SV-01 WCF サービス管理機能

■ 概要

本機能は、WCF(Windows Communication Foundation)を機能拡張し、WCF サービスクラスのインスタンス生成を管理することで、WCF サービスアプリケーションの開発を支援する機能である。

- WCF サービスのインスタンス管理とエンドポイント設定の簡略化
  大規模な WCF サービスを構築する場合、Visual Studio の「WCF サービスライブラリ」プロジェクト上で WCF サービスクラスを実装し、「WCF サービスアプリケーション」プロジェクトを使って WCF サービスを IIS 上にホスティングするといった開発スタイルが想定される。
  このような場合、「WCF サービスライブラリ」プロジェクトで WCF サービスクラスを作成すると、エンドポイントの設定は、Visual Studio により、そのプロジェクトのアプリケーション構成ファイル（App.config）に自動で生成される。このため、開発者は各「WCF サービスライブラリ」プロジェクトに生成されたエンドポイントの設定を、「WCF サービスアプリケーション」プロジェクト上の Web 構成ファイル（Web.config）にマージする作業が発生する。
  これは、以下の 2 つの点で問題となる。
  - バインディングやビヘイビアの設定などは、システム全体で共通的なポリシーを持たせることが望ましく、個別にエンドポイントの設定をさせると統制が取りづらくなる
  - 分散開発における、業者間での構成ファイルのマージにおける競合の問題

  本機能では、個々のエンドポイントの設定をしていなければシステムで定義した標準（デフォルト）設定を適用し WCF サービスを生成するよう WCF の機能を拡張している。これにより、「WCF サービスアプリケーション」の Web.config の記述を簡略化しマージ作業にかかるコストを削減すると同時に、システム全体の統制を取りやすい設計／製造作業が可能となる。
  本機能を使用する場合、「WCF サービスアプリケーション」プロジェクトには WCF サービスファイル（.svc ファイル）のみを配置する。クライアント AP より.svc ファイルが指し示すエンドポイントアドレスに要求があると、エンドポイントアドレスの URI の仮想パスをもとに、命名規約に従って実行対象の WCF サービスクラスを決定する。
  命名規約は、FW 構成ファイル(TerasolunaFramework.config)で指定したベース名前空間と、仮想パスのディレクトリ名を連結した文字列を名前空間とし、svc ファイルの拡張子を除くファイル名を WCF サービスの実装クラス名とする。また、サービスコントラクトは、WCF サービスの実装クラス自身および実装するインタフェース群を検索し取得する。
  その後、Web.config に記述したデフォルトのバインド設定やビヘイビアの設定を適用し、WCF サービスクラスをホストしたサービスホスト(System.ServiceModel.ServiceHost 継承クラス)を生成して公開する。
  TERASOLUNA フレームワークでは、ServiceHost 継承クラスの標準実装として TerasolunaServiceHost クラスを、サービスホストの作成には、System.ServiceModel.Activation.ServiceHostFactory 継承クラスを提供する。また、典型的な処理パターンに合わせて、デフォルトのバインディングやビヘイビア設定を適用した ServiceHostFactory 継承クラスも併せて提供している。

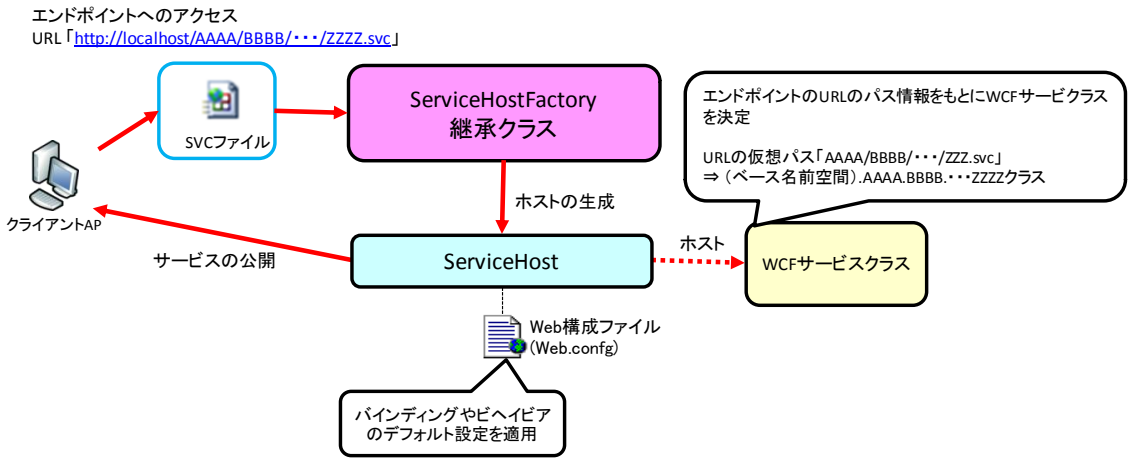

図 1 WCF サービス管理機能の動作概念図

- WCF サービスにおけるフレームワークの起動
  ServiceHostFactory 継承クラスへの初回アクセス時に「CM-01 アプリケーション起動・終了機能」を呼び出すことで TERASOLUNA フレームワークを起動する。

図 2 サーバ AP における TERASOLUNA フレームワークの起動

- WCF サービスのインスタンス管理

本機能では System.ServiceModel.Dispatcher.IInstanceProvider インタフェースを実装し「CM-02 インスタンス管理機能」により、WCF サービス実装クラスのインスタンスを生成するよう WCF を機能拡張している。
これにより、UnityContainer で管理しているオブジェクトを WCF サービス実装クラスに注入（DI）できるようになるので、TERASOLUNA フレームワークが持つ各種機能を利用することが可能になる。

