WCF 自習書

第 1 部 サービス開発総論

developer & platform evangelism

制作者: エディフィストラーニング株式会社

目次

はじめに

このシリーズの自習書では、ソフトウェア開発者がシステム連携や分散システムを実装するうえでキーテクノロジとなる WCF（Windows Communication Foundation）について、今後のさらなる習得や活用の際に役立つよう、その概要や実装のポイントについて取り上げます。

まず、第 1 部となる本編では、WCF の全体像を概観し、WCF が提供する各種テクノロジや機能の選択肢について確認し、それぞれの特徴について取り上げるほか、メリットやデメリットをふまえた各テクノロジや機能の使い分けの指針なども示します。また、同時に第 1 部では、WCF の感触をつかんでいただくため、WCF を用いた簡単なプログラムコード例を紹介します。

そして、第 2 部では、各機能について具体的な実装方法、たとえば、WCF 関連のクラスライブラリを用いたコード書き方や、構成ファイルなどの実行環境の構築方法、また、WCF に関する Visual Studio 2010 の使用方法について、それぞれのポイントを解説します。

註: 本ドキュメントでは、.NET Framework のバージョンは 4 を前提にしていますが、バージョン 4 における WCF の新機能のみを扱うわけではなく、WCF の全般的な特徴を概観します。バージョン 4 における WCF の新機能について一通り把握されたい場合は、以下のアドレスを参照してください。
http://msdn.microsoft.com/ja-jp/library/dd456789.aspx
Windows Communication Foundation の新機能
また、以下のアドレスには WCF の各機能の詳細についての解説やサンプルコードなどが包括的に記載されています。この自習書シリーズをお読みになった後、必要に応じて参照してください。
http://msdn.microsoft.com/ja-jp/library/dd456779.aspx
Windows Communication Foundation （WCF 関連資料のルートノード）
http://msdn.microsoft.com/ja-jp/library/ms730846.aspx
ドキュメントのガイド （MSDN ライブラリにおける WCF 関連資料の包括的なリンク一覧）

前提知識

本ドキュメントでは、.NET Framework の基礎知識や Visual Studio などの、 .NET の開発環境に関する基礎知識の説明については割愛しており、本ドキュメントをお読みになる上での前提知識として必要になります。また、本シリーズではプログラムコードの例を取り上げる場合があり、その場合は紙面の都合もあって、コードの特徴的な部分に絞り説明しております。そのため、基本的な言語文法の知識も前提知識として必要になります。

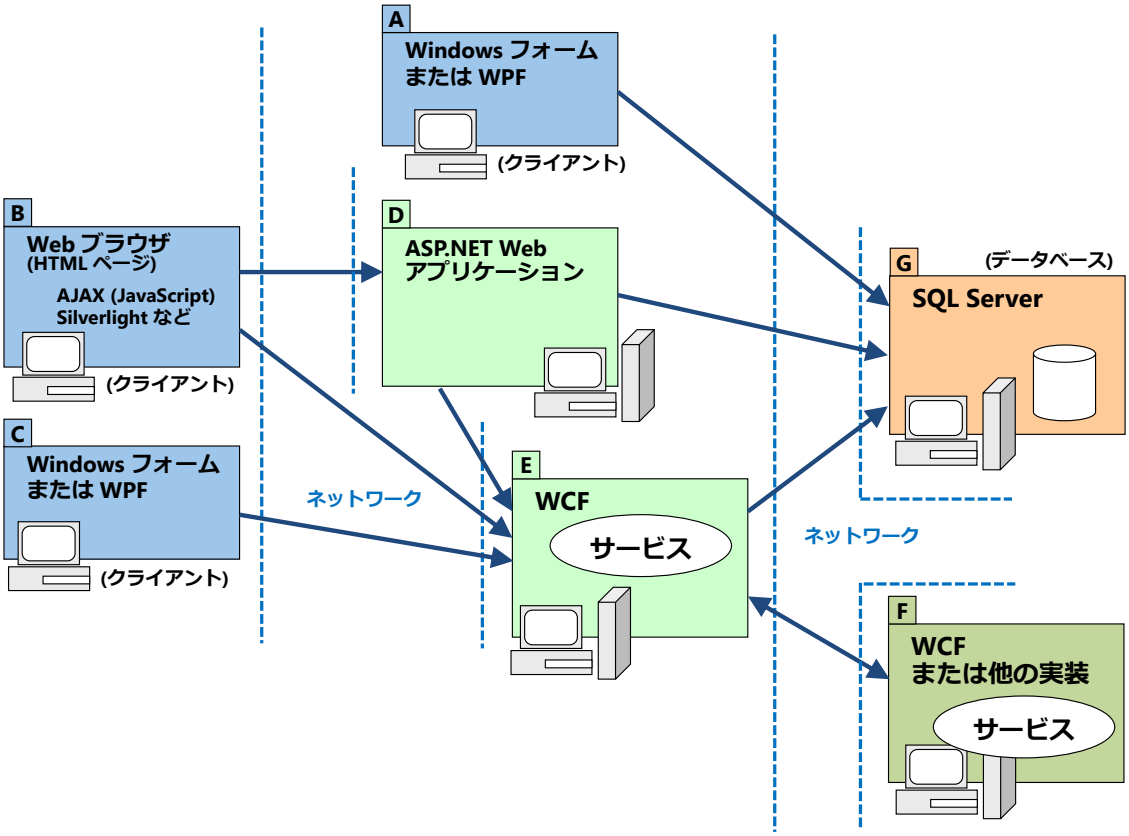
第 1 章 サービスを利用したシステムの全体像

WCF（Windows Communication Foundation）とは、.NET Framework におけるサービスの実装や、複数のサービス間の連携を行うためのフレームワークです。まずは、この WCF 自体の特徴や各機能について解説する前に、.NET Framework 関連のテクノロジを用いたシステムの全体像を把握し、その全体像の中で、サービスとそれを実現する WCF の必要性や役割を確認することにしましょう。

1.1 .NET におけるシステムの全体像

次の図は、.NET Framework を用いた 2 層システムや 3 層システムなどのシステム全体の構成において、典型的なパターンをいくつか表しています。この図には、主な .NET テクノロジを使用した個所が示されているほか、WCF を用いてサービスを実装した箇所が示されています。

図 1. .NET Framework を用いたシステムの全体像



この図の大きな四角形それぞれ（A から F）が、複数の層からなるシステムにおいて 1 つの層を表しています。（典型的には、これらの層ごとの実装は、物理的に別々のマシンに配置することが多いですが、同一のマシンに配置することもできます。そういう意味では、この層は論理的な実装単位であると言えます。）

このうち、A と G の組み合わせは、一般的なクライアントとサーバーからなる 2 層システムです。それ以外は、3 層以上の多層システムの構成要素であり、また、ここでは F が外部のシステムを表しています。この図が示すように、一般にサービスが関係するケースは、3 層以上の多層システムの場合です。

多層システムの構成要素のうち、B や C がクライアントであり、D が Web サーバーに配置された Web アプリケーションです。ここでのテーマでもあるサービスは、E のように中間層としての役割があり、D の Web アプリケーションの要求を受けて、中継役として G のデータベースにアクセスしたり、C のクライアントから直接要求を受けてデータベースにアクセスしたりする場合もあります。また、他のシステムと連携する際に、他のサービス(ここでは G)とやり取りする際にも利用されます。

このような全体像の中で、中間層としてのサービスの実装（E や F）に使用されるのが WCF ですが、これをふまえ、その必要性や役割について確認してみましょう。

> **註:** 上記の図 1 には、Windows Azure のような「クラウド」については明示していませんが、エンドユーザーが直接使用するクライアント（A から C）以外は、クラウド上で同様の実装を構築することが可能です。この一連の自習書では、WCF 自体の実装に焦点を当てていることから、サンプルプログラムの環境は従来のサーバー環境（オンプレミス）を使用しますが、基本的な WCF のプログラムコードはクラウド上の環境にも適用できます。

1.2 システムの中での WCF の必要性と役割

WCF の必要性や役割を考えるにあたり、まずは、WCF によって実現される「サービス」に関して、その一般的な特徴から考えてみましょう。

一般に多層システムなどの分散システムは、負荷分散や保守性の向上のために使用されます。そのような分散システムを構築する文脈においては、「サービス」とは、他の部分との依存関係が少ない「独立して実行可能な」アプリケーションの一種であり、他の部分と「接続可能な」性格を備えています。通常、1 つのサービスは 1 つの業務処理（ビジネスプロセス）などを行う単位として構築され、これらの複数のサービスの組み合わせ、システム統合を実現する手段として利用されます。

このようなサービスが必要とされる背景には、以下の点が挙げられます。

（a）システム間や、または業務処理間の相互連携

（b）変化を続けるビジネスニーズへの迅速な対応

（c）開発生産性の向上、効率化

このうち、(a) のように既存の複数のシステムを組み合わせたり、既存の業務処理を行うアプリケーションを組み合わせたりすることは、それぞれが単体で利用されるよりも、相乗効果によって、より大きな付加価値を生み出すことにつながり、サービスのような接続可能なものが必要とされています。

また、（b）に挙げたように昨今の変化が激しいビジネス環境では、急激なニーズの変化にも対応できるように、システム構成や各システム構成要素の機能を迅速かつ柔軟に変更する必要があります。そのためには、1 つの構成要素の変更が他へ影響することを最小限に抑えつつ、柔軟に構成要素の組み合わせを変えることが出来る必要があります。サービスのような独立性の強いアプリケーションは、これらの必要性に応えることができます。

さらに（c）に挙げたように、接続可能で独立性のあるアプリケーションを作る場合、ネットワークプロトコルや、やり取りするメッセージの形式など、様々な要件に対応した実装を、効率よく開発することが要求されます。一般に、サービス構築のために提供されるテクノロジには、このようなアプリケーションの開発生産性を高めるフレームワークやツールが必要となります。

WCF を使用すれは、このような必要性から生まれたサービスを迅速に構築することができ、これら（a）から（c）の 3 つのニーズに応えることができます。

たとえば、WCF を用いれば、コントラクトと呼ばれるサービスとクライアントがやり取りする際の規約（インターフェイス）のみを公開し、その内部実装はサービスの外部から切り離すことで、独立性の高いコンポーネントを作成できます。

また、WCF は様々なプロトコルやトポロジをサポートしており、様々な環境でのシステム構築や相互運用に優れ、柔軟にビジネスニーズの変更にも対応できます。

さらに、.NET Framework に基づいて、サービスおよびサービスのクライアントを構築するためのフレームワークが提供されており、開発生産性の向上につながるほか、信頼性やセキュリティなど .NET Framework が本来備えている様々な恩恵を受けることができます。

それでは、このようなニーズに応える役割を持つ WCF には、具体的にどんな機能があるのか、次章では改めて確認してみることにしましょう。

第 2 章 WCF オーバービュー

この章では、前章で触れたサービスのニーズに応える WCF が何であるのか、まずは改めて WCF の定義や基本的特徴を確認します。また、このドキュメントがソフトウェア開発者の方を対象としていることもあるので、実際の WCF の感触をつかんでいただくため、基本的なサンプルプログラムを用いて、その特徴や開発環境について確認します。そして、WCF の様々な機能や関連するテクノロジのラインアップを紹介します。

2.1 改めて WCF とは

WCF（Windows Communication Foundation）は、.NET Framework 3.0 から導入された、サービスの実装やサービスによる連携（サービスを利用するクライアントを含む）を実現するフレームワークです。具体的には、クラスライブラリやそれらを用いたサービスの実行環境、また関連する開発ツールとして提供されています。

従来からも、このようなサービスやサービス連携を含む分散アプリケーションを実現するために、.NET のテクノロジとして、次表のものが提供されていました。従来であれば、アプリケーションの形態に応じて、これらのテクノロジを使い分けていましたが、WCF はこれらのテクノロジを統合する位置付けにあります。

表 1. 分散アプリケーションを構築する様々な従来のテクノロジ

名称	説明
ASP.NET XML Web サービス	ASP.NET に基づくテクノロジの 1 つであり、HTTP ベースの基本的な Web サービスを実装するために使用する
Enterprise Services	中間層として利用できる COM+ベースの .NET 向けコンポーネントを作成するための属性ベースのテクノロジ
Web Services Enhancements	様々な WS-* 仕様に対応したサービスを構築するためのテクノロジ
System.Messaging（名前空間）	一方向のメッセージ送信を実現するために、 MSMQ を .NET 環境で利用するためのライブラリ群
.NET リモート処理	位置を意識せずに、リモートサーバーに配置されたオブジェクトにアクセスするためのテクノロジ

たとえば、以前であれば、HTTP ベースのサービスを手早く開発するめに、ASP.NET Web サービスを利用できましたが、TCP によるバイナリ転送を行う必要が出てくれば、.NET リモート処理などの別のテクノロジを用いて、サービスが適用できるよう書き換える必要がありました。WCF であれば、HTTP も TCP によるバイナリ転送も、統合された 1 つのプログラミングモデルで表現することができ、これによって開発生産性を高めることができます。

また、WCF は単純にこれらに置き換わるわけではなく、これらのテクノロジに基づく既存サービスとの相互連携や、これらのテクノロジを WCF のサービスの中で利用できる仕組みも用意されています。

註: 表 1 の各テクノロジと WCF との関係について言及した資料については、以下のアドレスに掲載されたリンク集から参照することができます。たとえば、.NET リモート処理アプリケーションを WCF へ移行する方法、COM+アプリケーションとの統合方法、WCF における MSMQ の利用方法、また、WCF のクライアントから WSE 3.0 ベースのサービスにアクセスする方法など、移行や相互運用に関する資料へのリンクが掲載されています。
http://msdn.microsoft.com/ja-jp/library/ms730846.aspx ドキュメントのガイド
（このアドレスの Web ページ内の下部にある「Windows Communication Foundation の他のテクノロジの統合」というタイトル以下の各項目のリンクを参照してください。）

さらに、WCF には様々なソフトウェアの品質を高めるための、以下に挙げる機能や側面を備えています。

（a）信頼性やセキュリティの強化（暗号化やメッセージの署名など）

（b）相互運用性の向上（WS-*仕様のサポート、REST サポートなど）

（c）保守性の向上（構成ファイルによる容易な変更）

（d）豊富な機能

・HTTP、TCP、名前付きパイプなど様々なプロトコルのサポート
・要求/応答、単方向、双方向など様々なメッセージング形態のサポート
・セキュリティや信頼性を高める様々なバリエーション、トランザクションサポートなど
・柔軟なホスト形態（ASP.NET、アプリケーション、Windows サービスなど）

