



Pesquisar



Desenvolvimento - WCF

Criando e consumindo serviços com WCF

O WCF é a plataforma da Microsoft disponível desde o .NET Framework 3.0 utilizada para cuidar da comunicação entre sistemas. O WCF é uma união de Web Service, WSE, Remoting e COM+, tudo isso em uma única plataforma, simples de usar, robusta e de fácil integração.

por André Baltieri



Publicidade



Introdução

Atualmente, cada vez mais os serviços estão presentes em nossas aplicações, seja consumindo um serviço, ou servindo uma informação. No cenário de aplicações distribuídas, o



Anuncie | Fale Conosco | Publique

**Linha de Código**

Curtir Página 15 mil curtidas

6 amigos curtiram isso



ser visto no serviço.

A Figura 1 ilustra um endpoint.

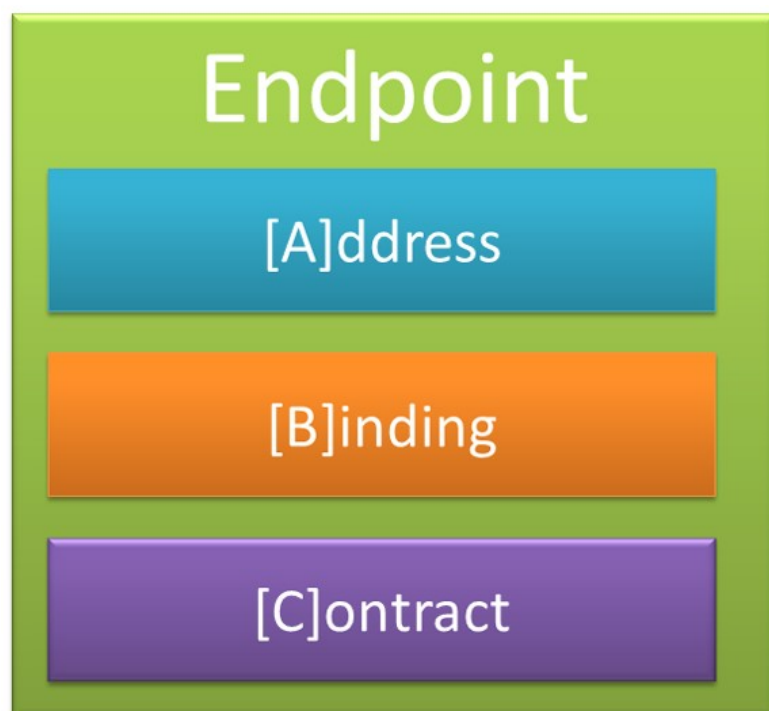


Figura 1 - Endpoints.

Hospedagem de Serviços

Posteriormente, quando tínhamos apenas Web Services, havia a necessidade de possuir o IIS (Internet Information Server) instalado para que o serviço pudesse ser publicado e acessado. Junto ao WCF, veio um novo conceito de `strong>` host (hospedagem) destes serviços que criamos, possibilitando inclusive a hospedagem de serviços em aplicações do tipo Console Application.

Desta forma temos as seguinte opções de host:

Self-host - Hospedado em qualquer aplicação .NET gerenciada.
Windows Service - Hospedado por um processo gerenciado pelo Windows.
IIS - Hospedado em um web site, no IIS.

Nota: No IIS 7 ou superior, também podemos hospedar serviços com disponibilidade via TCP.

Disponibilidade de Serviços

Basicamente, podemos disponibilizar nossos serviços de duas

Uma URL, um caso de uso para este cenário seria a necessidade de acesso a um serviço de fora da rede da empresa. Este tipo de disponibilidade é menos performático que o TCP. TCP - Acessível somente de dentro da rede da empresa. Este tipo de disponibilidade é mais performático que o HTTP, porém menos acessível já que fica restrito para a redes internas.

Caso faça-se necessário, podemos utilizar ambas disponibilidades. Podemos tomar como base novamente um cenário corporativo, onde um funcionário precisar acessar a rede da empresa tanto de dentro quanto de fora da empresa. O que acontece neste caso é que podemos criar ambos tipos de hospedagem e tornar o serviço disponível tanto dentro da rede, quanto fora.

