

BL-09 メッセージ管理機能

■ 概要

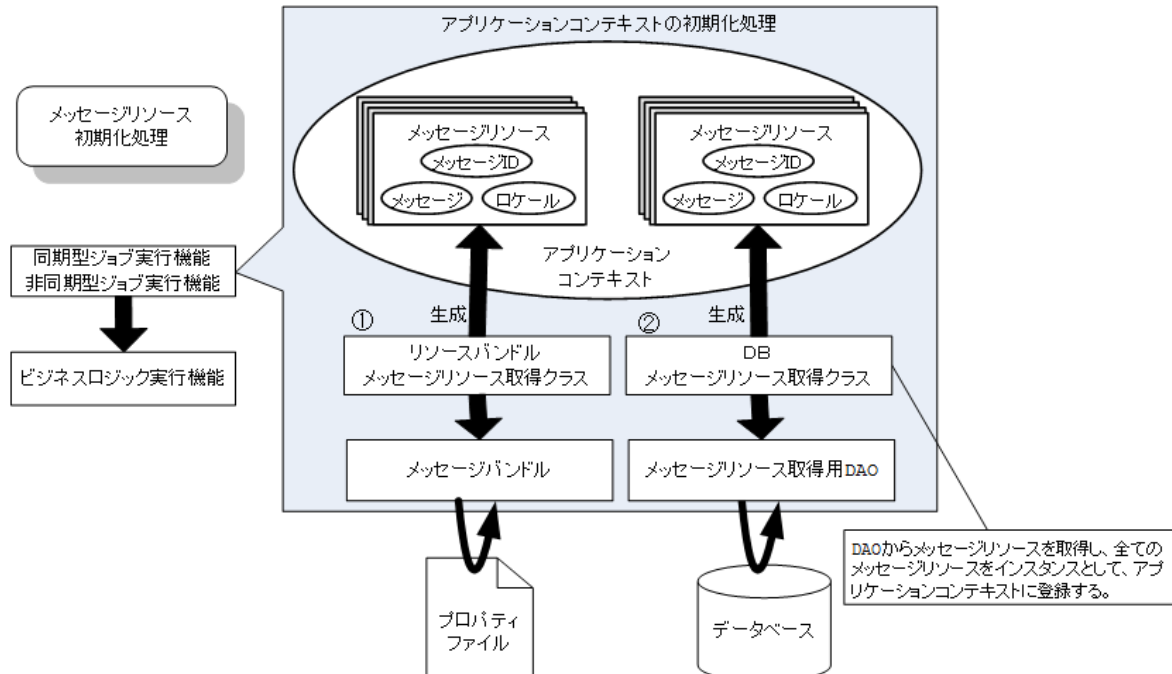
◆ 機能概要

- 主にログに出力する文字列（メッセージリソース）を管理する機能であり、プロパティファイルやデータベース内のテーブルに定義したメッセージリソースをビジネスロジックで取得するための機能を提供する。
- 本機能は Terasoluna Server Framework for Java ver. 2.x の『CE-01 メッセージ管理機能』と同等である。

◆ 概念図・解説

- 同期型ジョブ実行機能、または、非同期型ジョブ実行機能の起動時にメッセージリソースがアプリケーションコンテキストに登録される。
- ビジネスロジックではアプリケーションコンテキストからメッセージを取得する。