これらの各特徴については、改めて後述の「2.3 WCF の様々な機能と実装」で取り上げます。

いずれにしても、WCF を利用すれば、上記の様々な機能をサポートする付加価値の高いサービスを実現することができます。

2.2 WCF（基本部分）の実装概要と開発環境

ソフトウェア開発者の方にとって、WCF の基本的な特徴を確認するには、実際の WCF を用いたプログラムコードの中から、実装の感触をつかんでいただくのがよいでしょう。ただし、第 2 部や第 3 部でサンプルコードを用いながら、実装方法について改めて解説するので、ここでは、いくつかのコードの断片を題材にして、主な基本的特徴を読み取ることにしましょう。

まず、WCF に関するソフトウェアの作り込みの観点から見て、その実装はサービスとクライアントに分類できます。このうち、WCF を用いて実装したサービスのことを「WCF サービス」と呼び、この WCF サービスにアクセスするクライアントのことを「WCF クライアント」とも呼んでいます。さらに、便宜的に次に挙げる構成要素に分けることができます。

WCF サービスの実装

（a）サービス コントラクト、および、それを実装したクラス

（b）ホスト アプリケーション

（C）サービスに関する構成

WCF クライアントの実装

（d）クライアント（サービスにアクセスするための実装）

（e）クライアントに関する構成

上記が、WCF を用いた実装のすべての構成要素というわけではありませんが、WCF を用いて基本的な実装を行うとなれは、まずは必要となる構成要素です（各構成要素の説明には後述）。

また、そもそも WCF サービスは相互運用性に優れ、WCF サービスとやり取りするためのコントラクト（規約）にさえ従えば、上記に挙げたクライアントは、いずれのプラットフォームでも実装可能なのですが、クライアントの作成にも WCF が提供するライブラリを利用でき、.NET Framework 対応のアプリケーションとして実装することで、.NET Framework の恩恵を受けることができます。

このような WCF サービスや WCF クライアントを開発するには、.NET Framework 3.0 以降に対応した開発環境が必要です。Visual Studio 2010 の場合は、.NET Framework 4 対応なので、標準で WCF 向けアプリケーションの開発ができます。プログラミング言語としては、.NET Framework に対応していれば、いずれの言語を使用しても開発することができますが、Visual C# や Visual Basic の場合であれば、Visual Studio 2010 に WCF 向けのプロジェクトテンプレートや項目テンプレートが用意されているので、効率よく開発することができます。

> **註:** Visual Studio 2010 を用いた具体的な WCF 向けアプリケーションの実装方法については、「第 2 部 サービス開発 WCF 編」、および、「第 3 部 サービス開発 WCF REST 編」で取り上げます。

次に、それぞれの構成要素について、具体例を用いながら、簡単に特徴を確認してみましょう。この後の例では、クライアントからサービスに対して要求をすると、"Hello!" という文字列が応答として返る簡単な例を扱います。

・サービスコントラクト、および、それを実装したクラス

サービスを構築するには、クライアントなどのサービス外部から、サービスにアクセスする際に使用するインターフェイス（コントラクト）と、それを実装したクラスを記述する必要があります。以下に例を示します。

例 1. 参考: サービス コントラクトとサービスクラス

（C#）

```
[ServiceContract]             //←[1]
public interface IMyService   //←[2]
{
    [OperationContract]  //←[3]
    string HelloWorld(); //←[4]
}

public class MyService : IMyService //←[5]
{
    public string HelloWorld() {    //←[6]
        return "Hello!";
    }
}
```

（Visual Basic）

```
<ServiceContract()>            '←[1]
Public Interface IMyService    '←[2]

    <OperationContract()>             '←[3]
    Function HelloWorld() As String   '←[4]

End Interface

Public Class MyService
    Implements IMyService   '←[5]

    Public Function HelloWorld() As String _
            Implements IMyService.HelloWorld  ←[6]
        Return "Hello!"
    End Function

End Class
```

ここでは、個々の細かい書き方ではなく、特徴を読み取ることにします。

サービスの実装は何らかの処理を行うためのものなので、[6]のようにメソッドのブロックとして実装しています。そして、そのようなメソッドを実行すべくクライアントがアクセスするためには、一般的なオブジェクト指向プログラミングの手法がとられています。オブジェクト指向プログラミングでは、オブジェクトの利用者がオブジェクトにアクセスできるようにするためには、インターフェイスを定義し、そのインターフェイスを実装したクラスを用意します。このサンプルコードでは、[2]のようにアクセスに使用するインターフェイスを定義し、[5]のように、そのインターフェイスを実装したクラスを用意しています。普段、オブジェクト指向プログラミングを用いている開発者にとっては、習得する上で特に難しくはない、馴染みのあるプログラミング モデルでしょう。

[2]のインターフェイスは、クライアントがサービスにアクセスし、やり取りするための接点であり、WCF では「サービス コントラクト」（または単に「コントラクト」）と呼ばれています。特に、.NET Framework を用いて WCF クライアントを実装する場合には、サービスにアクセスする上で、コードレベルで必要な情報は、このコントラクタだけであり、[6]の実装には依存しましせん。この仕組みによって、サービス実装の独立性が実現されています。なお、サービスコントラクトを実装したクラスのことを「サービス クラス」とも呼んでいます。

また、[2]のインターフェイスが、サービスコントラクトになるためには、[1]や[3]の宣言的な属性を付けて定義する必要があります。このように定義しておけば、クライアントはこのインターフェイスを使って、[4]の HelloWorld メソッドを呼び出すことで、ネットワークを介して、サービス側の実装を呼び出せるようになります。

注目すべき点は、使用するネットワークプロトコルやサービスとの具体的なハンドシェイクについて記述していない点です。単純に属性を宣言するだけでよく、開発生産性を向上させることができます。また、使用するトランスポート プロトコルの指定は、別途、後述の構成ファイルに行うことができるので、ソースコードとは切り離してプロトコルの変更を行うことができ、保守性も向上します。

・ホストアプリケーション

前述のように「サービスコントラクト」を実装した「サービスクラス」は、サービスの中核となる部分と言えますが、それだけではサービスをクライアントに提供できません。サービスクラスのインスタンスを作成し、クライアントからの要求を受け取ることができる環境を用意する必要があります。そのような環境が、ホストアプリケーションです。

WCF には、簡単にホストアプリケーションを構築できるフレームワークが用意されており、その形態も様々です。便宜的にホストアプリケーションの形態を分類すると、次のものが挙げられます。

（a）自己ホスト（スタンドアローンのマネージアプリケーション）

（b）.NET Framework を用いて実装した Windows サービス（NT サービス）

（c）インターネットインフォメーションサービス（IIS 5.1、IIS 6.0、IIS 7.0）

（d）Windows プロセス アクティブ化サービス（WAS）

そもそも .NET Framework では、WCF サービスのホストアプリケーションを構築するためのクラスライブラリが提供されているため、任意の .NET Framework 対応アプリケーションをホストアプリケーションとして実装できます。これが前述の（a）にあたります。このため、様々な既存のマネージアプリケーションに対して、WCF サービスを実装することもできます。同様に、（b）に挙げたように、.NET Framework を用いて Windows サービスを実装できるので、その際にホストアプリケーションとして実装することもできます。実際にホストアプリケーションとして最低限必要なコードは、例 2 に示すように、数行程度の実装で済みます。ServiceHost オブジェクトを作成し、Open メソッドを呼び出すだけです。

例 2. 参考: ホストアプリケーション

（C#）

```
var host = new ServiceHost(typeof(MyService));
host.Open();
```

（Visual Basic）

```
Dim host As New ServiceHost(GetType(MyService))
host.Open()
```

さらに、IIS も HTTP ベースのホストアプリケーションとして利用でき、この場合はホストアプリケーションのためのコーディングは不要です。また、IIS が持つ管理機能を利用できるほか、IIS 7.0 では ASP.NET の Web アプリケーションと WCF サービスが実行環境を共有でき、ASP.NET の認証やセッションなどの仕組みを WCF サービスから利用できます。また、Windows Vista や Windows Server 2008 以降では、IIS のプロセスモデルをもとにして、HTTP に依存しない新しい環境として、（d）の Windows プロセス アクティブ化サービスが導入されました。これを IIS とともに利用すると、IIS 上で非 HTTP ベースのホスティング環境を提供することもできます。

第 2 部では、具体的なホスティング環境の構築手順について、いくつかの例を改めて取り上げます。

註: 以下のアドレスには、各ホスティング環境でサポートされる機能や特徴、利用するためのシステム要件などが記載されています。
http://msdn.microsoft.com/ja-jp/library/ms730158.aspx ホスティングサービス

・サービスに関する構成

サービスをネットワーク上に公開するには、クライアントがアクセスに使用するアドレスやプロトコルをホストアプリケーションに設定するなど、サービスを構成する必要があります。WCF では、これらの構成を「構成ファイル」に別途記述することができ、プログラムコードとは切り離して保守管理することができます。

以下に、構成ファイルの中の特徴的な設定を示します。

例 3. 参考: サービス側の構成ファイル（抜粋）

```
<service name="MySrv.MyService">
  <host>
    <baseAddresses>
      <add baseAddress="http://localhost:8080/MyDir"/>  ←[1]
    </baseAddresses>
  </host>
  <endpoint address="MyService"              ←[2]
            binding="basicHttpBinding"       ←[3]
            contract="MySrv.IMyService" />   ←[4]
</service>
```

[2]の<endpoint>要素が「エンドポイント」と呼ばれるもので、クライアントからのアクセスポイントに関する構成になります。主な設定内容としては、[2]か[4]までの 3 項目です。

[2]はアドレス（[1]を起点にした相対アドレス）、[3]は「バインディング」と呼ばれる設定項目、[4]は、使用するコントラクトです。（それぞれの頭文字をとって、まとめて「ABC」と表現することもあります。）

このうちバインディングは、トランスポートプロトコル、エンコーディング、セッションの有無など、データをどうやり取りするか、その方法の全般的な構成を指定するものです。予め、決められた名前の構成がいくつか用意されており、[3]の「basicHttpBinding」の場合は、WS-I Basic Profile に準拠したもので、HTTP を使用したテキストベースの、セッションを伴わない基本的なやり取りです。

このように、やり取りで必要な構成をする際には、わざわざプログラムコードには記述することはなく、予め決められた名前のバインディングの構成を指定するだけで済み、ソフトウェア開発者は、そのサービスに本来必要とするビジネスロジックの実装に専念することができます。

また、必要に応じて、独自のプロトコルや独自のエンコーディングを使用したカスタムバインディングも定義できます。

> **註:** 以下のアドレスには、予めシステムに定義されたバインディングの一覧や、カスタムバインディングの作成方法に関する記載があります。
> http://msdn.microsoft.com/ja-jp/library/ms731092.aspx
> システムが提供するバインディングの構成
> http://msdn.microsoft.com/ja-jp/library/aa347793.aspx
> カスタム バインディング

・クライアント（サービスにアクセスするための実装）および構成

次に、WCF クライアントの特徴的な部分を確認しましょう。図 1 のシステム全体像では、WCF クライアントは、B や C に当たります。既に触れたように、そもそも WCF クライアントは、.NET Framework 対応アプリケーションである必要はなく、様々な環境との相互運用性を備えていますが、クライアントも .NET Framework 対応のアプリケーションすることで、開発生産性の向上などの .NET の恩恵を受けることができます。また、図 1 の B や C は、一般的な PC での Windows アプリケーションや Web ページだけでなく、Windows Phone や Pocket PC などのモバイルデバイスも該当します。

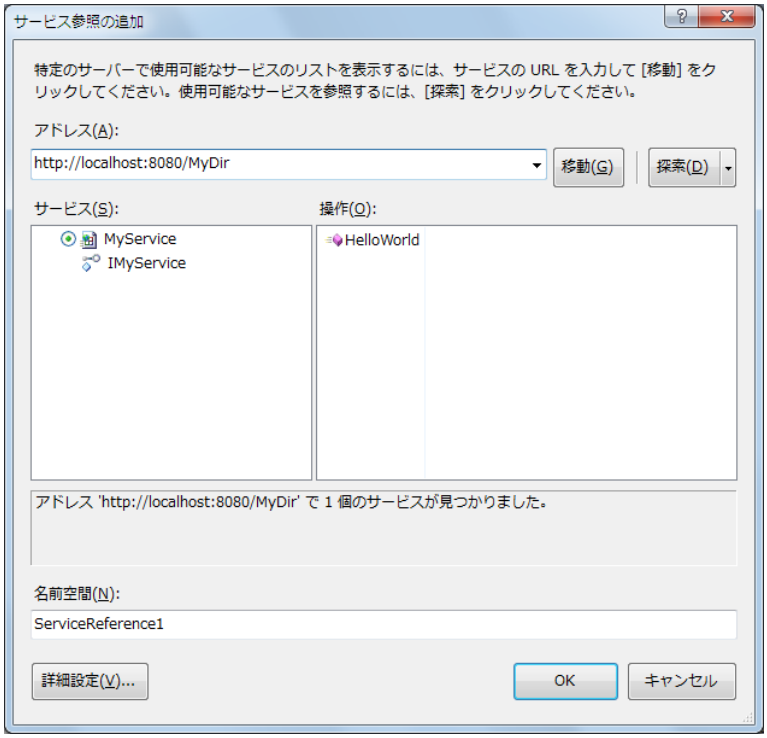
その実装手段にどのようなものがあるかまとめてみましょう。WCF クライアントの実装手段を便宜的に分類すると、以下のものが挙げられます。

表 2. 様々な WCF 向けクライアントの実装手段

分類	実装手段
①開発ツールよるプロキシ生成	Visual Studio の「サービス参照の追加」コマンドや、SvcUtil.exe を使用して、WCF サービスにアクセスするコード（プロキシ）を自動生成する。（内部的に WCF 関連ライブラリを使用）
②WCF 関連ライブラリ	たとえば、ChannelFactory クラス（チャネルファクトリ）など、Sytem.ServiceModel 名前空間の WCF 関連のクラスライブラリを使用する。
③汎用的なネットワーク関連のライブラリ	たとえば、WebClient クラスなど、System.Net 名前空間のネットワーク全般のクラスライブラリを使用する。
④クライアント スクリプト	ASP.NET AJAX において自動生成したプロキシを使用するなど、JavaScript を使用する

上記のうち①は、.NET Framework 環境で動作するクライアントアプリケーション開発において、最も開発生産性の高い実装方法です。もともと WCF サービスでは、サービスにアクセスする方法を WSDL や WS-MetaDataExchange を用いて、メタデータとして公開できます。表 2 の①の実装手段に挙げた「サービス参照の追加」というメニューコマンドから起動する対話形式のダイアログボックス（図 2）や SvcUtil.exe ツールを使用すれば、WCF サービスのアドレスを指定し、これらのメタデータを読み込ませることで、WCF サービスにアクセスするためのクライアントのコードを自動生成できます。

