 「イベントID」、「ロケール」が一致するテンプレートファイル

4. 「イベントID」が一致するテンプレートファイル

ロケールに関しては更に以下の順で評価されます。

1. 「 language 」「 country 」「 variant 」が全て一致
2. 「 language 」「 country 」が一致
3. 「 language 」が一致

> **コラム**
>
> テンプレートファイルが解決できなかった場合
>
> 上記手順を通してテンプレートファイルの解決が行えなかった場合、 IM-MessageHub ではデフォルトテンプレートファイルを利用します。
> デフォルトテンプレートファイルが配置されているパスは以下の通りです。
>
> %PUBLIC_STORAGE_PATH%/im_template/default_template.xml

例として、イベントに SampleMessageEvent 、配信先メディアIDに immh.imbox.appbox 、ロケールに日本語（ ja ）が指定された場合、IM-MessageHub は以下の順番でテンプレートファイルの解決を行います。

1. immh.imbox.appbox-sample.message_hub.event.SampleMessageEvent_ja.xml
2. immh.imbox.appbox-sample.message_hub.event.SampleMessageEvent.xml
3. sample.message_hub.event.SampleMessageEvent_ja.xml
4. sample.message_hub.event.SampleMessageEvent.xml
5. default_template.xml （デフォルトテンプレートファイル）

テンプレートフォーマット

テンプレートを作成するためのフォーマットの詳細は以下の通りです。

本文

配信を行うメッセージと、その付帯情報を定義します。
テンプレートの定義例は以下の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<template-data xmlns="http://www.intra-mart.co.jp/system/template/template-data"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.co.jp/system/template/template-data ../schema/template-data.xsd">
  <headers>
    <header name="subject"         value="Message Title" />
    <header name="url"             value="http://www.intra-mart.jp/" />
    <header name="application_info" value="Application-Information" />
  </headers>
  <body>
    Main Message Contents.
  </body>
</template-data>
```

設定が必要なタグとその説明は、以下の通りです。

- headers タグ

  テンプレートに関するヘッダー情報をまとめて定義するための親タグです。
  headers タグの内部には、 header タグを複数定義することが出来ます。

- header タグ

  テンプレートに関するヘッダー情報を定義します。

  header タグは属性として name と value を持ち、name にはヘッダー名を、 value にはヘッダー名に対応する値を設定します。
  intra-mart Accel Platform が提供している配信先メディアによっては、必ず設定しなければならないヘッダー情報が存在します。
  詳しくは「 *必須ヘッダ情報* 」を参照してください。

- body タグ

  配信を行うメッセージ本文を定義します。

置換処理（プレースホルダ）

IM-MessageHub のテンプレートは、プレースホルダを利用した置換処理に対応しています。

テンプレートファイルの例は以下の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<template-data xmlns="http://www.intra-mart.co.jp/system/template/template-data"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.co.jp/system/template/template-data ../schema/template-data.xsd">
  <headers>
    <header name="application_placeholder" value="${applicationPlaceholder}" />
  </headers>
  <body>
    Main Message Contents.

    --------------------------------

    Replace Value : ${replaceValue}
  </body>
</template-data>
```

- プレースホルダは、 `${＜プレースホルダ名＞}` というフォーマットで定義します。
  プレースホルダ名は、英数字大文字小文字のキャメルケースで定義する事を推奨します。
- テンプレートファイルに指定されたプレースホルダは、メッセージに指定された同名の属性に置換されます。

  例えば、メッセージの作成時に Message#setAttribute(" `foo` ", " **bar** ") と設定された場合、テンプレートファイル上の `${ foo }` は **bar** に置換されます。
  詳しくは「 *置換処理の対応* 」を参照してください。
- テンプレートファイル上でプレースホルダが設定可能な箇所は以下の通りです。
  - `header` タグの `value` 属性内
  - `body` タグ内

> **警告**
>
> プレースホルダ名に「.（ドット）」を含めないでください。
> 今後のアップデートにおいて、設定したプレースホルダが正しく動作しなくなる可能性があります。

完成したテンプレートファイルのサンプル

今回は、以下のようなテンプレートファイルを作成します。
（メディア共通テンプレートのサンプルは割愛します）

- immh.imbox.appbox-sample.message_hub.event.SampleMessageEvent_ja.xml （日本語）

```
<?xml version="1.0" encoding="UTF-8"?>
<template-data xmlns="http://www.intra-mart.co.jp/system/template/template-data"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.co.jp/system/template/template-data ../schema/template-data.xsd">
<headers>
   <header name="subject" value="IM-MessageHub プログラミングガイド サンプル"/>
   <header name="url"     value="${url}"/>
</headers>
<body>
サンプルアプリケーションからの通知です。
通知処理日時：${sendDate}
</body>
</template-data>
```

- immh.imbox.appbox-sample.message_hub.event.SampleMessageEvent_en.xml （英語）
  および、 immh.imbox.appbox-sample.message_hub.event.SampleMessageEvent.xml （ロケール指定なし）

```
<?xml version="1.0" encoding="UTF-8"?>
<template-data xmlns="http://www.intra-mart.co.jp/system/template/template-data"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.co.jp/system/template/template-data ../schema/template-data.xsd">
<headers>
   <header name="subject" value="IM-MessageHub Programming Guide Sample"/>
   <header name="url"     value="${url}"/>
</headers>
<body>
Notification from the Sample Application.
Notification processing Date : ${sendDate}
</body>
</template-data>
```

- immh.imbox.appbox-sample.message_hub.event.SampleMessageEvent_zh_CN.xml （中国語）

```
<?xml version="1.0" encoding="UTF-8"?>
<template-data xmlns="http://www.intra-mart.co.jp/system/template/template-data"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.co.jp/system/template/template-data ../schema/template-data.xsd">
<headers>
   <header name="subject" value="IM-MessageHub □程指南 □本"/>
   <header name="url"     value="${url}"/>
</headers>
<body>
通知从示例□用程序。
通知□理日期： ${sendDate}
</body>
</template-data>
```

メッセージの配信を依頼する

最後に、配信するメッセージを作成し、 IM-MessageHub に対してそのメッセージの配信を依頼します。
今回は、メッセージの作成、および、配信依頼を行うサンプルを作成します。

項目

メッセージを作成する

はじめに、配信するメッセージの作成方法を説明します。
メッセージを作成する上で必要な情報は以下の通りです。

- イベント
- 送信元のアドレス情報
- 宛先のアドレス情報リスト
- 任意の属性値

これらの情報をもとに、 IM-MessageHub で配信するメッセージを作成するサンプルを実装します。
メッセージを作成する処理の実装例は以下の通りです。

```
1   /**
2    * メッセージを作成します。
3    * @return メッセージ
4    */
5   public static Message createMessage(){
6     // イベントの作成
7     final Event event = new SampleMessageEvent();
8
9     // 送信元アドレスの作成
10    // 今回、送信元アドレスはアカウントコンテキストから取得しています。
11    AccountContext context = Contexts.get(AccountContext.class);
12    final Address fromAddress = new UserAddress(context.getUserCd());
13
14    // 送信先アドレスの作成
15    final List<Address> toAddressList = new ArrayList<Address>();
16
17    // サンプルとしていくつかの宛先を追加する。
18    toAddressList.add(new UserAddress("aoyagi")); // 青柳
19    toAddressList.add(new UserAddress("ueda")); // 上田
20    toAddressList.add(new UserAddress("hayashi")); // 林
21    toAddressList.add(new UserAddress("maruyama")); // 円山
22
23    // 属性の設定
24    final Map<String, Object> attributes = new HashMap<String, Object>();
25    attributes.put("url",      "http://www.intra-mart.jp/");
26    attributes.put("sendDate", DateTime.now(TimeZone.getTimeZone("Asia/Tokyo")));
27
28    // メッセージの作成
29    final Message message = new Message(event, fromAddress, toAddressList, attributes);
30
31    return message;
32  }
```

表：実装の詳細

行番号	説明
7	「 *イベントを定義する* 」で実装したクラスを元にイベントを作成します。
12	配信を行うユーザのユーザコードを取得し、ユーザコードを元に差出人のアドレスを作成します。 今回は、ユーザコードを `AccountContext` から取得しています。
15, 18-21	宛先のアドレスを作成します。 今回は、サンプルユーザコードを直接指定してアドレスを作成しています。
24-26	置換処理への対応を行います。 詳細は「 *置換処理の対応* 」を参照してください。
29	作成した各種情報を元にメッセージ（ `Message` ）を作成します。

置換処理の対応

テンプレートに定義されたプレースホルダを置換するためには、置換後の具体的な情報をメッセージの作成時に設定する必要があります。
今回のサンプルで、置換処理の対応を行っている箇所は以下の通りです。

```
1  // 属性の設定
2  final Map<String, Object> attributes = new HashMap<String, Object>();
3  attributes.put("url",      "http://www.intra-mart.jp/");
4  attributes.put("sendDate", DateTime.now(TimeZone.getTimeZone("Asia/Tokyo")));
5
6  // メッセージの作成
7  final Message message = new Message(event, fromAddress, toAddressList, attributes);
```

表：実装の詳細

行番号	説明
3-4	テンプレートに定義されたプレースホルダは、メッセージに指定された同名の属性に置換されます。 ここでは、メッセージの属性情報として利用される Map 形式の変数 attributes の `key` と `value` を以下のように設定しています。 ▪ `key` - プレースホルダ名 ▪ `value` - 置換後の具体的な情報

この処理によって、 IM-MessageHub は「 *完成したテンプレートファイルのサンプル* 」で作成したテンプレートファイルのプレースホルダを以下の通りに置換します。

- `${sendDate}` - メッセージの配信日付の情報に置換。
- `${url}` - メッセージに関連するURL情報に置換。

メッセージを配信する

次に、 IM-MessageHub へ作成したメッセージの配信を依頼する方法を説明します。
IM-MessageHub ではメッセージの配信を行うための `MessageService` クラスを提供しています。

- `jp.co.intra_mart.foundation.message_hub.service.MessageService`

メッセージを配信する処理の実装例は以下の通りです。
（この実装を含めた、サンプルクラスの全体像は「 *完成したメッセージ配信依頼クラスのサンプル* 」で提示しています。）

```
1  /**
2   * メッセージを配信します。
3   * @return メッセージを正常に送信できた場合はtrueを返します。
4   */
5  public static boolean deliver(){
6    try {
7      // MessageServiceをFactoryより取得する。
8      final MessageService messageService = MessageServiceFactory.getInstance().getMessageService();
9
10     // メッセージを作成し、配信をメッセージサービスに依頼します。
11     messageService.send(createMessage());
12
13   } catch (MessageServiceException e) {
14     return false;
15   }
16   return true;
17 }
```

表：実装の詳細

行番号	説明
8	`MessageService` を `MessageServiceFactory` より取得します。
11	「 *メッセージを作成する* 」で作成したメッセージを、 `MessageService#send` へ渡し、 IM-MessageHub へ配信依頼を行います。

完成したメッセージ配信依頼クラスのサンプル

今回は、以下のようなサンプルクラスを作成します。

```java
package sample.message_hub.send;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.TimeZone;

import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.i18n.datetime.DateTime;
import jp.co.intra_mart.foundation.message_hub.model.Address;
import jp.co.intra_mart.foundation.message_hub.model.Event;
import jp.co.intra_mart.foundation.message_hub.model.Message;
import jp.co.intra_mart.foundation.message_hub.model.UserAddress;
import jp.co.intra_mart.foundation.message_hub.service.MessageService;
import jp.co.intra_mart.foundation.message_hub.service.MessageServiceException;
import jp.co.intra_mart.foundation.message_hub.service.MessageServiceFactory;
import sample.message_hub.event.SampleMessageEvent;

/**
 * メッセージをIM-MessageHubを利用して作成から配信まで行うサンプルクラスです。
 */
public class SampleMessageSender {

    /**
     * メッセージを作成します。
     * @return メッセージ
     */
    public static Message createMessage(){
        // イベントの作成
        final Event event = new SampleMessageEvent();

        // 送信元アドレスの作成
        // 今回、送信元アドレスはアカウントコンテキストから取得しています。
        AccountContext context = Contexts.get(AccountContext.class);
        final Address fromAddress = new UserAddress(context.getUserCd());

        // 送信先アドレスの作成
        final List<Address> toAddressList = new ArrayList<Address>();

        // サンプルとしていくつかの宛先を追加する。
        toAddressList.add(new UserAddress("aoyagi")); // 青柳
        toAddressList.add(new UserAddress("ueda")); // 上田
        toAddressList.add(new UserAddress("hayashi")); // 林
        toAddressList.add(new UserAddress("maruyama")); // 円山

        // 属性の設定
        final Map<String, Object> attributes = new HashMap<String, Object>();
        attributes.put("url",      "http://www.intra-mart.jp/");
        attributes.put("sendDate", DateTime.now(TimeZone.getTimeZone("Asia/Tokyo")));

        // メッセージの作成
        final Message message = new Message(event, fromAddress, toAddressList, attributes);

        return message;
    }

    /**
     * メッセージを配信します。
     * @return メッセージを正常に送信できた場合はtrueを返します。
     */
    public static boolean deliver(){
        try {
            // MessageServiceをFactoryより取得する。
            final MessageService messageService = MessageServiceFactory.getInstance().getMessageService();

            // メッセージを作成し、配信をメッセージサービスに依頼します。
            messageService.send(createMessage());

        } catch (MessageServiceException e) {
            return false;
        }
        return true;
    }
}
```

サンプルの動作を確認する

メッセージ配信が正しく行われることを確認します。

サンプルアプリケーション（JSP）を作成する

これまで作成してきたファイルの動作を確認するために、サンプルアプリケーション（JSP）を作成します。
サンプルアプリケーションの配置例、および、作成例は以下の通りです。