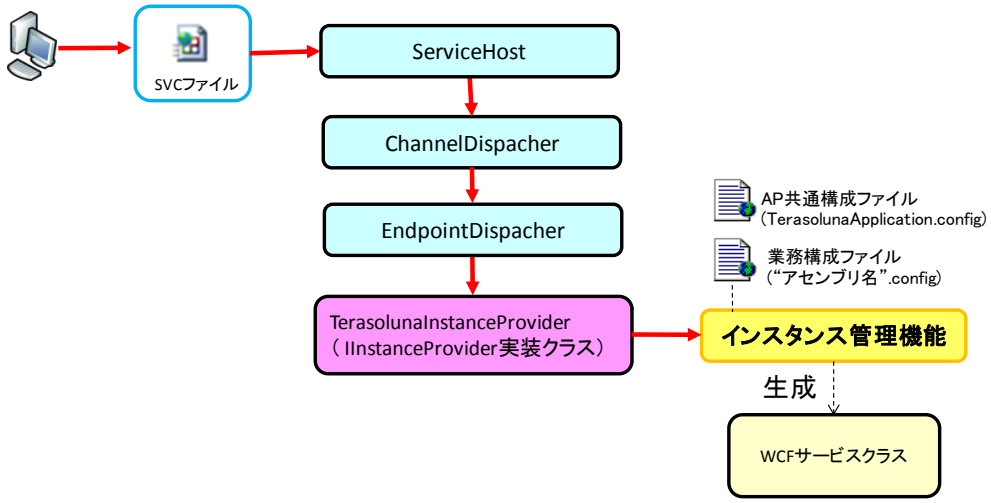

図 3 WCF サービスコントラクトのインスタンス管理

● サーバエラーハンドリング機能の適用

本機能では、WCF サービスホストの生成時に、「CL-05 クライアントエラーハンドリング機能」が提供する WCF の集約例外処理（System.ServiceModel.Dispatcher.IErrorHandler インタフェース実装クラス）の機構が、各 WCF サービスに適用されるような機能を提供している。
サーバエラーハンドリング処理の詳細は、「CL-05 クライアントエラーハンドリング機能」の機能説明書を参照のこと。

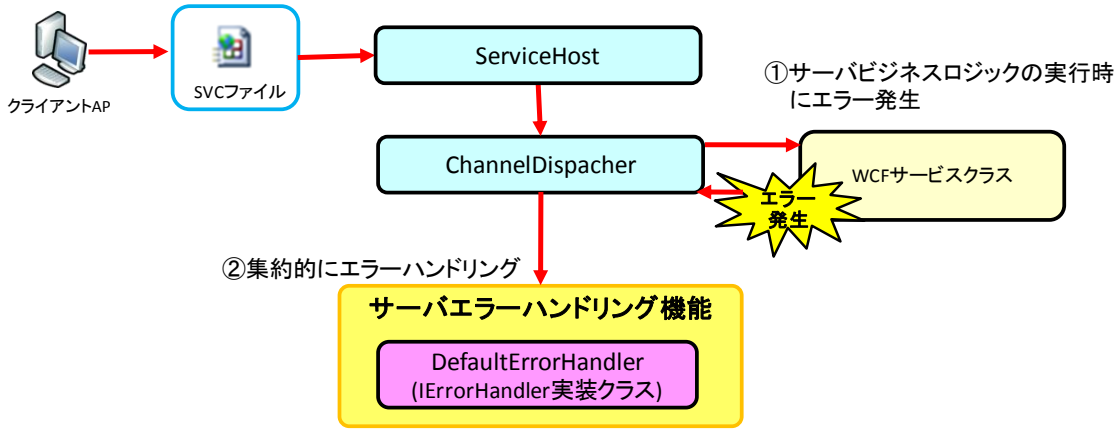

図 4 サーバエラーハンドリング機能の適用

■ 使用方法

◆ WCF サービスファイル（.svc ファイル）の作成

WCF サービスファイル（.svc ファイル）は、TERASOLUNA フレームワークにより提供される「WCF サービスアプリケーションプロジェクト」テンプレートを使ったプロジェクトに配置する。