Vale lembrar também, que estas opções são facilmente customizadas através do arquivo de configuração App.Config / Web.Config disponível nas aplicações.

Configurando um Serviço

O WCF conta com dois tipos de configuração, sendo eles:

Imperativo - Criado no próprio código gerenciado (C#, VB.NET).

Declarativo - Através de um arquivo de configuração (Web.Config, App.Config)

Quando trabalhamos com o método imperativo, temos a desvantagem de ter que recompilar o sistema a cada vez que uma modificação for feita, coisa que não ocorre no modo declarativo, onde as configurações estão escritas nos arquivos de configuração. Por serem arquivos no formato XML, podemos abri-los com o Notepad ou qualquer outro editor de texto e de forma fácil e simples alterar as configurações necessárias.

Contratos

No conceito de orientação à objetos, um contrato expõe quais membros de uma classe serão visíveis. No WCF o conceito é o mesmo, e através de interfaces podemos definir um contrato entre um serviço e as aplicações que irão consumi-lo, expondo somente os métodos desejados.

A utilização de contratos deve-se ao fato de melhorar (e muito)

adicionar um novo método a este serviço. Provavelmente isto irá impactar todas as aplicações que consomem este serviço, forçando elas a modificarem seus meios de acesso ao mesmo.

O versionamento do serviço permite que quando precisarmos criar um novo método, cria-se uma nova interface com este método e adiciona ao serviço (Interfaces permitem múltipla herança). Em seguida, podemos simplesmente criar um novo endpoint para que este seja consumido.

O WCF conta com os seguintes tipos de contratos:

Service Contract - Um contrato para um serviço. Define os detalhes do serviço, e será utilizado na interface de contrato.

Operational Contract - Define uma operação individual, e será aplicado na assinatura dos métodos da interface de contrato.

Data Contract - Define a serialização para objetos complexos.

Esta propriedade necessita da inclusão do namespace

System.Runtime.Serialization. Message Contract - Este contrato

descreve a mensagem SOAP completa. Fault Contract -

Utilizado para documentar erros no WCF.

Os contratos do WCF são utilizados como propriedades, decorando as classes e interfaces.

Criando um Self-host service

Como passado anteriormente, um serviço do tipo Self-host pode ser hospedado em qualquer aplicação .NET de código gerenciado. Para este exemplo, vou criar uma nova Console Application (Figura 2) no Visual Studio 2010 com o Framework 4.0, mas como já dito, o WCF está presente desde a versão 3.0 do .NET Framework.

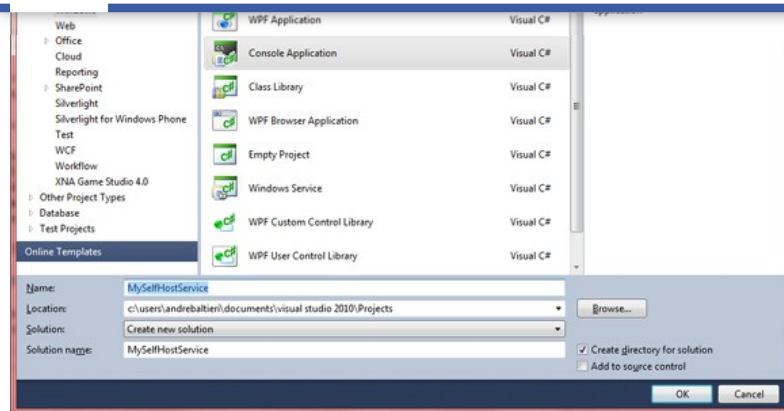


Figura 2 - Criando uma nova Console Application.

Com a aplicação criada, vamos criar uma nova classe, que representará um cliente (Customer). A Listagem 1 demonstra o código da classe criada, Customer.cs.

Listagem 1 - Classe Customer.cs.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace MySelfHostService
7  {
8      public class Customer
9      {
10         public int CustomerId { get; set; }
11         public string Name { get; set; }
12         public string Email { get; set; }
13         public string CreditCardNumber { get; set; }
14     }
15 }
16

```

Assim criamos o que chamamos de objeto complexo, que nada mais é do que um objeto customizado. Precisamos agora criar um contrato (Interface) que definirá o que poderá ser consumido deste serviço. Para isto, vamos criar uma nova interface e chamá-la de ICustomer.cs, como demonstrado na Listagem 2.

Listagem 2 - Interface (Contrato) ICustomer.cs.