図 2. サービス参照の追加



このコードは、WCF 関連クラスである ClientBase<T>派生クラスであり、「プロキシ」と呼ばれています。このクラスのメソッドを呼び出すだけで、クライアント アプリケーションは WCF サービスを呼び出すことができます。たとえば、WCF サービスがサービス コントラクトに HelloWorld メソッドを公開していれば、クライアントはそのプロキシのメソッドを呼び出すだけです。

例 4. 参考: プロキシの利用

（C#）

```
var proxy = new MyServiceClient();
var str = proxy.HelloWorld();  //メソッドを呼び出す
```

（Visual Basic）

```
Dim proxy As New MyServiceClient()
Dim str = proxy.HelloWorld()   'メソッドを呼び出す
```

なお、この例のメソッド呼び出しでは、WCF サービスからの応答が返るまで、メソッドの中で待機し、応答が返ったらメソッドを抜け出す「同期呼び出し」です。プロキシは「非同期呼び出し」もサポートしています。

また、このような開発生産性の高いプロキシは、以下に挙げる様々な形態の .NET Framework 対応アプリケーションで利用できます。

表 3 様々な WCF クライアントのアプリケーション形態

分類	アプリケーションの利用形態
.NET Framework	一般的なデスクトップ PC やノート PC での利用
.NET Compact Framework	Pocket PC や従来の Windows Mobile Phone での利用
Silverlight 向け .NET Framework	一般的な PC のブラウザー上の Web ページや Windows Phone 7 での利用

次に、前述の表 2 の②の WCF 関連のライブラリを使用した実装方法についてですが、この方法ではチャネルファクトリと呼ばれるクラスを用いて、手動でコーディングを行うことになります。たたし、自動生成されたプロキシに比べると、よりきめ細かい制御が可能になります。クライアントチャネルは、表 3 に挙げたどの形態のアプリケーションでも実装できます。

また、プロキシを使用する場合も、チャネルファクトリを使用する場合も、クライアント側での構成が必要です。サービス側と同様に、サービスとやり取りするためのエンドポイントを構成する必要があります。クライアントの構成も、クライアント側の構成ファイルに記述できます。プロキシを自動生成する方法では、クライアントの構成ファイルも自動生成されます。

このほか、表 2 の③にあるように、.NET Framework の汎用的なネットワーク関連のライブラリを使用することもできます。たとえば、HTTP ベースの WCF サービスであれば、WebClient クラス、WebRequest クラス、WebResponse クラスなども利用できます。これらクラスも、通常の PC から Windows Phone 7 まで、表 3 に挙げたどの形態のアプリケーションでも利用できます。このクライアントの実装方法では、プロキシやチャネルファクトリと異なり、WCF 固有ではないため、WCF 固有のエンドポイントの構成や構成ファイルを用意する必要はありません。

また、表 2 の④にあるように、クライアントスクリプトから WCF サービスも利用できます。一般に、JavaScript 環境では XMLHttpRequest というオブジェクトがサポートされており、このオブジェクトを使用すると、HTTP ベースの WCF サービスとやり取りすることができます。また、ASP.NET AJAX が提供する ScriptManager コントロールを Web ページに埋め込み、いくつかの設定をすると、その Web ページから WCF サービスにアクセスするための、JavaScript 版のプロキシを自動生成させることもできます。また、オープンソースの JavaScript のライブラリである jQuery を使用することもできます。

以上、WCF が何であるのか、感触をつかんでいただくため、WCF サービスと WCF クライアントの簡単なコードをまじえながら、基本的な構成要素や、それぞれの特徴を確認しました。

2.3 WCF の様々な機能と実装

次に、WCF にはどのような機能や実装方法があるのか、そのバリエーションを確認します。第 2 部や第 3 部では、具体的なサンプルコードを用いて、改めて主な機能や実装の特徴について解説するので、ここでは WCF の主な機能の品揃えを挙げ、それぞれの特徴を確認しておきます。

・データ コントラクト

前述のサービスコントラクト（例 1）では、クライアントとサービスとの間において、メソッド呼び出しの際の引数や戻り値などのデータのやり取りがあります。これらのデータは、ネットワークに流す際にシリアル化（エンコード）され、受け取る側で逆シリアル化(デコード)されます。

どのようにエンコードされるかは、バインディングの構成しだいであり、WCF では XML データや JSON、バイナリなど様々なデータ形式をサポートしています。また、標準的なデータ型については、どのような構造にエンコードされるかが予め決められたおり、カスタムデータ型については、エンコード時の出力結果の構造を定義することができます。

このような決め事は「データ コントラクト」といいます。この WCF のデータ コントラクトはプログラマーが任意に属性として指定できる柔軟性を備えています。たとえば、次の例 5 の Product クラスを引数としてやり取りすることを考えます。このクラスは、Id、Name、Price の 3 つの要素がありますが、Id と Price のみに DataMember 属性が付いているので、例 6 のように、この 2 つだけが XML へシリアル化されます。もちろん、構成によっては、JSON 形式にシリアル化されることもあります。

例 5 参考: カスタムデータ型のデータ コントラクト

（C#）

```
[DataContract]
public class Product
{
    [DataMember]
    public int Id;

    public string Name;

    [DataMember]
    public int Price;
}
```

（Visual Basic）

```
<DataContract()>
Public Class Product

    <DataMember()>
    Public Id As Integer

    Public Name As String

    <DataMember()>
    Public Price As Integer

End Class
```

例 6. 参考: シリアル化された出力結果

```
<Product>
  <Id>3000</Id>
  <Price>5980</Price>
</Product>
```

・メッセージングの形態

WCF では、クライアント間とサービスとの間でデータ（メッセージ）のやり取りをする際に、単純な同期呼び出し形式の要求/応答のようなメッセージングだけでなく、単方向や非同期の双方向など、様々な形態がサポートされています。これらは、バインディングの構成によって変更することができ、サービスコントラクトでも特定のメッセージング形態を使用することを要件として宣言できます。

たとえば、次例の IsOneWay プロパティが付いた属性では、双方向のサービス構成では利用できず、単方向を使用することが強制されます。

例 7. 参考: 単方向を示すサービス コントラクト

（C#）

```
[ServiceContract]
public interface IMyService
```

```
{
    [OperationContract(IsOneWay = true)]
    void Say(string msg);
}
```

（Visual Basic）

```
<ServiceContract()>
Public Interface IMyService

    <OperationContract(IsOneWay:=True)>
    Sub Say(ByVal msg As String)

End Interface
```

また、このようなコントラクトでの宣言を行えば、プログラマーにとっては、単方向であることが明示され、可読性も向上するでしょう。さらに、WCF サービスのメタデータにも反映され、クライアントの開発時に自動生成されるプロキシにも、単方向のメッセージングを使用することが反映されます。

・キューと信頼されるメッセージ送信

WCF ではメッセージを確実に伝える仕組みが用意されています。

まず、サービスがオフラインでもクライアントから受け付けることができるよう、キュー（MSMQ）を利用できます。このとき、キューへのメッセージ投函やキューからの取り出しを行うコードを書く必要はありません。クライアントがサービスのメソッドを呼び出すと、キューにそのメッセージは投函され、サービスがオンラインの場合、キューから自動的に取りだされ、サービスが呼び出されます。

また、WCF では WS-ReliableMessaging がサポートされており、トランスポート プロトコルのエラーに関係なく、送信が保証される仕組みが用意されています。一般に、トランスポートでエラーが発生したら、単純な実装では、例外が発生してクライアントは例外を認識するだけで、特別な実装をしない限り、再送されません。WCF では、このような例外が発生したら、WCF の実行基盤がサービスとクライアントとの間で WS-ReliableMessaging に基づきハンドシェイクを行なうことができ、メッセージ送信を再試行するなどして、確実に送り届けてくれます。クライアントは、透過的に正常に送信されたものと認識します。

> **註:** 詳細や具体例な構成については、以下のアドレスを参照してください。
> http://msdn.microsoft.com/ja-jp/library/ms732355.aspx キューと信頼できるセッション

・同期と非同期

WCF サービスの実行時の動作は、サービスクラスのメソッドとして実装されます。この実装では、一般的な同期呼び出し形式のほかに、非同期呼び出し形式もサポートします。

一般に同期呼び出しでは、クライアントがメソッドを呼び出すと、サービス側で処理が終わって結果を返すまで、クライアントは待機します。これに対して非同期呼び出しでは、クライアントは要求を出したのち、待機せずに別の処理を行うことができます。一般に、この非同期呼び出しでは、処理開始のメソッドと結果受け取りのメソッドという、1 組のメソッドを使用します。処理開始のメソッ

ドを呼び出すと、サービス側に処理を開始させ、クライアントはブロックされずに、すぐに別の処理にできます。WCF サービスでは、コントラクトに非同期呼び出し形式のメソッドを明示的に定義できます。たとえば、次の例のようになります。この例では、AsyncPattern プロパティが True に設定された BeginAdd メソッドと、EndAdd メソッドが、1 組のメソッドとして非同期呼び出しのために使用されます。

例 8. 参考: 非同期呼び出しを明示したサービス コントラクト

（C#）

```
[ServiceContract]
public interface ICalcService
{
    [OperationContract(AsyncPattern=true)]
    IAsyncResult BeginAdd(int a, int b, AsyncCallback cb, object asyncState);

    string EndAdd(IAsyncResult r);
}
```

（Visual Basic）

```
<ServiceContract()>
Public Interface ICalcService

    <OperationContract(AsyncPattern:=True)>
    Function BeginAdd(ByVal a As Integer, ByVal b As Integer,
        ByVal cb As AsyncCallback, ByVal asyncState As Object) As IAsyncResult

    Function EndAdd(ByVal r As IAsyncResult) As String

End Interface
```

WCF で特筆すべき点の 1 つは、このようなサービス側での同期と非同期の実装とは独立して、クライアント側のプロキシでも同期と非同期の 2 つのパターンを実装できる点です。

つまり、サービス側で同期呼び出しとしてサービスを実装しても、プロキシでは、同期呼び出しと非同期呼び出しの 2 種類が実装できます。また、サービス側で非同期呼び出しとして実装しても、プロキシでは、同期呼び出しと非同期呼び出しの 2 種類が実装できます。両者間の呼び出し形式の差異は、WCF の実行基盤が吸収します。これによって、実装の柔軟性が生まれます。

たとえば、WCF サービス側が同期呼び出しの実装であって、その呼び出しの処理時間が長い場合は、サービス側の実装をリファクタリングしなくとも、クライアント側では、プロキシを非同期呼び出しとして実装できます。この場合、プロキシが非同期形式で呼び出しを開始すると、クライアント側のバックグランドスレッドによって、サービスの同期呼び出しを開始し、クライアントのプライマリスレッドは、待機せずに別の処理を続けることができるのです。

・セキュリティ

WCF では、認証、承認、また、暗号化や署名などの一般に要求されるセキュリティ機能を標準装備しています。第 2 部以降では、具体的なセキュリティの実装コード例も示すので、ここでは主な特徴を確認しておきます。

まず、WCF では、HTTPS などのトランスポートレベルのものだけでなく、メッセージレベルでのセキュリティもサポートしています。メッセージレベルの暗号化を使用すれば、サービスにメッセージを送信する際に、途中で中継役のサーバーがプロトコルを変更する場合でも、最終目的地のサービスに到達するまではメッセージは復号されず、中継サーバーは内容を盗み見ることができません。これによって、クライアントとサービスとの間で、エンド・ツー・エンドのセキュリティを実現できます。

また、これらが WCF によって提供されるので、プログラマーはセキュリティ自身を実装するコードをわざわざ記述する必要がありません。通常は、バインディングの構成や属性の指定などで、セキュリティの構成や調整を行います。そのほか、IIS をホスティング環境に使用すれば、IIS の認証などのセキュリティ機能や ASP.NET が提供するセキュリティ機能もそのまま併用することができます。

さらに、これらの WCF のセキュリティでは、トランスポートプロトコルの標準的なセキュリティ、また、SOAP メッセージや WS-*仕様に基づく標準的なセキュリティも含み、相互運用性にも優れています。

> **註:** WCF が提供するセキュリティのラインアップや、より具体的な解説は、以下のアドレスにも記載されています。
> http://msdn.microsoft.com/ja-jp/library/ms732362.aspx
> Windows Communication Foundation セキュリティ

インスタンス、セッション、および同時実行制御

通常のサービスは、複数のクライアント(ユーザー)から、それぞれ、複数回呼び出される場合があります。一般に、HTTP でのやり取りでは、1 つの要求／応答ごとに独立した「ステートレス」なやり取りであり、ユーザーごとに複数回の要求／応答を 1 つの論理的なまとまりとする「セッション」のような概念を備えていません。仮に、このような複数の要求／応答をまたぐセッションを維持する特別な仕組みがない場合は、プログラマーがセッションを維持するコードを書いたり、明示的にサービス側でサービスのインスタンスを常駐するコードを書いたりする必要があります。

しかし、WCF ではいくつかのバインディングではセッションサポートしており、構成ファイルでバインディングを指定することで、セッションを有効にすることができます。

また、前述のサービスクラスのインスタンスのライフタイムを属性レベルで管理することができます。呼び出し単位でインスンタスを作成したり、セッション単位でインスタンスを生成したり、ホストアプリケーションに 1 つのインスタンスのみ（シングルトン）を常駐されることもできます。

さらに、複数のユーザーがアクセスする環境では、1 つのサービスインスタンスを複数のスレッドから呼び出す場合があります。このとき、複数のスレッドの並行実行を認めるか、それとも、最大 1 つのスレッドの実行のみ認めるかなどの同時実行制御も指定することができます。

以下の例は、シングルトンで、複数のスレッドの並行実行を許可した例であり、属性のプロパティを用いて（InstanceContextMode プロパティ、ConcurrencyMode プロパティ）、宣言形式で指定できます。

例 9. 参考: インスタンスのライフタイムと同時実行制御の宣言的な指定

（C#）

```
[ServiceBehavior(
```

```
    InstanceContextMode = InstanceContextMode.Single,
    ConcurrencyMode=ConcurrencyMode.Multiple)]
public class CalculatorService : ICalcService
{
    //省略
}
```