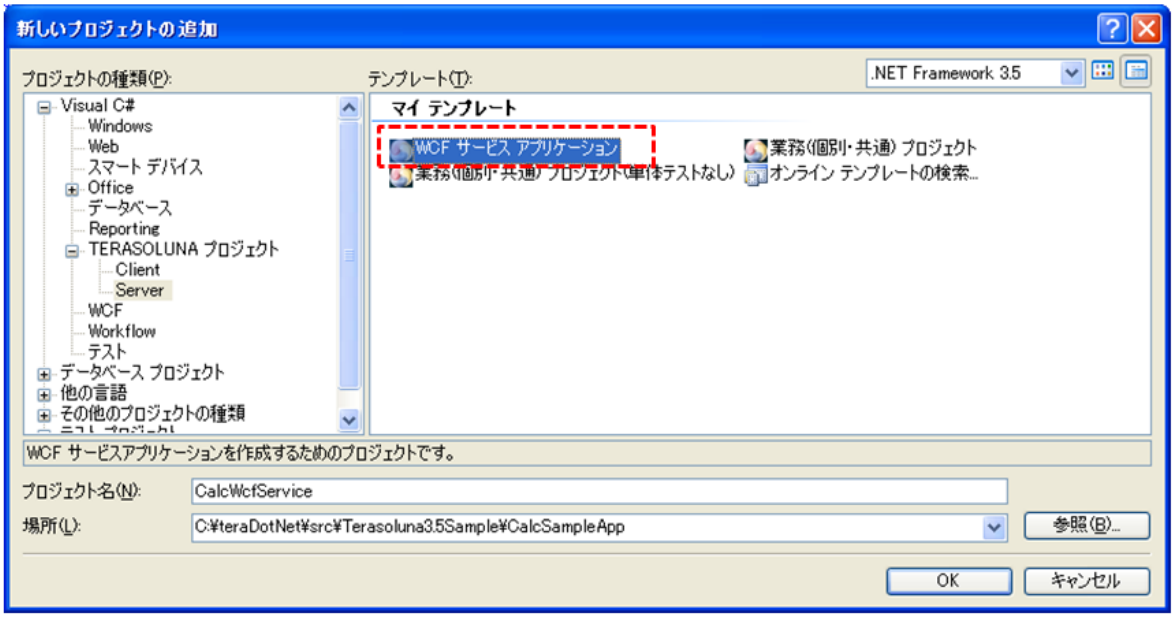

図 5　WCF サービスアプリケーションの追加

このとき、WCF サービスファイルが生成されたフォルダに、対となる WCF サービスクラス（Service Contract）も生成される。WCF サービスクラスは、後述する業務プロジェクトへ移動する。

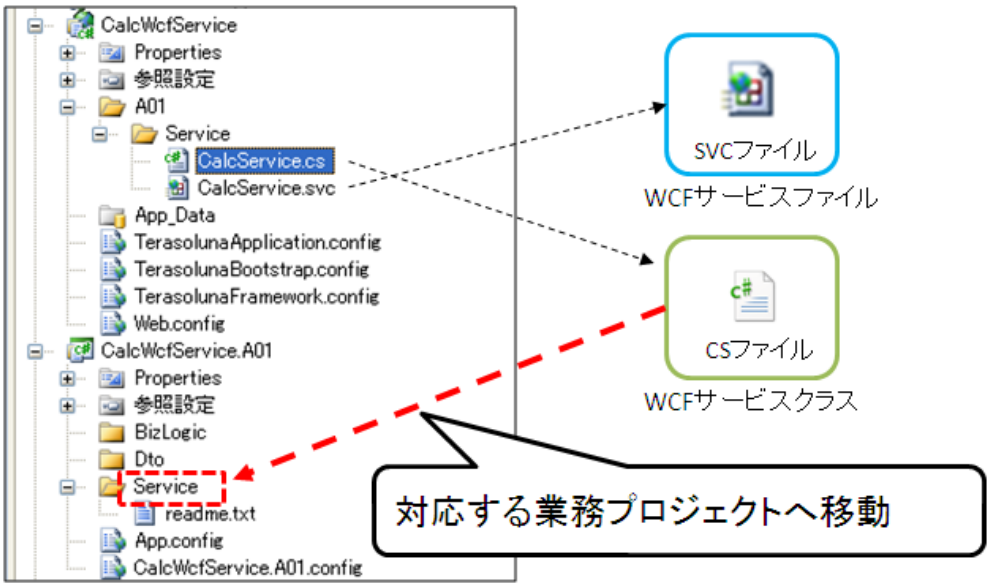

図 6　WCF サービスクラスの移動イメージ

WCF サービスを開発する標準的なパターンでは、WCF サービス(.svc ファイル)は、TERASOLUNA フレームワークにより提供される「WCF サービス(デフォルト)」アイテムテンプレートで作成する(ファイル転送をする場合などは、Mtom 対応の WCF サービスを利用する)。

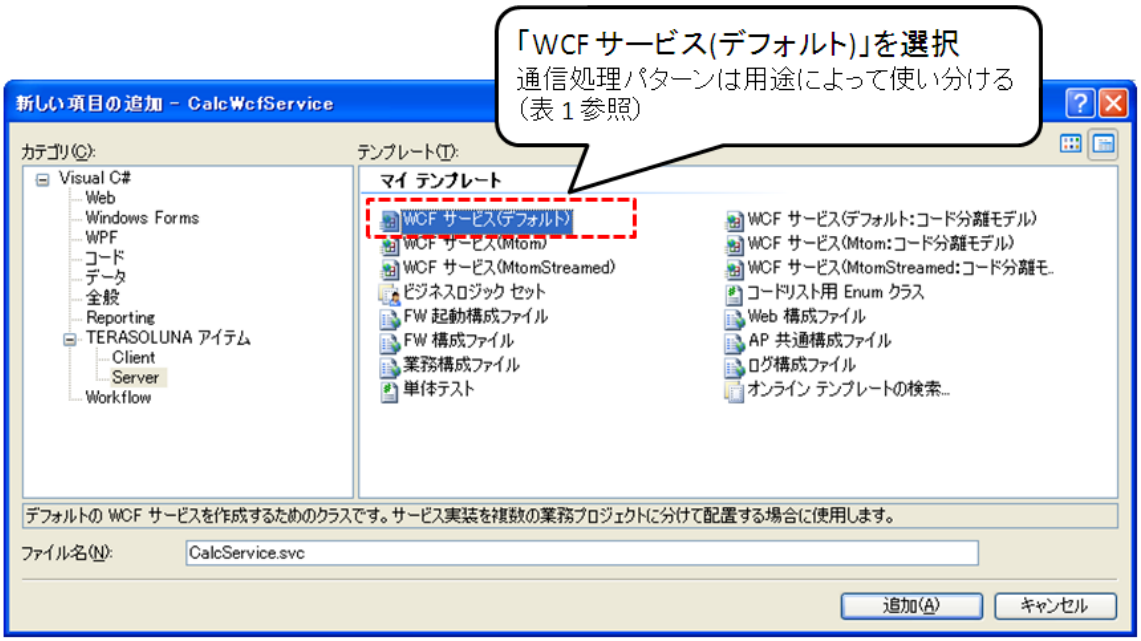

図 7 「WCF サービス(デフォルト)」の追加

カスタムアイテムテンプレートによって以下の svc ファイルが自動で作成される。
WCF ファイル(.svc ファイル)には、以下のように、@ServiceHost ディレクティブの Factory 属性として ServiceHostFactory 継承クラスが指定されており、実行時には ServiceHostFactory 継承クラスで指定された名前(ServiceHostManager の登録名)に応じたサービスホストが生成/起動される。

```
<%@ ServiceHost Language="C#" Debug="true"
        Factory="Terasoluna.Server.ServiceModel.DefaultServiceHostFactory" %>
```