サンプルアプリケーションパス：%CONTEXT_PATH%/sample/delivery.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui"%>
<%@ page import="sample.message_hub.send.SampleMessageSender" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<imui:head>
    <title>Sample Message Delivery Page</title>
</imui:head>

<div class="imui-title">
    <h1>SampleMessageDeliveryPage</h1>
</div>

<!-- 処理結果判定フラグ -->
<%! boolean isSuccess = false; %>

<!-- 配信処理 -->
<%
    // 作成したメッセージ配信処理を実行する。
    isSuccess = SampleMessageSender.deliver();
%>

<!-- 処理結果表示 -->
<div class="imui-form-container">
<% if(isSuccess) { %>
    <span class="im-ui-icon-common-32-confirmation"></span> 配信に成功しました。
<% } else { %>
    <span class="im-ui-icon-common-32-error"></span> 配信に失敗しました。
<% } %>
</div>
```

サンプルアプリケーション作成後、これまで作成した資材を intra-mart Accel Platform 上にデプロイします。

サンプルアプリケーションを実行し、メッセージ配信を確認する

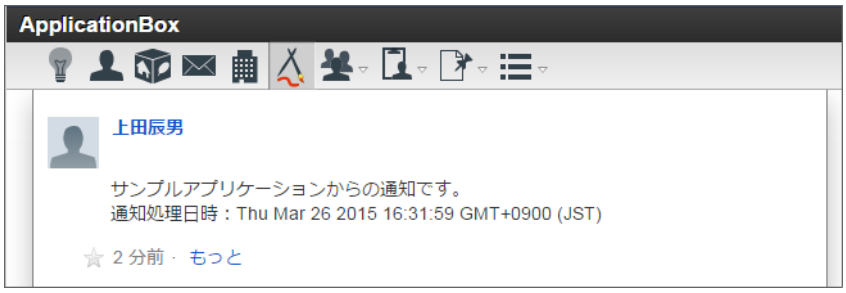
intra-mart Accel Platform を起動し、サンプルユーザ（ここでは ueda とします）でログイン後、以下のアドレスにアクセスします。

http://＜HOST＞:＜PORT＞/＜CONTEXT-PATH＞/sample/delivery.jsp

正常に処理が行われた場合、以下のような画面が表示されます。

図：サンプルアプリケーション実行例

サンプルアプリケーション実行後、「サイトマップ」-「IMBox」-「IMBox」 から IMBox を表示し、ApplicationBoxをクリックします。
正しく配信が行われている場合、以下のメッセージが表示されます。



図：サンプルユーザ ueda でログインし、サンプルを実行した結果

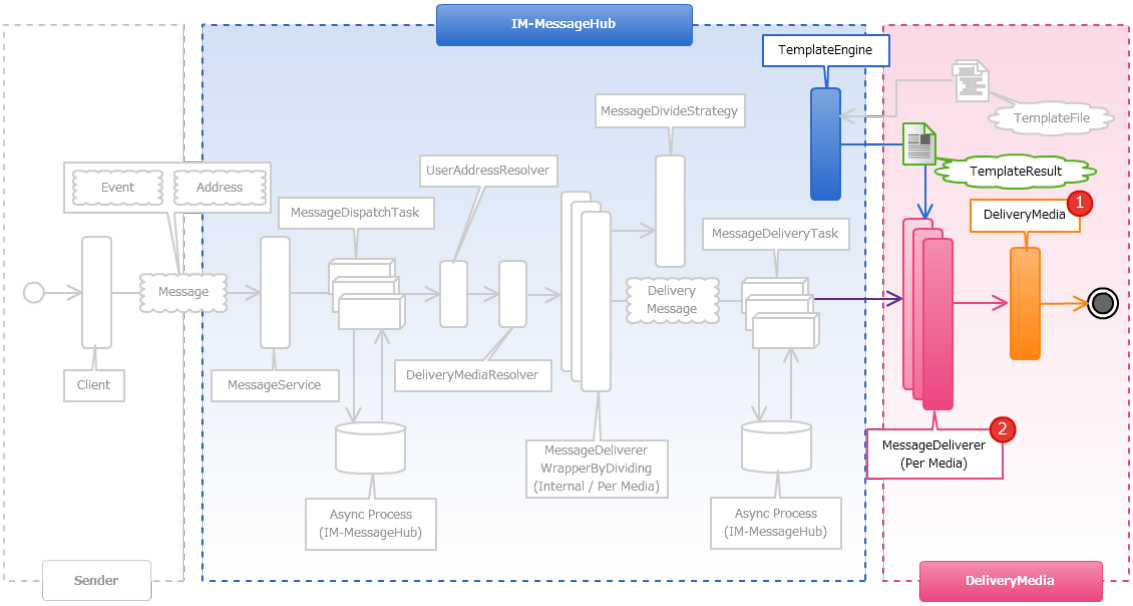
IM-MessageHub に配信先を追加する

全体の作業の流れ

ここでは IM-MessageHub を利用して配信されるメッセージを受け取る方法（配信先メディアを追加する方法）を説明します。
この章を通して以下のことを学びます。

- IM-MessageHub へ配信先メディアを追加する方法
  - 配信先メディアの定義
  - 配信クラスの作成

IM-MessageHub へ配信先メディアを追加するための流れは以下の通りです。



図：配信先を追加するための流れ

表：全体の作業リスト

手順	作成する資材	作業解説
1	配信先メディア、および、その定義（図中の DeliveryMedia ） ▪ 配信先メディアを定義するプラグインファイル	*配信先メディアを定義する*
2	配信先メディアへメッセージを配信するクラス（図中の MessageDeliverer(Per Media) ） ▪ 配信先メディアに対応する配信クラス	*配信クラスを作成する* *サンプルの動作を確認する*

本章では、 intra-mart Accel Platform から渡されるメッセージ情報を標準出力に配信（出力）するサンプルを作成します。

なお、流れに合わせてサンプルを実装し、実際に動作確認を行う際の前提条件は「 *サンプルコードについて* 」を参照してください。

配信先メディアを定義する

最初に、 IM-MessageHub が受け付けたメッセージを配信する新しい配信先メディアを定義します。
新しく配信先メディアを定義するためには、プラグインファイル（plugin.xml）を作成する必要があります。
作成するプラグインファイル（plugin.xml）の定義例は以下の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
   <extension point = "jp.co.intra_mart.message_hub.DeliveryMedias">
     <delivery-media
          id = "immh.sample.delivery"
          name-key = "CAP.Z.SAMPLE.APPLICATION.NAME"
          name = "Sample Delivery Media"
          deliverer-class ="sample.message_hub.delivery.SampleMessageDeliverer"
          version = "8.0.0"
          rank = "100" />
   </extension>
</plugin>
```

表：タグおよび属性の詳細

設定名称	説明
拡張ポイント	jp.co.intra_mart.message_hub.DeliveryMedias 固定です。
id	この配信先メディアを一意に表すユニークなIDを指定します。
name-key	表示名を定義したメッセージファイルのプロパティキー名を指定します。
name	表示名がメッセージファイルより解決できなかった場合の表示名を指定します。 英語で指定することを推奨します。
deliverer-class	後述する「*配信クラスを作成する*」にて作成した配信クラスの完全修飾クラス名（FQCN）を指定します。
version	この定義のバージョン情報を指定します。
rank	この定義の実行順序を指定します。 deliverer-class 属性で定義した配信クラスの実行順序に利用されます。 なお、 intra-mart Accel Platform が提供している配信先メディアに設定されている rank の値は「*配信先メディア一覧*」を参照してください。

配信先メディアに紐づく詳細な配信設定

IM-MessageHub では、配信先メディアに紐づく詳細な配信設定を行うことが可能です。

- 設定例

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
   <extension point = "jp.co.intra_mart.message_hub.DeliveryMedias">
      <delivery-media id = "...">
         <description>...</description>

         <display-setting
            visible = "true"
            order = "10"
            initial-check-state = "checked"
            allow-user-settings = "false"
         />

      </delivery-media>
   </extension>
</plugin>
```

表：属性の詳細

設定項目	説明
visible	配信先メディアに紐づく配信設定をユーザに対して表示するか否かを設定します。 設定可能な値は以下の通りです。 ▪ true - 表示する ▪ false - 表示しない 未設定の場合は、true （表示する）となります。 あわせて「*配信設定の表示可否(visible)の優先順位*」を参照してください。
order	配信先メディアの表示順を設定します。 正の整数値を指定してください。 未設定の場合は、9999 となります。 この値は、メッセージ通知設定画面での表示順として利用されます。 詳しくは「*メッセージ通知設定画面*」の「*画面項目*」を参照してください。 なお、 intra-mart Accel Platform が提供している配信先メディアに設定されている order の値は「*配信先メディア一覧*」を参照してください。
initial-check-state	配信先メディアに紐づく配信可否の初期値を設定します。 設定可能な値は以下の通りです。 ▪ checked - 配信を行う ▪ unchecked - 配信を行わない 未設定の場合は、メッセージ通知設定の初期値が有効となります。 詳しくは「*メッセージ通知設定画面における配信可否の初期値*」、および、「*配信可否の初期設定(initial-check-state)の優先順位*」を参照してください。

設定項目	説明
allow-user-settings	配信先メディアに紐づく配信設定の変更をユーザに対して許可するか否かを設定します。 設定可能な値は以下の通りです。 ▪ true - ユーザ変更可能 ▪ false - ユーザ変更不可能 未設定の場合は、true （ユーザ変更可能）となります。 あわせて「 *配信設定の表示可否(visible)の優先順位* 」を参照してください。

配信クラスを作成する

次に、 IM-MessageHub が配信するメッセージを受け取り、配信先メディアへ展開する配信クラスを作成します。
今回は、 IM-MessageHub から渡されるメッセージを標準出力に配信（出力）するサンプルを作成します。

配信クラスの実装

AbstractMessageDeliverer クラスを継承し、メッセージ配信処理を実装してください。

- jp.co.intra_mart.foundation.message_hub.delivery.AbstractMessageDeliverer

配信クラスの作成例は以下の通りです。

```java
package sample.message_hub.delivery;

import jp.co.intra_mart.foundation.message_hub.delivery.AbstractMessageDeliverer;
import jp.co.intra_mart.foundation.message_hub.delivery.MessageHubDeliverException;
import jp.co.intra_mart.foundation.message_hub.delivery.model.DeliveryMessage;
import jp.co.intra_mart.foundation.template.TemplateResult;

public class SampleMessageDeliverer extends AbstractMessageDeliverer<TemplateResult> {

    private final String HEADER_SUBJECT = "subject";

    private final String HEADER_URL = "url";

    @Override
    protected void deliver(TemplateResult templateResult, DeliveryMessage message) throws MessageHubDeliverException {
        // メッセージのヘッダー情報を取得
        String headerSubject = templateResult.getHeader(HEADER_SUBJECT);
        String headerUrl = templateResult.getHeader(HEADER_URL);

        // 配信エラーログの出力（ヘッダー情報が正しく取れなかった場合）
        if (headerSubject == null || headerUrl == null) {
            writeErrorLog(message, message.getTo());
            return; // 処理中断
        }

        // メッセージの本文を取得
        String body = templateResult.getBody();

        // 配信エラーログの出力（メッセージ本文が正しく取れなかった場合）
        if (body == null){
            writeErrorLog(message, message.getTo());
            return; // 処理中断
        }

        // 配信処理
        stdoutBodyMessage(body);

        // 配信成功ログの出力
        writeSuccessLog(message, message.getTo());

        /**
         * [参考]
         * 以下の処理は、引数の値がどの様な値として渡され、取得/利用可能かを表す補足となります。
         */

        // 引数の値の表示（Header）
        stdoutHeaderInfo(headerSubject, headerUrl);

        // 引数の値の表示(message)
        stdoutMessageDetail(message);
    }

    /**
     * 本文を標準出力へ配信（出力）します。
     *
     * @param body 本文情報
     */
    private void stdoutBodyMessage(String body) {
        StringBuilder logBuilder = new StringBuilder();

        logBuilder.append("\n===== [BODY] =====\n");
        logBuilder.append(body);
        logBuilder.append("===== [BODY-END] =====\n");

        System.out.println(logBuilder.toString());
    }

    /**
     * [参考]
     * ヘッダー情報を標準出力へ出力します。
     *
     * @param subject サブジェクト
     * @param url URL
     */
    private void stdoutHeaderInfo(String subject, String url) {
        StringBuilder logBuilder = new StringBuilder();

        logBuilder.append("\n===== [HEADER] =====\n");

        logBuilder.append("[SUBJECT] ").append(subject); // サブジェクト
        logBuilder.append("\n[URL] ").  append(url); // URL

        logBuilder.append("\n===== [HEADER-END] =====\n");