（Visual Basic）

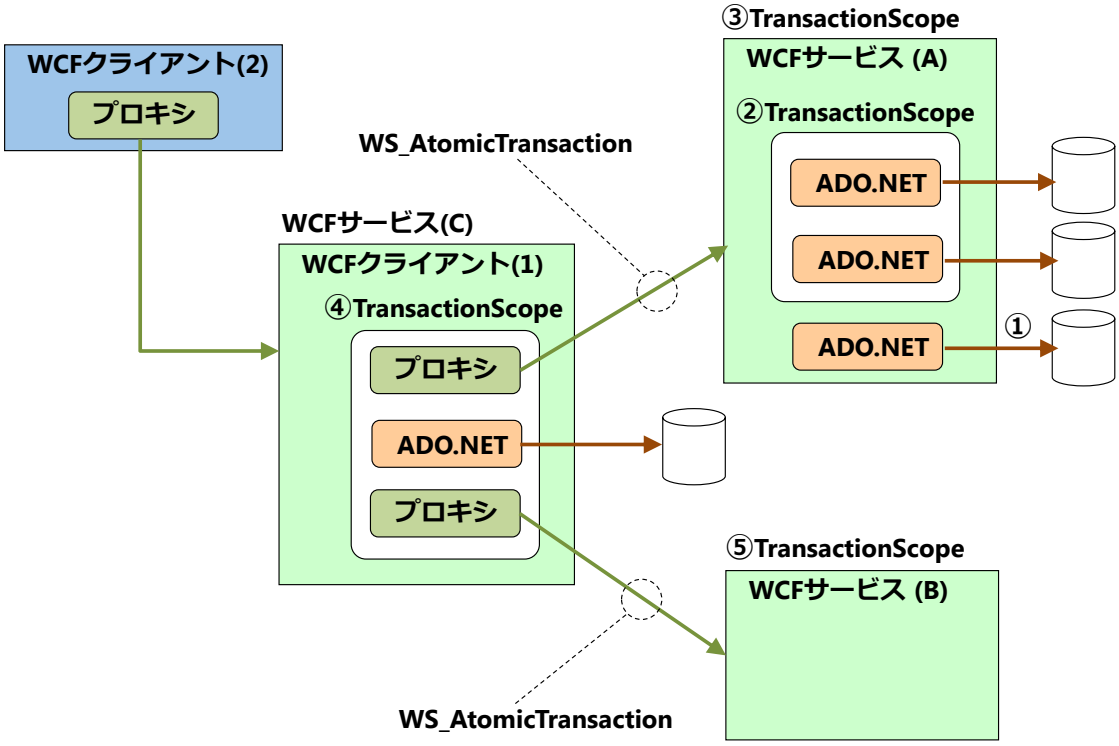
```
<ServiceBehavior(
    InstanceContextMode = InstanceContextMode.Single,
    ConcurrencyMode=ConcurrencyMode.Multiple)>
Public Class CalculationService
    Implements ICalcService

    '省略

End Class
```

> **註:** WCF におけるサービスインスタンス、セッション、同時実行制御については、以下のアドレスにも記載されています。
> http://msdn.microsoft.com/ja-jp/library/ms731193.aspx
> セッション、インスタンス化、および同時実行

・トランザクション

WCF サービスでは、様々な形でトランザクション操作を実装できます。次図では、WCF サービスを使用する中で、データベースへのアクセスを含むトランザクション操作の主なパターンを示しています。

図 3. WCF におけるトランザクションの利用



まずは、右上部の「WCF サービス（A）」のブロックに注目してください。①と②は、.NET Framework の一般的なトランザクションの仕組みを利用した方法です。

最も簡単な方法は、①のようにデータベースにアクセスする際に、ADO.NET のライブラリを用いて 1 つのリソースにアクセスしてトランザクションを制御する方法（ローカルトランザクション）です。また、②のように Enterprise Services（COM+）に基づく分散トランザクションを利用できます。このトランザクションでは、TransactionScope とよばれる 1 つの論理的なトランザクションブロックを構成し、その中で行う複数の分散リソースへのアクセスを、1 つのトランザクションとして構成できます。

さらに WCF では、③のように（図の右上部）、WCF サービス自体（正確には 1 回のサービスのメソッドの呼び出し自体）を、1 つの TransactionScope として構成することができます。そして、このように構成されたサービスは、④の WCF クライアント（図の中央）の TransactionScope の中から、プロキシを経由して呼び出すことで、④と③を 1 つのトランザクションに参加させることができます。このとき、③と④のやり取りには、トランザクションを実現するメッセージ交換の仕様として、WS_AtomicTransaction が利用できます。

④の観点からすると、④の TransactionScope のブロック内に示すように、③の WCF サービス呼び出しや、ADO.NET による直接のデータアクセス、また、⑤の WCF サービスの呼び出しなど、これらを 1 つの TransactionScope の管理下に置き、1 つのトランザクションとして制御することができるようになります。

また、図中央の WCF クライアント（1）は、実装できる環境が限られるので、実際のところ、④の実装を含む部分を WCF サービスとして公開し（WCF サービス（C））、図の左部分にある軽量なクライアント（2）からアクセスすることになるでしょう。

結局のところ、WS_AtomicTransaction を介して、WCF サービス（A）、WCF サービス（B）、および WCF サービス（C）の 3 つのサービスが、1 つのトランザクションの中で連携することになります。

なお、ここではトランザクション制御の対象として ADO.NET 対応のリソースを例に取りましたが、Enterprise Services に対応したいずれのリソースもこの仕組みに参加できます。たとえば、MSMQ なども、このトランザクションに参加できます。

> **註:** WCF におけるトランザクションについては、以下のアドレスにも記載されています。
> http://msdn.microsoft.com/ja-jp/library/ms730266.aspx トランザクション

・サービスの探索

WCF では、WS-Discovery をサポートしており、実行時に WCF クライアントはサービスを探索して見つけることができます。

これを行うために、WCF では探索用サービスを実装することが可能であり、WCF クライアントはそのサービスに問い合わせることで、必要な WCF サービスのエンドポイントを見つけることができます。

また、探索用サービスから逆に、「アナウンス メッセージ」というものを発信して、WCF クライアントに変更を通知することもできます。その結果、たとえば、WCF クライアントはアクセスすべきア

ドレスの変更を認識することができ、臨機応変にアドレスを変更するように実装しておくことが可能になります。

> **註:** WCF における探索については、以下のアドレスにも記載されています。
> http://msdn.microsoft.com/ja-jp/library/dd456782.aspx WCF Discovery

・ルーティング

WCF では、WCF クライアントが WCF サービスを呼び出すとき、クライアントからの要求を目的のサービスに伝える際の中継局となるサービスを実装することができます。

これを用いると、クライアントの要求に応じて異なるサービスへ振り分けることができ、複数の WCF サービスが存在する場合に、個々のエンドポイントを公開しなくとも、クライアントには単一の中継局のアクセスポイントを公開することができます。

また、この仕組みによって、複数のサービスへの負荷分散を実現したり、クライアントからの要求の優先度によって異なるサービスへ振り分けたり、サービス側でのバージョンアップに伴ってエンドポイントを入れ替えたりできます。これらのいずれの状況でも、WCF クライアントは WCF サービス側の変更や構成詳細を意識せずに、透過的にアクセスできます。

また、WCF クライアントがアクセスに使用するプロトコルを介して中継局が要求を受付け、それを別のプロトコルに変換して、WCF サービスを呼び出すこともできます。つまり、プロトコルブリッジとしての役割もあります。

> **註:** WCF におけるルーティングについては、以下のアドレスにも記載されています。
> http://msdn.microsoft.com/ja-jp/library/ee517421.aspx ルーティング

・管理と診断

WCF サービスでは、サービスの動作の分析や診断、管理のための機能が標準で提供されています。

これらを用いることで、開発時において品質の高いサービスを実現でき、また、運用時の管理も効率が向上します。また、このような管理機能をプログラマーがわざわざ実装する必要がないので、開発生産性を高めることができます。主な機能としては、以下のものが挙げられます。

（a）クライアントとサービス間でやり取りされるメッセージのトレース

（b）送信時、受信時のメッセージログ

（c）ETW 対応のイベント

（d）WCF 向けのパフォーマンスカウンター

（e）WMI に基づく WCF の構成管理

（e）構成や診断に関するツールの提供（構成エディター ツール、サービス トレース ビューアー）

> **註:** WCF における管理と診断については、以下のアドレスにも記載されています。
> http://msdn.microsoft.com/ja-jp/library/ms731055.aspx 管理と診断

・ワークフロー サービス ～Windows Workflow Foundation との連携～

ビジネスプロセスのワークフローを効率よく開発できる Windows Workflow Foundation（WF）には、WCF と連携する様々な仕組みが導入されています。.NET Framework 3.5 では、ワークフロー内の 1 つのプロセスであるアクティビティとして、WCF サービスや WCF クライアントを実装したアクティビティが標準装備されています。また、.NET Framework 4 では、ワークフローを WCF サービスとしてホスティングする仕組み（ワークフローサービス）も導入されています。これらの一連の仕組みを用いることで、WCF を用いたサービス連携の分散システムに、ワークフローベースのソリューションを導入することができます。

> **註:** Windows Workflow Foundations、および WCF におけるワークフロー サービスについては、以下のアドレスにも記載があります。
> http://msdn.microsoft.com/ja-jp/library/dd489441.aspx Windows Workflow Foundation
> http://msdn.microsoft.com/ja-jp/library/dd456788.aspx ワークフロー サービス

・WCF 配信

WCF には、Atom 形式や RSS 形式の配信フィードを扱うためのサービスを簡単に実装できる仕組みが用意されています。この仕組みのクラスライブラリを用いれば、配信フィードのデータ形式を意識せずに、配信すべきデータを通常のオブジェクトとして作成することができます。実際に配信する際には、WCF の実行基盤がこのオブジェクトを RSS 形式などのフィードへシリアル化してくれます。

また、Visual Studio 2010 にも、配信フィードを行うための WCF サービスを実装するプロジェクトテンプレートが用意されており、効率よく開発することができます。

> **註:** WCF 用いた配信フィードについては、以下のアドレスにも記載されています。
> http://msdn.microsoft.com/ja-jp/library/bb412202.aspx WCF 配信

・REST のサポート

WCF では、軽量で相互運用性が高いと言われている REST もサポートしています。詳しくは、次章で改めて概説するほか、第 3 部では具体例を紹介します。

・WCF RIA Services

WCF には、Silverlight などの RIA (Rich Internet Applications) 向けの N 層アプリケーション開発を簡素化するテクノロジが提供されており、これは「WCF RIA Services」と呼ばれています。

これについては、この第 1 部で章を設けて、改めて概説します。

・WCF Data Services

WCF では、データ提供に特化した WCF サービスを実装するためのテクノロジが提供されています。このテクノロジは、ADO.NET の拡張テクノロジとしての側面もあるので、以前は「ADO.NET Data Services と呼ばれていましたが、現在は「WCF Data Services」と呼ばれています。

これについては、この第 1 部で章を設けて、改めて概説します。

註: WCF Data Services については、以下のアドレスから自習書を入手することができます。
http://msdn.microsoft.com/ja-jp/data/gg615417 データ アクセス自習書
「データ アクセス自習書 (Visual Studio 2010/.NET4 版)」の中の
「第 3 部 SQL Server データ アクセス手法 - (4) WCF Data Services 編」

第 3 章 WCF REST オーバービュー

前章では、WCF の基本的な構成要素や、WCF が提供する各機能のラインアップなどについて、全般的な特徴を解説しました。この章では、WCF が提供する機能のうち、REST に関連する機能に焦点を当てて説明します。まずは、念のため REST とはそもそも何であるのか、その一般的な特徴を確認したのち、WCF において REST を提供するための基本的な実装方法やその特徴について確認します。

3.1 REST とは（一般論として）

REST とは Representational State Transfer の略です。これは特定の厳密な仕様を指している言葉ではなく、一般には、 SOAP メッセージは使わずに、標準的な HTTP 要求を利用してアクセスする軽量な仕組みを指しています。

ここで、SOAP と REST を対比しながら、軽量である REST の特徴を確認してみましょう。

通常、SOAP メッセージを使用してサービスの特定の機能を呼び出す場合、XML 形式のデータである SOAP エンベロープをクライアントが構築して、これをサービス側へ送信します。たとえば、次の例は、サービス側に加算を行う機能として Add メソッドが実装されており、クライアントがその機能を呼び出す場合の典型的なエンベロープの例です。

例 10. 参考: サービスの特定機能を呼び出す SOAP エンベロープ

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
  </s:Header>
  <s:Body>
    <Add xmlns="http://tempuri.org/">
      <a>10</a>
      <b>20</b>
    </Add>
  </s:Body>
</s:Envelope>
```

ここでは、5 行目の<Add>要素ブロックが呼び出すべき機能（メソッド）を表し、<Add>要素ブロックの内側にある<a>要素と<b>要素は、Add メソッドに渡される引数です。単に、2 つの引数を渡して Add メソッドを呼び出すだけですが、そのためにクライアントは、ここに挙げたように<Envelope>要素や<Header>要素、また<Body>要素など、決められたタグのメッセージを組み立ててサービスに送信する必要があります。

これに対して REST の場合は、サービスの機能を呼び出すために、アドレスを表す URI の一部に含めて要求します。たとえば、前述の SOAP エンベロープと同様に、Add メソッドを呼び出す場合には、次のようにアドレスを指定して、HTTP GET プロトコルメソッドを用いてサービスを呼び出すだけです。

例 11. REST ベースのサービスの特定機能を呼び出す URI

```
http://app.abcdef.com/Add?a=10&b=20
```

このように、アドレス内の仮想ディレクトリ名（Add）やクエリ文字列（?a=10&b=20）を用いて、呼び出すべき機能を指定します。クライアントは、本格的な SOAP メッセージを組み立てる必要はなく、アドレス指定を出来るクライアント環境であれば、このサービスを呼び出すことができます。

また、呼び出した結果として受け取る応答メッセージも、SOAP メッセージではなく、単純なテキスト形式のデータ（後述）を受けることができます。さらに、REST では SOAP や XML データを使用しないことから、それらに基づく WS-＊仕様も一般的に使用しません。

このように REST では、軽量で単純なデータのやり取りができるので、クライアントの要件にはそれほど縛りはなく、様々な種類のクライアントから同じサービスを利用することができます。Web ページ上の JavaScript などクライアントスクリプトからも、比較的簡単に利用できます。一般に、REST 対応のサービスを使用するケースは、Web ブラウザーを含め、より多くの種類のクライアントからアクセスさせる状況にあるときです。

なお、サービスが REST 対応である状況のことを、「RESTful」と表現することがあります。WCF サービスも RESTful なサービスとして実装することができます。次項では、RESTful な WCF サービスの実装の特徴について確認していきます。

3.2 WCF における REST の実装概要と開発環境

WCF では、REST に対応したサービス、すなわち、RESTful なサービスを簡単に実装することができます。REST に対応させるには、構成ファイルなどで、REST 向けのバインディングを選択するように構成して、サービスコントラクトに専用の属性を付けるだけです。ホスト アプリケーションやサービス コントラクト、サービスクラスの基本的な構成や実装方法は、前章「WCF オーバービュー」で説明した方法と同様です。このあとは、REST 対応の実装における特徴的な面をいくつか確認しましょう。

> **註:** .NET Framework 4 の WCF では、REST 対応の一連の仕組みのことを「WCF Web HTTP プログラミング モデル」と呼んでいます。MSDN のドキュメントなどで関連事項を調べる際には、この用語を検索キーワードにしてみてください。

・宣言的な属性による指定

WCF のサービス コントラクトでは、特定の URI の HTTP 要求に対して、どのメソッドを呼び出すのかという対応付けを、属性を用いて宣言的に行うことができます。次に例を示します。

例 12. 特定の URI とメソッドとの対応付け

