SV-03 サーバエラーハンドリング機能

■ 概要

サーバ AP で発生した以下のエラーを集約的に例外処理する機能を提供する。

- 入力値検証エラー
  - 必須入力チェックや形式チェックなど入力 DTO のみでチェックできるエラー。
    クライアント AP からの再要求により復帰可能。
- 業務エラー
  - DB との突き合わせや複雑なビジネスルールによるチェックなど、ビジネスロジック処理時に発生するエラー。クライアント AP から再要求により復帰可能。
- システムエラー
  - DB やネットワークのエラーなど、システム異常や AP バグ等による復帰不可能なエラー。

TERASOLUNA フレームワークは通信基盤として WCF(Windows Comunication Foundation)を採用している。サーバ AP で発生したエラーをクライアントへエラー電文として返却する際、SOAP Fault をエラー電文として取り扱うこととする。WCF では、FaultContract を定義することで、<detail>要素に格納することができる。

「CL-05 クライアントエラーハンドリング機能」は、SOAP Fault の<detail>要素に、ある特定の形式でエラー情報が格納されていた場合、集約的に例外を処理する。本機能では、クライアント AP で集約的に例外処理可能なエラー情報を格納する FaultContract として TerasolunaSoapFault クラスを提供する。

また、WCF サービスで発生した例外を集約的に処理する仕組みとして System.ServiceModel.Dispatcher.IErrorHandler インタフェースが提供されている。本機能は IErrorHandler の標準実装クラスとして DefaultErrorHandler クラスを提供する。DefaultErrorHandler は WCF の例外処理の仕組みを、TERASOLUNA フレームワークが提供する集約例外処理の仕組み（IExceptionHandler インタフェース）に委譲する。

サーバで集約例外処理を実施する IExceptionHandler 実装クラスとして標準で以下の 2 つのクラスを提供し、エラーの種類に応じてクライアント AP へ SOAP Fault を送信するとともに、ログの出力といった後処理を実施する。

- DefaultResponseExceptionHandler クラス
  - エラーの種類に応じて SOAPFault を作成する。
- DefaultLogExceptionHandler クラス
  - エラーの種類に応じてログを出力する。

- サーバ入力値検証エラー
  サーバ側の入力値検証は、「SV-02 サーバ入力値検証機能」により実施する。
  「SV-02 サーバ入力値検証機能」では、入力値検証エラーになった場合には、FaultException<ValidationFault>クラスをスローする。
  DefaultErrorHandler クラスは DefaultResponseExceptionHandler クラスに処理を委譲する。DefaultResponseExceptionHandler は、FaultException<ValidationFault>の Detail プロパティ(ValidationFault オブジェクト)を解析し、サーバ入力値検証エラー用の SOAP Fault を作成する。