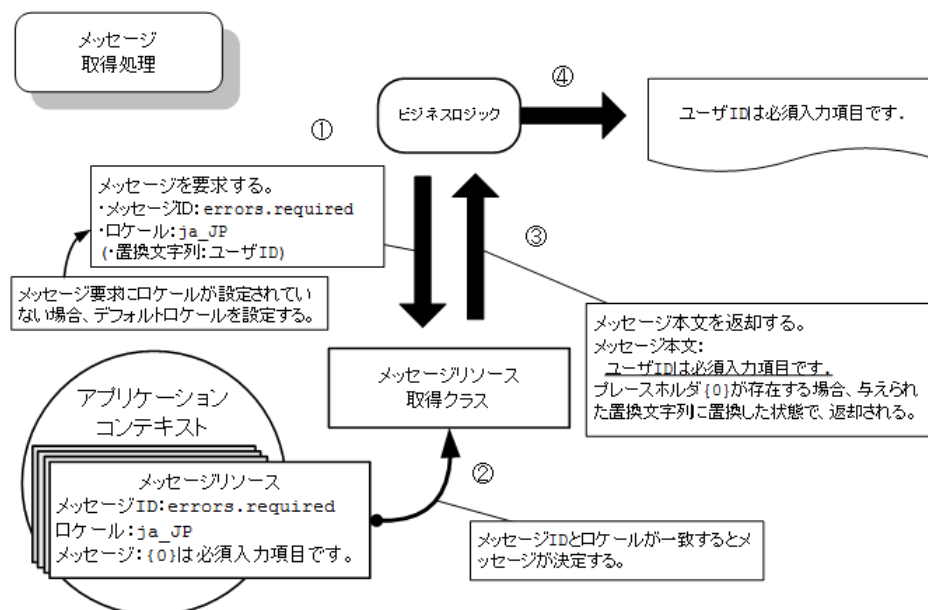
- メッセージリソース初期化処理の概念図



メッセージリソースは後述の設定により、プロパティファイル、または、データベースから生成する。

- ① リソースバンドルメッセージリソースを生成する
リソースバンドルを使用してプロパティファイルを読み込み、メッセージリソースのインスタンスを作成し、アプリケーションコンテキストに登録する。
- ② データベースメッセージリソースを生成する
メッセージリソース取得用 DAO を使用して、データベース内に定義されたすべてのメッセージを取り出し、{メッセージ ID, ロケール, メッセージ文字列} の組としてメッセージリソースのインスタンスを作成し、アプリケーションコンテキストに登録する。

● メッセージ取得処理の概念図



- ① ビジネスロジックからリソースメッセージを要求する
取得したいメッセージのメッセージ ID およびロケール文字列、(必要に応じて) プレースホルダ用の置換文字列を引数に指定して呼び出す。
- ② メッセージリソース取得クラスはメッセージリソースを検索する
メッセージ ID、ロケールを検索キーにアプリケーションコンテキストに登録されたメッセージリソース内を検索する。
- ③ メッセージリソース取得クラスはメッセージを返却する
②で検索したメッセージをビジネスロジックに返却する。メッセージにプレースホルダ (図では `{0}`) が存在する場合は、メッセージリソース取得クラスのメソッド呼び出し時に引数として渡された置換文字列に置き換えたメッセージを返却する。
- ④ ビジネスロジックで取得したメッセージを使用する
取得したメッセージをログ出力する。

■ 使用方法

◆ コーディングポイント

- ソフトウェアアーキテクトが行うコーディングポイント（リソースバンドル）

➤ メッセージリソース取得 Bean の定義

以下のように"messageSource"という識別子の Bean を AdminContext.xml または commonContext.xml のどちらかに定義する。どちらに定義するかは、後述の「業務開発者が行うコーディングポイント」で説明する。

class 属性には Spring Framework が提供する ResourceBundleMessageSource を指定し、メッセージが定義されたクラスパス上のプロパティファイルを列挙する。

◇ Bean 定義ファイル定義例（AdminContext.xml / commonContext.xml）



定義するプロパティファイルが多い場合は、以下のようにリストで指定できる。

◇ Bean 定義ファイル定義例（AdminContext.xml / commonContext.xml）

```
<bean id="messageSource"
      class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basenames">
    <list>
      <value>common-messages</value>
      <value>B000001-messages</value>
      <value>B000002-messages</value>
      ... (省略) ...
    </list>
  </property>
</bean>
```

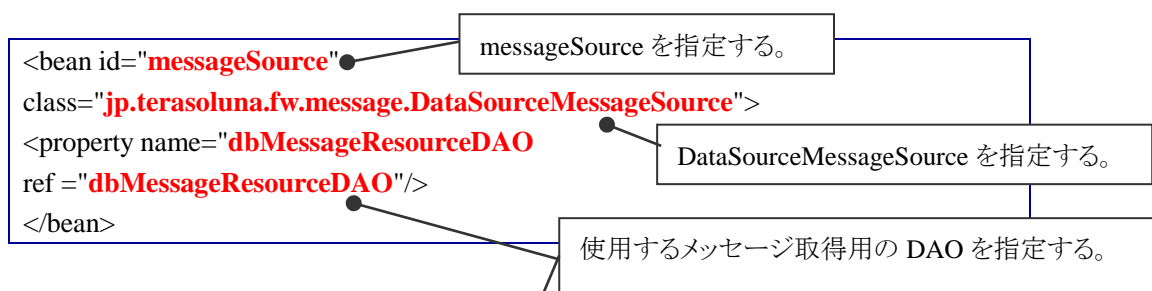
- ソフトウェアアーキテクトが行うコーディングポイント（データベースメッセージ）