        System.out.println(logBuilder.toString());
    }
```

```
86      }
87
88      /**
89       * [参考]
90       * 配信予定のメッセージ詳細を標準出力へ出力します。
91       *
92       * @param message 配信用メッセージ
93       */
94      private void stdoutMessageDetail(DeliveryMessage message) {
95          StringBuilder logBuilder = new StringBuilder();
96
97          logBuilder.append("\n===== [MESSAGE-DETAIL] =====\n");
98          logBuilder.append("[EVENT] ").          append(message.getEvent().getId()); // イベントID
99          logBuilder.append("\n[FIRST_TO_NAME] ").append(message.getTo().get(0).getName()); // 最初の宛先ユーザ名
100         logBuilder.append("\n[FROM_NAME] ").    append(message.getFrom().getName()); // 送信者ユーザ名
101         logBuilder.append("\n===== [MESSAGE-DETAIL-END] =====\n");
102
103         System.out.println(logBuilder.toString());
104     }
105 }
```

表：実装の詳細

行番号	説明
15	AbstractMessageDeliverer#deliver をオーバーライドし、配信先メディアへの配信処理を実装します。 メソッドの引数として渡ってくる値の詳細は以下の通りです ▪ templateResult（TemplateResult） テンプレートの変換後の処理結果を反映したクラスです。 このクラスには、プレースホルダを置換済みのヘッダー情報や、本文情報が含まれています。 ▪ message（DeliveryMessage） 配信用のメッセージモデルクラスです。 このクラスには、ユーザレベルに解決済みの差出人情報や宛先情報、配信の契機となったイベント情報などが含まれています。 なお、ここで取得できる宛先情報は分割済みの宛先情報となります。 メッセージの分割についての詳細は「 *配信メッセージの分割* 」を参照してください。
17-18	テンプレートのヘッダー情報を、 TemplateResult クラスから取得します。
27	テンプレートの本文情報を、 TemplateResult クラスから取得します。
22, 31	メッセージ配信に必要な情報が正常に取得出来なかった場合、配信エラーログを出力し処理を中断します。 ログの種類については「 *ログの種類と出力方法* 」を参照してください。
36	テンプレートの本文を、標準出力に配信（出力）します。 この処理が、サンプルの実際の「配信処理」となります。
39	メッセージ配信が正常に行われた場合、配信成功ログを出力します。 ログの種類については「 *ログの種類と出力方法* 」を参照してください。
47	テンプレートのヘッダーに設定された情報を出力します。
50	配信メッセージから取得可能な情報のいくつかを出力します。

ログの種類と出力方法

AbstractMessageDeliverer には、メッセージ配信に関するログを出力するためのメソッドが用意されています。
AbstractMessageDeliverer に用意されているログ出力用のメソッドは以下の通りです。
（各メソッドの引数、および、その詳細は、APIドキュメント「 AbstractMessageDeliverer 」を参照してください）

- writeSuccessLog
  - メッセージの送信に成功したことを示す情報ログを出力します。
- writeSkipLog
  - メッセージの送信をスキップしたことを示す情報ログを出力します。
- writeWarningLog
  - メッセージの送信に失敗し、以降の処理をスキップしたことを示す警告ログを出力します。
- writeErrorLog
  - メッセージの送信が失敗したことを示すエラーログを出力します。

配信処理実装時には、利用用途に応じて適切なログ出力を行ってください。

サンプルの動作を確認する

追加した配信先メディアに、メッセージが正しく配信されることを確認します。

動作確認を行うために、「 *IM-MessageHub を利用してメッセージを配信する* 」で作成したサンプルアプリケーションを利用します。
本章で作成した資材、および、「*IM-MessageHub を利用してメッセージを配信する* 」を intra-mart Accel Platform 上にデプロイします。

intra-mart Accel Platform を起動し、サンプルユーザ（ここでは ueda とします）でログイン後、以下のアドレスにアクセスします。

http://＜HOST＞:＜PORT＞/＜CONTEXT-PATH＞/sample/delivery.jsp

以下がコンソールに出力されます。

```
===== [BODY] =====
サンプルアプリケーションからの通知です。
通知処理日時：Tue Mar 20 2015 23:59:59 GMT+0900 (JST)
===== [BODY-END] =====

===== [HEADER] =====
[SUBJECT] IM-MessageHub プログラミングガイド サンプル
[URL] http://www.intra-mart.jp/
===== [HEADER-END] =====

===== [MESSAGE-DETAIL] =====
[EVENT] sample.message_hub.event.SampleMessageEvent
[FIRST_TO_NAME] 青柳辰巳
[FROM_NAME] 上田辰男
===== [MESSAGE-DETAIL-END] =====
```

付録

配信先メディア

配信先メディア一覧

intra-mart Accel Platform が提供している配信先メディアは以下の通りです。

表：配信先メディア一覧

配信先メディア	配信先メディアID	ランク	オーダー	説明	提供バージョン
IMBox の ApplicationBox	immh.imbox.appbox	100	100	IMBoxのApplicationBoxに配信します。	2014 Winter(Iceberg)
IM-Notice の デスクトップ通知	immh.im_notice.desktop	150	150	IM-Notice デスクトップアプリケーション（デスクトップ版 IM-Notice ）に配信します。	2014 Winter(Iceberg)
IM-Notice の 通知履歴	immh.im_notice.history	200	200	配信内容を IM-Notice の通知履歴として保存します。IM-Notice （デスクトップアプリケーション、iOS版、Android版）で通知履歴を参照できます。	2014 Winter(Iceberg)
IM-Notice の モバイル通知	immh.im_notice.mobile	250	250	IM-Notice モバイルアプリケーション（iOS, Android版 IM-Notice ）に配信します。	2014 Winter(Iceberg)
メール	immh.im_mail.legacy	300	300	メールに配信します。 この配信先メディアでは、MailTemplateクラス で利用可能なメールテンプレートを利用します。 詳しくは「 *メールテンプレートの利用* 」を参照してください。	2014 Winter(Iceberg)
Slack のメッセージ通知	immh.im_slack.message	350	350	Slack に配信します。	2021 Spring(Bergamot)

必須ヘッダ情報

intra-mart Accel Platform が提供している配信先メディアの必須ヘッダ情報を示します。
特定の配信先メディア用にテンプレートを定義する場合は、下記のヘッダを必ず定義してください。

表：必須ヘッダ情報一覧

配信先メディア	必須ヘッダ	説明
IMBox の ApplicationBox	なし	
IM-Notice の デスクトップ通知	subject	メッセージの件名
	url	メッセージのリンク先URL
IM-Notice の 通知履歴	subject	メッセージの件名
	url	メッセージのリンク先URL
IM-Notice の モバイル通知	subject	メッセージの件名
	url	メッセージのリンク先URL
メール	なし	

テンプレートの置換処理の拡張

IM-MessageHub が利用するテンプレート機能は、置換処理を拡張することが可能です。
この章では、その拡張方法を説明します。

> **注意**
> この章では「 *IM-MessageHub を利用してメッセージを配信する* 」で作成したサンプルを利用します。

項目

置換処理の拡張とは

IM-MessageHub が利用するテンプレート機能では、置換処理時にプログラムによる独自の処理を追加することができます。
例えば、メッセージの属性に格納された「商品コード」を元に、「商品名」をマスタより取得し出力するといった拡張が可能です。

ここでは、メッセージの配信日時「 `${sendDate}` 」を、宛先ユーザのタイムゾーンに合わせて変換する方法について説明します。

置換処理を拡張するための実装クラスを作成する

置換処理を拡張するには、 `ReplaceEventHandler` インタフェースを実装します。

- jp.co.intra_mart.foundation.template.handler.ReplaceEventHandler

置換処理を拡張した実装クラスのサンプルは以下のとおりです。

```java
package sample.template.handler;

import java.util.TimeZone;

import jp.co.intra_mart.foundation.i18n.datetime.DateTime;
import jp.co.intra_mart.foundation.message_hub.delivery.model.AccountSource;
import jp.co.intra_mart.foundation.message_hub.delivery.model.DeliveryMessage;
import jp.co.intra_mart.foundation.message_hub.template.TemplateSearchCondition;
import jp.co.intra_mart.foundation.template.TemplateContext;
import jp.co.intra_mart.foundation.template.handler.ReplaceEventHandler;
import sample.message_hub.event.SampleMessageEvent;

public class SampleReplaceEventHandler implements ReplaceEventHandler {

    // このサンプル拡張で取り扱うプレースホルダのキー
    private static final String PLACEHOLDER_SEND_DATE= "sendDate";

    @Override
    public boolean isSupported(Object templateSearchCondition) {
        // IM-MessageHub が利用する場合のみ置換処理は有効
        if (!TemplateSearchCondition.class.isInstance(templateSearchCondition)) {
            return false;
        }

        TemplateSearchCondition condition = TemplateSearchCondition.class.cast(templateSearchCondition);

        // 今回サンプルで作成した SampleMessageEvent でのみこの置換処理は有効
        if (!SampleMessageEvent.class.isInstance(condition.getEvent())) {
            return false;
        }

        return true;
    }

    @Override
    public Object onReplace(TemplateContext context, String key, Object value, String... args) {
        final Object returnValue = value;

        // このサンプルで扱うプレースホルダの場合のみ置換処理を行う
        if (!PLACEHOLDER_SEND_DATE.equals(key)) {
            return returnValue;
        }

        if (!DateTime.class.isInstance(value)) {
            return returnValue;
        }

        // TemplateContext に格納されている配信メッセージを取得
        final DeliveryMessage message = context.get(DeliveryMessage.class.getName());

        // （先頭の）宛先ユーザのタイムゾーンを取得
        final AccountSource accountSource = message.getTo().get(0).getExtension(AccountSource.class);
        if (accountSource == null || accountSource.getTimeZone() == null) {
            return returnValue;
        }

        // メッセージにセットされた DateTime を、宛先ユーザのタイムゾーンに変換
        DateTime original = DateTime.class.cast(value);
        DateTime converted = original.withTimeZone(accountSource.getTimeZone());

        return converted;
    }

}
```

表：実装の詳細

行番号	説明
16	拡張する置換処理に紐づくプレースホルダのキーです。 今回のサンプルでは、テンプレートに ${sendDate} というプレースホルダがあった場合に置換処理を行います。
19	ReplaceEventHandler の isSupported メソッドを実装します。 メソッドの引数（templateSearchCondition）として渡ってくる値は、テンプレートを特定するための情報を表すオブジェクトです。 IM-MessageHub では TemplateSearchCondition クラスのインスタンスが渡されます。 TemplateSearchCondition クラスについては、APIドキュメント「 TemplateSearchCondition 」を参照してください。 今回のサンプルでは、以下の条件をすべて満たす場合にのみ置換処理を実行します。 ▪ IM-MessageHub によるテンプレートの利用である。 ▪ 今回のサンプルで作成したイベントによる配信である。
21	この置換処理を、 IM-MessageHub が利用する場合のみ有効とする判定処理です。 具体的には、引数が TemplateSearchCondition クラスのインスタンスであるかをチェックしています。
28	今回のサンプルで作成したイベントによる配信であるかをチェックしています。
36	ReplaceEventHandler の onReplace メソッドを実装します。 このメソッドは、テンプレートファイル内のプレースホルダ置換処理が実行される際に呼び出されます。 詳しくは、APIドキュメント「 ReplaceEventHandler 」を参照してください。
40	変換対象のプレースホルダのキー名が、サンプルで扱うキー名と同じであるかを判定しています。
49	TemplateContext から配信メッセージを取得しています。 ▪ jp.co.intra_mart.foundation.template.TemplateContext 配信メッセージ(DeliveryMessage ）に設定されている属性は、テンプレートコンテキスト（ TemplateContext ）にKey-Valueのペアでコピーされています。 また、配信メッセージ自体も、DeliveryMessage のクラス名をキーとしてテンプレートコンテキストに格納されています。 テンプレートコンテキストの詳細はAPIドキュメント「 TemplateContext 」を参照してください。 今回のサンプルでは、後者の方法を用いて、宛先ユーザのタイムゾーンを取得しています。
52-55, 58-59	メッセージの配信日時を、宛先ユーザのタイムゾーンの日時に変換しています。 なお、IM-MessageHub では、宛先ユーザの「ロケール」と「タイムゾーン」が同一の宛先ごとに配信メッセージが分割されています。この仕様により、今回のサンプルで利用するタイムゾーンは、先頭の宛先ユーザに設定されているタイムゾーンを利用しています。

> **注意**
>
> テンプレートコンテキストから取得できる宛先情報を利用する場合の注意点
>
> テンプレートコンテキストから取得した配信メッセージに設定されている宛先は、 IM-MessageHub によって配信メッセージの分割処理が行われた後のものになります。
> そのため、テンプレートコンテキストから取得できる宛先情報は、メッセージ配信アプリケーションが指定した宛先の一部となる場合があります。
> （メッセージの分割についての詳細は「 *配信メッセージの分割* 」を参照してください）

実装したクラスを有効にするためにプロバイダ構成ファイルを作成し、実装クラスの FQCN（完全修飾クラス名）を指定します。

- META-INF/services/jp.co.intra_mart.foundation.template.handler.ReplaceEventHandler

```
sample.template.handler.SampleReplaceEventHandler
```

置換処理を拡張した結果を確認する

作成した資材を intra-mart Accel Platform 上にデプロイし、拡張した置換処理が正常に動作しているかを確認します。
確認する手順は以下の通りです。

1. サンプルユーザ（ここでは ueda ）のタイムゾーンを「GMT-04:00」に変更します。
   - 今回のサンプルでは、通知処理日時を「 Asia/Tokyo(GMT+09:00) 」で設定しているため、それ以外のタイムゾーンに変更してください。
2. タイムゾーンを変更した上で、配信処理を実行します。
3. IMBox の ApplicationBox を表示し、配信された内容を確認します。

正しく配信が行われている場合、以下のメッセージが IMBox の ApplicationBox に表示されます。

図：拡張した置換処理が適用されたメッセージ結果

イベント設定

IM-MessageHub では、イベントに関する詳細情報 や イベントをカテゴライズするための情報を定義することが可能です。

- **アプリケーション** - イベントを発生させているアプリケーションに関する情報
- **イベントカテゴリ** - イベントをカテゴライズするための情報
- **イベント** - メッセージ配信処理の起因となるイベントに関する情報

IM-MessageHub では、これらの情報を利用して「 *メッセージ通知設定画面* 」におけるチェック状態の初期設定などを制御しています。

この章では、アプリケーション、イベントカテゴリ、および、イベントの設定に関する詳細を説明します。

項目

アプリケーションの設定

イベントを発生させているアプリケーションに関する情報です。
プラグイン形式で設定を行います。

- 設定例

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension point = "jp.co.intra_mart.message_hub.Applications">
    <application
        id       = "sampleapplication"
        name-key = "CAP.Z.SAMPLE.APPLICATION.NAME"
        name     = "SampleApplication"
        version  = "8.0.0"
        rank     = "900" >
      <description message-cd="CAP.Z.SAMPLE.APPLICATION.DESCRIPTION">The Sample Application.</description>
    </application>
  </extension>
</plugin>
```