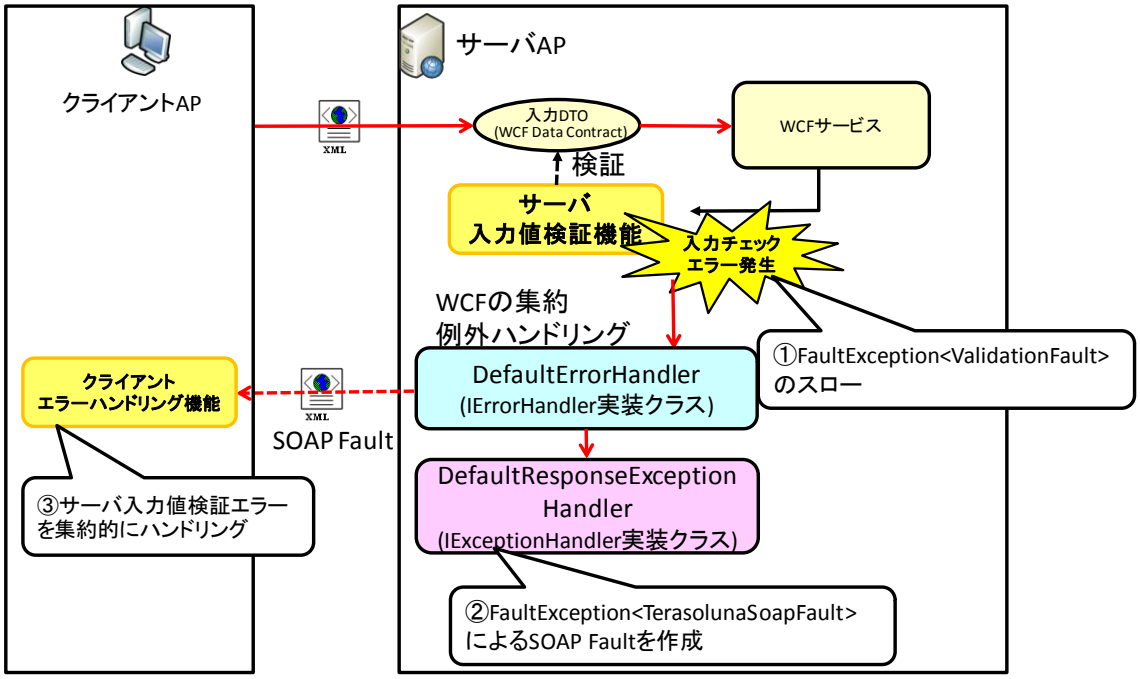

図 1 サーバ入力値検証エラー時のエラーハンドリングの動作概念図

- サーバ業務エラー

  NET には検査例外の機構がないため、業務エラーはメソッドの戻り値で扱うことが一般的である。TERASOLUNA フレームワークでは、ビジネスロジッククラスを POCO(Plain Old CLR Object)で実装でき、開発者にとって柔軟な開発が可能なアーキテクチャを採用しているため、戻り値だけで業務エラーかどうかをフレームワークが一律判定することは困難である。

  そこで、TERASOLUNA フレームワークでは、ビジネスロジッククラスの中で業務エラーを表す例外(BizLogicException)をスローすることで集約的に業務エラーを処理する機能を提供する。ビジネスロジック内で BizLogicException がスローされると、DefaultErrorHandler クラスは、DefaultResponseExceptionHandler クラスに処理を委譲する。

  DefaultResponseExceptionHandler は、BizLogicException のエラー情報をもとにサーバ業務エラー用の SOAP Fault を生成する。

  また、DefaultLogExceptionHandler に処理を委譲し、BizLogicException のエラー情報をもとに Warn ログを出力する。

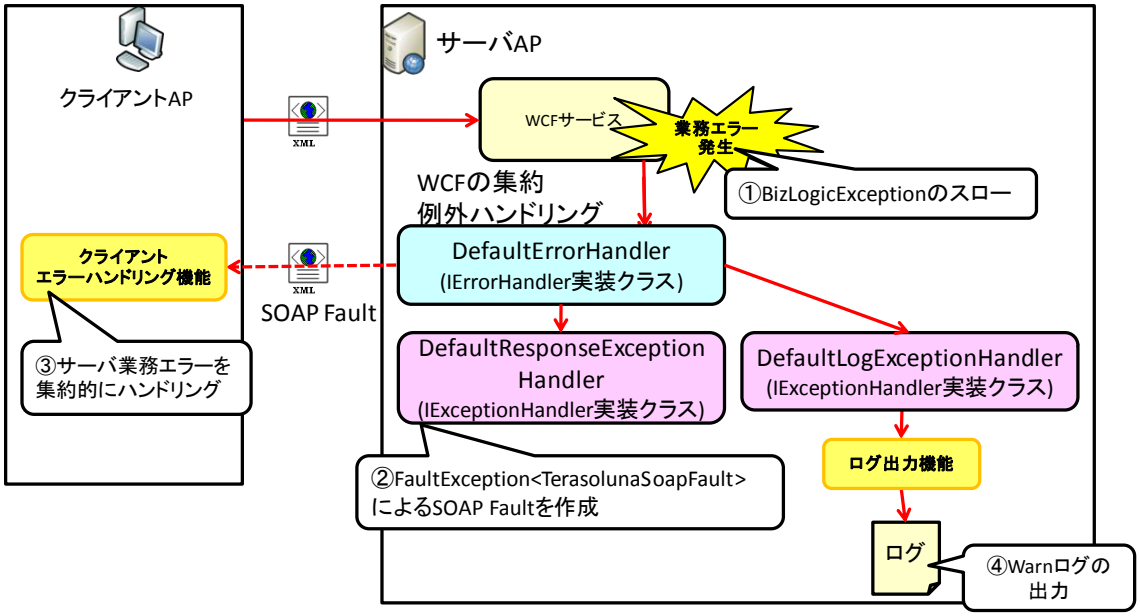

図 2 サーバ業務エラー時のエラーハンドリングの動作概念図

- サーバシステムエラー
  前述の業務エラー用の例外クラス(BizLogicException)を除き、.NET で発生する例外(System.Exception 継承クラス)は、全てシステムエラーとして扱う。
  例外がスローされると、DefaultErrorHandler クラスは、DefaultResponseExceptionHandler クラスに処理を委譲する。
  DefaultResponseExceptionHandler は、サーバシステムエラー用の SOAP Fault を返却する。また、DefaultLogExceptionHandler に処理を委譲し、例外情報をもとに Error ログを出力する。

