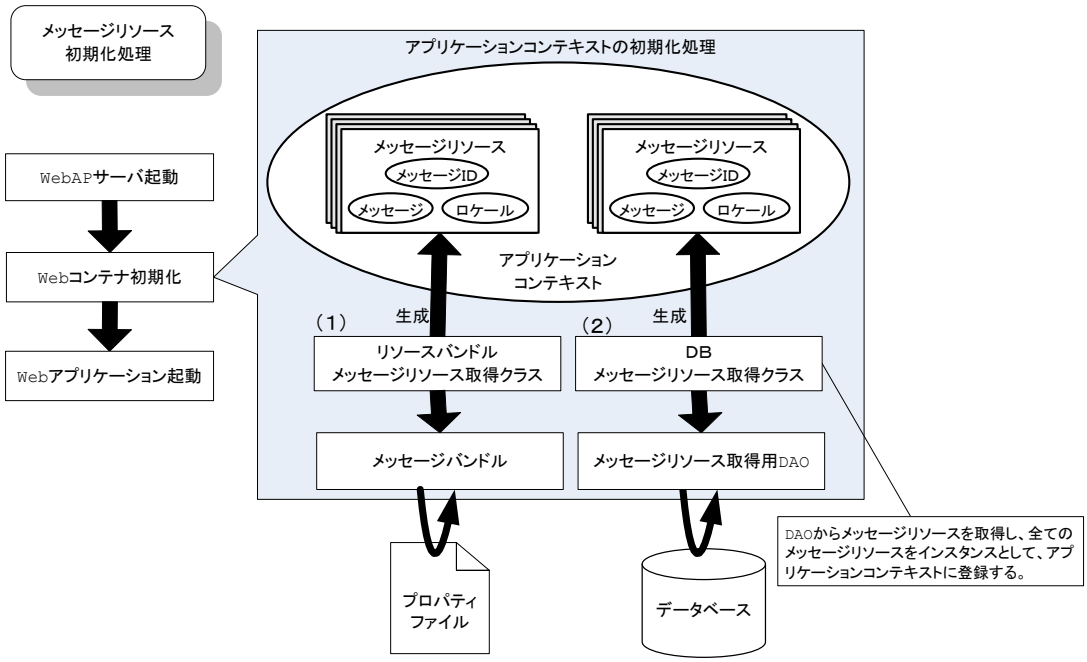
CE-01 メッセージ管理機能

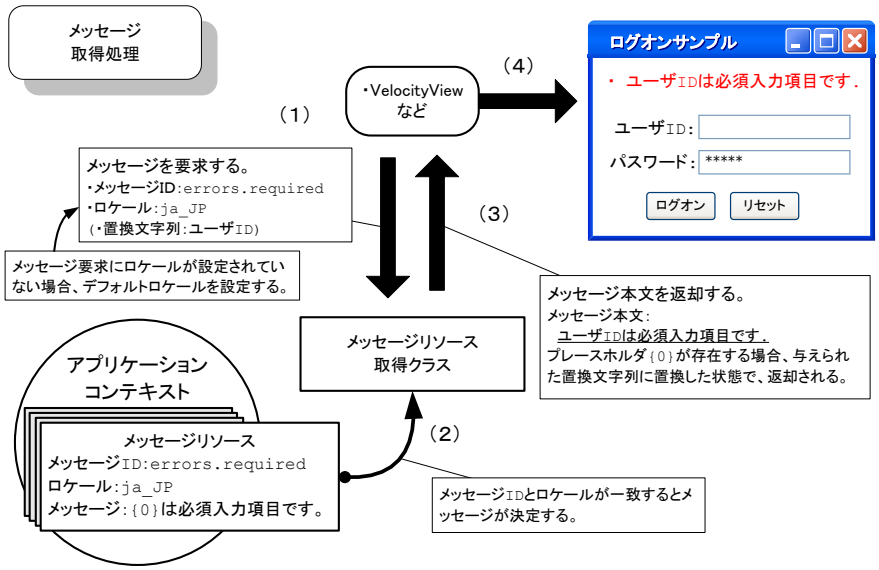
■ 概要

◆ 機能概要

- アプリケーションユーザなどに対して表示する文字列(メッセージリソース)を、定義できる。
- メッセージリソースは、プロパティファイルやデータベース内のメッセージ定義テーブルに定義することができる。
- 国際化に対応しており、ユーザのロケールに応じたメッセージを定義できる。