```
[ServiceContract]
public interface ICalcService
{
    [OperationContract]
    [WebGet(UriTemplate = "Add?a={a}&b={b}")]
    int Add(int a, int b); int Add(int a, int b);
}
```

ここでは、Add メソッドに WebGet 属性が付いています。この属性の詳細は、第 3 部で改めて扱うので、ここでは基本的な特徴を読み取ってみましょう。

特に、WebGet 属性の UriTemplate プロパティに指定された "Add?a={a}&b={b}" は、この Add メソッドを呼び出す際に、どのように URI に指定するか、その相対 URI が記述されています。この例の場合、例 11 のようにアドレスを指定して呼び出すことができます。この WCF サービスを実装する際には、プログラマーがわざわざ HTTP 要求を解析して、呼び出すべき機能を決定する必要はありません。このほか、WebInvoke 属性を代わりに使用すると、HTTP POST や HTTP PUT、HTTP DELETE などとも対応付けることができます（これらも詳しくは第 3 部で取り上げます）。

・軽量なメッセージへのシリアル化

WCF の REST では、データをネットワーク上に送信するために、そのデータをシリアル化する際には、XML 形式のほか、JSON（JavaScript Object Notation）形式にも対応しています。

JSON は、JavaScript の構文に準じたデータの表現形式であり、XML に比べると、非常に軽量なデータ表現です。たとえば、例 5 のデータコントラクトのように定義された Product クラスのオブジェクトが、JSON 形式にシリアル化されると、次のようなテキストになります。

例 13. JSON 形式にシリアル化された Product オブジェクト

```
{"Id":3000,"Price":5980}
```

この例のように、単純な「名前:値」の対になった項目が、カンマを区切り文字として列挙された形式になります。このような簡単な構造であれば、多くのクライアント環境でデータを読み取ることが可能でしょう。特に、クライアント側で JavaScript を用いていた場合には親和性があり、JavaScript の eval 関数を用いて、簡単にオブジェクトに逆シリアル化できます。例 13 の場合は、Id と Price という 2 つのメンバーを持つオブジェクトになります。

・REST における WCF のホスト環境

ホスト アプリケーションに関しては、バインディングの構成を REST 対応にすればよく、REST 向けの特別なコードを記述する必要はありません。前章の「2.2 WCF（基本部分）の実装概要と開発環境」の中の「ホストアプリケーション」で触れた（a）自己ホストから（d）Windows プロセス アクティブ化サービスまで、いずれの形態も利用できます。

> **註:** ただし、WCF には前章の例 2 で取り上げたホストアプリケーションのクラス ServiceHost 以外にも用途に応じたクラスがあり、REST 向けに特化した WebServiceHost クラスも用意されています。このクラスには、予め REST に関する構成がされており、このような特定のホスト用のクラスを使用すると、構成はより簡潔にできます。WebServiceHost クラスは第 3 部で取り上げます。

もちろん、IIS 上で動作する ASP.NET Web アプリケーションの環境と共存できるので、REST 対応のサービスも、ASP.NET の認証やセッションを共有できます。典型的なパターンとしては、図 1 の.NET Framework を用いたシステムの全体像のうち、D と E を IIS 上の 1 つのアプリケーション環境に共存させ、B のクライアントの Web ページから、JavaScript を使用して、IIS 上の REST 対応サービスにアクセスする方法です。また、B のクライアントに挙げた Web ページ上の Silverlight からもアクセスできます。

ただし、Silverlight の場合には、サービス側の別の選択肢として、WCF RIA Services や WCF Data Services を使用する方法もあります。これらは次章以降で補足説明をします。

なお、REST はクライアントに汎用性がある点は優れていますが、そのトレードオフとして、セキュリティ面では、SOAP を使用しないこともあり、WS-＊仕様に準じたメッセージ レベルでの暗号化はできません。しかし、IIS 上で SSL を用いた接続を使用することで、REST の場合もトランスポート レベルの暗号化を行うことがとでき、一般的な Web アプリケーションにおけるセキュリティを確保できます。

・様々なクライアント

改めて、REST 対応サービスの WCF クライアントの実装手段の選択肢を確認しましょう。

REST 対応サービスのクライアントを構築する場合、もちろんメタデータからプロキシを生成する方法もあります。しかし、REST 対応サービスの特徴の一つは、様々なクライアントで軽量な実装が可能である点なので、やはり典型的な実装方法としては、簡単なプログラムコードを使用して、HTTP GET などをクライアントから送信する方法が挙げられるでしょう。

表 2 に当てはめると、③の汎用的なネットワーク関連のライブラリを使用する方法、および、④のクライアントスクリプトにおいて、XMLHttpRequest オブジェクトを直接使用する方法などが該当します。もちろん、他の WCF サービスと同様に、①のプロキシや②のチャネルファクトリを使用する方法も利用できます。

参考までに例を挙げると、③のネットワーク関連のライブラリ（WebClient クラス）を使用して、例 12 のサービスコントラクトにある Add メソッドを呼び出す場合、クライアント側のサービスを呼び出す部分のコードの断片は、次のようになるになります。

例 14. 参考: WebClient クラスを使用して REST 対応のサービスにアクセスする

（C#）