```

1  namespace MySelfHostService
2  {
3      public interface ICustomer
4      {
5          Customer CreateCustomer(int customerId, st
6          Customer GetCustomerByName(string customerl

```

Na interface `ICustomer` temos duas observações a serem levadas em conta, sendo a primeira o uso de array no retorno do método `Customer[] GetAllCustomers()`. Estamos acostumados com a utilização de listas (`List<T>`) quando falamos em conjuntos de objetos, porém temos que pensar que como este sistema será distribuído e consumido por outros sistemas que talvez não sejam feitos em .NET, provavelmente estes sistemas que estão consumindo os serviços não suportarão o `List`, como é o caso do VB6. Podemos muito bem ter uma aplicação desenvolvida em VB6 que irá consumir este serviço. Ela entenderá um array de `Customers`, mas não um `List` de `customers`.

A outra observação vai para o fato de não incluirmos o número do cartão de crédito do cliente no método `CreateCustomer`. Como a intenção aqui é exemplificar o que um contrato pode ou não expor de um serviço, vamos optar futuramente por mesmo tendo uma propriedade para armazenar o número do cartão de crédito do cliente, não expô-la.

Por fim, teremos uma classe que representará o serviço em si, e irá conter a implementação dos métodos especificados no contrato (Interface `ICustomer`), como mostrado na Listagem 3.

Listagem 3 - Classe `CustomerService.cs`

```
1  using System.Collections.Generic;
2  using System.Linq;
3
4  namespace MySelfHostService
5  {
6      public class CustomerService:ICustomer
7      {
8          ///
9          /// Lista de clientes
10         ///
11         List customers = new List();
12
13         ///
14         /// Método construtor
15         /// Carrega os clientes
16         ///
17         public CustomerService()
18         {
19             for (int i = 1; i
20             /// Implementação do método CreateCustomer
21             /// Cria um novo cliente e retorna o mesmo
```

```
25     /// Email
26     /// Cliente (Customer)
27     public Customer CreateCustomer(int customerId, string name, string email)
28     {
29         Customer customer = new Customer();
30         customer.CustomerId = customerId;
31         customer.Name = name;
32         customer.Email = email;
33
34         return customer;
35     }
36
37     ///
38     /// Busca na lista de clientes gerada no banco de dados
39     /// pelo nome
40     ///
41     /// Nome do cliente
42     /// Cliente
43     public Customer GetCustomerByName(string name)
44     {
45         return (from c in customers where c.Name == name select c).FirstOrDefault();
46     }
47
48     ///
49     /// Retorna todos os clientes
50     ///
51     /// array de clientes
52     public Customer[] GetAllCustomers()
53     {
54         return customers.ToArray();
55     }
56 }
57 }
```

Basicamente implementamos a interface `ICustomer` e codificamos seus métodos. Neste caso, criei uma lista simples de clientes, apenas para fins de estudos, mas é um cenário de um sistema, aqui seria feito o acesso à dados por exemplo.

Decorando classes e interfaces

Até o momento, apenas criamos um tipo customizado, uma interface e uma classe que implementa a interface. Temos agora que decorar estas classes e interfaces com contratos que o WCF nos fornece, utilizando os atributos do .NET Framework.

Para que possamos decorar a interface, precisaremos adicionar uma referência ao `System.ServiceModel`, e para decorar nosso tipo customizado, precisamos adicionar referência ao `System.Runtime.Serialization`.

Começaremos com a decoração da interface `ICustomer`, que

Listagem 4 - Decorando a interface (Contrato) ICustomer.cs.