◆ 概念図





◆ 解説

● メッセージリソース初期化処理

(1) リソースバンドルメッセージリソース
リソースバンドルを利用してプロパティファイルを読み込み、アプリケーションコンテキストに保持する。

(2) ＤＢメッセージソース
メッセージリソース取得用 DAO を用いて、データベース内に定義された全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} の組としてアプリケーションコンテキストに保持する。

● メッセージリソース取得処理

(1) 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を引数に指定して呼び出す。

(2) メッセージリソースを検索する
アプリケーションコンテキストのメッセージリソース内を検索して該当するメッセージを取得する。

(3) メッセージを返却する
取得したメッセージを要求元に返却する。

(4) メッセージを利用する
View やビジネスロジックなどでメッセージを利用する。

※メッセージリソース中にプレースホルダ(概念図中の「{0}」)を定義しておくことで、引数に指定した文字列を動的に埋め込んだメッセージを取り出すことができる。

■ 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）
  以下のように “messageSource”という識別子の Bean を準備することで、この機能を利用できる。
  - Bean 定義ファイルサンプル（applicationContext.xml）

```
<bean id="messageSource"
   class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basenames"
            value="application-messages, system-messages"/>
</bean>
```

messageSource を指定する。

読み込むプロパティファイルをカンマ区切りで列挙する。ファイル名の”.properties”は省略する。

プロパティファイルはクラスパス上に配置する。

定義するプロパティファイルが多い場合は、下記のようにリストの形で指定することもできる。

- Bean 定義ファイルサンプル（applicationContext.xml）

```
<bean id="messageSource"
   class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basenames">
    <list>
      <value> application-messages </value>
      <value>system-messages</value>
    </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（ＤＢメッセージ)
  以下のように “messageSource”という識別子の Bean を準備することで、この機能を利用できる。
  - メッセージリソース取得 Bean の定義
    TERASOLUNA Server Framework for Java が提供している DataSourceMessageSource クラスを指定し、DAO(後述)を DI する。BeanID は”messageSource”である必要がある。
    - Bean 定義ファイルサンプル（applicationContext.xml）

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

messageSource を指定する。
DataSourceMessageSource を指定する。
利用するメッセージ取得用の DAO を指定する。

  - 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。TERASOLUNA Server Framework for Java ではこのインタフェースのデフォルト実装として DBMessageRecourceDAOImpl クラスを提供している。
    - Bean 定義ファイルサンプル(dataAccessContext-local.xml)

```
<bean id="dbMessageResourceDAO"
class="jp.terasoluna.fw.message.DBMessageResourceDAOImpl">
  <property name="dataSource" ref bean="dataSource"/>
</bean>
<bean id="dataSource" ……>
</bean>
```

DBMessageResourceDAOImpl を指定する。
dataSource を指定する。

  - データソースの定義
    『CB-01 データベースアクセス機能』を参照して設定する。
  - メッセージ文字列の定義

メッセージ文字列はデータベース中の以下のテーブルに格納しておく:
・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
```

テーブルスキーマを自由に定義することも可能である。後述「メッセージリソースのテーブルスキーマをデフォルト値から変更する」を参照されたい。

- 業務開発者が行うコーディングポイント
  - メッセージの取得方法
    - 例外メッセージの取得（Rich 版の場合）

      Velocity ビューを利用することで、例外メッセージの取得をコーディングレスで行える。詳細は『RB-02 レスポンスデータ生成機能』の説明書を参照のこと。

    - その他、正常系メッセージなどの取得

      DI コンテナで管理するクラスが、上記、『例外ハンドリング機能』を利用せずにメッセージを取得する場合は、以下のクラスを利用する。

      org.springframework.context.support ApplicationObjectSupport

      上記クラスで定義されている MessageSourceAccessor 内の getMessage メソッドを使用する。詳細については MessageSourceAccessor の JavaDoc を参照のこと。各ビジネスロジックが直接、getMessage メソッドを使用することはせず、メッセージ取得用クラスなどの業務共通クラスから利用することを推奨する。

- メッセージ使用例
  - メッセージ取得クラスインタフェースサンプル

```
public interface MessageAccessor {
    //メッセージをそのまま取り出す場合
    public String getMessage(String code, Object[] args);
    …省略…
}
```