```
int a = 10, b = 20;
WebClient client = new WebClient();
Stream data = client.OpenRead(
    String.Format(
        "http://localhost:8085/WebSite1/MyService.svc/Add?d1={0}&d2={1}", a, b));
StreamReader reader = new StreamReader(data);
string result = reader.ReadToEnd();
```

（Visual Basic）

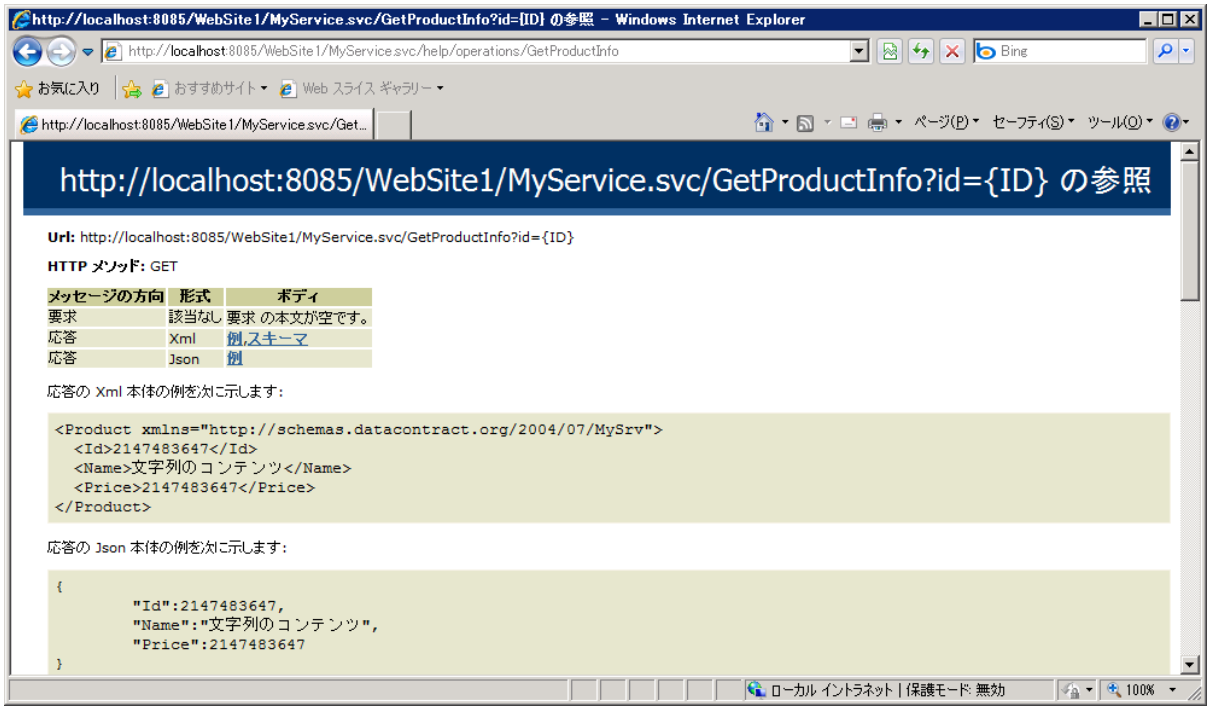
```
Dim a As Integer = 10, b As Integer = 20
Dim client As New WebClient()
Dim data As Stream = client.OpenRead(
    String.Format(
        "http://localhost:8085/WebSite1/MyService.svc/Add?d1={0}&d2={1}", a, b))
Dim reader As New StreamReader(data)
Dim result As String = reader.ReadToEnd()
```

実際にサービスを呼び出すのは 3 行目の OpenRead メソッドであり、サンプルコードの最終行で、サービスからの応答を文字列として取得するまで、わずか数行で済みます。（HTTP 応答としての Add メソッドの戻り値として JSON 形式を採用している場合、スカラー値の戻り値は、単純なテキストとして返ります。）

なお、例 14 で使用した WebClient クラスは、WCF 固有のクラスではない点を思い出してください。WCF が登場する .NET Framework 3.0 よりも以前の環境でも利用できます。そういう意味で、この方法は .NET Framework の中でも汎用性のある手法です。

また、以前のバージョンの REST 対応の WCF サービスでは、人が可読できるヘルプページが提供できませんでしたが、.NET Framework 4 では新機能として、REST 対応のヘルプページが提供できるようになりました。クライアント側でメタデータによるプロキシ生成を行わず、手動でコードを記述する際に、このページを参照すれば、クライアントからどのようにアクセスして、どんなデータを交換するのかという点を把握でき、開発作業の効率も向上します。図 4 のヘルプページには、アクセスする際のアドレスや、やり取りされるデータ形式、スキーマなどが掲載されています。

図 4. REST 対応の WCF サービスが提供するヘルプページ



> **註:** このほか、.NET Framework 4 では REST に関する新機能がいくつか追加されています。詳しくは、以下のアドレスのページ内の「WCF REST」という項を参照してください。
> http://msdn.microsoft.com/ja-jp/library/dd456789.aspx
> Windows Communication Foundation の新機能

・開発環境

WCF サービスを REST 対応にするには、前章の「2.2 WCF（基本部分）の実装概要と開発環境」で説明したものと同様であり、特別な違いはありません。Visual Studio 2010、または Visual Studio 2008 を利用できます。

ただし、WCF が利用可能な .NET Framework 3.0 からですが、REST がサポートされたのは .NET Framework 3.5 からなので、Visual Studio 2010 で作成するプロジェクトのターゲットを、3.5 または 4 に設定する必要があります。

以上、WCF における REST の特徴について確認しました。のちほど第 3 部では、具体的な実装例を紹介していきます。

第 4 章 WCF RIA Services オーバービュー

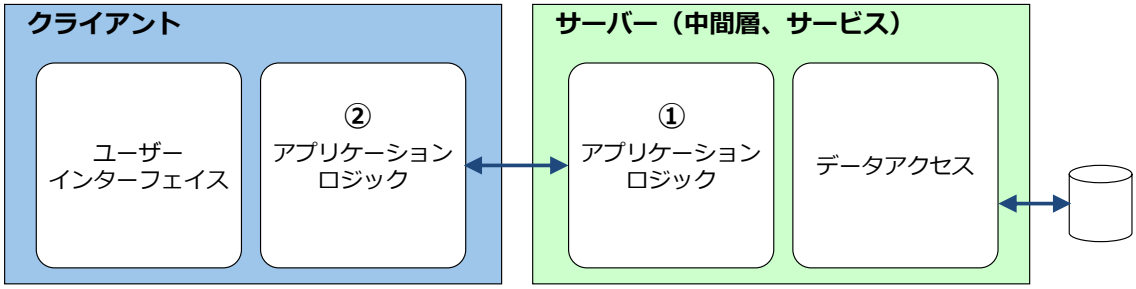
この章では、WCF が提供する一連のテクノロジのうち、WCF テクノロジをベースにした一種の拡張テクノロジと見ることができる「WCF RIA Services」について概説します。まずは、WCF RIA Services が何であるかという点も含め、基本的な特徴や用途を確認したのち、WCF RIA Services を用いた基本的な実装方法のパターンや特徴について確認します。

4.1 WCF RIA Services とは

既に触れたように、WCF RIA Services とは、Silverlight などの RIA (Rich Internet Applications) 向けの N 層アプリケーション開発を簡素化するための、WCF ベースのテクノロジです。N 層のどの部分で開発生産性を向上させるか確認してみましょう。

次図は、一般的な N 層アプリケーションについて、中間層やクライアントを構成する部分を簡略化したものです。

図 5. 一般的な中間層とクライアント

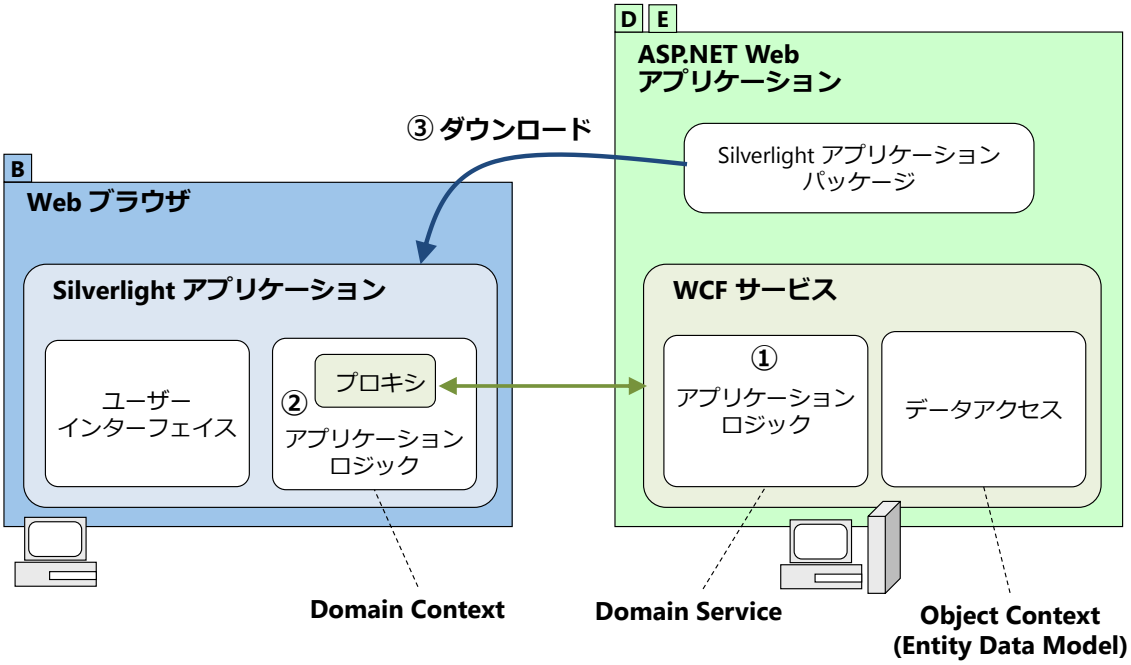


この図に示すように、サーバー側では中間層のロジック（①）を作り込むほか、それに連動したクライアント側のロジック（②）も作成する必要があります。また、保守を行う段階では、サーバー側の中間層のロジックが変更されると、それに伴って、クライアント側のロジックを変更する必要が生ずる場合があります。

WCF RIA Services を使用すると、これらの開発効率化を図ることができます。WCF RIA Services には、この部分のためのフレームワークやツール、また、中間層の部分を WCF サービスとして構築するための仕組みが用意されています。WCF RIA Services のクラスライブラリを使用すれば、これらの両者のロジックを含む連携部分のコーディングを簡素化することができます。また、Visual Studio 2010 向けの WCF RIA Services 用プロジェクトテンプレートも用意されており、このプロジェクトテンプレートには、前図のように構成された実装部分のひな形も用意されています。

この WCF RIA Services を使用してシステムを構成する場合、上記の図 5 の構成を、図 1 のシステム全体の構成図に当てはめてみると、典型的な構成は次図のようになります。以下の図 6 の①と②の実装は、図 5 の①と②に相当する部分です。

図 6. WCF RIA Services を用いた典型的な構成



中間層のサービスは、図 1 の全体図では E の WCF サービスであり、ごく一般的な実装の 1 つは、図 6 のように D と E とを 1 つの Web アプリケーションに共存させ、ASP.NET の認証やセッションを共有する方法です。また、B の部分のクライアントでは、ブラウザー上に表示された Web ページに Silverlight アプリケーションを含んでおり、この部分にクライアントとしてのロジック（②）が実装されています。

ただし、B の Web ページを開いた際には、Silverlight アプリケーションは Web サーバーである D からダウンロードされるので（③）、結局のところ、クライアントのロジックもサーバー側で管理できます。また、サービス側の①の部分では、Domain Service と呼ばれる実装を使用します。この Domain Service は、サービス側のビジネスロジックをカプセル化したオブジェクトです。既定では データアクセスを行う実装として、ADO.NET Entity Framework の Entity Data Model に準拠した ObjectContext を利用できる、Domain Service 用クラスが用意されています。

また、②の部分としては、サービス側の Domain Service にクライアントからアクセスするための実装（Domain Context）が用意されています。これらの実装は、WCF RIA Services のフレームワークとしてほとんど用意されているので、プログラマーはアプリケーション固有の実装に専念できます。

さらに、Visual Studio 2010 向けの WCF RIA Services 対応プロジェクトを使用すると、この一連の構成を行うための各種テンプレートが用意されており、効率よく、WCF RIA Services を用いたアプリケーションを構築することができます。

註: WCF RIA Services は Silverlight 向けに最適化された実装ではありますが、Silverlight 以外でもクライアントを実装することができます。たとえば、ASP.NET Web ページ上で、DomainDataSource コントロールを使用すると、サーバーサイドで Web ページが構築される際に、Domain Service からデータを取り込み、同じ Web ページ上の GridView コントロール（クリッド形式の表示の UI）にデータを表示できるようになります。このような Silverlight 以外のクライアントの扱いについては、以下のアドレスに記載があります。
http://msdn.microsoft.com/ja-jp/library/gg602749(VS.91).aspx
Silverlight 以外のクライアントへのアクセス

4.2 WCF RIA Services の実装概要と開発環境

本ドキュメントでは、WCF RIA Services に関するコーディング方法の詳細については割愛しますが、ここで、WCF RIA Services に対応するアプリケーションを実装する上での開発環境や実装の特徴について確認します。

・開発環境

WCF RIA Services に対応した WCF やそのクライアントの Silverlight アプリケーションには、実行環境として Silverlight 4 のランタイムおよび、WCF RIA Services の実装が必要です。また、Visual Studio 2010 を用いて、対応するサービスやクライアントを開発する場合には、これらの実行環境のほかに、WCF RIA Services 向けのアドオンツールを Visual Studio 2010 へ追加する必要があります。

これらの実行環境や開発環境が 1 つになったものが、以下の名前でマイクロソフト ダウンロード センターに公開されています。

Microsoft Silverlight 4 Tools for Visual Studio 2010

これには、Silverlight 4 のライタイム、WCF RIA Service で必要となる Silverlight 4 向けのクラスライブラリのほか、Silverlight 4 に対応した Visual Studio 2010 のデザイナーや、WCF RIA Services 向けのアプリケーションを作成するためにプロジェクトテンプレートが拡張され、そのほか関連するテンプレートも追加されます。Visual Studio 2010 のほか、Visual Web Developer 2010 Express でも利用できます。

・サービスとクライアントの同時開発と自動連携

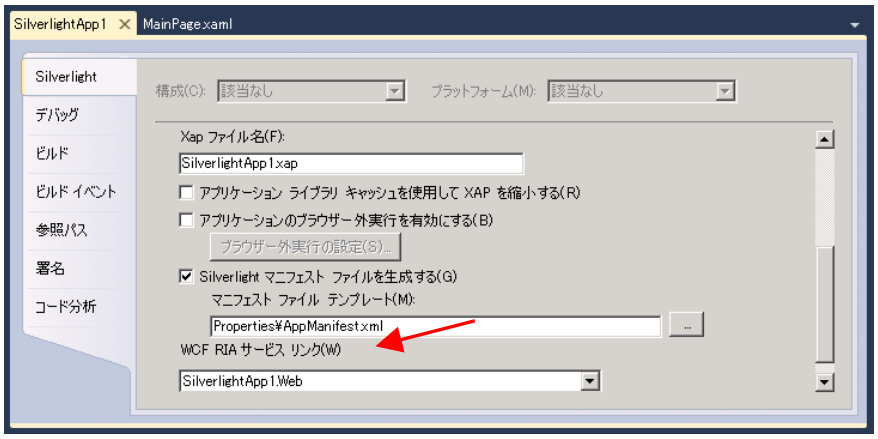
前述のツールをインストールしたのち、Visual Studio 2010 の Silverlight アプリケーション プロジェクトは Silverlight 4 対応になります（Visual Studio 2010 をインストールした直後は、Silverlight 3 対応です）。このテンプレートを選択してプロジェクトを新規作成した場合、Silverlight アプリケーション プロジェクトの一般的な構成として、次の図 7 のようなに 2 つのプロジェクトとなります。

図 7. Silverlight アプリケーションの一般的なプロジェクト構成

これらは、クライアントの実装にあたる Silverlight アプリケーションと、それを提供するサーバー側の ASP.NET Web アプリケーションのプロジェクトです。もともと、この後者のサーバー側のプロジェクトは、Silverlight アプリケーションをダウンロードさせるための Web サイトに当たりますが、このプロジェクトに、WCF RIA Services を実装することで、図 5 や図 6 に挙げたようなクライアントと中間層の実装を、1 つのソリューションの中で開発することができます。

特に注目すべき点は、Silverlight アプリケーションのプロジェクトのプロパティには、次図に示す設定項目（WCF RIA サービス リンク）がある点です。

図 8. WCF RIA Services にアクセスすることを有効化するオプション

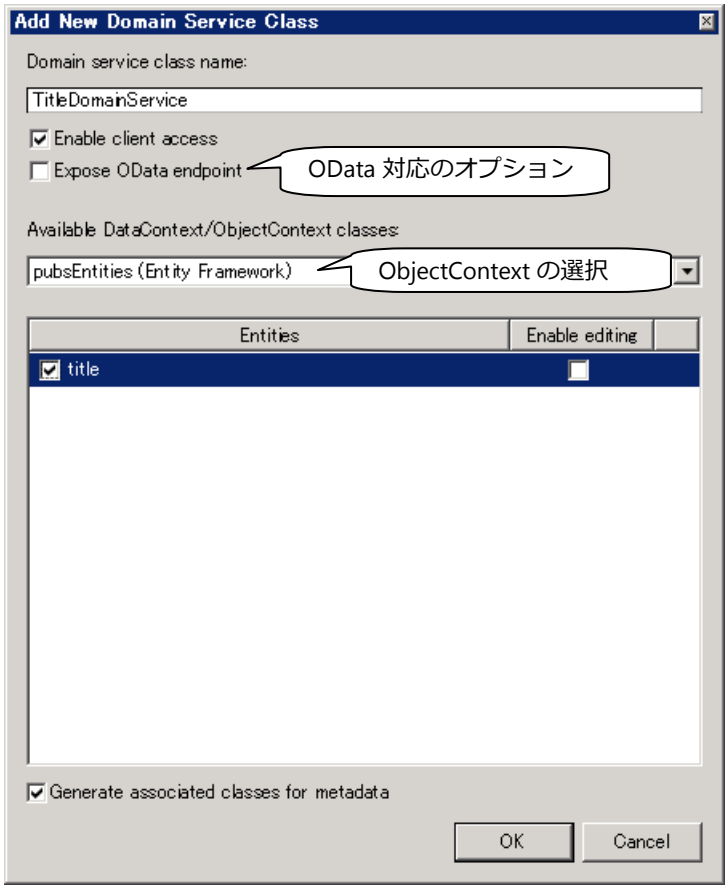


この項目には、クライアントからアクセスすべき WCF RIA Services を提供するプロジェクトを指定します。これを指定した状態でビルドを行うと、サーバー側の WCF RIA Services にアクセスするためのプロキシがクライアントプロジェクトの中に自動生成されます。つまり、クライアントとサービスとの連携をするコードが自動生成されるのです。

Domain Service とデータ操作の提供

既に触れたように、WCF RIA Services では、サービス側のロジックを Domain Service として実装します。前述のアドオンツールをインストールすれば、Visual Studio 2010 の項目テンプレートには、Domain Services のひな形も追加されています。このテンプレートを追加する過程では、次図に示す設定画面が表示されます。

図 9. Domain Service の新規追加



この画面では、このテンプレートを使用する際には、Domain Service の操作対象として Entity Framework のエンティティに関する操作を行う ObjectContext を選択することもできます。もちろん、ここで使用する ObjectContext は、Entity Framework でエンティティのデザインをビジュアルに行う EDM デザイナーで作成したものを流用できます。

> **註:** ADO.NET Entity Framework については、次のアドレスから自習書を入手することができます。
> http://msdn.microsoft.com/ja-jp/data/gg615417 データ アクセス自習書
> 「データ アクセス自習書 (Visual Studio 2010/.NET4 版)」の中の
> 「第 3 部 SQL Server データ アクセス手法 - (3) ADO.NET Entity Framework 編」

そのほか、REST ベースのデータ操作を行う OData (Open Data Protocol) に対応するオプションも用意されています。また、WCF RIA Services のエンドポイントは、WCF サービスに基づいており、データ形式として、バイナリ、JSON、SOAP、また単純な XML など、様々なデータを利用することができます。

註: OData (Open Data Protocol) は様々な環境でデータを共有するための規約です。SharePoint Server 2010 をはじめ、多くのマイクロソフト製品で採用されています。加えて、他社製品での採用もすすめられているほか、JavaScript や PHP などの言語に対応したライブラリが提供されています。詳しくは次のアドレスをご覧ください。

http://www.odata.org/ Open Data Protocol(英語)

・ユーザーインターフェイスとの容易な連携

WCF RIA Services では、WCF サービス側の Domain Service が提供するデータをクライアントが受け取り、クライアント アプリケーション上のユーザーインターフェイスに対して、容易にデータバインディングを行うことができます。

そもそも、従来から Silverlight アプリケーションでは、WCF サービスに問い合わせてデータなどを取得する場合、非同期呼び出しが用いられます。つまり、WCF サービスに問い合わせてから、その応答が返るまでの間、Silverlight アプリケーションは待機しているわけではなく、呼び出しそのものがバックグランドで行われます。そのため、WCF サービスからの応答が返るまでの間、Silverlight アプリケーションのユーザーインターフェイスがハングすることはありません。

このことは、応答性のよいユーザーインターフェイスを実装する上で役立つ仕組みですが、その反面、本来は WCF サービスの呼び出しに関する実装が、やや複雑になりました。WCF サービスを呼び出す際に、同期呼び出しのように単純なプロキシのメソッド呼び出しではなく、WCF サービスを呼び出したのち、WCF サービスからの応答をイベントとして受け取るなどの実装が必要でした。

しかし、WCF RIA Services では、このような非同期呼び出しもカプセル化した実装を提供しており、クライアントはこれを利用することで、次例のように、非同期呼び出しを伴うデータバインディングをわずか数行で実現することができます。

例 15. 非同期呼び出しによって取得するデータとのバインディング（WCF RIA Services の場合）

（C#）

```
private PubsDomainContext dc = new PubsDomainContext(); //←[1]

public MainPage()
{
    InitializeComponent();

    LoadOperation<title> loadOp = dc.Load(dc.GetTitlesQuery()); //←[2]
    dataGrid1.ItemsSource = loadOp.Entities;                    //←[3]
}
```

（Visual Basic）

```
Private dc As New PubsDomainContext()          '←[1]

Public Sub New()
    InitializeComponent()