➤ メッセージリソース取得 Bean の定義

以下のように"messageSource"という識別子の Bean を AdminContext.xml または commonContext.xml のどちらかに定義する。どちらに定義するかは、後述の「業務開発者が行うコーディングポイント」で説明する。

class 属性にはフレームワークが提供する DataSourceMessageSource を指定し、後述の「メッセージリソース取得用の DAO」を DI する。

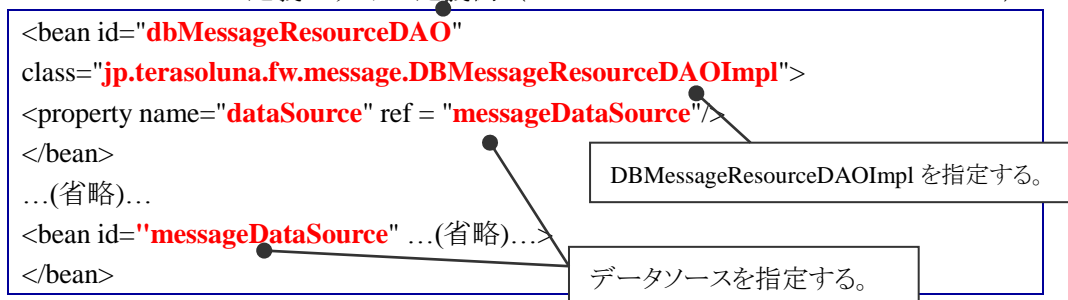
◇ Bean 定義ファイル定義例（AdminContext.xml / commonContext.xml）



➤ メッセージリソース取得用 DAO の Bean 定義

DBMessageResourcesDAO インタフェースを指定し、データソースを DI する。フレームワークではこのインタフェースのデフォルト実装として DBMessageResourceDAOImpl クラスを提供している。

◇ Bean 定義ファイル定義例（AdminContext.xml / commonContext.xml）



➤ データソースの定義

『BL-06 データベースアクセス機能』を参照して設定する。

➤ メッセージリソースの定義

メッセージリソースはデータベース中の以下のテーブルに格納する。

- ・テーブル名 : MESSAGES
- ・メッセージコードを格納するカラム名 : CODE
- ・メッセージ本文を格納するカラム名 : MESSAGE

DBMessageResourceDAOImpl が発行する SQL は以下である。

| |
|--|
| SELECT CODE,MESSAGE FROM MESSAGES |
|--|

テーブルスキーマや SQL を変更する場合は、後述の「データベースメッセージリソース取得方法」を参照すること。

- 業務開発者が行うコーディングポイント

- MessageAccessor を使用する場合のメッセージ取得方法

ビジネスロジックで MessageAccessor を DI し、使用する。

commonContext.xml に定義したメッセージリソースを取得する。

メッセージリソースへのアクセスは Spring Framework が提供する MessageSource を使用してもよいが、フレームワークでは、ロケールを業務開発者に意識させずにメッセージリソースへアクセスできる MessageAccessor インタフェースを提供しているため、原則 MessageAccessor を使用する。

◇ ビジネスロジック実装例

```
public class SampleBLogic implements BLogic {  
  
    @Inject  
    MessageAccessor messageAccessor;  
  
    //ビジネスロジック  
    public int execute (BLogicParam arg0)  
    ...中略...  
    } catch(RuntimeException e) {  
        if(log.isEnabled()){  
            log.error(  
                messageAccessor.getMessage("errors.runtimeexception", null)  
            );  
            return 255;  
        }  
    }  
    ...(省略)...  
}
```

MessageAccessor を DI する。

MessageAccessor のメソッドを使用して、メッセージを取得する。
置換文字列がない場合、第 2 引数は null。

➤ MessageUtil を使用する場合のメッセージ取得方法（非推奨）¹

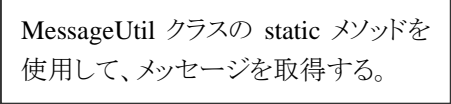
ビジネスロジックで MessageUtil の static メソッドを呼びだし、使用する。