```
1 using System.ServiceModel;
2
3 namespace MySelfHostService
4 {
5     [ServiceContract]
6     public interface ICustomer
7     {
8         [OperationContract]
9         Customer CreateCustomer(int customerId, string name);
10
11         [OperationContract]
12         Customer GetCustomerByName(string customerName);
13
14         [OperationContract]
15         Customer[] GetAllCustomers();
16     }
17 }
18
```

Para a interface, utilizamos o atributo `ServiceContract`, e para os métodos utilizamos o atributo `OperationalContract`, ambos explicados no começo do artigo. Temos agora o contrato decorado, mas isto não basta. Como iremos trafegar um tipo customizado (`Customer.cs`), precisamos decorá-lo também, para dizer ao contrato quais membros deste tipo devem ser trafegados e expostos. A Listagem 5 mostra a versão final da classe `Customer.cs`, decorada.

Listagem 5 - Decorando a classe `Customer.cs`.

```
1 using System.Runtime.Serialization;
2
3 namespace MySelfHostService
4 {
5     [DataContract]
6     public class Customer
7     {
8         [DataMember]
9         public int CustomerId { get; set; }
10
11         [DataMember]
12         public string Name { get; set; }
13
14         [DataMember]
15         public string Email { get; set; }
16
17         public string CreditCardNumber { get; set; }
18     }
19 }
20
```


configurar este serviço, informar seu endereço, forma de acesso e contrato que expõe suas informações.

Configurando um serviço

Como comentado previamente, existem dois meios para se configurar um serviço no WCF, imperativo e declarativo. Por ser de mais fácil manutenção, criaremos do modo declarativo, através do arquivo de configuração App.Config.

Caso sua aplicação não possua este arquivo, adicione-o.

No App.Config, dentro da sessão configurations, existe um área reservada para as configurações dos serviços, esta área é o `system.serviceModel`.

Dentro desta área podemos configurar vários serviços, inclusive devemos notar inclusive que temos uma tag `services`, no plural, indicando que mais de um serviço pode ser configura dentro deste service model.

A Figura 3 mostra as tags de configuração iniciais.



Figura 3 - Configurando os serviços.

Note que temos no nome do serviço, o namespace seguido do nome da classe que implementa os membros definidos pelo contrato. Esta ligação que permite identificar para qual serviço estas configurações se aplicam.

Configurando Host

Ainda dentro da configuração de serviços, temos a necessidade de configurar o host, que definirá a disponibilidade deste serviço. Como vimos previamente, podemos disponibilizar um serviço via HTTP ou via TCP, ou melhor, em ambos, como mostrado na Figura 4.



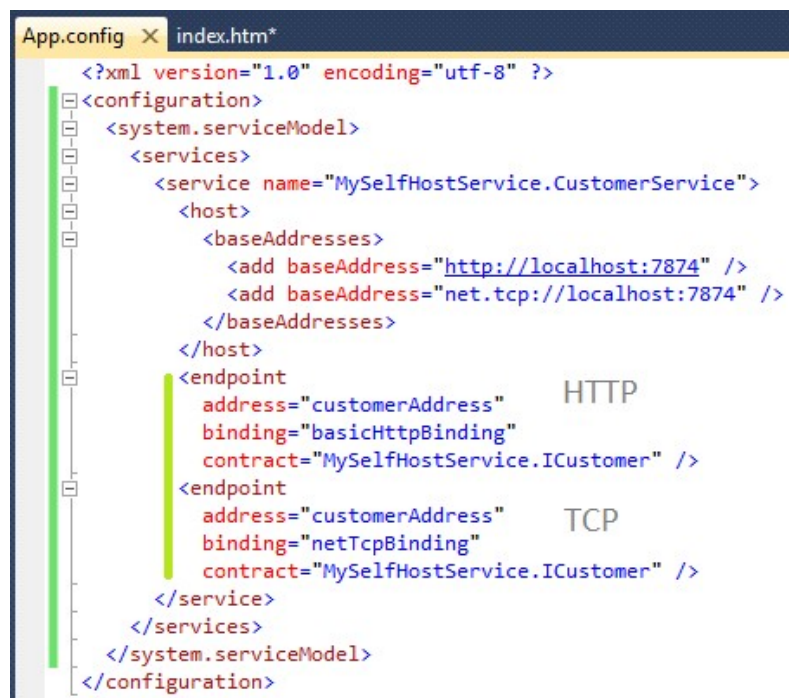
```
<system.serviceModel>
  <services>
    <service name="MySelfHostService.CustomerService">
      <host>
        <baseAddresses>
          <add baseAddress="http://localhost:7874" />
          <add baseAddress="net.tcp://localhost:7874" />
        </baseAddresses>
      </host>
    </service>
  </services>
</system.serviceModel>
</configuration>
```

Figura 4 - Configurando hosts.

Configurando Endpoints

Até o momento, temos o serviço e os seus hosts configurados, porém, também precisamos configurar seus endpoints, que são justamente os pontos por onde este serviço poderá ser acessado.

Para cada tipo de acesso (HTTP, TCP) teremos que criar um endpoint diferente, que conterá o address, binding e contract para aquele ponto de acesso, como mostrado na Figura 5.