リスト 1 WCF サービスファイル(.svc ファイル)の記述(DefaultServiceHostFactory の例)

表 1 標準提供する通信処理パターンとカスタムテンプレート

通信処理パターン	用途	ServiceHostFactory クラス	カスタムテンプレート名
デフォルト	標準的な通信	DefautServiceHostFactory	WCF サービス(デフォルト)
MTOM	ファイルアップロード・ダウンロード(オンメモリ)	MtomServiceHostFactory	WCF サービス(Mtom)

通信処理パターン	用途	ServiceHostFactory クラス	カスタムテンプレート名
MTOM ストリーミング転送モード	ファイルアップロード・ダウンロード(ストリーミング)	MtomStreamedServiceHostFactory	WCF サービス(MtomStreamed)

◆ WCF サービスクラスの作成

TERASOLUNA フレームワークにより提供されるサーバ用の「業務(個別・共通)プロジェクト」テンプレートを使ったプロジェクトに配置する。

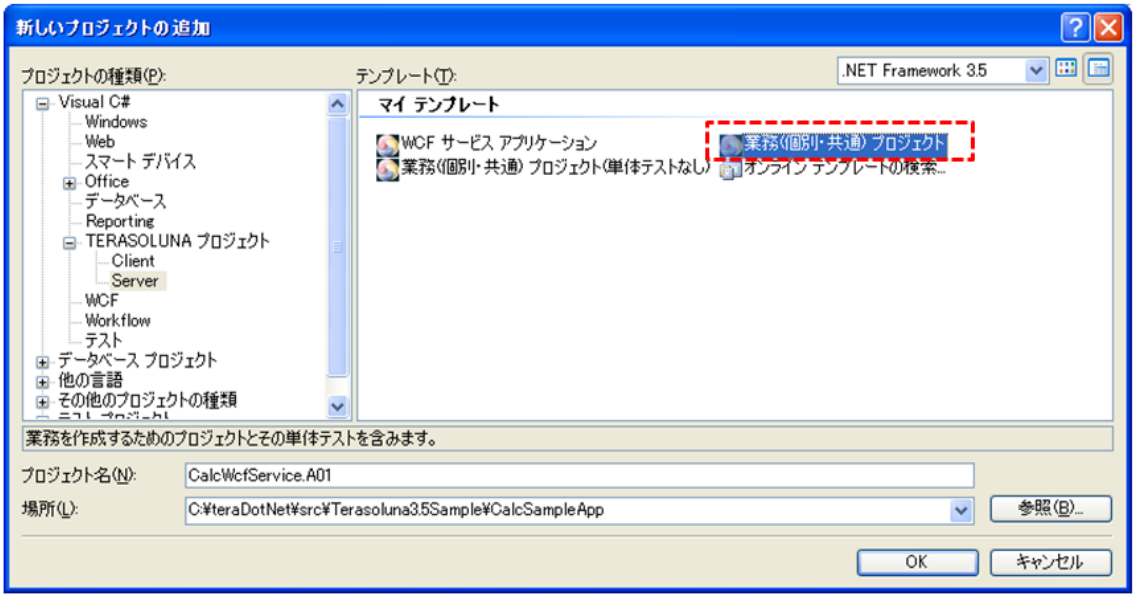

図 8 サーバ用「業務(個別・共通) プロジェクト」の追加

WCF サービスクラス(Service Contract 実装クラス)は、「WCF サービスアプリケーションプロジェクト」において、前述の「WCF サービス(デフォルト)」アイテムテンプレートにより WCF サービスファイル(.svc ファイル)と対で生成される。

図 9 WCF サービスクラスの生成

```
[ServiceContract]
public class CalcService
{
      [OperationContract]
      public void DoWork();
}

// この cs ファイルは、業務プロジェクトの対応するフォルダに移動して下さい
//   …
// クラス名と svc ファイル名は、同じにして下さい。
```

リスト 2「WCF サービス(デフォルト)」テンプレートにより生成された WCF サービスクラス

「WCF サービスアプリケーションプロジェクト」で生成された WCF サービスクラス(Service Contract 実装クラス)は、対応する業務プロジェクトの適切なフォルダへ移動する。配置先の業務プロジェクトの構成は、WCF サービスクラスの名前空間と一致させる(C#プロジェクトの場合[1])。