業務共通機能担当者が作成する。

ビジネスロジック開発者が使用するメソッドを規定する。

  - メッセージ取得クラス実装クラスサンプル

```
public class MessageAccessorImpl extends ApplicationObjectSupport implements
MessageAccessor {
    //メッセージをそのまま取り出す場合
    public String getMessage(String code, Object[] args) {
        return getMessageSourceAccessor().getMessage(code, args);
    }
    …省略…
}
```

業務共通機能担当者が作成する。

ApplicationObjectSupport クラス内の MessageSourceAccessor オブジェクトの getMessage メソッドを利用する。

➢ ビジネスロジックサンプル

```
public class SampleBLogic implements BLogic {
    //メッセージ出力クラス用setter
    MessageAccessor msgAcc = null;
    public void setMsgAcc(MessageAccessor msgAcc){
        this. msgAcc = msgAcc;
    }

    //ビジネスロジック
    public ResultBean sampleLogic(String teamId) throws Exception {
        ResultBean result = new ResultBean();

        String outPutMessage = null;
        outPutMessage =  msgAcc.getMessage("welcome",teamId);

        result.setResult(outPutMessage,……(省略)……);
        return result;
    }
}
```

ビジネスロジック開発担当者が作成する。

上記、メッセージ出力クラスを DI するためのセッターを記述する。

メッセージ出力クラスからメッセージ取得メソッドを利用する。

➢ Bean 定義ファイルサンプル(applicationContext.xml)

```
<!–メッセージ出力クラス →
<bean id="msgAcc" class="jp.terasoluna.fw.message.MessageAccessorImpl"/>

<!-- 業務ロジッククラス -->
<bean id="sampleBLogic" class="jp.terasoluna.sample.service.impl.SampleBlogic">
  <property name="messageAccessor" ref="msgAcc"/>
</bean>
```

業務共通機能担当者が記述する。

ビジネスロジック開発担当者が記述する。

➢ メッセージリソースの再定義方法

Web アプリケーション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することができる。再定義には以下のメソッドを使用する。なお、クラスタ環境化では、クラスタごとに再定義する必要があるので注意されたい。以下のメソッドを使用する。

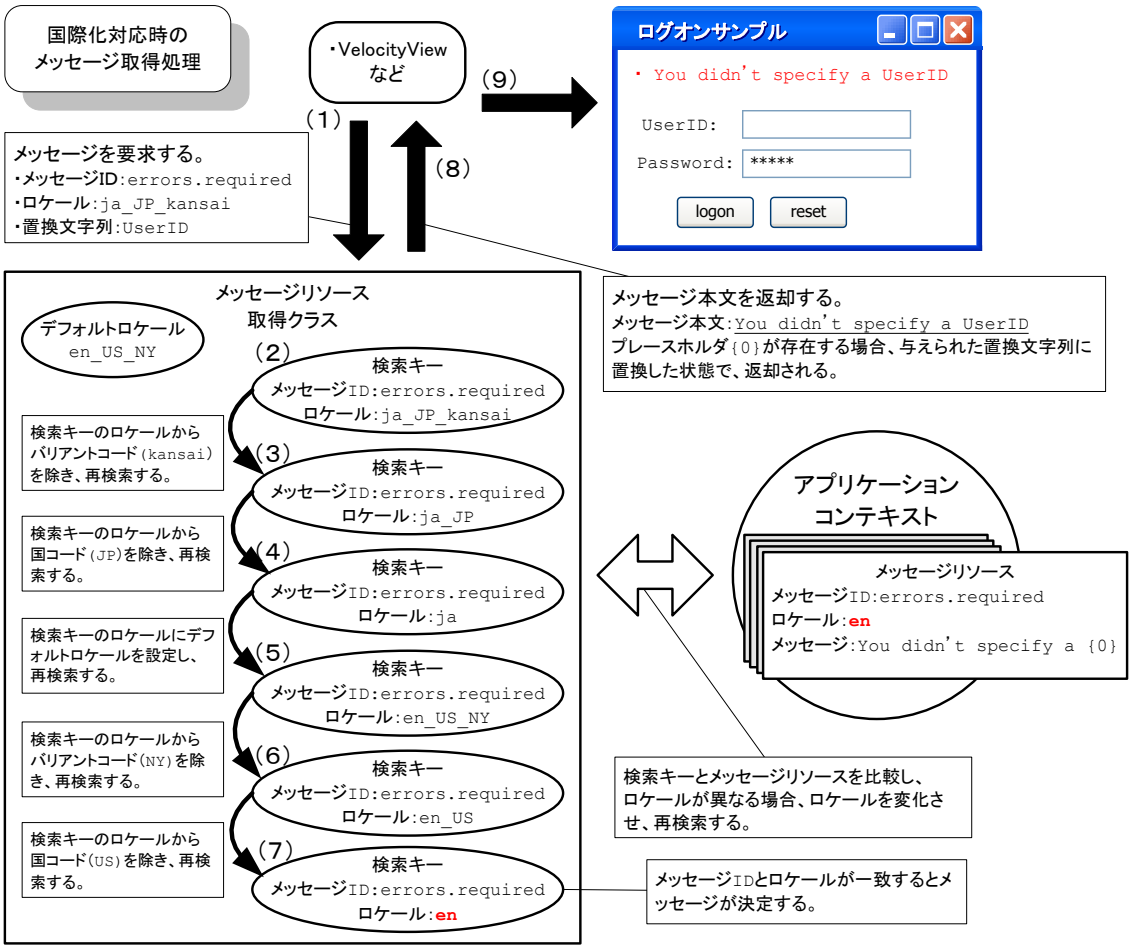
jp.terasoluna.fw.message.DataSourceMessageSource クラスの reloadDataSourceMessage メソッド