表：タグおよび属性の詳細

設定項目	説明
拡張ポイント	jp.co.intra_mart.message_hub.Applications 固定です。
id	このアプリケーションを一意に表すユニークなIDを指定します。
name-key	アプリケーション名を定義したメッセージファイルのプロパティキー名を指定します。
name	アプリケーション名がメッセージファイルより解決できなかった場合の代替アプリケーション名を指定します。 英語で指定することを推奨します。
version	この定義のバージョン情報を指定します。
rank	この定義の順序を指定します。 「メッセージ通知設定」画面の表示順に利用されます。
message-cd	アプリケーションに関する説明を定義したメッセージファイルのプロパティキー名を指定します。
description	アプリケーションに関する説明がメッセージファイルより解決できなかった場合の代替文を指定します。 英語で指定することを推奨します。

注意

アプリケーションIDに指定可能な文字は「英数字」、および、「_（アンダースコア）」のみです。
ただし、先頭は「英字」としてください。

コラム

アプリケーション名ついて

- アプリケーション名は、サイトマップ、または、グローバルナビに表示している名称と同じ名称とすることを推奨します。
- アプリケーション名は、「メッセージ通知設定」画面のタブ名として利用されます。
  タブ名が長すぎるとアプリケーションが多くなった場合に複数行表示されてしまうため、短めの名称を推奨します。

コラム

アプリケーションの設定では、visible や initial-check-state などの詳細な配信設定を行うことはできません。

コラム

アプリケーションに設定するランクについて

アプリケーションに設定するランクは、サイトマップと同じ順序となることを推奨します。
具体的には、サイトマップに関するソート番号と同じ値に設定することを推奨します。

イベントカテゴリの設定

イベントをカテゴライズするための情報です。プラグイン形式で設定を行います。
なお、イベントカテゴリは作成しなくても構いません。

- 設定例

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension point = "jp.co.intra_mart.message_hub.EventCategories">
    <event-category
      id       = "sample.event.category"
      name-key  = "CAP.Z.SAMPLE.EVENT.CATEGORY.NAME"
      name      = "Sample Event Category"
      parent-id = "sampleapplication"
      version   = "8.0.0"
      rank      = "800" >
      <description message-cd="CAP.Z.SAMPLE.EVENT.CATEGORY.DESCRIPTION">The Sample Event Category.</description>
    </event-category>
  </extension>
</plugin>
```

表：タグおよび属性の詳細

設定項目	説明
拡張ポイント	jp.co.intra_mart.message_hub.EventCategories 固定です。
id	このイベントカテゴリを一意に表すユニークなIDを指定します。
name-key	イベントカテゴリ名を定義したメッセージファイルのプロパティキー名を指定します。
name	イベントカテゴリ名がメッセージファイルより解決できなかった場合の代替イベントカテゴリ名を指定します。 英語で指定することを推奨します。
parent-id	このイベントカテゴリが所属する *アプリケーション* のID、 または、 *イベントカテゴリ* のIDを指定します。
version	この定義のバージョン情報を指定します。
rank	この定義の順序を指定します。 「メッセージ通知設定」画面の表示順に利用されます。
message-cd	イベントカテゴリに関する説明を定義したメッセージファイルのプロパティキー名を指定します。
description	イベントカテゴリに関する説明がメッセージファイルより解決できなかった場合の代替文を指定します。 英語で指定することを推奨します。

コラム

イベントカテゴリ名・説明について

- イベントカテゴリはオプションですが、アプリケーションのメニューが機能ごとフォルダわけされている場合、同じようにイベントカテゴリを作成することを推奨します。
- イベントカテゴリの説明は、「メッセージ通知設定」画面上でツールチップとして表示されます。
  「メッセージ通知設定」画面上のツールチップ表示サイズは固定となっているため、スクロールが発生しない適切な文字数としてください。

コラム

イベントカテゴリの設定では、visible や initial-check-state などの詳細な配信設定を行うことはできません。

イベントの設定

メッセージ配信処理の起因となるイベントに関する情報です。
プラグイン形式で設定を行います。

- 設定例

```xml
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension point = "jp.co.intra_mart.message_hub.Events">
    <event
      id       = "sample.message_hub.event.SampleEvent"
      name-key  = "CAP.Z.SAMPLE.EVENT.NAME"
      name      = "Sample Event"
      parent-id = "sampleapplication"
      version   = "8.0.0"
      rank      = "700">
      <description message-cd="CAP.Z.SAMPLE.EVENT.DESCRIPTION">Sample Event active.</description>
    </event>
  </extension>
</plugin>
```

表：タグおよび属性の詳細

設定項目	説明
拡張ポイント	jp.co.intra_mart.message_hub.Events 固定です
id	このイベントを一意に表すユニークなIDを指定します。
name-key	イベント名を定義したメッセージファイルのプロパティキー名を指定します。
name	イベント名がメッセージファイルより解決できなかった場合の代替イベント名を指定します。 英語で指定することを推奨します。
parent-id	このイベントが所属するアプリケーションのアプリケーションID、または、イベントカテゴリのイベントカテゴリIDを指定します。
version	この定義のバージョン情報を指定します。
rank	この定義の順序を指定します。 「メッセージ通知設定」画面の表示順に利用されます。
message-cd	イベントに関する説明を定義したメッセージファイルのプロパティキー名を指定します。
description	イベントに関する説明がメッセージファイルより解決できなかった場合の代替文を指定します。 英語で指定することを推奨します。

コラム

イベント名・説明について

イベントの発生タイミングをわかりやすく記述してください。記述例は以下の通りです。

表：イベント名・説明の記述例

	記述書式	備考
イベント名	～したとき 例：「スケジュールが登録されたとき」	▪ 「メッセージ通知設定」画面上で表示可能な文字数としてください。
イベント説明	～が～されたときに通知します。 例：「あなたが参加者に含まれるスケジュールが登録されたときに通知します。」	▪ 通知に関して条件がある場合は、合わせて記述します。 ▪ ただし、条件や内容は、配信処理の実装に依存して変更される可能性があるため、影響がないような記述にしてください。 ▪ イベントの説明は、「メッセージ通知設定」画面上でツールチップとして表示されます。 「メッセージ通知設定」画面上のツールチップ表示サイズが固定となっているため、できるだけ スクロールが発生しない文字数としてください。

イベントに紐づく詳細な配信設定

イベントの設定では、イベントに紐づく詳細な配信設定を行うことが可能です。

- 設定例

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension point = "jp.co.intra_mart.message_hub.Events">
    <event>
      <description>...</description>

      <display-setting
        visible           = "true"
        initial-check-state = "unchecked"
        allow-user-settings = "false" >
      </display-setting>
    </event>
  </extension>
</plugin>
```

表：属性の詳細

設定項目	説明
visible	イベントに紐づく配信設定をユーザに対して表示するか否かを設定します。 設定可能な値は以下の通りです。 ▪ true - 表示する ▪ false - 表示しない 未設定の場合は「*配信先メディア定義*」で設定された値が有効となります。 詳しくは「*配信先メディアを定義する*」、および、「*配信設定の表示可否(visible)の優先順位*」を参照してください。
initial-check-state	イベントに紐づく配信可否の初期値を設定します。 この設定値は、ユーザが「メッセージ通知設定」画面でイベントに紐づく配信設定を行っていない場合に利用されます。 設定可能な値は以下の通りです。 ▪ checked - 配信する ▪ unchecked - 配信しない 未設定の場合は「*配信先メディア定義*」で設定された値が有効となります。 詳しくは「*配信先メディアを定義する*」、および、「*配信可否の初期設定(initial-check-state)の優先順位*」を参照してください。
allow-user-settings	イベントに紐づく配信設定の変更をユーザに対して許可するか否かを設定します。 設定可能な値は以下の通りです。 ▪ true - ユーザ変更可能 ▪ false - ユーザ変更不可能 未設定の場合は「*配信先メディア定義*」で設定された値が有効となります。 詳しくは「*配信先メディアを定義する*」、および、「*配信設定の変更可否(allow-user-settings)の優先順位*」を参照してください。

イベントと配信先メディアに紐づく詳細な配信設定

イベントに紐づく詳細な配信設定は、**配信先メディアを指定して** 設定することも可能です。

- 設定例

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension point = "jp.co.intra_mart.message_hub.Events">
    <event>
      <description>...</description>

      <display-setting>
        <delivery-media
          id              ="immh.imbox.appbox"
          initial-check-state ="unchecked"
          allow-user-settings ="false" />
      </display-setting>
    </event>
  </extension>
</plugin>
```

表：属性の詳細

設定項目	説明
id	設定を行いたい配信先メディアの配信先メディアIDを指定します。 intra-mart Accel Platform が提供している配信先メディアについては「*配信先メディア一覧*」を参照してください。

設定項目	説明
initial-check-state	イベントに紐づく配信可否の初期値を、配信先メディア単位で設定します。 この設定値は、ユーザが「メッセージ通知設定」画面でイベントに紐づく配信設定を行っていない場合に利用されます。 設定可能な値は以下の通りです。 ▪ checked - 配信する ▪ unchecked - 配信しない 未設定の場合は「 *イベントに紐づく詳細な配信設定* 」で定義された設定が有効となります。 詳しくは「 *配信可否の初期設定(initial-check-state)の優先順位* 」を参照してください。
allow-user-settings	イベントに紐づく配信設定の変更をユーザに対して許可するか否かに関して、配信先メディア単位で設定します。 設定可能な値は以下の通りです。 ▪ true - ユーザ変更可能 ▪ false - ユーザ変更不可能 未設定の場合は「 *イベントに紐づく詳細な配信設定* 」で定義された設定が有効となります。 詳しくは「 *配信設定の変更可否(allow-user-settings)の優先順位* 」を参照してください。

コラム
「 *イベントと配信先メディアに紐づく詳細な配信設定* 」に 配信設定の表示可否設定(visible) はありません。

詳細な配信設定の優先順位

詳細な配信設定の優先順位は以下の通りです。

配信可否の初期設定(initial-check-state)の優先順位

1. 「 *イベントと配信先メディアに紐づく詳細な配信設定* 」 の initial-check-state の 値
2. 「 *イベントに紐づく詳細な配信設定* 」 の initial-check-state の 値
3. 「 *配信先メディア定義* 」で設定された initial-check-state の 値
4. 「 *メッセージ通知設定画面における配信可否の初期値* 」

全て未設定の場合、配信可否の初期設定は **配信する** となります。

配信設定の表示可否(visible)の優先順位

1. 「 *イベントに紐づく詳細な配信設定* 」 の visible の 値
2. 「 *配信先メディア定義* 」で設定された visible の 値

全て未設定の場合、配信設定は **表示** されます。

配信設定の変更可否(allow-user-settings)の優先順位

1. 「 *イベントと配信先メディアに紐づく詳細な配信設定* 」 の allow-user-settings の 値
2. 「 *イベントに紐づく詳細な配信設定* 」 の allow-user-settings の 値
3. 「 *配信先メディア定義* 」で設定された allow-user-settings の 値

全て未設定の場合、配信設定は **変更可能** となります。

メッセージ通知設定画面

IM-MessageHub では メッセージ通知設定画面 が提供されています。 この画面を利用することで、ユーザはアプリケーションから配信される IM-MessageHub 経由のメッセージに対する受信可否を柔軟に設定することが可能となります。

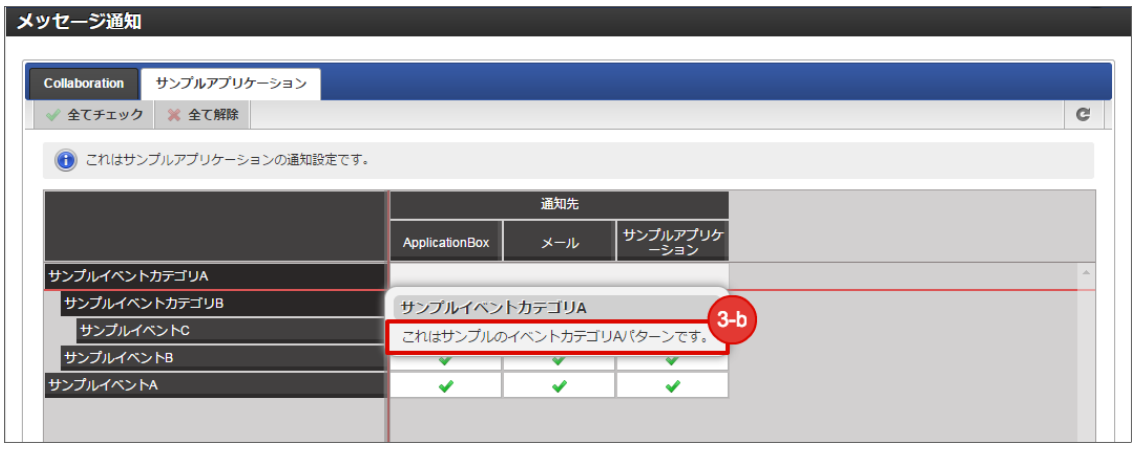
コラム
メッセージ通知設定の詳細な操作方法については「 一般ユーザ操作ガイド 」を参照してください。

画面項目

メッセージ通知設定画面の「画面項目」と「設定項目」の関係は以下の通りです。

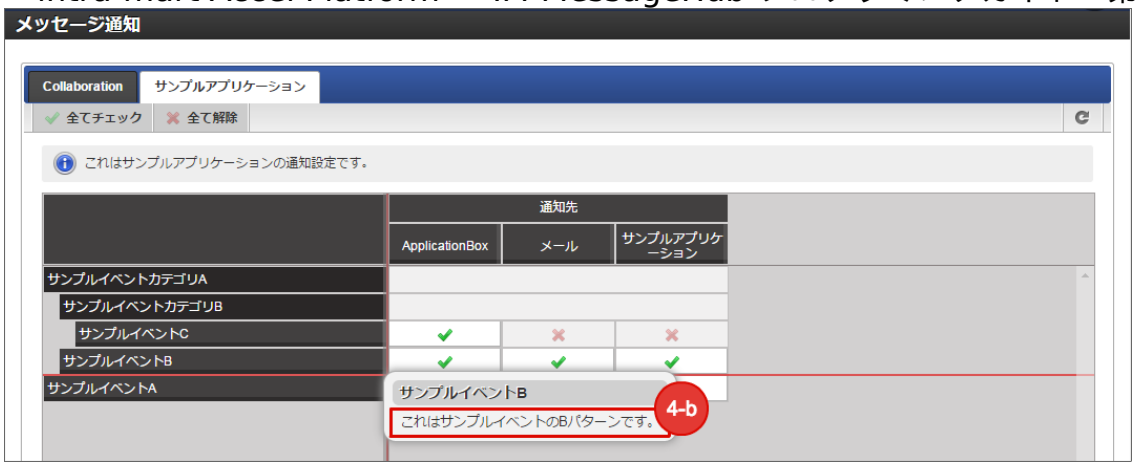
図：メッセージ通知設定画面 - 詳細

項番	画面項目	設定項目
1	アプリケーション名	「 *アプリケーションの設定* 」 の name-key, name 表示順は、プラグイン情報の取得順です。
2	アプリケーションの説明	「 *アプリケーションの設定* 」 の message-cd, description
3-a	イベントカテゴリ名	「 *イベントカテゴリの設定* 」 の name-key, name 表示順は、プラグイン情報の取得順です。
4-a	イベント名	「 *イベントの設定* 」 の name-key, name 表示順は、プラグイン情報の取得順です。
5	配信先メディア名	「 *配信先メディアを定義する* 」 の name-key, name 表示順は、「 *配信先メディアに紐づく詳細な配信設定* 」 の order が関係します。 具体的には以下の順序で表示されます。 1. order が設定されて **いる** 配信先メディアを、order の昇順で表示。 2. order が設定されて **いない** 配信先メディアを、プラグイン情報の取得順で表示。
6	配信設定	配信可否の初期設定(initial-check-state) 、配信設定の表示可否(visible)、および、配信設定の変更可否(allow-user-settings)に従って表示されます。 詳しくは「 *詳細な配信設定の優先順位* 」を参照してください。



図：メッセージ通知設定画面 - イベントカテゴリの説明

項番	画面項目	設定項目
3-b	イベントカテゴリの説明 （イベントカテゴリ名にマウスカーソルを合わせた際に表示されます）	「 *イベントカテゴリの設定* 」 の message-cd, description



図：メッセージ通知設定画面 - イベントの説明

項番	画面項目	設定項目
4-b	イベントの説明 （イベント名にマウスカーソルを合わせた際に表示されます）	「 *イベントの設定* 」 の message-cd, description

特定のアプリケーションのメッセージ通知設定画面について

「メッセージ通知設定」画面は、特定のアプリケーションのみを対象とした設定画面を表示することができます。
これにより、ポップアップ表示 や IFrame を利用して、アプリケーション専用のメッセージ通知設定画面を提供することが可能です。
アクセスするためのURLは以下の通りです。

- http://<HOST>:<PORT>/<CONTEXT_PATH>/user/settings/notification/application/`${applicationId}`

  `${applicationId}` には、表示対象の「 *アプリケーションID* 」を指定します。

テーマ設定

ポップアップ や IFrame で表示する場合 ヘッダー部分 は不要です。 そのために以下の設定を行ってください。

- 設定例

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

    <!-- 表示対象のアプリケーションのアプリケーションIDを"sample"としています。 -->
    <path>/user/settings/notification/application/sample</path>

</theme-head-with-container-path-config>
```