```
App.config X index.htm*
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="MySelfHostService.CustomerService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:7874" />
            <add baseAddress="net.tcp://localhost:7874" />
          </baseAddresses>
        </host>
        <endpoint
          address="customerAddress" HTTP
          binding="basicHttpBinding"
          contract="MySelfHostService.ICustomer" />
        <endpoint
          address="customerAddress" TCP
          binding="netTcpBinding"
          contract="MySelfHostService.ICustomer" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Figura 5 - Configurando endpoints.

Os endpoints possuem um endereço, que neste caso chamei de customerAddress, um binding que pode ser basicHttpBinding ou netTcpBinding e um contract que deve ser o nome do contrato

WSDL

Para que um serviço possa ser lido e entendido, quem consome este serviço faz uso de um documento que chamamos de WSDL (Web Service Description Language), que pode ser visto adicionando o parâmetro WSDL no final da url (<http://localhost:8776/MeuServico?WSDL>), e que contém todas as informações sobre o serviço requisitado.

Não é necessário criarmos manualmente este documento, o próprio WCF faz isto para nós, porém, precisamos no arquivo de configuração, dizer a ele quais contratos e serviços esta aplicação está expondo para que então o documento WSDL seja gerado.



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="MySelfHostService.CustomerService"
        behaviorConfiguration="wsdlConfiguration">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:7874" />
            <add baseAddress="net.tcp://localhost:7874" />
          </baseAddresses>
        </host>
        <endpoint
          address="customerAddress"
          binding="basicHttpBinding"
          contract="MySelfHostService.ICustomer" />
        <endpoint
          address="customerAddress"
          binding="netTcpBinding"
          contract="MySelfHostService.ICustomer" />
        </service>
      </services>
      <behaviors>
        <serviceBehaviors>
          <behavior name="wsdlConfiguration">
            <serviceMetadata httpGetEnabled="true" />
          </behavior>
        </serviceBehaviors>
      </behaviors>
    </system.serviceModel>
  </configuration>
```

Figura 6 - Configurando behaviors.

Basicamente criamos um behavior (Comportamento), e nomeamos ele como `wsdlConfiguration`. Para que a aplicação que esteja consumindo este serviço esteja habilitada a gerar seu WSDL, adicionamos a tag `serviceMetadata httpGetEnabled` e atribuímos seu valor para verdadeiro.

Para finalizar precisamos então criar mais um endpoint, que será o ponto de acesso para o WSDL gerado. Ao contrário dos outros endpoints, neste endpoint os dados são padrões.

A Figura 7 mostra o arquivo App.Config final.



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="MySelfHostService.CustomerService"
        behaviorConfiguration="wsdlConfiguration">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:7874/" />
            <add baseAddress="net.tcp://localhost:7884/" />
          </baseAddresses>
        </host>
        <endpoint
          address="customerAddress"
          binding="basicHttpBinding"
          contract="MySelfHostService.ICustomer" />
        <endpoint
          address="customerAddress"
          binding="netTcpBinding"
          contract="MySelfHostService.ICustomer" />
        <endpoint
          address="mex"
          binding="mexHttpBinding"
          contract="IMetadataExchange" />
        </service>
      </services>
      <behaviors>
        <serviceBehaviors>
          <behavior name="wsdlConfiguration">
            <serviceMetadata httpGetEnabled="true" />
          </behavior>
        </serviceBehaviors>
      </behaviors>
    </system.serviceModel>
  </configuration>
```

Figura 7 - Arquivo App.Config na versão final.