以下に、WCF サービスクラスの実装例を示す。

---

[1] VB プロジェクトにおける構成は、「WCF サービス アプリケーション」テンプレートで出力される「readme.txt」を参照のこと

サービスとして公開するメソッドに OperationContract 属性を付与する。入力値検証を実施したい場合は、ValidationBehavior 属性を適宜付与する。ValidationBehavior 属性についての詳細は「SV-02 サーバ入力値検証機能」の機能説明書を参照のこと。

```
[ServiceContract]
public class CalcService
{
    [OperationContract]
    [ValidationBehavior(RuleSet = "RS01")]
    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 3 WCF サービスクラスの実装例

◆ Web 構成ファイル(Web.config)の記述

本機能では、Web 構成ファイル(Web.config)にデフォルトのバインディングやビヘイビア等の設定をしておき、それら設定の組み合わせを、ServiceHostFactory 継承クラスと対応づけることができる。TERASOLUNA フレームワークで標準提供している ServiceHostFactory 継承クラスは、以下の表の 3 つである。なお、プロジェクトの要件により、新たな定義の組み合わせを持つ ServiceHostFactory を追加定義し、前述の WCF サービスファイル(.svc ファイル)で指定することで、サービスホストを切り替えることも可能である。なお、ServiceHostFactory 継承クラスの追加定義の方法については、後述する。

表 2 ServiceHostFactory と対応するデフォルト設定

ServiceHostFactory クラスの種類	登録名	binding の種類	binding の構成	servceBehavior の構成	endpointBehavior の構成
			name 属性の値	name 属性の値	name 属性の値
DefautServiceHostFactory	－(なし)	wsHttpBinding	DefaultBinding	DefaultServiceBehavior	DefaultEndpointBehavior
MtomServiceHostFactory	Mtom	wsHttpBinding	MtomBinding		
MtomStreamedServiceHostFactory	MtomStreamed	basicHttpBinding	MtomStreamedBinding		

上記の設定項目について、プロジェクトテンプレートにより生成される Web.config には、あらかじめ以下の表のような設定値が格納されており、パラメータの設定作業を簡略化できるようになっている。アーキテクトは性能やセキュリティなどの非機能要件に応じて、デフォルトのパラメータ設定を細かく調整する。

表 3 Web.config のテンプレートによる binding 構成の初期設定値

name 属性の値	messageEncoding 属性の値	transferMode 属性の値	その他設定内容
DefaultBinding	Text(デフォルト)	-	-
MtomBinding	Mtom	-	maxReceivedSize 属性および readerQuotas タグの maxArrayLength 属性を 10485760(10MB)に設定。送受信するファイルのサイズ等の要件により、適宜調整する。
MtomStreamed Binding	Mtom	Streamed	maxReceivedSize 属性を 10485760(10MB)に設定。送受信するファイルのサイズ等の要件により、適宜調整する。

表 4 Web.config のテンプレートによる serviceBehavior 構成設定値

name 属性の値	設定内容
DefaultServiceBehavior	serviceMetadata タグの httpGetEnabled 属性を”true”に設定 serviceDebug タグの includeExceptionDetailInFaults 属性を”true”(既定値のまま)に設定。

表 5　Web.config のテンプレートによる endpointBehavior 構成設定値

name 属性の値	設定内容
DefaultEndpointBehavior	特になし。

なお、構成ファイルにサービス定義および従属するエンドポイントの定義を明記した場合、、上記のデフォルト設定よりも優先させることができる。システムの共通的な設定と異なる設定を個別にしたい場面では、/configuration/system.servcemodel/service/タグを使用してサービスを適宜定義すること。
以下に、Web.config の記述例を示す。

```
<system.serviceModel>
  <bindings>
    <wsHttpBinding>
      <!-- DefaultServiceHostFactoryのBindingの設定（デフォルト） -->
      <binding name="DefaultBinding" />
      <!-- MtomServiceHostFactoryのBindingの設定(Mtom) -->
      <binding name="MtomBinding" messageEncoding="Mtom"
               closeTimeout="00:01:00" openTimeout="00:01:00" receiveTimeout="10:00:00"
                 sendTimeout="00:01:00"    maxBufferPoolSize="524288"
                 maxReceivedMessageSize="10485760">
        <readerQuotas maxDepth="32" maxStringContentLength="8192"
                 maxArrayLength="10485760"
              maxBytesPerRead="4096" maxNameTableCharCount="16384" />
      </binding>
    </wsHttpBinding>
    <basicHttpBinding>
      <!-- MtomStreamedServiceHostFactoryのBindingの設定（Mtomストリーミング転送モード） -->
      <binding name="MtomStreamedBinding" messageEncoding="Mtom" transferMode="Streamed"
               closeTimeout="00:01:00" openTimeout="00:01:00" receiveTimeout="10:00:00"
                 sendTimeout="00:01:00" maxBufferSize="65536" maxBufferPoolSize="524288"
                 maxReceivedMessageSize="10485760"/>
    </basicHttpBinding>
  </bindings>
  <behaviors>
    <serviceBehaviors>
      <!-- デフォルトのServiceBehaviorの設定 -->
      <behavior name="DefaultServiceBehavior">
        <serviceMetadata httpGetEnabled="true" />
        <serviceDebug includeExceptionDetailInFaults="true" />
      </behavior>
    </serviceBehaviors>
    <endpointBehaviors>
      <!-- デフォルトのEndpointBehaviorの設定 -->
      <behavior name="DefaultEndpointBehavior">
      </behavior>
    </endpointBehaviors>
  </behaviors>
  <services>
  </services>
</system.serviceModel>
```

リスト 4　Web.config の記述例

◆ FW 構成ファイル(TerasolunaFramework.config)の記述

本機能を利用するためには、FW 構成ファイル(TerasolunaFramework.config)の /configuration/unity/containers/container/extensions/add タグに、以下の UnityContainerExtension 継承クラスを記述する。

- Terasoluna.Server.ServiceModel.ServiceModelExtension クラス
  - 本機能を利用するため必要なデフォルトの Extension

また、/configuration/unity/containers/container/instances/add タグで、name 属性を「BaseNamespace」として、エンドポイントの URL より WCF サービスクラスを決定する際に使用するベース名前空間を、文字列型の値で指定する。

以下に、TerasolunaFramework.config の設定例を示す。
なお、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ファイルを生成した場合、下記内容はデフォルト設定されている。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
・・・
  <unity>
    <typeAliases>
      <typeAlias alias="string" type="System.String" />
    </typeAliases>
    <containers>
      <container>
        <types>
        </types>
        <instances>
          <!-- WCFサービスを取得するためのベース名前空間名の設定 -->
          <add name="BaseNamespace" type="string" value="CalcWcfBusinessApplication" />
        </instances>
        <extensions>
・・・
          <!-- WCFサービス管理機能の設定 -->
          <add type="Terasoluna.Server.ServiceModel.ServiceModelExtension,
                Terasoluna.Server.ServiceModel" />
        </extensions>
      </container>
    </containers>
  </unity>
</configuration>
```