詳しくは「 設定ファイルリファレンス 」-「 テーマの適用方法設定 HeadWithContainerThemeBuilder 」を参照してください。

配信先メディア解決

IM-MessageHub では、発生したイベントに紐づくメッセージの配信可否を配信先メディア毎に解決します。
この解決は、配信先メディア解決インタフェース (DeliveryMediaResolver) を実装したクラスで行います。

- jp.co.intra_mart.foundation.message_hub.resolver.media.DeliveryMediaResolver

DeliveryMediaResolver は、プラグイン形式で提供されており、プラグイン情報の取得順に実行されます。
なお、配信先メディア解決の概要は「 *配信先メディア解決* 」を参照してください。

配信先メディア解決の追加方法

IM-MessageHub は配信先メディア解決を開発者が追加可能な機構となっています。
ここでは、配信先メディア解決を作成する方法、および、作成した配信先メディア解決を IM-MessageHub で有効にする方法を説明します。

> **注意**
>
> この章では「 *IM-MessageHub を利用してメッセージを配信する* 」で作成したサンプルを利用します。

配信先メディア解決の作成

DeliveryMediaResolver インタフェースを実装し、配信先メディア解決を行う処理を作成します。

ここでは、「 *IM-MessageHub を利用してメッセージを配信する* 」で作成した SampleMessageEvent イベントに関する配信の場合、全ての配信先メディアへの配信を行わないようにするサンプルを作成します。

```
1  package sample.message_hub.media.resolver;
2
3  import jp.co.intra_mart.foundation.message_hub.delivery.model.DeliveryAddress;
4  import jp.co.intra_mart.foundation.message_hub.delivery.model.DeliveryMedia;
5  import jp.co.intra_mart.foundation.message_hub.model.Event;
6  import jp.co.intra_mart.foundation.message_hub.model.Message;
7  import jp.co.intra_mart.foundation.message_hub.resolver.media.DeliveryMediaResolveException;
8  import jp.co.intra_mart.foundation.message_hub.resolver.media.DeliveryMediaResolveResult;
9  import jp.co.intra_mart.foundation.message_hub.resolver.media.DeliveryMediaResolveResultMap;
10 import jp.co.intra_mart.foundation.message_hub.resolver.media.DeliveryMediaResolver;
11 import sample.message_hub.event.SampleMessageEvent;
12
13 public class SampleMediaResolver implements DeliveryMediaResolver<DeliveryAddress> {
14
15  // 配信を行わないイベント
16  private static final Event IGNORE_EVENT = new SampleMessageEvent();
17
18  @Override
19  public void resolve(DeliveryMediaResolveResultMap resultMap, Event event, DeliveryAddress address, Message message) throws
20 DeliveryMediaResolveException {
21   // 特定のイベントの場合、配信を行わない
22   if (IGNORE_EVENT.getId().equals(event.getId())) {
23    // どの配信先メディアにも配信を行わない。
24    for (final DeliveryMedia media : resultMap.getDeliveryMediaSet()) {
25     resultMap.updateResult(media, DeliveryMediaResolveResult.DISABLE);
26    }
27   }
28  }
29
30  @Override
31  public Class<DeliveryAddress> getDeliveryAddressClass() {
32   return DeliveryAddress.class;
33  }
 }
```

表：実装の詳細

行番号	説明
19	DeliveryMediaResolver#resolve メソッドの実装です。 このメソッドの引数として渡ってくる値の詳細は以下の通りです。 ▪ resultMap（DeliveryMediaResolveResultMap） 配信先メディアと配信解決の判定結果を管理するモデルクラスです。 このモデルクラスへ、キーとして判定を行った DeliveryMedia クラス、値として判定結果（ DeliveryMediaResolverResult ）をセットします。 判定結果として設定可能な値は以下の通りです。 ▪ DeliveryMediaResolveResult.ENABLE - 配信する。 ▪ DeliveryMediaResolveResult.DISABLE - 配信しない。 ▪ DeliveryMediaResolveResult.UNKNOWN - この処理段階では解決を保留する。 ▪ event（Event） 配信メッセージに設定される配信元のイベントです。 ▪ address（DeliveryAddress） 配信メッセージに設定される宛先のアドレスです。 ▪ message（Message） 配信される予定のメッセージです。
21	今回のサンプルで作成したイベントによる配信であるかをチェックしています。
23-24	引数の resultMap に格納されている、配信先メディアのセットを取り出し、各配信先メディアに対して「配信しない（ DeliveryMediaResolveResult.DISABLE ）」という判定結果を設定しています。

作成した配信先メディア解決の有効化

次に作成した配信先メディア解決を有効にします。

IM-MessageHub は配信先メディア解決をプラグインによって管理しています。
作成した配信先メディア解決を有効にするためのプラグインファイル（plugin.xml）の記述例は以下の通りです。
この例を反映させた環境で再度サンプルのJSPを実行すると、 IMBox の ApplicationBox への配信が行われないようになります。

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <plugin>
3    <extension point = "jp.co.intra_mart.message_hub.DeliveryMediaResolver">
4      <delivery-media-resolver-config
5        name    = "Sample Media Resolver"
6        id      = "sample.message_hub.media.resolver"
7        version = "8.0.0"
8        rank    = "50">
9        <resolver>
10         <resolver-class>sample.message_hub.media.resolver.SampleMediaResolver</resolver-class>
11       </resolver>
12     </delivery-media-resolver-config>
13   </extension>
14 </plugin>
```

表：タグおよび属性の詳細

設定項目	説明
拡張ポイント	jp.co.intra_mart.message_hub.DeliveryMediaResolver 固定です。
id	この配信先メディア解決を一意に表すユニークな ID を指定します。
rank	この定義の実行順序を指定します。 resolver-class に定義した配信クラスの実行順序に利用されます。 なお、 intra-mart Accel Platform が提供している配信先メディア解決に設定されている rank の値は「*DeliveryMediaResolver 一覧*」を参照してください。
resolver-class	作成した DeliveryMediaResolver の完全修飾クラス名（FQCN）を指定します。

注意

配信先メディア解決のランクについて

新しく作成した配信先メディア解決に設定するランクは「 **10 - 299** 」の範囲で設定してください。
また、複数の配信先メディア解決を作成する場合は、以下の優先順位で設定することを推奨します。

1. アプリケーションレベルで配信を停止したい場合。
2. アプリケーションレベルで強制的に配信を行いたい場合。
3. アプリケーションが独自に提供する通知設定機能で配信可否を決定したい場合。

コラム

予約済みの配信先メディア解決のランク一覧

intra-mart Accel Platform では、以下の配信先メディア解決のランクが予約されています。
新しく作成する配信先メディア解決のランクには利用しないでください。

予約済みランク一覧

ランク	用途
～9	メッセージ配信の停止、および、配信先メディアを限定する場合。
300～499	IM-MessageHub が提供する機能によって決定する優先順位を元に、メッセージの配信可否を判定する場合。
500～999	メッセージの配信可否が決定している場合でも、強制的に配信可否を更新する場合。
9999	メッセージの配信可否が決定しなかった際の初期状態を決定する場合。

DeliveryMediaResolver 一覧

intra-mart Accel Platform が提供している DeliveryMediaResolver は以下の通りです。

表：DeliveryMediaResolver 一覧

クラス名（単純名）	ランク	説明	提供バージョン
CheckEventRestrictionDeliveryMediaResolver	5	宛先ユーザがアプリケーションライセンスを持たない場合、メッセージ配信を行わないようにするためのクラスです。 （イベントの親となるアプリケーション／イベントカテゴリにアプリケーションライセンスの設定がある場合のみ）	2015 Spring(Juno)
IMBoxSendToMeDeliveryMediaResolver	10	IMBox の「私にメール」に関するメッセージの配信先メディアを「メール」に限定するクラスです。	2014 Winter(Iceberg)
IMBoxStandardDeliveryMediaResolver	20	IMBox に関するメッセージの配信先メディアを解決するクラスです。 2015 Winter(Lydia) よりIMBoxの通知設定は「 *メッセージ通知設定画面* 」に統合されたため使用されていません。	2014 Winter(Iceberg) ～ 2015 Summer(Karen)
ImWorkflowDeliveryMediaResolver	50	IM-Workflow に関するメッセージの配信先メディアを解決するクラスです。	2014 Winter(Iceberg)
UserSettingDeliveryMediaResolver	400	「 *メッセージ通知設定画面* 」の設定を元にメッセージの配信先メディアを解決するクラスです。 「配信する」・「配信しない」の判定は以下の優先順位で行われます。 1. 「 *メッセージ通知設定画面* 」 の 設定値 2. 「 *イベントと配信先メディアに紐づく詳細な配信設定* 」 の initial-check-state の 値 3. 「 *イベントに紐づく詳細な配信設定* 」 の initial-check-state の 値 4. 「 *配信先メディア定義* 」で設定された initial-check-state の 値	2015 Spring(Juno)
LegacyMailTemplate PreconditionCheckDeliveryMediaResolver	500	配信先メディアが「メール (immh.im_mail.legacy)」の場合、メール配信の前提条件が整っているかをチェックするクラスです。	2014 Winter(Iceberg)
BlackListDeliveryMediaResolver	9999	メッセージの配信可否が決定しなかった配信先メディアに対して、「配信する」という結果を設定するクラスです。 メッセージの配信可否が決定しなかった配信先メディアに対する判定結果を変更する方法は、「 *配信可否の初期状態の変更* 」を参照してください。	2014 Winter(Iceberg)

配信可否の初期状態の変更

IM-MessageHub では、ユーザによる配信設定が未設定などの理由で、配信先メディアに対するメッセージの配信可否が決定しなかった場合のために、2つの DeliveryMediaResolver を提供しています。
（パッケージ名は、いずれも jp.co.intra_mart.foundation.message_hub.resolver.media です。）

表：提供するDeliveryMediaResolver

クラス名	説明
BlackListDeliveryMediaResolver	メッセージの配信可否が決定しなかった配信先メディアに対して「**配信する**」という結果を設定するクラスです。（初期状態ではこのクラスが有効となっています）
WhiteListDeliveryMediaResolver	メッセージの配信可否が決定しなかった配信先メディアに対して「**配信しない**」という結果を設定するクラスです。

ここでは、初期状態で有効となっている BlackListDeliveryMediaResolver を、WhiteListDeliveryMediaResolver に変更する方法を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension point = "jp.co.intra_mart.message_hub.DeliveryMediaResolver">
    <delivery-media-resolver-config
        name    = "Overwrite Default Delivery Media Resolver"
        id      = "jp.co.intra_mart.message_hub.StandardDeliveryMediaResolver"
        version = "8.0.1"
        rank    = "9999">
      <resolver>
        <resolver-class>jp.co.intra_mart.foundation.message_hub.resolver.media.WhiteListDeliveryMediaResolver</resolver-class>
      </resolver>
    </delivery-media-resolver-config>
  </extension>
</plugin>
```