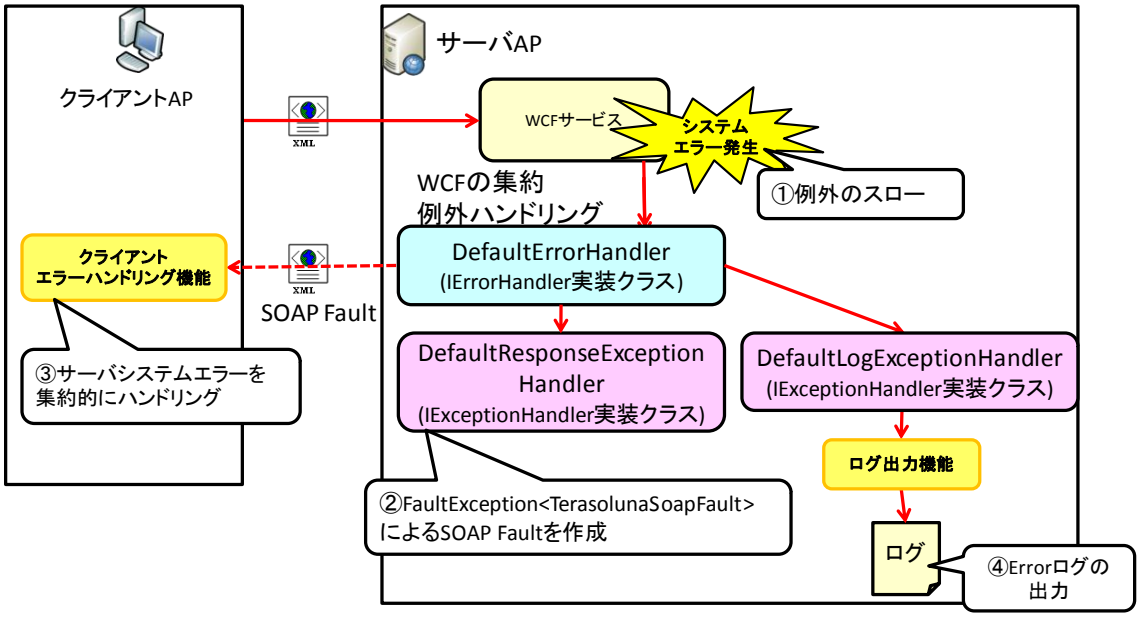

図 3 サーバシステムエラー時のエラーハンドリングの動作概念図

■ 使用方法

◆ 業務エラー用の例外(BizLogicException)の作成

ビジネスロジックで業務エラーを返却する場合、
BizLogicException をスローすることで、フレームワークにより集約的に例外を処理することができる。以下に、通常使用する BizLogicException クラスのコンストラクタを示す。

表 1 BizLogicException クラスのコンストラクタ

項番	コンストラクタ	引数
1	public BizLogicException (string errorType, string bizLogicErrorType, IList<ErrorInfo> errors)	第１引数： エラー種別を表す任意の文字列。 ErrorType プロパティと対応。 第２引数： 業務エラーの分類を表す任意の文字列。 BizLogicErrorType プロパティと対応。 第３引数： エラー要因の詳細な情報(ErrorInfo)のリスト。 Errors プロパティと対応。 エラー要因が複数あれば ErrorInfo を複数指定する。

第1引数に対応する ErrorType プロパティは、「CL-03 イベント処理実行機能」における
イベント処理結果(EventProcessResult オブジェクト)の文字列として格納される。
BizLogicException.ErrorType の格納値とイベント処理結果の対応関係を以下に示す。

表 2 BizLogicException と EventProceResultType の定数値の対応関係

項番	BizLogicException.ErrorType プロパティの格納値	対応するイベント処理結果 （EventProcResultType の定数値）
1	BizLogicFailure	ServerBizLogicFailure （サーバ AP 側ビジネスロジックで業務エラーが発生）
2	ValidationFailture	ServerValidationFailure （サーバ AP 側で入力値検証エラーが発生）

通常は、BizLogicExceptionErrorType の定数値として「BizLogicFailure」を指定する。
これによりイベント処理結果が「サーバ業務エラー」として処理される。
また、「ValidationFailure」を指定することで、サーバビジネスロジック内で入力値検証エラーを扱うことも可能である。ただし、通常は「SV-02 サーバ入力値検証機能」により入力 DTO の検証を実施するため業務開発者が使用する必要通常はない。
なお、業務エラーの種類によって処理方法を振り分けたい場合などは、追加定義した任意の文字列を格納し「完了処理」フェーズでの処理を拡張することで柔軟に対応できる。