    Dim loadOp As LoadOperation(Of title) =
        dc.Load(dc.GetTitlesQuery())           '←[2]
    dataGrid1.ItemsSource = loadOp.Entities    '←[3]
End Sub
```

詳細の手順は割愛しますが、特徴だけ確認しておきましょう。これは、Silverlight アプリケーションの上にある DataGrid（dataGrid1）に対して、WCF サービス側から取り込んだテーブルを表示するためにバインディングを行っている例です。必要なのは、[1]から[3]までの 3 つの文だけです。特に重要な点は、クライアントでは、サービス側が提供する Domain Service へ操作をカプセル化した DomainContext オブジェクト（[1]のオブジェクト）を使うことができる点であり、このオブジェクトはプロキシとして非同期呼び出しも内部的に行うので、プログラマーがわざわざサービスにアクセスするための非同期の手順を記述する必要がありません。参考までに、次章の WCF Data Services には、非同期呼び出しの典型的なサンプルコード（例 16）を掲載しましたが、それに比べると、特定のオブジェクトをプロパティに設定するだけなので、非常に簡潔であることが分かります。

以上、開発者の観点から、WCF RIA Services の特徴について確認しました。これを利用することで、Silverlight クライアントからサービスに対してデータ操作を行う実装を簡潔に実現することができます。

註: WCF RIA Services については、次のアドレスから学習を目的としたサンプルアプリケーションを入手することができます。
http://msdn.microsoft.com/ja-jp/silverlight/hh219352
Visual Studio 2010/Silverlight 4 ソリューション サンプル (MVVM) 編　～ Silverlight 4 + WCF RIA Services + MVVM ～

第 5 章 WCF Data Services オーバービュー

この章では、WCF が提供する一連のテクノロジのうち、WCF テクノロジの拡張テクノロジと言える「WCF Data Services」について概説します。まずは、WCF Data Services が何であるかという点も含め、基本的な特徴や用途を確認したのち、WCF Data Services を用いた基本的な実装方法の特徴について確認します。

> **註:** WCF Data Services については、以下のアドレスから自習書を入手することができます。
> http://msdn.microsoft.com/ja-jp/data/gg615417 データ アクセス自習書
> 「データ アクセス自習書 (Visual Studio 2010/.NET4 版)」の中の
> 「第 3 部 SQL Server データ アクセス手法 - (4) WCF Data Services 編」

5.1 WCF Data Services とは

WCF Data Services とは、データに関する操作を提供することを主な目的とした WCF サービスを構築するための、.NET Framework テクノロジの 1 つです。WCF Data Services には、REST ベースのデータ操作を行う OData（Open Data Protocol）に基づく WCF サービスを構築するためのフレームワークが用意されており、これを用いることで、クライアントがデータの操作を行うサービスを効率よく開発することができます。

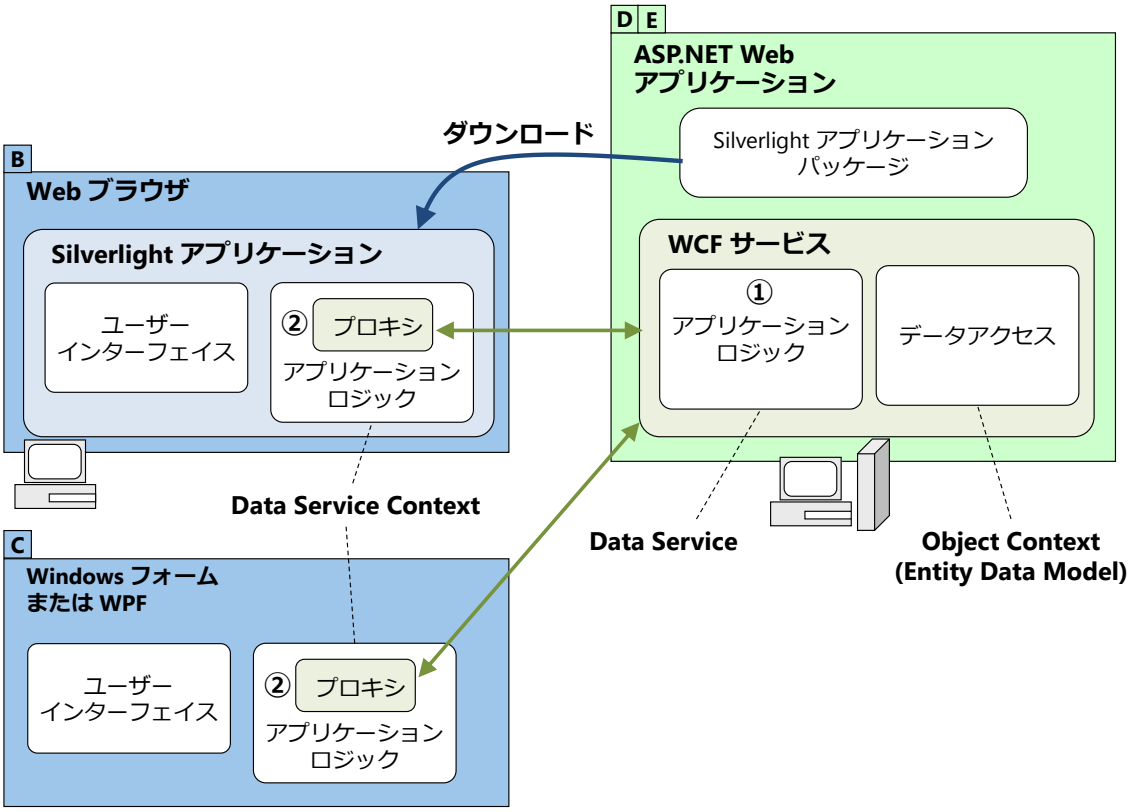
WCF Data Services に基づいて実装したサービスは REST 対応であるので、一般的に使用される HTTP GET などプロトコルが利用できればよく、やり取りされるデータも、テキストベースの ATOM や JSON を使用するので、それほど複雑な実装環境を必要しません。よって、クライアント側は .NET Framework に限らず、幅色い環境で利用できます。ただし、WCF Data Services では、 .NET Framework ベースのクライアント実装のための仕組みやライブラリも用意されており、これらを用いることで、クライアント側の開発生産性を向上させることができます。WCF Data Services 向けのクライアント側のライブラリは、通常の PC 向けの .NET Framework 3.5 SP1 以降のほか、Silverlight 版の .NET Framework 3 以降にも用意されています。

また、サービス側では、前章の WCF RIA Services と同様に、直接データアクセスを担当する実装は、ADO.NET Entity Framework の Entity Data Model に準拠した ObjectContext を利用できます。また、これ以外にも、このデータアクセスに相当する実装を構築するフレームワークとして「データ サービス プロバイダー」や「リフレクション プロバイダー」と呼ばれるものが用意されています。

> **註:** データ サービス プロバイダーに関する全般的な情報は、以下のアドレスを参照してください。
> http://msdn.microsoft.com/ja-jp/library/dd672591.aspx
> データ サービス プロバイダー (WCF Data Services)
>
> http://msdn.microsoft.com/ja-jp/library/dd723653.aspx
> リフレクション プロバイダー (WCF Data Services)

これらをふまえ、WCF Data Services を利用する様子を図 1 のシステム全体構成に当てはめると、次図のようになります。

図 10. WCF Data Services を用いた典型的な構成



前章の WCF RIA Services の図 6 の構成と非常に類似しています。違いは、Silverlight に限定していない点です。もともと、WCF Data Services のほうが先に登場しているので、WCF RIA Services は WCF Data Services と同様の構成で、より Silverlight 向けに最適化しているという見方もできるでしょう。

図 10 の WCF Data Services の場合も、WCF サービス側のアプリケーション ロジック（①）は、専用の Data Service と呼ばれるクラスが用意されており、[B]や[C]のクライアント側のアプリケーション ロジック（②）も、相応の一連のクラスが用意されています。これらは、Visual Studio 2010（または Visual Studio 2008 SP1）の項目テンプレートとして用意されていて、プログラマーが追加する固有部分はわずかです。

WCF Data Services では、上図の構成パターンに当てはまるような場合において、非常に少ないコード量で、目的のシステム構成を実現することができます。なお、この図では割愛しましたが、このサービスは REST ベースで公開されるので、HTTP プロトコルによる基本的な操作（HTTP GET など）ができる環境であれば、どのようなクライアントからもアクセスすることが可能です。

5.2 WCF Data Services の実装概要と開発環境

次に、WCF Data Services に対応するアプリケーションを実装する上での開発環境や実装の特徴について確認します。

・開発環境

WCF Data Services に対応した WCF サービスは、.NET Framework 3.5 SP1 以降が必要であり、開発のためには、Visual Studio 2008 SP1 以降が必要です。（正確にいえば、.NET Framework 3.5 SP1 や Visual Studio 2008 SP1 で作成する場合は、ADO.NET Data Services と呼ばれています。）もちろん、この自習書の前提環境である Visual Studio 2010 でも利用できます。

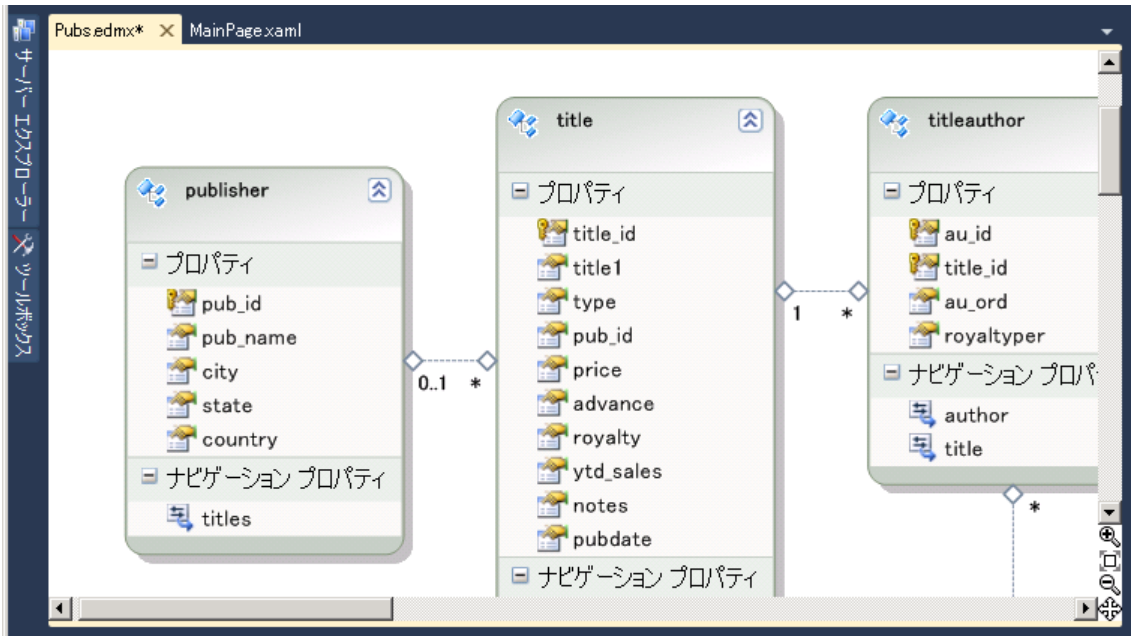
クライアント用の WCF Data Services のライブラリを使用する場合も同様です。なお、Silverlight の場合は、Silverlight 3 以降が必要になります。Visual Studio 2008 の場合は、別途、Silverlight 3 向けのアドインが必要です。Visual Studio 2010 では、既定で Silverlight 3 の開発環境が備わっています。

・Visual Studio の各種テンプレートやウィザードによるサービスの効率的な開発

図 10 の構成を構築する際に Visual Studio を用いれば、その構成の主要部分は、項目テンプレートやツール類を利用して、自動的に生成することができます。

たとえば、図 10 の WCF サービスにおけるデータアクセスの実装として、Entity Data Model を使用するのであれば、Visual Studio 上でビジュアルにデータ モデルをデザインする EDM デザイナーが利用できます（図 11）。

図 11. EDM デザイナー

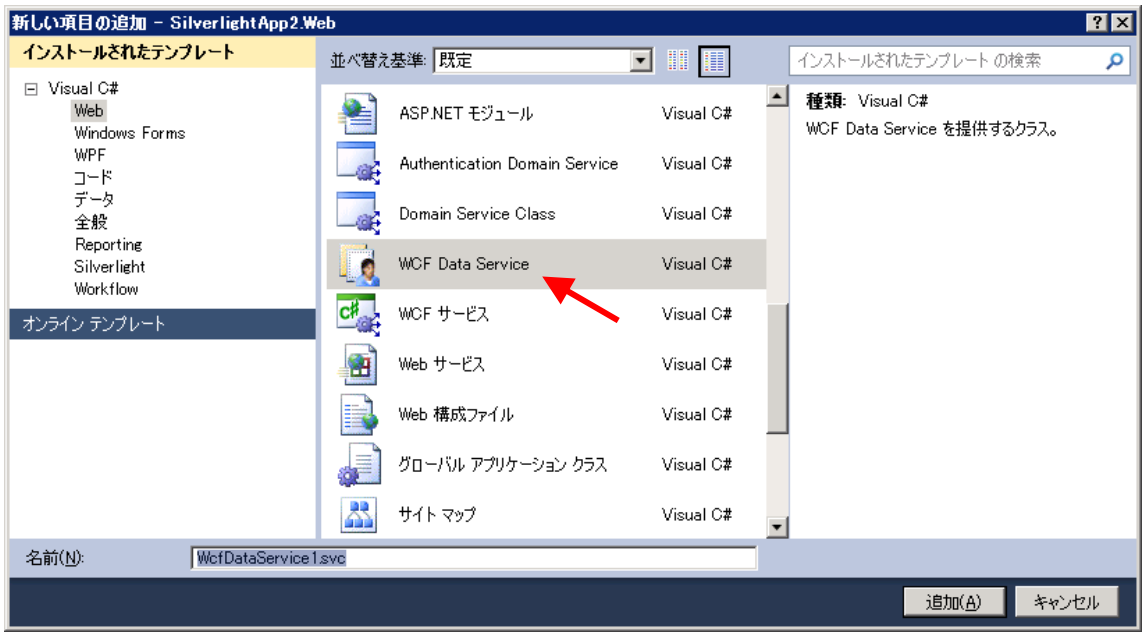


これを利用することで、必要なエンティティや、データ操作などをカプセル化した Object Context のコードを自動生成することができます。

また、図 10 の Data Service にあたる部分は、予め項目テンプレートが用意されています（図 12）。これを使用すると、Data Service の実装部分にあたるクラスのソースコードが自動生成されるほか、WCF の構成ファイルや、エンドポイントとなる .svc ファイルも自動的に生成されます。プログラマーが追加すべき最低限のコードは、公開すべきデータのセキュリティなどわずかです。例 16 では、参考までに、「titles」という名前のエンティティを読み取り専用で公開するよう指定したサンプルコード

を示しています。Data Service 本体のクラスの実装コード全体は、ここに示した数行が全てです。（実際は、このコードのほとんどは項目テンプレートを使用して自動生成されおり、プログラマーが入力した箇所は、[1]のジェネリッククラスの型パラメータの指定と、[2]のエンティティを読み取り専用として公開する部分です。）

図 12. WCF Data Services のサービス側の項目テンプレート



例 16. WCF Data Services における Data Service 実装クラスのサンプルコード

（C#）