Com o nosso sereviço configurado, precisamos agora iniciar o serviço quando a nossa aplicação iniciar. Para isto precisamos criar um ServiceHost que contenha as informações do nosso serviço. Ele será responsável pelo host do mesmo.

O interessante aqui, é que podemos apenas passar o tipo do nosso serviço (typeof(CustomerService)) que o ServiceHost já sabe onde obter as informações do mesmo. O que ele faz é ir na sessão services do App.Config e buscar por este tipo nos nomes dos serviços configurados ali. Por isso a importância de ter o namespace.nomedoserviço no parâmetro "name" da

A Listagem 6 mostra como iniciar um serviço self-host em uma ConsoleApplication.

Listagem 6 - Iniciando um serviço self-host.

```
1
2 static void Main(string[] args)
3 {
4     ServiceHost host = new ServiceHost(typeof(Custo
5     host.Open();
6     Console.WriteLine("Serviço rodando...");
7     Console.WriteLine("Tecle para finalizar.");
8     Console.ReadKey();
9     host.Close();
10 }
```

Testando o serviço

O Visual Studio (a partir da versão 2008 se não me engano) traz com ele uma ferramenta chamada WCF Test Client, que permite-nos testar os serviços rodando sem a necessidade de criar uma aplicação e consumi-lo.

Para acessar esta ferramenta, clique sobre Start (Iniciar)> All Programs (Todos os Programas)> Visual Studio 2010 (No meu caso)> Visual Studio Tools> Visual Studio Command Prompt (2010)

Deste modo, o prompt de comando do Visual Studio abrirá, digite então o seguinte comando:

WcfTestCliente.exe <endereço do serviço>

Caso você esteja seguindo a risca o artigo, ficaria assim:

WcfTestClient.exe <http://localhost:7874/>

Nota: Em alguns casos pode ser necessário rodar o Visual Studio Command Prompt como administrador.

Nota: Seu serviço deve estar rodando para que o aplicativo possa se conectar a ele.

Deste modo, teremos a tela do WCF Test Client aberta e listando os serviços, como mostrado na Figura 8.

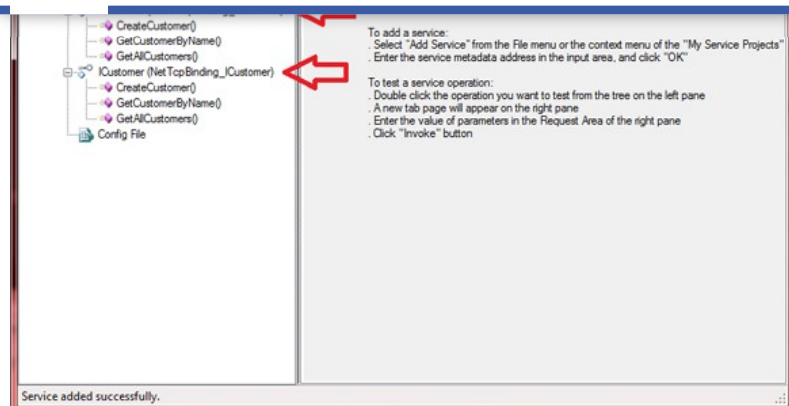


Figura 8 - WCF Test Client.

Podemos notar que ele mostra tanto o serviço HTTP quanto o TCP, e com os métodos para invocarmos. Neste ponto, fiquem à vontade em testar da forma que acharem interessante.

Criando serviços apartir de templates

Até o momento, criamos um serviço do tipo self-host. Legal, mas pudemos notar que tivemos que fazer muita coisa manualmente, e isso foi desnecessário (Necessário apenas para aprendizado).

O Visual Studio conta com templates que trazem os serviços prontos, bastando apenas alterar o nome dos arquivos, classes e contratos.

Para este exemplo, criei um novo WCF Service Application, acessando o menu File> New Project e selecionando o template, como mostrado na Figura 9.