第2引数に対応する BizLogicErrorType は、業務エラーの分類を表す任意の文字列で、

業務エラーをさらに分類したいときなどに、開発者が自由に使用する。
第3引数に対応するの Errors プロパティは、Terasoluna.ErrorHandling.ErrorInfo クラスに格納する。以下に、ErrorInfo のコンストラクタを示す。

項番	コンストラクタ	引数
1	public ErrorInfo(string errorId, IList<string> arguments, string message, object rawData)	第１引数： エラーID。 エラーを特定するためのエラーコード。 第２引数： メッセージの置換文字列 第３引数： メッセージ 第４引数： エラー情報が格納されていた元データ

なお、BizLogicErrorType や Errors は、「CL-03 イベント処理実行機能」におけるイベント処理結果の文字列として格納される。詳細は、「CL-03 イベント処理実行機能」の機能説明書を参照のこと。

以下に、BizLogicException を使った業務エラーの記述例を示す。

```
public void Login(LoginInputDto inputDto)
{
    using (TransactionScope scope = new TransactionScope())
    {
        UserTableAdapter ta = new UserTableAdapter();
        CalcDataSet.UserDataTable userTable = new CalcDataSet.UserDataTable();
        ta.FillByIdAndPassword(userTable, inputDto.UserId, inputDto.Password);
        if (userTable.Count == 0)
        {
            ///IDとパスワードが合致しなかった場合
            ///業務エラーとして、BizLogicExceptionをスロー
            throw new BizLogicException(
                BizLogicExceptionErrorType.BizLogicFailure,
                CalcResource.ERROR_CALC_MSG002,
                new List<ErrorInfo>()
                {
                    new ErrorInfo("ERROR_CALC_MSG003", null,
                        CalcResource.ERROR_CALC_MSG003, null)
                });
        }
    }
}
```

リスト 1　BizLogicException を使った業務エラーの記述例

◆ 集約例外処理の設定

本機能を利用するためには、アーキテクトは、FW 起動構成ファイル(TerasolunaBootstrap.config)に、集約例外処理の設定をする。
IExceptionHandler 実装クラスは、「CM-02 インスタンス管理機能」を利用して生成される。UnityAB の DI 設定により IExceptionHandler の実装クラスを指定することで、集約例外処理の設定をすることができる。
集約例外ハンドリングクラスは、システムで1つあればよいので、<lifetime>タグは「singleton」(ContainerControlledLifetimeManager)を指定する。
また、<type>タグの name 属性で指定した名前と、例外を捕捉するポイントは1対1に対応しており、表 3 の例外捕捉ポイントについて、IExceptionHandler 実装クラスの設定を行う必要がある。

表 3 例外発生ポイントとの対応関係

項番	type の name 属性の値	例外捕捉ポイント	標準で設定する IExceptionHandler クラス
1	ServiceModelErrorHandlerHandleError	DefaultErrorHandler クラスが実装する IErrorHandler.HandleError メソッド。エラーログの記録、システム通知、アプリケーションのシャットダウンなどのエラー関連動作を実装するポイント。	DefaultLogExceptionHandler
2	ServiceModelErrorHandlerProvideFault	DefaultErrorHandler クラスが実装する IErrorHandler.ProvideFault メソッド。クライアントに返されるカスタムのエラーメッセージ作成処理を実装するポイント。	DefaultResponseExceptionHandler