AdminContext.xml に定義したメッセージリソースを取得する。

MessageUtil は MessageAccessor をジョブごとに管理するためのラッパークラスであり、MessageAccessor を使用した場合と同じである。

✧ ビジネスロジック実装例

```
public class SampleBLogic implements BLogic {  
  
    //ビジネスロジック  
    public int execute (BLogicParam arg0) {  
        ...中略...  
    } catch(RuntimeException e) {  
        if(log.isDebugEnabled()){  
            log.error(MessageUtil.getMessage("errors.runtimeexception"));  
            return 255;  
        }  
    }  
    ...中略...  
}  
}
```



MessageUtil クラスの static メソッドを使用して、メッセージを取得する。

¹各ジョブのビジネスロジックで MessageAccessor を DI して使用した場合でも同等のことができるため、本クラスは ver3.6.0 より非推奨としている。

➤ データベースメッセージの更新方法

データベースメッセージを使用している場合、ジョブの起動中にアプリケーションコンテキスト内のメッセージをデータベースから再取得し、更新できる。更新には以下のメソッドを使用する。なお、更新にあたってはアプリケーションコンテキストの参照範囲に注意すること。たとえば、同期型ジョブ実行機能の場合、既に起動済みの他のジョブには影響を与えない。

**jp.terasoluna.fw.message.DataSourceMessageSource クラスの
reloadDataSourceMessage メソッド**

各ビジネスロジックが直接 reloadDataSourceMessage メソッドを使用することはせず、業務共通クラスから使用することを推奨する。

■ リファレンス

◆ 構成クラス

| | クラス名 | 概要 |
|---|--|---|
| 1 | jp.terasoluna.fw.message.DataSourceMessageSource | メッセージを生成、発行するクラス。 |
| 2 | jp.terasoluna.fw.message.DBMessage | データベースのメッセージリソースを保持するクラス。 |
| 3 | jp.terasoluna.fw.message.DBMessageResourceDAO | データベースよりメッセージリソースを抽出するための DAO インタフェース。 |
| 4 | jp.terasoluna.fw.message.DBMessageResourceDAOImpl | データベースよりメッセージリソースを抽出するため DBMessageResourceDAO の実装クラス。 |
| 5 | jp.terasoluna.fw.message.DBMessageQuery | データベースよりメッセージリソースを抽出するために実際のデータベースアクセスを行うクラス。 |
| 6 | jp.terasoluna.fw.batch.message.MessageAccessor | MessageSource に簡易アクセスするためのインタフェース。 |
| 7 | jp.terasoluna.fw.batch.message.MessageAccessorImpl | MessageSource に簡易アクセスするためにフレームワークが提供するデフォルトの実装クラス。 |
| 8 | jp.terasoluna.fw.batch.util.MessageUtil | MessageAccessor をジョブごとに管理するためのラッパークラス。 |

◆ 拡張ポイント

- なし

■ 関連機能

- なし

■ 使用例

- 機能網羅サンプル (terasoluna-batch-functionsample)

■ 備考

◆ データベースメッセージリソース取得方法の変更

メッセージリソース取得用 DAO の Bean 定義のプロパティを変更すると、データベースメッセージリソースを取得する方法を変更できる。

変更できるプロパティは以下のとおり。

| プロパティ名 | デフォルト値 | 概要 |
|----------------|----------|-------------------|
| tableName | MESSAGES | テーブル名 |
| codeColumn | CODE | メッセージコードを格納するカラム名 |
| messageColumn | MESSAGE | メッセージ本文を格納するカラム名 |
| findMessageSql | 枠外参照 | メッセージを取得する SQL 文 |