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利用することを推奨する。

■ 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

- 概要図



- 解説

(1) 取得したいメッセージのメッセージ ID、ロケールを検索キーとして、また必要な場合は置換文字列を引数として渡す。なお、ロケールはクライアントのリクエストから取得する。取得出来ない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。

(2) 与えられたメッセージ ID とロケールを検索キーとし、メッセージの検索をする。

(3) (2)でメッセージが決定されず、検索キーのロケールにバリアントコードがある場合、バリアントコードを除き、再検索する。

(4) (3)でメッセージが決定されず、検索キーのロケールに国コードがある場合、国コードを除き、再検索する。

(5) (4)でメッセージが決定されない場合は、検索キーの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)でメッセージが決定されず、検索キーのロケールにバリアントコードがある場合、

バリアントコードを除き、再検索する。
(7) (6)でメッセージが決定されず、検索キーのロケールに国コードがある場合、国コードを除き、再検索する。
(8) 決定されたメッセージを返却する。決定されたメッセージにプレースホルダ（概念図では｛0｝）が存在する場合（MessageFormat 型）は引数として渡された置換文字列に置き換える。
(9) 取得したメッセージを用い、画面に表示する。

- 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設定
    メッセージリクエストにロケールが設定されていない場合、及びメッセージリソース内にメッセージリクエストで要求されたロケールが見つからない場合に使用される。設定しない場合はデフォルトロケールの初期設定（サーバー側 VM のロケール）が使用される。
    - Bean 定義ファイルサンプル（applicationContext.xml）

```
<bean id="messageSource"
    class="jp.terasoluna.fw.message.DataSourceMessageSource">
  <property name="dbMessageResourceDAO"
     ref bean="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
    データベースのロケールに対応するカラムからの読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
・バリアントコードカラム

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、バリアント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アント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。

注意点としては、**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**、そのメッセージは**アプリケーションから参照されない**点である。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。

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通り。

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	バリア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通り。

SELECT メッセージコード, (言語コード), (国コード), (バリアントコード), メッセージ本体 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 定義ファイルサンプル（dataAccessContext-local.xml）

```
<bean id=DBMessageResourceDAO
    class="jp.terasoluna.fw.message.DBMessageResourceDAOImpl">
  <property name="dataSource" ref bean="dataSource"/>
  <property name=tableName value="DBMESSAGES"/>
  <property name=codeColumn value="BANGOU"/>
  <property name=languageColumn value="GENGO"/>
  <property name=countryColumn value="KUNI</value>
  <property name=variantColumn value="HOUGEN</value>
  <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。
バリアントコードのカラム名を指定す

DBのテーブル名及びカラム名は以下の様な設定となる。
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
メッセージのバリア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文は以下の通り。
SELECT BANGOU,GENGO,KUNI,HOUGEN,HONBUN FROM DBMESSAGES