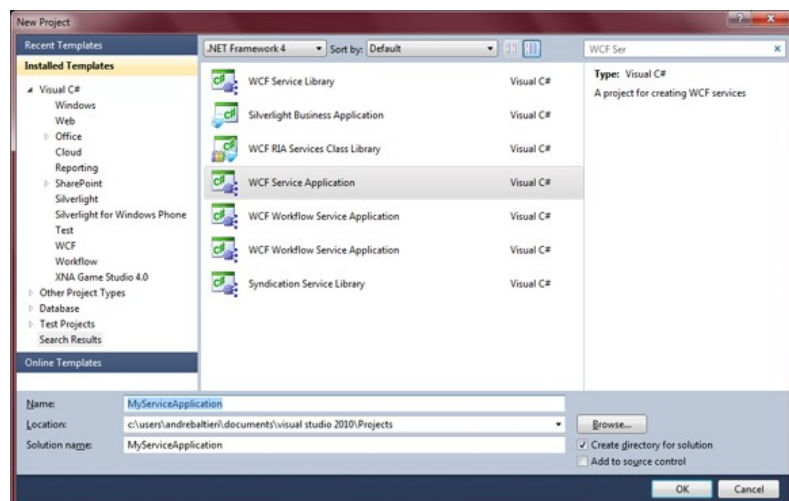


Figura 9 - Criando um novo serviço apartir de um template.

mostrado na Figura 10.

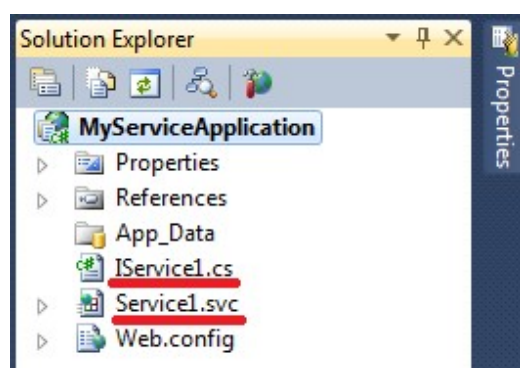


Figura 10 - Solução do projeto.

Nota: Este tipo de serviço só suporta hospedagens HTTP. IIS 7 ou superior também suporta hospedagem de serviços TCP.

Da mesma forma que anteriormente, podemos criar classes e decorá-las, modificar as configuração no arquivo de configuração (Agora Web.Config) e etc.

Rodando a aplicação, novamente teremos o WCF Test Cliente nos auxiliando a testar este serviço, porém agora ele foi iniciado automaticamente, sem a necessidade de utilizar o prompt de comando do Visual Studio.

Conclusão

O WCF é uma plataforma que une recursos antes espalhados em um único ponto, tornando a plataforma robusta. Os recursos de customização estão facilitados, podendo ser acessados e modificados através de arquivos de configuração como Web.Config e App.Config.

A possibilidade de hospedar os serviços não apenas no IIS torna o WCF ainda mais rico, trazendo flexibilidade e aumentando o leque de possibilidades de consumo deste mesmo serviço.

Para finalizar, temos a opção de disponibilizar este serviço via TCP, que é mais performático que o HTTP, porém menos acessível. Se mesmo assim ainda não basta, podemos ter a disponibilidade tanto HTTP quanto TCP em um mesmo serviço.

Referências

<http://www.linhadecodigo.com.br/tag/wcf>

WCF Brasil

<http://wcfbrasil.ning.com/>

Espero que tenham gostado e até o próximo artigo!



André Baltieri - Trabalha com desenvolvimento de aplicações web a mais de 7 anos, e com ASP.NET desde 2003. É líder da comunidade Inside .NET (<http://www.insidedotnet.com.br/>) e do projeto Learn MVC .NET (<http://learn-mvc.net/>). Bacharelado em Sistemas de Informação, atualmente trabalha com desenvolvimento e suporte de aplicações web em ASP.NET/C# em projetos internacionais e ministra treinamentos e consultorias sobre a plataforma .NET.



Leia também

Verificando disponibilidade de um serviço WCF ou WebService
C#

WCF em Aplicação Gerenciável (Managed Application)
WCF

WCF - Comunicação P2P
WCF

WCF – Gerenciamento de Instância
WCF

Comunicação Local no Silverlight
Silverlight

DESENVOLVIMENTO ▼

FRONT-END ▼

BANCO DE DADOS

LOGIN