```
public class WcfDataService1 : DataService<pubsEntities>  //←[1]
{
    public static void InitializeService(DataServiceConfiguration config)
    {
        config.DataServiceBehavior.MaxProtocolVersion =
            DataServiceProtocolVersion.V2;
        config.SetEntitySetAccessRule("titles", EntitySetRights.All);  //←[2]
    }
}
```

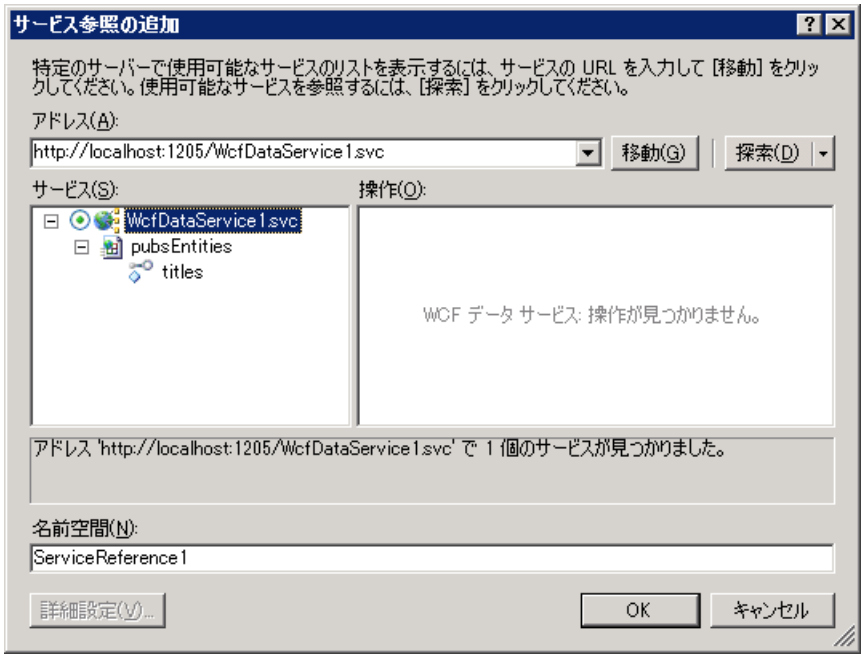
（Visual Basic）

```
Public Class WcfDataService1
       Inherits DataService(Of pubsEntities)  '←[1]

    Public Shared Sub InitializeService(ByVal config As DataServiceConfiguration)

        config.DataServiceBehavior.MaxProtocolVersion =
            DataServiceProtocolVersion.V2
        config.SetEntitySetAccessRule("titles", EntitySetRights.All)  '←[2]
    End Sub

End Class
```

クライアント側の開発

WCF Data Services はメタデータも提供しています。図 10 のクライアントの②のアプリケーションロジックの部分は、クライアントの開発環境において、「サービス参照の追加」を行って、このメタデータを取得することで自動生成できます。次図は、すでに取り上げたサービス参照の追加のダイアログボックスであり、WCF Data Services にアクセスしてメタデータを取り込んだ場合には、WCF Data Services にアクセスするための固有の実装コードが自動生成されます。

図 13. サービス参照の追加による WCF Data Services クライアントのコード生成



これを用いることで、クライアント側の主な実装である Data Service Context が生成されるほか、やり取りされるデータの受け皿となるエンティティ クラスや、サービスにアクセスするためのプロキシも自動生成されます。

通常の PC 版の .NET Framework 環境において、プロキシを用いた呼び出しでは、同期呼び出しと非同期呼び出しの両方をサポートしています。Silverlight の場合は、非同期呼び出しのみサポートしています。（むしろ、ブラウザーのバックグランドでサービスからデータを取り寄せるほうがユーザビリティの観点では望ましいので、非同期呼び出しのほうが適当でしょう。）

参考までに、以下に非同期呼び出しを用いて、サービス側が提供する titles テーブルのデータを、Silverlight 上の DataGrid コントロールに表示する例を掲載しておきます。前章の例 15 には、WCF RIA Services を用いた同様の例を示しました。WCF RIA Services を用いた場合に比べると、[1]のような非同期呼び出しの明示的な開始を記述したり、[2]の非同期処理完了時のハンドラーを記述したりと、比較的コード量は増えますが、その反面、非同期呼び出しの際のバックグランドの実行制御を直接コーディングできるので、よりきめ細かい制御が可能です。

例 16. 非同期呼び出しによって取得するデータとのバインディング（WCF Data Services の場合）

（C#）- Silverlight 3 の場合

```
    private pubsEntities ctx =
        new pubsEntities(
```

```
            new Uri("http://localhost:1205/WcfDataService1.svc/"));
    private ObservableCollection<title> titles;
    private IAsyncResult currentResult;

    public MainPage()
    {
        InitializeComponent();
        var query = ctx.titles;
        query.BeginExecute(OnQueryCompleted, query);   //←[1]
    }

    private void OnQueryCompleted(IAsyncResult result) //←[2]
    {
        currentResult = result;
        Dispatcher.BeginInvoke(PopulateData);
    }

    private void PopulateData()
    {
        titles = new ObservableCollection<title>();
        var query = (DataServiceQuery<title>)currentResult.AsyncState;
        var resultQuery = query.EndExecute(currentResult);
        foreach (var p in resultQuery)
        {
            titles.Add(p);
        }
        dataGrid1.ItemsSource = titles;
    }
```

（C#）- Silverlight 4 の場合

```
    private pubsEntities ctx =
        new pubsEntities(
            new Uri("http://localhost:1205/WcfDataService1.svc/"));
    private DataServiceCollection<title> titles;

    public MainPage()
    {
        InitializeComponent();
        titles = new DataServiceCollection<title>();
        titles.LoadCompleted += titles_LoadCompleted;
        titles.LoadAsync(ctx.titles);  //←[1]
    }

    void titles_LoadCompleted(object sender, LoadCompletedEventArgs e) //←[2]
    {
        dataGrid1.ItemsSource = titles;
    }
```

（Visual Basic）- Silverlight 3 の場合

```
    Private ctx As New pubsEntities(
        New Uri("http://localhost:1205/WcfDataService1.svc/"))
    Private titles As ObservableCollection(Of title)
    Private currentResult As IAsyncReslt

    Public Sub New()
        InitializeComponent()
        Dim query = ctx.titles
        query.BeginExecute(AddressOf OnQueryCompleted, query)    '←[1]
    End Sub
```

```
    Private Sub OnQueryCompleted(ByVal result As IAsyncResult)   '←[2]
        currentResult = result
        Dispatcher.BeginInvoke(PopulateData)
    End Sub

    Private Sub PopulateData()
        titles = New ObservableCollection(Of title)()
        Dim query =
            CType(currentResult.AsyncState, DataServiceQuery(Of title))
        Dim resultQuery = query.EndExecute(currentResult)
        For Each p In resultQuery
            titles.Add(p)
        Next
        dataGrid1.ItemsSource = titles
    End Sub
```

（Visual Basic）- Silverlight 4 の場合

```
    Private ctx As New pubsEntities(
        New Uri("http://localhost:1205/WcfDataService1.svc/"))
    Private titles As DataServiceCollection(Of title)

    Public Sub New()
        InitializeComponent()
        titles = New DataServiceCollection(Of title)()
        AddHandler titles.LoadComplete, AddressOf titles_LoadCompleted
        titles.LoadAsync(ctx.titles)  '←[1]
    End Sub

    Private Sub titles_LoadCompleted(
            ByVal sender As Object, ByVal e As LoadCompletedEventArgs)  '←[2]
        dataGrid1.ItemsSource = titles
    End Sub
```

以上、開発者の観点から、WCF Data Services の特徴について確認しました。これを利用することで、REST ベースの OData プロトコルに基づいて公開する WCF サービスを簡単に実装することができ、また、クライアントが .NET Framework の場合には、Silverlight アプリケーションも、通常の PC 向けアプリケーションも効率よく実装することができます。

第 6 章 まとめとして ～WCF 各テクノロジの比較～

この第 1 部では、WCF の全体像を概観し、WCF が提供する各テクノロジの特徴について解説してきました。この章では、最後にまとめとして、それらのテクノロジを比較して相違点を挙げ、利用する状況やシナリオに応じて、テクノロジを選択する上での指針を改めて確認していきます。

6.1 WCF 各テクノロジの相違点、選択の指針と利用シナリオ

第 1 部では、WCF のテクノロジを次のように分類して、章ごとに説明してきました。ここに挙げた各項目は、WCF サービスを実装する際に使用するテクノロジを決める際の選択肢と見ることもできます。

（a）WCF の基本部分、および関連するテクノロジ各種（第 2 章）

（b）WCF REST（第 3 章）

（c）WCF RIA Services（第 4 章）

（d）WCF Data Services（第 5 章）

もちろん、これらの選択肢は相互排他的に 1 つを選択するという性格のものではなく、複数を同時に利用することもありますが、それぞれは採用するか否かの判断対象となる選択肢であるといえるでしょう。

では、どのような観点から、これらの選択肢を選べばよいでしょうか。選択する上での 1 つの重要な考え方として、「汎用的なもの」か「特化したものか」という観点が挙げられます。

一般に WCF に限らず、複数のテクノロジの選択肢がある場合、より汎用的なものと、特定のシナリオに最適化され用途が特化したものがあります。

通常、汎用的なテクノロジは文字通り、様々な状況で利用が可能なので、一度作成したソフトウェアコンポーネントを様々な状況で再利用ができ、また、システム仕様が変更されても、そのコンポーネントの大幅な修正はなく、引き続き利用できる場合が多くあります。しかし、汎用的なテクノロジは、基本機能や関連機能などの品揃えが多く、いざ目的の実装を実現しようと思うと、それらを使い込むためにコーディングの手間が比較的かかる場合もあります。

一方、特定の用途に最適化されたテクノロジは、構築すべきシステム構成が、そのテクノロジの用途にうまく合致した場合、そのテクノロジのフレームワークがそのまま利用でき、手作業によるコーディング量が少なく、大幅に開発生産性を挙げることができます。その反面、そのテクノロジの本来の用途からはずれるような場合、かえって修正等のコーディングが増える場合もあります。

よて、これらのテクノロジの中から選択する指針としては、まず、構築すべきシステムに特化されたテクノロジの適用が可能であるかを考え、無理であれば、より汎用的なものを選択するということになるでしょう。もちろん、現時点でのシステムの仕様が、特化されたテクノロジに向いているかどうかだけでなく、将来的に仕様変更や拡張を行った場合にも、その特化されたテクノロジを適用できる余地があるか否かも検討する必要があります。

前述の WCF の選択肢の場合、（a）が最も汎用性が高いものであり、（b）が REST に特化したものになり、（c）や（d）は、さらに特定した用途に最適化しているテクノロジであるといえます。よって、まずは（c）や（d）のような特化したテクノロジの適用が可能であるかを考え、無理であれば、より汎用的なもの、（b）、そして、（a）というように、選択すべき候補を検討することになるでしょう。これらを踏まえて、それぞれの選択肢の特徴をまとめると次のようになります。

表 4 WCF の主なテクノロジの選択肢

分類	特化度合	特徴
（a）WCF の基本部分、関連すテクノロジ各種		WCF の基本的なサービス、およびクライアントの実装ほか、様々なメッセージング形態、簡単なデータから SOAP、WS-＊などの本格的なメッセージ交換、セキュリティや信頼性、運用時の管理機能など様々な機能を提供する。
（b）WCF REST	☆☆	REST ベースの WCF サービスに適している。REST に関しては汎用性が高い。ただし、SOAP や WS-＊などの高度なメッセージ交換には向かない。また、以下の（c）や（d）に比べると作り込む部分は多い。
（c）WCF RIA Services	☆☆☆	特に、Silverlight アプリケーションを WCF クライアントとして用い、データを操作するロジックを伴う場合には、最適である。なお、OData ほかプロトコルやメッセージ形式には汎用性があるが、クライアント向けに標準で提供されているライブラリやフレームワークは Silverlight に限定される。
（d）WCF Data Services	☆☆☆	特に、REST 形式の OData プロトコルを用いたデータ操作のサービスに向いている。クライアント向けのライブラリは、Silverlight だけでなく、PC 向けの .NET Framework にも提供されており、この面では（c）よりも汎用的だが、コード量は（c）よりもやや多くなる。

実際に、この中から選択する場合には、この表に挙げた特徴のほか、各章で取り上げたそれぞれの典型的なシステム構成図のパターンなども参考にしてください。それらをふまえ、まず、（c）や（d）が適用できるシナリオであるか否かを検討することから始めることになるでしょう。

まず、（c）や（d）では、どちらもデータ操作に関わるサービスに特化している点では共通しています。どちらも、既定で Entity Framework の EDM が利用できます。特に、（c）と（d）との間での違いは、（c）が Silverlight 向けに最適化されていることでしょう。主なクライアントが Silverlight であるのなら（c）が特に有効です。

また、データ操作に限らず、より汎用的だが REST ベースサービスに限定するのなら、（b）が利用できます。ただし、（b）では WS-＊仕様が利用できないので、サービス間の分散トランザクションなど、高度な機能を一部実現できない場合があります。

そして、（b）、（c）および（d）のどれにも当てはまらず、より汎用的でオールマイティとなる選択肢が（a）です。もちろん、（a）の全機能が他の選択肢と相互排他的な関係にあるわけではなく、（b）、（c）および（d）を使用する場合でも、たとえば、セキュリティ面で（a）の機能を併用する

ことはあります。その意味で、（b）、（c）または（d）のいずれか 1 つを使用して実装するプログラマーも、（a）は共通して習得すべき知識であるといえます。

以上、この第 1 部では総論として、WCF の全体像や、WCF が提供する各テクノロジの特徴について解説してきました。これらをふまえ第 2 部からは各論になり、第 2 部では選択肢（a）である WCF の基本機能や関連機能を用いたサンプルコードを例に、各機能の実装方法について確認していきます。