表：タグおよび属性の詳細

設定項目	説明
拡張ポイント	jp.co.intra_mart.message_hub.DeliveryMediaResolver 固定です。
id	jp.co.intra_mart.message_hub.StandardDeliveryMediaResolver 固定です。
resolver-class	jp.co.intra_mart.foundation.message_hub.resolver.media.WhiteListDeliveryMediaResolver 固定です。

注意

- version, rank は、既存のプラグインよりも優先して読み込まれるように設定してください。
- BlackListDeliveryMediaResolver を、WhiteListDeliveryMediaResolver に変更した場合は、 必ず「 *メッセージ通知設定画面における配信可否の初期値* 」の変更も行ってください。

メッセージ通知設定画面における配信可否の初期値

「 *配信可否の初期状態の変更* 」を行った場合、メッセージ通知設定画面における配信可否の初期値を、変更した DeliveryMediaResolver にあわせて変更する必要があります。

メッセージ通知設定画面における配信可否の初期値は、プラグイン形式で管理されています。

ここでは、初期状態で有効となっている「**配信する**」を、WhiteListDeliveryMediaResolver にあわせて「**配信しない**」に変更する方法を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension point = "jp.co.intra_mart.message_hub.NotificationConfig">
    <notification-config
        id = "jp.co.intra_mart.message_hub.NotificationConfig.Default"
        default-check-state = "unchecked" />
  </extension>
</plugin>
```

表：タグおよび属性の詳細

設定項目	説明
拡張ポイント	jp.co.intra_mart.message_hub.NotificationConfig 固定です。
id	jp.co.intra_mart.message_hub.NotificationConfig.Default 固定です。

設定項目	説明
default-check-state	メッセージ通知設定画面における配信可否の初期値を設定します。 設定可能な値は以下の通りです。 ・checked - 配信する ・unchecked - 配信しない 未設定の場合は、checked （配信する）となります。

メールテンプレートの利用

IM-MessageHub では、通常のテンプレートファイル以外にメールモジュールで利用可能なメールテンプレートを扱うことが可能です。具体的には、MailTemplateクラス で利用可能なメールテンプレートを扱うことができます。

> **コラム**
>
> 「メールモジュール」の詳細は以下の通りです。
>
> | | |
> | --- | --- |
> | モジュール名 | メールモジュール（標準機能/基盤機能） |
> | モジュールID | jp.co.intra_mart.im_javamail |

この章では、IM-MessageHub でメールテンプレートを利用する方法を説明します。

項目

メールテンプレートファイル

メールテンプレートファイルとは、メールモジュールで利用するテンプレート情報を定義した XML ファイルです。
メールテンプレートファイルを作成する上での規約、および、基本的な記述方法は以下の通りです。

ファイル名

メールテンプレートファイルの命名規約は以下の通りです。

```
<テンプレート名>_<ロケール>.xml
```

ファイル名を構成する要素（ <～> によって囲まれている部分）の詳細は以下の通りです。

表：構成する要素の詳細

テンプレート名	メールテンプレートを表す任意の名称を指定します。
ロケール	メールテンプレートファイルのロケールを指定します。 言語ごとにメールテンプレートファイルを作成することで、多言語に対応することが可能です。 ロケールは <language>_<country>_<variant> 形式で指定します。

配置場所

メールモジュールは以下のディレクトリをルートディレクトリとして、メールテンプレートファイルを管理します。

```
%CONTEXT_PATH%/WEB-INF/conf/mail_template/
```

作成したメールテンプレートファイルは、このルートディレクトリ配下に配置してください。
なお、メールモジュールはルートディレクトリ配下のサブフォルダを含めて管理対象としています。

新規に作成したメールテンプレートファイルを配置する際には intra-mart Accel Platform 標準で同梱されているメールテンプレートファイルと区別するために専用のフォルダを作成することを推奨します。

メールテンプレートのテンプレートフォーマット

メールテンプレートの作成例は以下の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<mail-template xmlns="http://www.intra-mart.co.jp/system/mail/template"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.co.jp/system/mail/template ../../../schema/mail-template.xsd ">
   <headers>
      <header name="Organization" value="IM-MessageHub Sample" />
   </headers>
   <from address="sekine@intra-mart.jp" personal="SEKINE" />
   <replyTo>
      <mail address="maruyama@intra-mart.jp" personal="MARUYAMA"/>
   </replyTo>
   <to>
      <mail address="aoyagi@intra-mart.jp" personal="AOYAGI" />
   </to>
   <cc>
      <mail address="ueda@intra-mart.jp" personal="UEDA_CC"/>
   </cc>
   <bcc>
      <mail address="ueda@intra-mart.jp" personal="UEDA_BCC"/>
   </bcc>
   <subject>Mail Title</subject>
   <body>
      Main Body Contents
   </body>
</mail-template>
```

表：設定が必要なタグとその詳細

タグ名	説明
headers	送信するメールに指定するメールヘッダー情報をまとめて定義するための親タグです。 headers タグの内部には、 header タグを複数定義することが出来ます。
header	送信するメールに指定するメールヘッダー情報を定義します。 header タグは属性として name と value を持ち、 name にはメールヘッダー名を、 value にはメールヘッダーに対応する値を設定します。
from	差出人アドレスを定義します。 from タグは属性として address と personal を持ち、 address には差出人のアドレスを、 personal には差出人の表示名を設定します。
replyTo	返信先アドレスをまとめて定義するための親タグです。 replyTo タグの内部には、 mail タグを複数定義することが出来ます。
to	宛先アドレスをまとめて定義するための親タグです。 to タグの内部には、 mail タグを複数定義することが出来ます。
cc	カーボンコピー先のアドレスをまとめて定義するための親タグです。 cc タグの内部には、 mail タグを複数定義することが出来ます。
bc	ブラインドカーボンコピー先のアドレスをまとめて定義するための親タグです。 bcc タグの内部には、 mail タグを複数定義することが出来ます。
mail	メールアドレス情報を定義します。 mail タグは属性として、 address と personal を持ち、address にはアドレスを、 personal には表示名を設定します。
subject	送信するメールの件名を定義します。
body	送信するメールの本文を定義します。

注意

タグの定義順について

メールテンプレートで利用するタグには定義順が設定されています。
以下の順序で定義してください。

1. headers
2. from
3. replyTo
4. to
5. cc
6. bcc
7. subject
8. body

メールテンプレート用アノテーション

メールテンプレートを利用するには、Event クラスにメールテンプレートの情報をアノテーションで付加する必要があります。

- アノテーション
  - jp.co.intra_mart.foundation.template.annotation.Template
- 引数
  - mediaId
    - immh.im_mail.legacy を指定してください。
  - path
    - テンプレートファイルのパスを <%CONTEXT_PATH%/WEB-INF/conf/mail_template > からの相対パスで指定してください。

メールテンプレートを利用するイベントの作成例は以下の通りです。

```
1  package sample.message_hub.event;
2
3  import jp.co.intra_mart.foundation.message_hub.model.Event;
4  import jp.co.intra_mart.foundation.template.annotation.Template;
5
6  @Template(mediaId="immh.im_mail.legacy", path="jp.co.intra_mart/sample/mailtemplate")
7  public class SampleLegacyMailEvent extends Event {
8
9   private static final long serialVersionUID = -8837862755292447983L;
10
11 }
```

メッセージの作成

メールテンプレートを利用したメッセージの作成方法は、「 *IM-MessageHub を利用してメッセージを配信する* 」で説明したメッセージの作成方法と変わりません。

ここでは、メール配信に関する詳細な設定方法について説明します。

Cc、Bcc の設定方法

IM-MessageHub では、メール配信を行う際の宛先メールアドレスは To として扱われます。
Cc、Bcc として扱う場合は、アドレス情報に属性を設定します。

表：属性名と属性値

属性名	属性値
im_mail.cc	メールアドレスを Cc として扱う場合、この属性値に 文字列 true を設定してください。
im_mail.bcc	メールアドレスを Bcc として扱う場合、この属性値に 文字列 true を設定してください。

宛先メールアドレスを Cc、Bcc として扱うサンプルは以下の通りです。

```java
package sample.message_hub.create;

import java.util.ArrayList;
import java.util.List;

import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.message_hub.mail.delivery.MailMessageDelivererConstants;
import jp.co.intra_mart.foundation.message_hub.model.Address;
import jp.co.intra_mart.foundation.message_hub.model.Event;
import jp.co.intra_mart.foundation.message_hub.model.Message;
import jp.co.intra_mart.foundation.message_hub.model.UserAddress;
import sample.message_hub.event.SampleLegacyMailEvent;

public class SampleLegacyMailCreator {

  /**
   * メッセージを作成します。
   */
  public static Message create() {

    // イベントの作成（メールテンプレート用のイベントを設定）
    final Event event = new SampleLegacyMailEvent();

    // 送信元アドレスの作成
    // 今回、送信元アドレスはアカウントコンテキスから取得しています。
    AccountContext context = Contexts.get(AccountContext.class);
    final Address fromAddress = new UserAddress(context.getUserCd());

    // 宛先アドレスの作成
    final List<Address> toAddressList = new ArrayList<Address>();

    // To設定
    toAddressList.add(new UserAddress("aoyagi"));

    // Cc設定
    Address ccAddress = new UserAddress("ueda");
    ccAddress.setAttribute(MailMessageDelivererConstants.ATTRIBUTE_NAME_FOR_CC, Boolean.TRUE.toString()); // Cc設定用パラメータ
    toAddressList.add(ccAddress);

    // Bcc設定
    Address bccAddress = new UserAddress("ueda");
    bccAddress.setAttribute(MailMessageDelivererConstants.ATTRIBUTE_NAME_FOR_BCC, Boolean.TRUE.toString()); // Bcc設定用パラメータ
    toAddressList.add(bccAddress);

    // メッセージの作成
    final Message message = new Message(event, fromAddress, toAddressList);
    // ... message属性の設定は省略します ...

    // 作成したメッセージを返します。
    return message;
  }
}
```

コラム

- 属性名の指定には、以下の定数を利用してください。
  - MailMessageDelivererConstants.ATTRIBUTE_NAME_FOR_CC
  - MailMessageDelivererConstants.ATTRIBUTE_NAME_FOR_BCC
- Cc、および、Bcc の設定を両方有効にした場合、対象の宛先情報は Cc と Bcc どちらにも設定されます。

メールアドレス解決の拡張

IM-MessageHub では、メール配信を行う際に、宛先、および、送信元のメールアドレス解決を行います。
この解決は、以下のインタフェースの実装が行います。

表：提供インタフェース一覧

インタフェース	説明
ToMailAddressResolver	配信メッセージに指定された宛先のメールアドレスを解決します。
FromMailAddressResolver	配信メッセージに指定された送信元のメールアドレスを解決します。

パッケージ名は、いずれも jp.co.intra_mart.foundation.message_hub.mail.delivery です。

例えば、このインタフェースを実装することにより以下を実現することができます。

- アプリケーションがメールアドレス 1 と 2 に送るためのチェックボックスを用意し、その設定値を元にメールアドレスを解決する。
- 別のシステムで管理されているメールアドレスを利用する。

各インタフェースの詳細は、APIドキュメントを参照してください。

拡張例

ToMailAddressResolver、および、FromMailAddressResolver インタフェースの実装は、 ServiceLoaderUtil#loadPriority(Class) を利用して順番に実行されます。

インタフェースの実装を有効化するために以下を行ってください。

- 「プロバイダ構成ファイル」を配置してください。
  （詳しくは、 APIドキュメント「 java.util.ServiceLoader 」を参照してください）
- 実装クラスに 優先度 (@Priority) を設定してください。
  （詳しくは、APIドキュメント「 ServiceLoaderUtil 」、および、「 @Priority 」を参照してください）