メッセージを取得する SQL 文のデフォルトは以下のとおり。

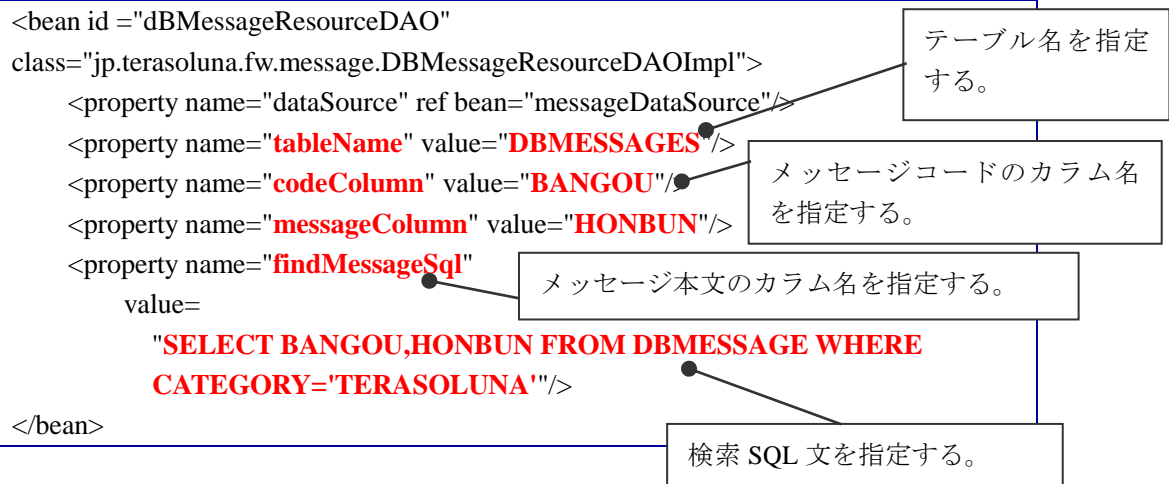
```
SELECT メッセージコード, (言語コード), (国コード), (バリエントコード), メッセージ本文 FROM テーブル名
```

SQL 文を変更する場合でも、codeColumn プロパティおよび messageColumn プロパティで指定したカラムは取得する必要がある。なお、言語コード、国コード、バリエントコードについては、後述の「メッセージの国際化対応」を参照すること。

例として、以下のような変更を加える場合の定義例を示す。

- ・テーブル名 : DBMESSAGES
- ・メッセージコードを格納するカラム名 : BANGOU
- ・メッセージ本文を格納するカラム名 : HONBUN
- ・メッセージを取得する SQL 文 : 『SELECT BANGOU,HONBUN FROM DBMESSAGE WHERE CATEGORY='TERASOLUNA'』

◇ Bean 定義ファイル定義例 (AdminContext.xml / commonContext.xml)



◆ メッセージリソースの追加

"messageSource"という識別子の Bean として設定したメッセージリソースだけではメッセージを決定できない場合に使用されるメッセージリソースを追加できる。

➤ 第2メッセージリソース取得 Bean の定義

以下のように“parentMessageSource プロパティ”に別のメッセージリソースを指定する。BeanID は”messageSource”とは別の名前を付与する必要がある。追加したメッセージリソース取得クラスが `AbstractMessageSource` の継承クラスであれば、さらに“parentMessageSource プロパティ”を指定してメッセージリソースを追加できる。

◇ Bean 定義ファイル定義例 (AdminContext.xml / commonContext.xml)

```
<bean id="messageSource"
class="jp.terasoluna.fw.message.DataSourceMessageSource">
  <property name="parentMessageSource"
    ref="secondMessageSource"/>
  <property name="dbMessageResourceDAO"
    ref="dbMessageResourceDAO"/>
</bean>

<bean id="secondMessageSource"
  class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basenames" value="applicationResources,errors"/>
</bean>
```