■ リファレンス

◆ 構成クラス

	クラス名	概要
1	DataSourceMessageSource	メッセージを生成、発行するクラス
2	DBMessageResourceDAOImpl	DB よりメッセージリソースを抽出する DBMessageResourceDAO の実装クラス
3	MessageSource	メッセージ取得のためのメソッドを規定したインタフェイスクラス

◆ 拡張ポイント

- なし

■ 関連機能

- なし

■ 使用例

- TERASOLUNA Server Framework for Java (Web 版) 機能網羅サンプル
  - WG-01 **メッセージ管理機能**

- TERASOLUNA Server Framework for Java (Rich 版) 機能網羅サンプル
  - UC110 DB **メッセージ管理**
    - jp.terasoluna.rich.functionsample.dbmessage.*

■ 備考

◆ メッセージリソースのテーブルスキーマをデフォルト値から変更する

- テーブル名、カラム名をプロジェクト側で独自に指定する場合
  テーブル名及び各カラム名のすべてもしくは一部を設定することでデータベースのテーブル名及びカラム名を自由に変更できる。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通り。

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

メッセージ取得 SQL 文の SELECT 節のフォーマットは以下の通り。

SELECT メッセージコード, メッセージ本体

なお、この設定では国際化に未対応となる。国際化対応が必要な場合は、前述の『メッセージの国際化対応』の項目を参照のこと。

例）データベースのテーブル名及びカラム名を以下の様にする場合
・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

➢ Bean 定義ファイルサンプル（dataAccessContext-local.xml）

```
<bean id="dBMessageResourceDAO"
class="jp.terasoluna.fw.message.DBMessageResourceDAOImpl">
  <property name="dataSource" ref bean="dataSource"/>
  <property name="tableName" value="DBMESSAGES"/>
  <property name="codeColumn" value="BANGOU"/>
  <property name="messageColumn" value="HONBUN"/>
</bean>
```

テーブル名を指定する。

メッセージコードのカラム名を指定する。

メッセージ本文のカラム名を指定する。

メッセージ取得 SQL 文は以下の通り。
SELECT BANGOU,HONBUN FROM DBMESSAGES

- 上記『テーブル名、カラム名をプロジェクト側で独自に指定する場合』に加え、
  プロジェクト独自の SQL 文を設定する場合
  findMessageSql プロパティで独自の SQL 文を設定できる。設定する SQL 文には、

codeColumn プロパティおよび messageColumn で指定したカラムが必要となる。
設定する項目は以下の通り。

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	-	メッセージを取得する SQL 文

例）メッセージ取得 SQL 文を『SELECT BANGOU,HONBUN FROM DBMESSAGE WHERE CATEGORY='TERASOLUNA'』とする場合。
・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

➢ Bean 定義ファイルサンプル（dataAccessContext-local.xml）

```
<bean id ="dBMessageResourceDAO"
    class="jp.terasoluna.fw.message.DBMessageResourceDAOImpl">
  <property name="dataSource" ref bean="dataSource"/>
  <property name="tableName" value="DBMESSAGES"/>
  <property name="codeColumn" value="BANGOU"/>
  <property name="messageColumn" value="HONBUN"/>
  <property name="findMessageSql"
    value=
    "SELECT BANGOU,HONBUN FROM DBMESSAGE WHERE CATEGORY='TERASOLUNA'"
  />
</bean>
```

テーブル名を指定する。

メッセージコードのカラム名を指定する。

メッセージ本文のカラム名を指定する。

検索 SQL 文を指定する。

◆ 第2メッセージリソースの使用

メッセージリソースを追加できる。追加したメッセージリソースは前述で"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でメッセージが決定できない場合に利用される。以下のように “parentMessageSource プロパティ”に別のメッセージリソースへの参照を指定することで、この機能を利用できる。

✧ 第２メッセージリソース取得 Bean の定義

利用したいメッセージリソース取得クラスを指定する。BeanID は”messageSource”とは別の名前を付与する必要がある。

AbstractMessageSource の継承クラスであれば、この“parentMessageSource プロパティ”を利用できるので、さらに第 3、4 とリンクすることが可能である。

➢ Bean 定義ファイルサンプル（applicationContext.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。