以下に、TerasolunaBootstrap.config の記述例を示す。TERASOLUNA フレームワークが提供するカスタムテンプレートを利用して、当該構成ファイルを生成した場合、下記の設定はデフォルトでなされている。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
…
  <unity>
    <typeAliases>
…
      <typeAlias alias="singleton"
                 type="Microsoft.Practices.Unity.ContainerControlledLifetimeManager,
                       Microsoft.Practices.Unity,
                     Version=1.2.0.0,
                     Culture=neutral,
                     PublicKeyToken=31bf3856ad364e35" />
      <typeAlias alias="IExceptionHandler"
                   type="Terasoluna.ExceptionHandling.IExceptionHandler,
                         Terasoluna" />
    </typeAliases>
    <containers>
      <container>
        <types>
          <!-- 集約例外処理の設定 -->
          <!-- IErrorHandler.HandleErrorメソッドのポリシー -->
          <type type="IExceptionHandler" name="ServiceModelErrorHandlerHandleError"
mapTo="Terasoluna.Server.ServiceModel.ExceptionHandling.DefaultLogExceptionHandler,
                       Terasoluna.Server.ServiceModel" >
            <lifetime type="singleton" />
          </type>
          <!-- IErrorHandler.ProvideFaultメソッドのポリシー -->
          <type type="IExceptionHandler" name="ServiceModelErrorHandlerProvideFault"
mapTo="Terasoluna.Server.ServiceModel.ExceptionHandling.DefaultResponseExceptionHandler,
                       Terasoluna.Server.ServiceModel" >
            <lifetime type="singleton" />
          </type>
      </container>
    </containers>
  </unity>
</configuration>
```

リスト 2　TerasolunaBootstrap.config の記述例

◆ ログの出力設定

DefaultLogExceptionHandler クラスでは、「CM-06 ログ出力機能」を使って発生した例外情報をログに出力する。
TraceSource を使った標準のログ出力機能を利用する場合、アーキテクトは、DefaultLogExceptionHandler クラスが存在するアセンブリのアセンブリ名である「Terasoluna.Server.ServiceModel」をカテゴリ名として設定する必要がある。
以下に、アプリケーション構成ファイル(Web.config)の設定例を示す。
ログ出力の詳細な設定方法は、「CM-06 ログ出力機能」の機能説明書や MSDN のドキュメントを参照のこと。

```
  <system.diagnostics>
    ・・・
   <sources>
        <!-- Terasoluna.Server.ServiceModel.dllに存在するクラスのデフォルトログ -->
        <source name="Terasoluna.Server.ServiceModel" switchName="FrameworkLogSwitch"
          switchType="System.Diagnostics.SourceSwitch">
            <listeners>
                <add name="XMLLogFile"/>
                <add name="ConsoleLog"/>
                <remove name="Default"/>
            </listeners>
        </source>
        ・・・
    </sources>
      <!-- ログスイッチの設定 -->
    <switches>
      <add name="FrameworkLogSwitch" value="Warning" />
      <add name="DefaultLogSwitch" value="All" />
    </switches>
    ・・・
  </system.diagnostics>
</configuration>
```

リスト 3　アプリケーション構成ファイル（Web.config）のログの設定例

■ 内部構成

◆ 構成クラス

本機能を構成するクラスを以下に示す。

表 4　構成クラス一覧

項番	クラス名	説明
Terasoluna.Server.ServiceModel 名前空間		
1	DefaultErrorHandler	WCF の集約例外ハンドラ(IErrorHandler)クラス。TERASOLUNA フレームワークの集約例外ハンドラ(IExceptionHandler)へ委譲する。
Terasoluna.Server.ServiceModel.ExceptionHandlering 名前空間		
2	DefaultResponseExceptionHandler	FaultException<TerasolunaSoapFault> による SOAPFault を生成する IExceptionHandler 実装クラス。
3	DefaultLogExceptionHandler	エラーの種類に応じてログを出力する IExceptionHandler 実装クラス。

■ 拡張ポイント

集約例外処理を変更したい場合は、IExceptionHandler インタフェースを実装し、TerasolunaBootstrap.config の設定を変更する。
以下に、実装例を示す。

```
public class SampleExceptionHandler：IExceptionHandler
{
    private static readonly ILogger _log = LogManager.GetLogger();
    // 例外に対する後処理を実装
    public bool HandleException(Exception exceptionToHandle, string name)
    {
        if (_log.IsErrorEnabled)
        {
            _log.Error(exceptionToHandle.Message, exceptionToHandle);
        }
        return true;
    }
}
```

リスト 4 IExceptionHandler の実装例

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
・・・
  <unity>
    <containers>
      <container>
        <types>
          <!-- 例外ハンドリング処理の設定 -->
          <!-- IErrorHandler.HandleErrorメソッドのポリシー -->
          <type type="IExceptionHandler" name="IErrorHandlerHandleError"
      mapTo="CalcWcfBusinessApplication.ExceptionHandling.SampleExceptionHandler,
                        CalcWcfBusinessApplication " >
            <lifetime type="singleton" />
          </type>
・・・
        </types>
        <extensions>
        </extensions>
      </container>
    </containers>
  </unity>
</configuration>
```

リスト 5 TerasolunaBootstrap.config の設定例

■ 関連機能

- CM-02 インスタンス管理機能
- SV-01 WCF サービス管理機能
- CM-06 ログ出力機能
- SV-02 サーバ入力値検証機能