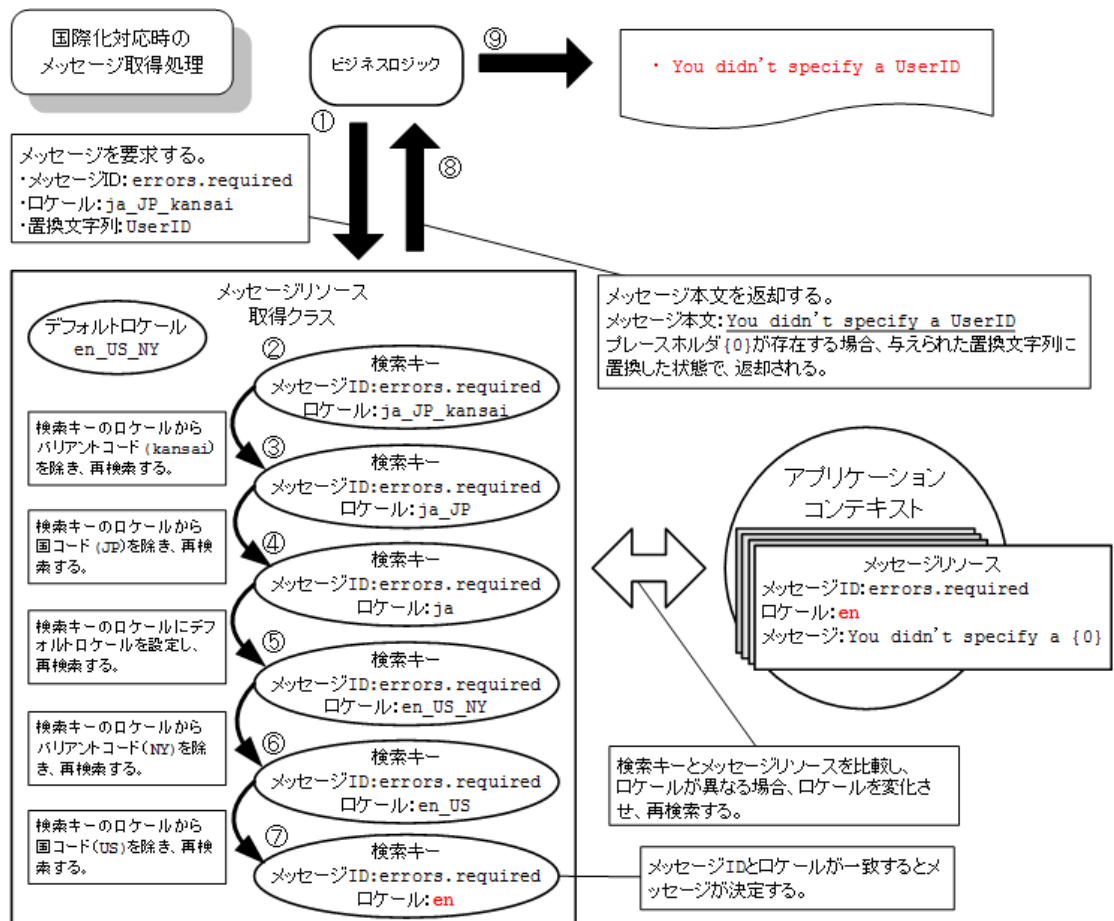
messageSource を指定する。
優先して検索されるメッセージリソースとなる。

次に参照される

第2のメッセージリソースを指定する。
上記 messageSource 内にメッセージが存在しなかった場合、ここで指定したメッセージリソース内を検索する。

◆ メッセージの国際化対応

● 国際化対応時のメッセージ決定ロジック



- ① ビジネスロジックからリソースメッセージを要求する
取得したいメッセージのメッセージ ID およびロケール文字列、(必要に応じて)プレースホルダ用の置換文字列を引数に指定して呼び出す。
- ② メッセージリソース取得クラスはメッセージリソースを検索する
メッセージリソース取得クラスは、与えられたメッセージ ID とロケールを検索キーとし、メッセージを検索する。メッセージ ID、ロケールを検索キーにアプリケーションコンテキストに登録されたメッセージリソース内を検索する。なお、ロケールが引数として渡されなかった場合はサーバー側で設定されているデフォルトロケールが設定される。
- ③ ②でメッセージが検索できなかった場合は、検索キーのロケールからバリエーションコード（図では kansai）を除き、再検索する。
- ④ ③でメッセージが検索できなかった場合は、検索キーのロケールから国コード（図では JP）、バリエーションコードを除き、再検索する。

- ⑤ ④でメッセージが検索できなかった場合は、ロケールにデフォルトロケールを設定し、再検索する。
- ⑥ ⑤でメッセージが検索できなかった場合は、デフォルトロケールからバリエーションコード（図では **NY**）を除き、再検索する。
- ⑦ ⑥でメッセージが検索できなかった場合は、デフォルトロケールから国コード（図では **US**）、バリエーションコードを除き、再検索する。
- ⑧ メッセージリソース取得クラスはメッセージを返却する
②～⑦のいずれかで検索したメッセージをビジネスロジックに返却する。メッセージにプレースホルダ（図では {0}）が存在する場合は、メッセージリソース取得クラスのメソッド呼び出し時に引数として渡された置換文字列に置き換えたメッセージを返却する。
- ⑨ ビジネスロジックで取得したメッセージを使用する
取得したメッセージをログに出力する。