ここでは、ToMailAddressResolver インタフェースの実装例、および、実装の有効化方法について説明します。

- 実装例

```java
package sample.message_hub.address.resolver;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Locale;

import jp.co.intra_mart.common.annotation.Priority;
import jp.co.intra_mart.foundation.mail.javamail.model.MailAddress;
import jp.co.intra_mart.foundation.message_hub.mail.delivery.ToMailAddressResolver;
import jp.co.intra_mart.foundation.message_hub.model.Event;
import sample.message_hub.event.SampleLegacyMailEvent;

/**
 * 宛先に対するメールアドレス解決用インターフェースのサンプル実装
 */
@Priority(value = 500)
public class SampleToMailAddressResolver implements ToMailAddressResolver {

    @Override
    public List<MailAddress> getMailAddress(final String userCd, final Date baseDate, final Locale locale) {

        // 独自のアプリケーションが管理する情報から、メールアドレスを取得する。
        final List<String> addressList = getAddressListByApplicationMaintainDB(userCd, baseDate);

        final List<MailAddress> result = new ArrayList<MailAddress>();
        for (final String address : addressList) {
            result.add(new MailAddress() {
                @Override
                public String getAddress() {
                    return address;
                }

                @Override
                public String getPersonal() {
                    return userCd; // ユーザコードで代替
                }
            });
        }

        return result;
    }

    @Override
    public boolean isSupported(final Event event) {
        // 今回作成したサンプルイベントの場合のみ適用
        return SampleLegacyMailEvent.class.isInstance(event);
    }

    private List<String> getAddressListByApplicationMaintainDB(final String userCd, final Date baseDate) {
        // ...
        // アプリケーションがDB上で管理するアドレス情報を、引数のユーザコードおよび基準日から取得するメソッド。
        // ...
    }

}
```

実装を有効化するために、プロバイダ構成ファイルを作成し、実装クラスの完全修飾クラス名（FQCN）を指定します。

- META-INF/services/jp.co.intra_mart.foundation.message_hub.mail.delivery.ToMailAddressResolver

```
sample.message_hub.address.resolver.SampleToMailAddressResolver
```

ToMailAddressResolver 一覧

intra-mart Accel Platform が提供している ToMailAddressResolver は以下の通りです。

表：ToMailAddressResolver一覧

クラス名（単純名）	優先度	説明	提供バージョン
IMBoxToMailAddressResolver	100	IMBox で指定されている宛先のメールアドレスを解決するクラスです。	2014 Winter(Iceberg)
StandardToMailAddressResolver	未設定	メッセージの宛先に指定されたユーザコード と ロケールを元に、IM-共通マスタ のプロファイルに設定されているメールアドレスを解決します。 基準日を「システム日時」としてユーザを検索します。 具体的には、MailAddressResolverUtil.getMailAddressContainer(String, Date, Locale) の結果を 以下の順番で検索し、最初に見つかったメールアドレスを返却します。 1. MailAddressContainer.getMailAddress1() 2. MailAddressContainer.getMailAddress2() 以下の場合、空のListが返却されます。 - 指定されたユーザコードと基準日（=システム日時）に紐づくユーザが存在しなかった場合 - メールアドレスがいずれも見つからなかった場合 このように、この実装クラスが返却するListの要素数は 0 または 1 です。これは、複数のメールアドレスが返却されることはないということを意味します。 このクラスは、全てのEventに対して有効です。 最後に読み込まれることを想定した標準実装であるため、優先度は指定されていません。	2014 Winter(Iceberg)

FromMailAddressResolver 一覧

intra-mart Accel Platform が提供している FromMailAddressResolver は以下の通りです。

表：FromMailAddressResolver一覧

クラス名（単純名）	優先度	説明	提供バージョン
IMBoxToMailAddressResolver	100	IMBox で指定されている送信元のメールアドレスを解決するクラスです。	2014 Winter(Iceberg)

クラス名（単純名）	優先度	説明	提供バージョン
StandardToMailAddressResolver	未設定	メッセージの送信元に指定されたユーザコード と ロケールを元に、IM-共通マスタ のプロファイルに設定されているメールアドレスを解決します。 基準日を「システム日時」としてユーザを検索します。 本クラスの getMailAddress(String, Date, Locale) メソッドは、StandardToMailAddressResolver#getMailAddress(String, Date, Locale) の 最初の要素を返却します。 結果が空のListだった場合は null を返却します。 このクラスは、全てのEventに対して有効です。 最後に読み込まれることを想定した標準実装であるため、優先度は指定されていません。	2014 Winter(Iceberg)

IM-MessageHub の非同期処理機能

ここでは IM-MessageHub の非同期処理機能と設定方法について説明します。

IM-MessageHub の非同期処理機能とは

IM-MessageHub では、メッセージの配信処理などの非同期タスクの管理に、独自の非同期機構を利用しています。
この非同期機構では、非同期タスククラスごとにキューを保持し、IM-MessageHub の各非同期タスクを処理します。
IM-MessageHub が保持する非同期タスクキューは以下の通りです。

表：非同期タスククラスと詳細

タスククラス名	説明
jp.co.intra_mart.system.message_hub.task.MessageDispatchTask	メッセージを受け付けるためのキューです。
jp.co.intra_mart.system.message_hub.delivery.strategy.MessageDeliveryTask	MessageDeliverer に処理を依頼するためのキューです。

IM-MessageHub の非同期処理機能の設定

IM-MessageHub の持つ非同期機構では、設定ファイルによって、実行するタスククラス毎の実行スレッド数やキューの詳細を変更することが可能です。
標準では、IM-Juggling 上で設定ファイルは提供されません。
変更を行う場合には設定ファイルを新しく作成し、WARへ含めた上でアプリケーションサーバにデプロイする必要があります。

IM-Juggling のプロジェクト上に作成する設定ファイル名とパスは以下の通りです。

- `<conf/im-message-hub-worker-config.xml>`

設定例は以下の通りです。

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

 <task
   type             ="jp.co.intra_mart.system.message_hub.task.MessageDispatchTask"
   thread-count       ="1"
   queue-size         ="1000"
   polling-timeout    ="6000"
   queueing-retry-count ="100"
   queueing-timeout   ="300"/>

 <task
   type             ="jp.co.intra_mart.system.message_hub.delivery.strategy.MessageDeliveryTask"
   thread-count       ="3"
   queue-size         ="1000"
   polling-timeout    ="6000"
   queueing-retry-count ="100"
   queueing-timeout   ="300"/>

</im-message-hub-worker-config>
```

表：属性の詳細

設定項目	説明	デフォルト値
type	設定を行う非同期タスククラスの完全修飾クラス名（FQCN）を指定します。	
thread-count	実行スレッド数を指定します。	`3`
queue-size	キューのサイズを指定します。	`2147483647`（Integer.MAX_VALUE）

設定項目	説明	デフォルト値
polling-timeout	キューからの取得処理のタイムアウトまでの時間（ミリ秒）を指定します。	60000 [ms]
queueing-retry-count	キューへの登録処理の再試行回数を指定します。	100
queueing-timeout	キューへの登録処理のタイムアウトまでの時間（ミリ秒）を指定します。	300 [ms]

IM-MessageHub の非同期処理機能のデフォルト設定

標準では、IM-MessageHub では以下のデフォルト設定が行われています。

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

 <task
   type="jp.co.intra_mart.system.message_hub.task.MessageDispatchTask"
   max-thread-count="5"/>

 <task
   type="jp.co.intra_mart.system.message_hub.delivery.strategy.MessageDeliveryTask"
   max-thread-count="25"/>

</im-message-hub-worker-config>
```

この設定により、それぞれのタスクキューに対してデフォルトのスレッド数が設定されています。

表：非同期タスククラスとデフォルトスレッド数

タスククラス名	スレッド数
jp.co.intra_mart.system.message_hub.task.MessageDispatchTask	5
jp.co.intra_mart.system.message_hub.delivery.strategy.MessageDeliveryTask	25

> **注意**
>
> これらの設定は、設定ファイルを変更することによって上書きされます。
> 作成した設定ファイルには、必ず２つのタスクキューを設定し、スレッド数の設定を行うようにしてください。

配信メッセージの分割

ここでは配信メッセージの分割と設定方法について説明します。

配信メッセージの分割とは

IM-MessageHub の行う配信処理は、各配信先メディアに対してそれぞれ一度行われます。
この時、配信先メディアへの配信処理によっては、一度に配信できない場合があります。
このような場合のために、IM-MessageHub では、配信メッセージを分割してから配信処理を実行します。

IM-MessageHub で標準で用意されている配信メッセージの分割処理は以下の通りです。

1. *宛先ユーザ情報による分割*
2. *宛先の数による分割*

なお、メッセージの分割処理は、上記の順に実行されます。

宛先ユーザ情報による分割

全てのメッセージを一度に配信した場合、宛先ユーザによっては自分のロケールと異なるロケールでメッセージが通知されることとなります。
IM-MessageHub では、メッセージの本文が同一となるように、宛先ユーザの「ロケール」と「タイムゾーン」が同一の宛先ごとに配信メッセージを分割します。

この処理は、配信先メディアや設定によらず、常に実行されます。

宛先の数による分割

IM-MessageHub を利用したメール配信の場合、一つのメッセージに多くの宛先（TO）が指定された状態で配信処理を行うと、メールサーバへの過負荷や規制の対象となる可能性が考えられます。

この問題を回避するために、 IM-MessageHub には、配信クラス毎に宛先の配信上限数を指定することができます。
メッセージの宛先が設定された上限数を越えたメッセージは、分割されて配信されます。

IM-MessageHub は宛先の上限数を配信先メディアを定義するプラグインファイル内に設定します。
具体的には、「 *配信先メディアを定義する* 」で作成したプラグインファイルに設定します。