リスト 5　TerasolunaFramework.config の設定例

■ TIPS

以下では、本機能を利用する際、開発者がよく直面する問題について対処方法をまとめている。
開発時に実装方法に困った場合に参考にするとよい。

◆ WCF のトレース機能の利用

通信周りの問題解析時には、System.Diagnostics のトレース機能を使用して、WCF サービスの処理の追跡や、SOAP メッセージの確認を実施することができる。
詳細は、MSDN のドキュメント[2]等を参照のこと。

トレースの利用には、Web 構成ファイル(Web.config)に設定が必要である。
以下に、WCF のトレースの設定例を示す。

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <diagnostics>
      <messageLogging logEntireMessage="true"
                      logMalformedMessages="false"
                      logMessagesAtServiceLevel="true"
                      logMessagesAtTransportLevel="false"
                      maxMessagesToLog="3000"
                      maxSizeOfMessageToLog="200000"/>
    </diagnostics>
  </system.serviceModel>
  <system.diagnostics>
    <!-- WCFトレースの設定 -->
      <source name="System.ServiceModel"
        switchValue="Information, ActivityTracing"
        propagateActivity="true">
        <listeners>
          <add name="WCFTraceLog" />
        </listeners>
      </source>
      <source name="System.ServiceModel.MessageLogging">
        <listeners>
          <add name="WCFTraceLog" />
        </listeners>
      </source>
     </sources>
    <sharedListeners>
      <add name="WCFTraceLog" type="System.Diagnostics.XmlWriterTraceListener" initializeData=
"C:¥TEMP¥WcfSeverTraces.svclog" />
    </sharedListeners>
  </system.diagnostics>
</configuration>
```

リスト 6　Web.config における WCF トレースの設定例

[2] トレースとメッセージログ
http://msdn.microsoft.com/ja-jp/library/ms751526.aspx

出力されたログは、「サービストレースビューワ」(SvcTraceViewer.exe)で閲覧可能である。
なお、クライアントのログを追加で読み込むことで、クライアント-サーバ間のやり取りを追跡することができる。

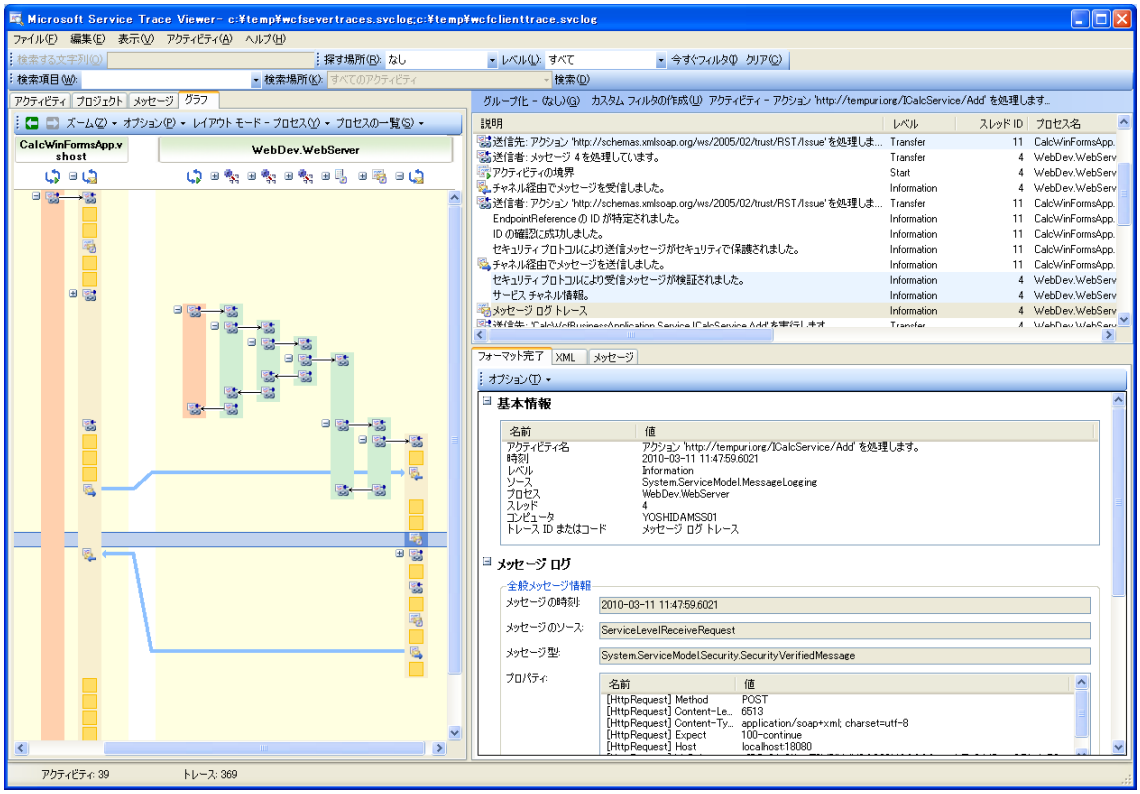

図 10 サービストレースビューワ

図 11 サービストレースビューワで閲覧した SOAP メッセージ(メッセージログ)の例

■ 内部構成

◆ クラス図

以下に、本機能の主要なクラスを表すクラス図を示す。

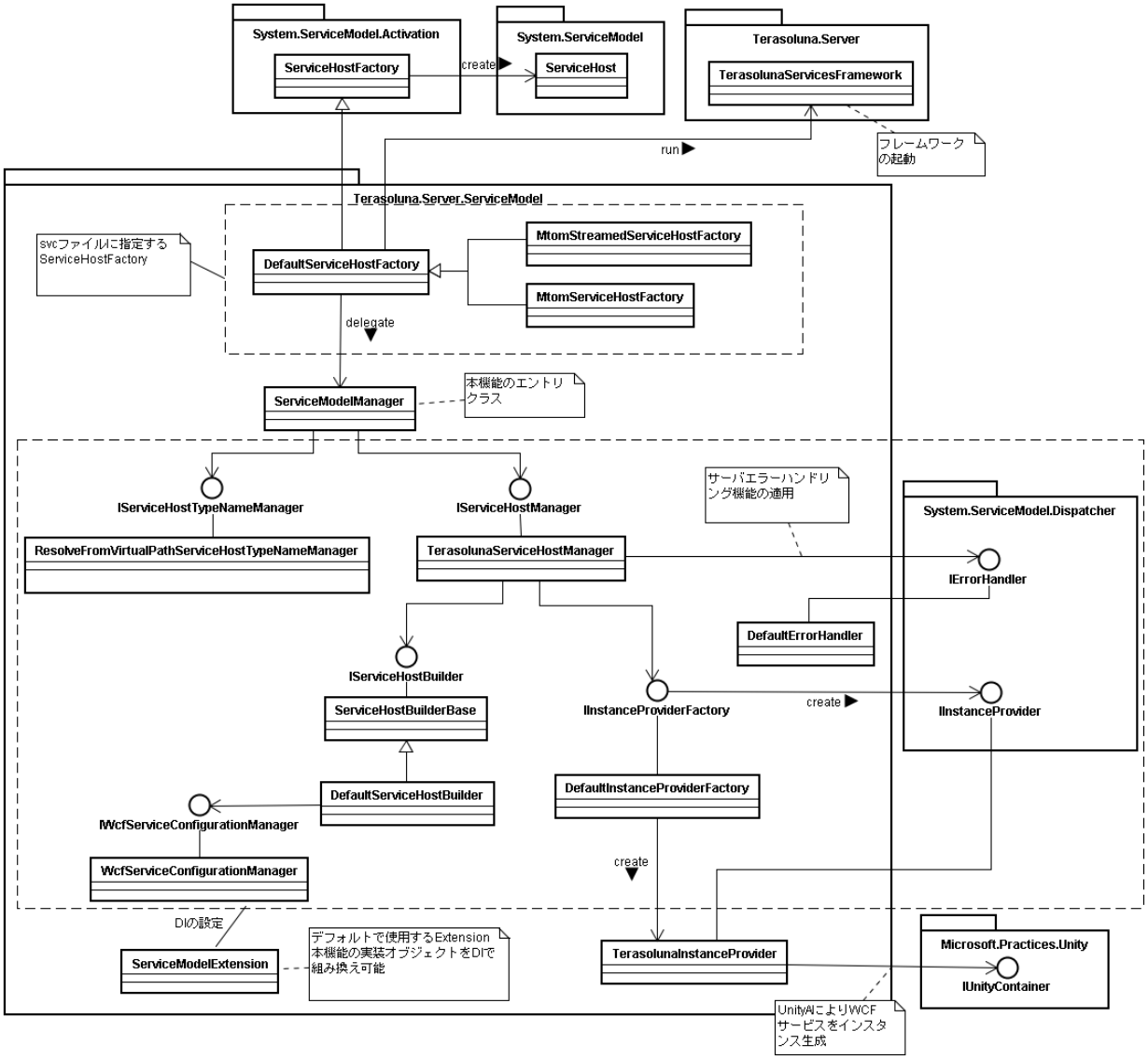

図 12 クラス図

◆ 構成クラス

以下に本機能を構成するクラスを示す。

表 6 構成クラス一覧

項番	クラス名	説明
Terasoluna.Server.ServiceModel 名前空間		