- ソフトウェアアーキテクトが行うコーディングポイント

- デフォルトロケールの変更

MessageAccessor や MessageUtil は同期型ジョブ実行機能、または、非同期型ジョブ実行機能が動作している VM のロケールを使用してメッセージを検索するが、データベースメッセージリソースを使用する場合、デフォルトロケールを変更できる。

- ✧ Bean 定義ファイル定義例 (AdminContext.xml / commonContext.xml)

```
<bean id="messageSource"
      class="jp.terasoluna.fw.message.DataSourceMessageSource">
  <property name="dbMessageResourceDAO"
    ref="dbMessageResourceDAO"/>
  <property name="defaultLocale" value="ja_JP"/>
</bean>
```

デフォルトロケールを指定する。

- 国際化対応カラムの有効化

データベースメッセージリソースを使用する場合、ロケールを判断するためのデータベースのカラムを Bean 定義ファイルに設定し、ロケールに対応するカラムからの読み込みを有効にする必要がある。ロケールに対応するカラムは以下の3つがある。

- ・言語コードカラム
- ・国コードカラム
- ・バリエーションコードカラム

設定の優先順位は、言語コードカラムが一番高く、国コードカラム、バリエーションコードカラムの順に低くなる。言語コードカラムを指定せずに、国コードカラムやバリエーションコードカラムを指定しても無効となる。

これらのカラムのうち、言語コードカラムの指定によってデータベースに登録されたメッセージの認識が以下のように変化する。

- ・言語コードカラムを指定しない場合は、すべてのメッセージがデフォルトロケールとして認識される。(defaultLocale プロパティを指定した場合はその値となる)
- ・言語コードカラムを指定した場合は、言語コードカラムに指定したとおりに認識される。

言語コードカラムを指定し、言語コードカラムに null や空文字のメッセージをデータベースに登録した場合は、そのメッセージはジョブから参照されない。 null や空文字で登録したメッセージがデフォルトロケールとして認識されるわけではない点に注意すること。

以下のプロパティで設定されていない値はデフォルトの値が使用される。設定する項目は以下のとおり。

| プロパティ名 | デフォルト値 | 概要 | 備考 |
|----------------|--------|-----------------------|------------|
| languageColumn | null | 言語コードを格納する カラム名 | 国際化対応時のみ設定 |
| countryColumn | null | 国コードを格納するカ ラム名 | 国際化対応時のみ設定 |
| variantColumn | null | バリエントコードを格 納するカラム名 | 国際化対応時のみ設定 |

メッセージ取得 SQL 文のフォーマットは以下のとおり。

SELECT メッセージコード, (言語コード), (国コード), (バリエントコード), メッセージ本文 **FROM** テーブル名 **FROM** テーブル名

() 内は設定した値のみが有効になる。デフォルトでは無効になっており、カラム名を設定すると有効になる。

◇ Bean 定義ファイル定義例 (AdminContext.xml / commonContext.xml)

```
<bean id=DBMessageResourceDAO
  class="jp.terasoluna.fw.message.DBMessageResourceDAOImpl">
  <property name="dataSource" ref bean="messageDataSource"/>
  <property name="tableName" value="DBMESSAGES"/>
  <property name="codeColumn" value="BANGOU"/>
  <property name="languageColumn" value="GENGO"/>
  <property name="countryColumn" value="KUNI"/>
  <property name="variantColumn" value="HOUGEN"/>
  <property name="messageColumn" value="HONBUN"/>
</bean>
```

国際化対応する場合のみ設定。
言語コードのカラム名を指定する。

国際化対応する場合のみ設定。
国コードのカラム名を指定する。

国際化対応する場合のみ設定。
バリエントコードのカラム名を指定する。

データベースのテーブル名およびカラム名は以下のような設定となる。

テーブル名 = DBMESSAGES

メッセージコードを格納するカラム名 = BANGOU

メッセージの言語コードを格納するカラム名 = GENGO

メッセージの国コードを格納するカラム名 = KUNI

メッセージのバリエントコードを格納するカラム名 = HOUGEN

メッセージ本文を格納するカラム名 = HONBUN

検索 SQL 文は以下のとおり。

SELECT BANGOU,GENGO,KUNI,HOUGEN,HONBUN **FROM** DBMESSAGES