宛先の上限数を 50 とする設定例は以下の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
   <extension point = "jp.co.intra_mart.message_hub.DeliveryMedias">
      <delivery-media
            id   = "immh.sample.delivery"
            name = "Sample Delivery Media"
            name-key = "CAP.Z.SAMPLE.APPLICATION.NAME"
            deliverer-class ="sample.message_hub.delivery.SampleMessageDeliverer"
            version = "8.0.0"
            rank = "100" >
         <init-param>
            <param-name>max-address-size</param-name>
            <param-value>50</param-value>
         </init-param>
      </delivery-media>
   </extension>
</plugin>
```

表：属性の詳細

タグ名	説明
param-name	max-address-size を指定してください。
param-value	最大宛先数とする数値を指定してください。 デフォルトでは 0 （分割しない）になります。

IM-MessageHub のシーケンス図

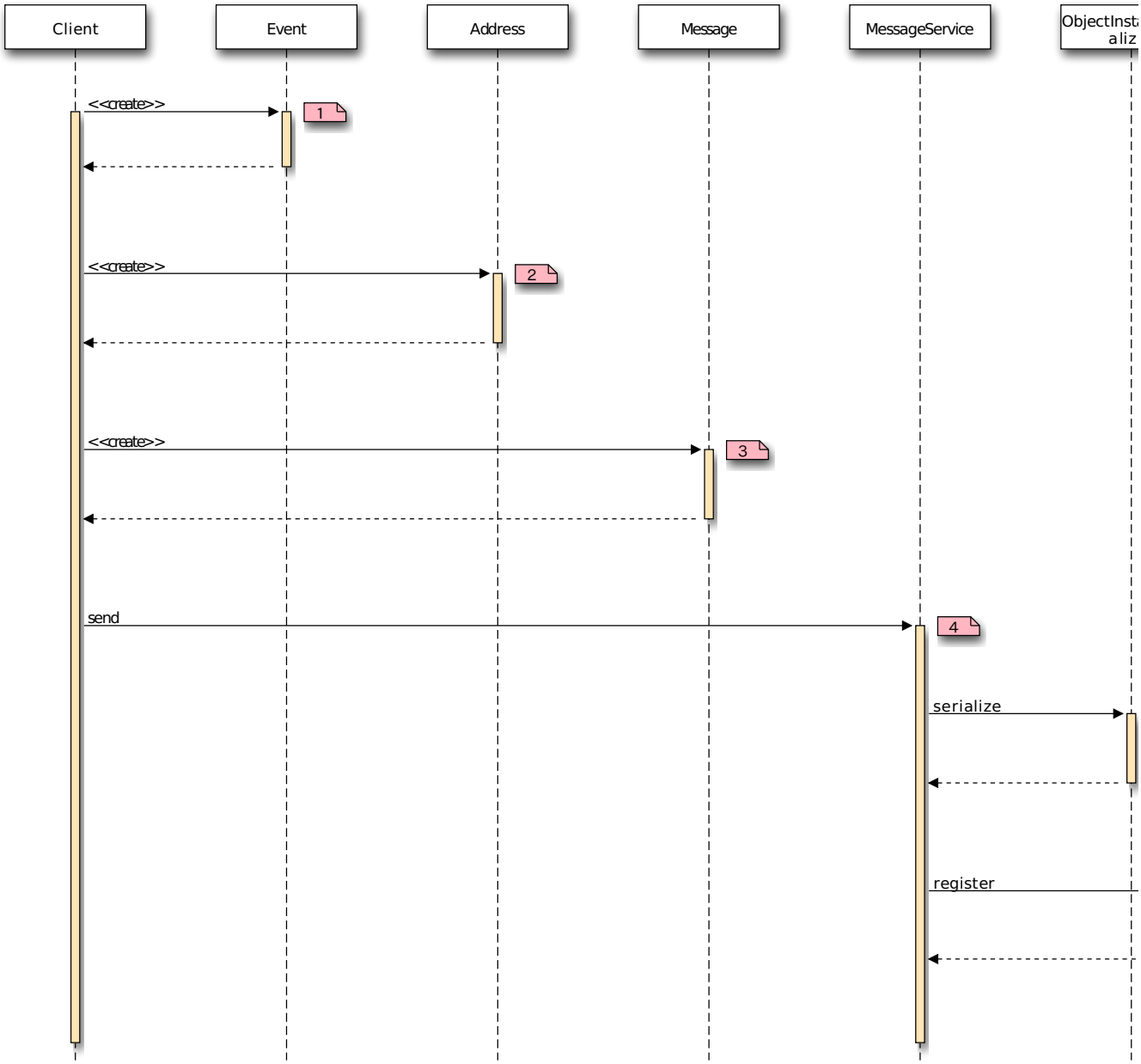
IM-MessageHub の処理のシーケンス図とその詳細を説明します。

項目

メッセージ配信を受け付ける

IM-MessageHub は、メッセージ配信の依頼を受け付けた後、すぐにメッセージ配信アプリケーションに対してレスポンスを返します。
受け付けたメッセージ配信の依頼は、非同期の配信処理タスクとして別スレッドで実行されます。
これにより、メッセージ配信アプリケーションは、メッセージ配信処理の完了を待つこと無く、後続の処理を続けることが可能となります。

メッセージ配信アプリケーションがメッセージを作成してから、IM-MessageHub に対して配信依頼を行うまでのシーケンス図は以下の通りです。



Name	Description
Client	メッセージ配信アプリケーション
Event	jp.co.intra_mart.foundation.message_hub.model.Event
Address	jp.co.intra_mart.foundation.message_hub.model.Address
Message	jp.co.intra_mart.foundation.message_hub.model.Message
MessageService	jp.co.intra_mart.foundation.message_hub.service.MessageService
ObjectInstanceSerializer	jp.co.intra_mart.system.message_hub.util.ObjectInstanceSerializer
TaskRegisterer	jp.co.intra_mart.system.message_hub.task.TaskRegisterer

シーケンスの詳細は以下の通りです。

1. メッセージ配信アプリケーションは、配信するメッセージがどの様なイベントによるものかを表すクラスを生成します。
2. メッセージ配信アプリケーションは、送信者／宛先のユーザのアドレス情報を表すクラスを生成します。
3. メッセージ配信アプリケーションは、前工程で作成したイベント、アドレス情報をもとにメッセージを作成します。
4. メッセージ配信アプリケーションは、作成したメッセージの配信処理を MessageService に依頼します。
5. MessageService は、受け付けたメッセージ配信処理を非同期の配信処理タスクとして登録します。その後、すぐにメッセージ配信アプリケーションに対してレスポンスを返します。以降、メッセージ配信に関わる処理は非同期で動作します。

メッセージを配信先メディアに配信する

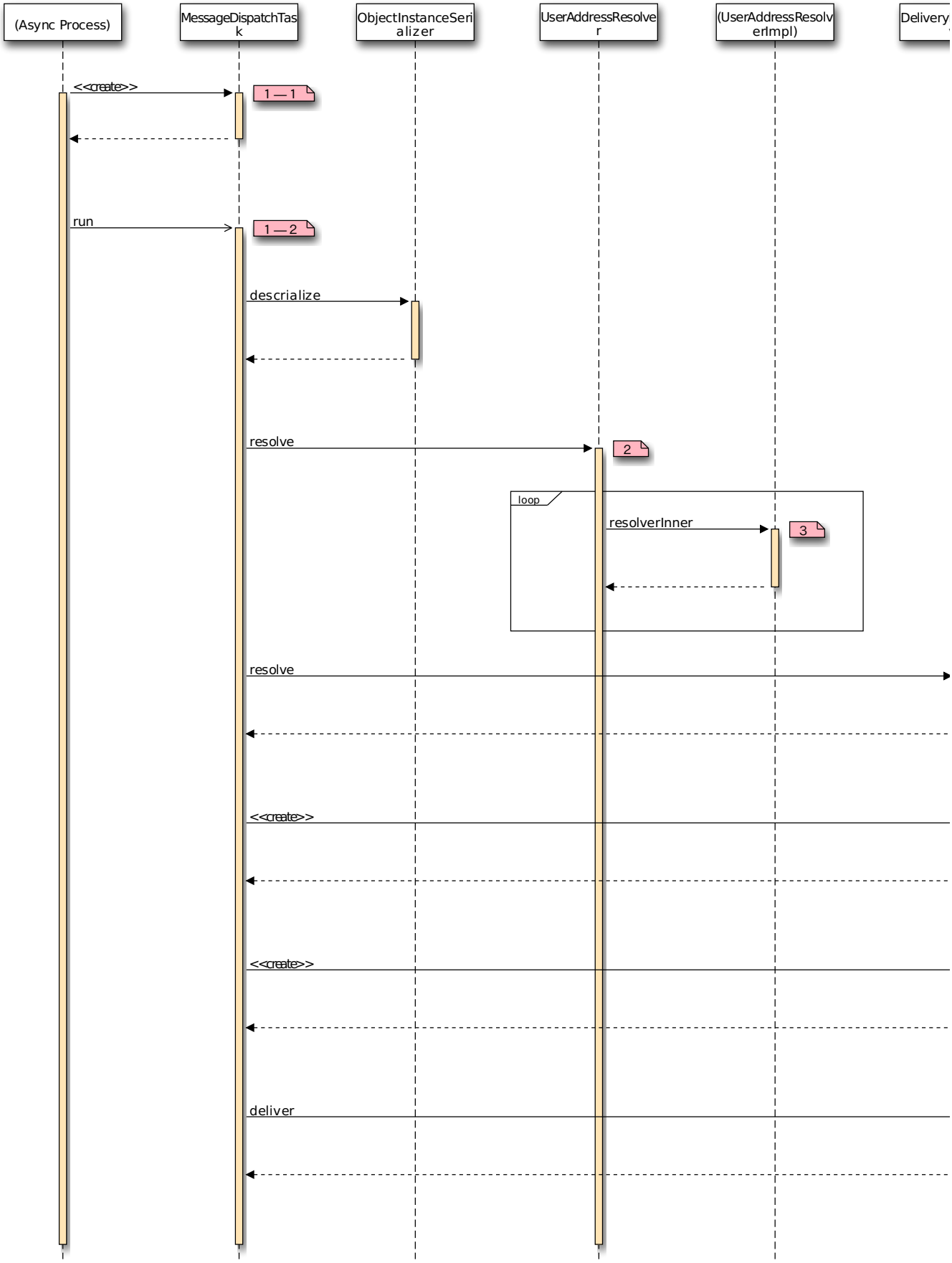
IM-MessageHub は「*メッセージ配信を受け付ける*」で受け付けたメッセージを、登録されている配信先メディアに対して配信します。

配信処理は以下の順に実行されます。

1. *メッセージの配信を配信先メディアに依頼する*
2. *メッセージを分割して配信を実行する*

メッセージの配信を配信先メディアに依頼する

IM-MessageHub が受け付けたメッセージを解析し、配信先メディアごとの処理に分割するまでのシーケンス図は以下の通りです。



Name	Description
(Async Process)	非同期処理
MessageDispatchTask	jp.co.intra_mart.system.message_hub.task.MessageDispatchTask
ObjectInstanceSerializer	jp.co.intra_mart.system.message_hub.util.ObjectInstanceSerializer
UserAddressResolver	jp.co.intra_mart.foundation.message_hub.resolver.address.UserAddressResolver
(UserAddressResolverImpl)	UserAddressResolverの実装クラス
DeliveryMediaResolver	jp.co.intra_mart.foundation.message_hub.resolver.media.DeliveryMediaResolver
DeliveryMessage	jp.co.intra_mart.foundation.message_hub.delivery.model.DeliveryMessage
MessageDelivererWrapperByDividing	jp.co.intra_mart.system.message_hub.delivery.strategy.MessageDelivererWrapperByDividing

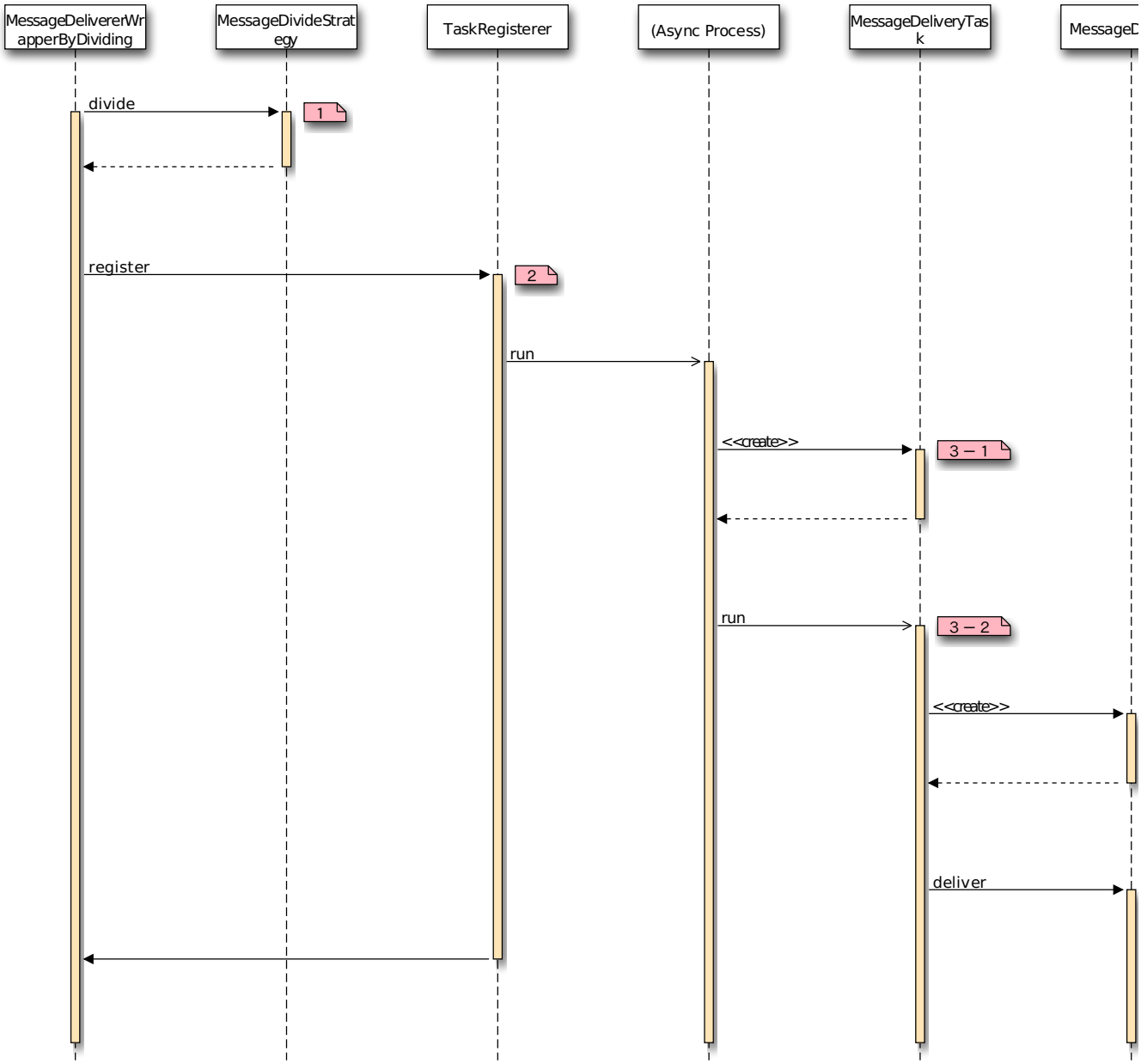
シーケンスの詳細は以下の通りです。

1. 非同期の配信処理タスクが生成され（１－１）、実行されます（１－２）。
2. 配信を行うメッセージに設定されている送信者/宛先の情報を元に、ユーザ情報を解決します。
3. 具体的なユーザ情報を解決するために、設定されている送信者/宛先の種類に適した UserAddressResolver の実装を呼び出します。
4. 配信先メディアを解決します。
5. 前工程で解決されたユーザ情報、配信先メディア情報をもとに配信メッセージを作成します。
6. メッセージを分割して配信を実行する MessageDelivererWrapperByDividing を生成し（６－１）、配信処理を委譲します（６－２）

メッセージを分割して配信を実行する

MessageDelivererWrapperByDividing クラスでは、配信処理を実行するために渡されたメッセージの宛先情報を解析します。
宛先情報を解析した結果、必要であればメッセージを分割した上で、再度非同期の配信処理タスクを作成し、配信を依頼します。

メッセージを分割して配信を実行する処理に関するシーケンス図は以下の通りです。



Name	Description
MessageDelivererWrapperByDividing	jp.co.intra_mart.system.message_hub.delivery.strategy.MessageDelivererWrapperByDividing
MessageDivideStrategy	jp.co.intra_mart.foundation.message_hub.delivery.strategy.MessageDivideStrategy
TaskRegisterer	jp.co.intra_mart.system.message_hub.task.TaskRegisterer
(Async Process)	非同期処理
MessageDeliveryTask	jp.co.intra_mart.system.message_hub.delivery.strategy.MessageDeliveryTask
MessageDeliverer	jp.co.intra_mart.foundation.message_hub.delivery.MessageDeliverer

シーケンスの詳細は以下の通りです。

1. MessageDelivererWrapperByDividing クラスが、MessageDivideStrategy を利用してメッセージを分割します。
2. 分割したメッセージを、それぞれ非同期の配信処理タスクとして登録します。
3. 分割したメッセージの配信処理を行うタスクが生成され（３－１）、実行されます（３－２）。
4. メッセージに設定されている宛先情報および配信先メディア情報から、実際に配信処理を行う MessageDeliverer クラスを生成し（４－１）、配信処理を実行します（４－２）。