1	DefaultServiceHostFactory	デフォルトのバインディングやビヘイビアの設定を適用する ServiceHost を作成する System.ServiceModel.Activation.ServiceHostFactory 継承クラス。
2	MtomServiceHostFactory	MTOM におけるデフォルトのバインディングやビヘイビアの設定を適用する ServiceHost を作成する System.ServiceModel.Activation.ServiceHostFactory 継承クラス。
3	MtomStreamedServiceHostFactory	MTOM ストリーミング転送モードにおけるデフォルトのバインディングやビヘイビアの設定を適用する ServiceHost を作成する System.ServiceModel.Activation.ServiceHostFactory 継承クラス。
4	ServiceModelManager	本機能のエントリクラス。
5	IServiceHostManager	System.ServiceModel.ServiceHost オブジェクトを管理するためのインタフェース
6	IServiceHostBuilder	System.ServiceModel.ServiceHost オブジェクトを生成するためのインタフェース
7	IServiceHostTypeNameManager	WCF サービスクラス名を解決するためのインタフェース
8	IInstanceProviderFactory	System.ServiceModel.Dispatcher.IInstanceProvider オブジェクトを生成するためのインタフェース
9	IWcfServiceConfigurationManager	WCF に関する設定情報を管理するためのインタフェース
10	TerasolunaServiceHostManager	IServiceHostManager のデフォルト実装クラス。
11	ServiceHostBuilderBase	IServiceHostBuilder が持つ共通機能を実装するクラス。
12	DefaultServiceHostBuilder	IServiceHostBuilder のデフォルト実装クラス。
13	ResolveFromVirtualPathServiceHostTypeNameManager	IServiceHostTypeNameManager のデフォルト実装クラス。WCF サービスの URL の仮想パスをもとにサービスクラス名を決定する。
14	DefaultInstanceProviderFactory	IInstanceProviderFactory のデフォルト実装クラス。

項番	クラス名	説明
15	WcfServiceConfigurationManager	IWcfServiceConfigurationManager のデフォルト実装クラス。
16	TerasolunaInstanceProvider	System.ServiceModel.Dispatcher.IInstanceProvider のデフォルト実装クラス。
17	SetErrorHandlerBehavior	System.ServiceModel.Dispatcher.IErrorHandler オブジェクトを全てのチャネルに設定するための System.ServiceModel.Description.ServiceBehavior 実装クラス
18	SetInstanceProviderBehavior	System.ServiceModel.Dispatcher.IInstanceProvider オブジェクトを全てのチャネルに設定するための System.ServiceModel.Description.ServiceBehavior 実装クラス
19	ServiceModelExtension	本機能を提供するデフォルトの Extension クラス。

■ 拡張ポイント

WCF サービスのバインディングやビヘイビアの組み合わせを新たに追加したい場合には、組み合わせに対応する ServiceHostFactory 継承クラスを新規作成する。以下に、実装例を示す。

```
public class SampleServiceHostFactory：DefaultServiceHostFactory
{
    protected override ServiceHost CreateServiceHost(Type serviceType, Uri[] baseAddresses)
    {
        IServiceHostManager manager = ServiceModelManager.GetServiceHostManager("NewConfig");
        return manager.CreateServiceHost(serviceType, baseAddresses);
    }
}
```

リスト 7　ServiceHostFactory 継承クラスの実装例

また、ServiceModelExtension クラスを継承して、サービスホストの定義を追加で登録する。
以下に、Exntension クラスの実装例を示す。
ServiceModelExtension.RegisterServiceHostManager メソッドの第 1 引数は、ServiceHostManager クラスの登録名であり、ServiceHostFactory 継承クラスの実装で指定する文字列（ServiceModelManager.GetServiceHostManager()の文字列）と一致させる（上記例の場合、“NewConfig”）。
また、ServiceModelExtension.RegisterServiceHostManager メソッドの第 2 引数は、ServiceModelExtension.RegisterServiceHostBuilder メソッドの第 1 引数で指定する ServiceHostBuilder クラスの登録名と一致させる。
ServiceModelExtension.RegisterServiceHostManager メソッドの第2引数はサービスメタデータ（WSDL）を公開するかのフラグ（ServiceMetadataBehavior.HttpGetEnabled プロパティ）に対応する。なお、Web.config で serviceMetadata タグの httpGetEnabled 属性の値を設定している場合、その設定が優先される。デフォルト提供されているものは、第2引数が false に設定されているため、Web.config で設定を上書かない限り（true にしない限り）、WSDL は公開されない。
第3引数～第6引数は、Web.config に事前定義したバインディングやビヘイビアの name 属性の値を指定する。

```
public class SampleServiceModelExtension : ServiceModelExtension
{
    protected override void Setup()
    {
        base.Setup();
        RegisterServiceHostManager(
            "NewConfig",      //ServiceHostManagerの登録名
            "NewConfig"       //参照するServiceHostBuilderの登録名
            );
        RegisterServiceHostBuilder(
            "NewConfig",                  //ServiceHostBuilderの登録名
            false,                        //サービスメタデータを公開するかどうか
            "basicHttpBinding",       //binding名
            "NewBindingConfig",       //web.configのBinding構成のname属性
            "NewServiceBehavior",     //web.configのServiceBehavior構成のname属性
            "NewEndpointBehavior"); //web.configのEndBehavior構成のname属性
    }
}
```

リスト 8　ServiceModelExtension の拡張例

Extension クラス作成後、FW 構成ファイル(TerasolunaFramework.config)でデフォルトの Extension クラスの代わりに設定する。以下に記述例を示す。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
・・・
  <unity>
    <containers>
      <container>
        <extensions>
・・・
          <!-- WCFサービス管理機能の拡張 -->
          <add type=" CalcWcfBusinessApplication.ServiceModel.SampleServiceModelExtension,
                CalcWcfBusinessApplication" />
        </extensions>
      </container>
    </containers>
  </unity>
</configuration>
```

リスト 9　TerasolunaFramework.config の記述例

Web.config には、Extension クラスで、RegisterServiceHostBuilder メソッドの引数の値に対応するバインディングやビヘイビアの設定をする。

```
<system.serviceModel>
  <bindings>
    <basicHttpBinding>
      <binding name="NewBindingConfig" />
    </basicHttpBinding>
  </bindings>
  <behaviors>
    <serviceBehaviors>
      <behavior name="NewServiceBehavior">
        <serviceMetadata httpGetEnabled="true" />
        <serviceDebug includeExceptionDetailInFaults="true" />
      </behavior>
    </serviceBehaviors>
    <endpointBehaviors>
      <behavior name="NewEndpointBehavior">
      </behavior>
    </endpointBehaviors>
  </behaviors>
</system.serviceModel>
```

リスト 10　Web.config の記述例

業務開発者は、対象の WCF サービスファイル（.svc ファイル）において、@ServiceHost ディレクティブの Factory 属性に、アーキテクトが追加した ServiceHostFactory 継承クラスを指定する。

```
<%@ ServiceHost Language="C#" Debug="true"
        Factory="CalcWcfBusinessApplication.ServiceModel.SampleServiceHostFactory" %>
```

リスト 11　WCF サービスファイル（.svc ファイル）の記述例

■ 関連機能

- CM-02 インスタンス管理機能
- CL-05 クライアントエラーハンドリング機能