EBook Gratis

APRENDIZAJE wcf

Free unaffiliated eBook created from
Stack Overflow contributors.

#wcf

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: wcf

It is an unofficial and free wcf ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official wcf.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con wcf

Observaciones

Esta sección proporciona una descripción general de qué es wcf y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de wcf, y vincular a los temas relacionados. Dado que la Documentación para wcf es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Examples

Demostración de servicio tranquilo de WCF

svc

```
 public class WCFRestfulService : IWCFRestfulService
    {
       public string GetServiceName(int Id)
        {
            return "This is a WCF Restful Service";
        }
    }
```

Interfaz

```
[ServiceContract(Name = "WCRestfulService ")]
public interface IWCFRestfulService
{
    [OperationContract]
    [WebInvoke(Method = "GET", ResponseFormat = WebMessageFormat.Json, BodyStyle =
WebMessageBodyStyle.Wrapped, UriTemplate = "GetServiceName?Id={Id}")]
    string GetServiceName(int Id);
}
```

Marcado svc (haga clic con el botón derecho en el archivo svc y haga clic en la vista MarkUp)

```
<%@ ServiceHost Language="C#" Debug="true" Service="NamespaceName.WCFRestfulService"
CodeBehind="WCFRestfulService.svc.cs" %>
```

Configuración web

```
<services>
      <service name="NamespaceName.WCFRestfulService"
behaviorConfiguration="ServiceBehaviour">
        <endpoint address="" binding="webHttpBinding"
contract="NamespaceName.IWCFRestfulService" behaviorConfiguration="web"/>
      </service>
```

```
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="ServiceBehaviour">
          <!-- To avoid disclosing metadata information, set the values below to false before
deployment -->
          <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
          <!-- To receive exception details in faults for debugging purposes, set the value
below to true.  Set to false before deployment to avoid disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
        <behavior name="web">
          <webHttp/>
        </behavior>
      </endpointBehaviors>
    </behaviors>
```

Ahora simplemente ejecute el servicio o el host en un puerto. Y acceda al servicio usando " http: // hostname / WCFRestfulService / GetServiceName? Id = 1 "

Servicio WCF simple

Los requisitos mínimos para el servicio WCF es un ServiceContract con un OperationContract.

Contrato de servicios:

```
[ServiceContract]
public interface IDemoService
{
    [OperationContract]
    CompositeType SampleMethod();
}
```

Implementación del contrato de servicios:

```
public class DemoService : IDemoService
{
    public CompositeType SampleMethod()
    {
        return new CompositeType { Value = "foo", Quantity = 3 };
    }
}
```

Archivo de configuración:

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.5.2" />
    <httpRuntime targetFramework="4.5.2"/>
```

```
  </system.web>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata httpGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
multipleSiteBindingsEnabled="true" />
  </system.serviceModel>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true"/>
    <directoryBrowse enabled="true"/>
  </system.webServer>
</configuration>
```

DTO:

```
[DataContract]
public class CompositeType
{
    [DataMember]
    public string Value { get; set; }

    [DataMember]
    public int Quantity { get; set; }
}
```

Lea Empezando con wcf en línea: https://riptutorial.com/es/wcf/topic/992/empezando-con-wcf

Capítulo 2: Cómo utilizar un contenedor de inyección de dependencia con un servicio WCF

Examples

Cómo configurar un servicio WCF para usar un contenedor de inyección de dependencias (Castle Windsor)

Este ejemplo tiene dos partes: algunos pasos para agregar Castle Windsor a su servicio WCF, y luego un ejemplo simple y concreto para mostrar cómo configuramos y usamos el contenedor de Windsor.

Eso hace que el ejemplo sea un poco largo. Si ya comprende el uso de un contenedor DI, es probable que solo le interesen los pasos de la placa de la caldera. Si el uso de un contenedor DI no es familiar, se necesita un poco más, ya que todo funciona de principio a fin, antes de que tenga sentido.

Pasos repetitivos

1. Agregue el paquete Nuget de las instalaciones de integración WCF de Castle Windsor a su aplicación de servicio WCF. Esto agregará referencias a Castle Windsor así como a algunos componentes específicamente para servicios WCF.

2. Agregue una Clase de aplicación global (global.asax) a su proyecto: Agregar> Nuevo elemento> Visual C #> Web> Clase de aplicación global.
   El código que configura su contenedor debe llamarse desde el método `Application_Start` .
   Para mantenerlo organizado, podemos poner toda nuestra configuración de contenedores en una clase separada. No tenemos que hacerlo. No importa. Verás esto hecho de manera diferente en diferentes ejemplos. Lo que importa es que se llama desde el método `Application_Start` porque esa es la *raíz* de su *composición* , donde comienza la aplicación. La idea es configurar su contenedor cuando se inicie la aplicación y luego no volver a tocarlo directamente. Simplemente se queda en segundo plano haciendo su trabajo.

3. Crea una clase para configurar el contenedor. Esto hace dos cosas:

   - `ContainerInstance.AddFacility<WcfFacility>()` le dice al código WCF de Windsor que use este contenedor en particular al crear instancias de sus servicios WCF.
   - `ContainerInstance.Install(FromAssembly.This())` le dice a Windsor que escanee `This` conjunto (en otras palabras, su proyecto WCF) en busca de clases que implementen `IWindsorInstaller` . Esas clases proporcionarán instrucciones para decirle a su contenedor cómo resolver dependencias. (Vamos a crear uno unos pasos más tarde.)

```
public static class Container
```

```
{
    private static readonly IWindsorContainer ContainerInstance = new WindsorContainer();

    public static void Configure()
    {
        ContainerInstance.AddFacility<WcfFacility>();
        ContainerInstance.Install(FromAssembly.This());
    }
}
```

4. Llame a este método desde `Application_Start` en su global.asax:

```
public class Global : System.Web.HttpApplication
{
    protected void Application_Start(object sender, EventArgs e)
    {
        Container.Configure();
    }
}
```

5. Crear un instalador. Esta es solo una clase que implementa `IWindsorInstaller` . Este está vacío. No hace nada. Vamos a añadir a esta clase en unos pocos pasos. Cuando llamamos `ContainerInstance.Install(FromAssembly.This())` , `ContainerInstance` se pasa a la `Install` método de modo que podemos registrar las dependencias con el contenedor.

```
public class WindsorInstaller : IWindsorInstaller
{
    public void Install(IWindsorContainer container, IConfigurationStore store)
    {
        // Nothing here yet!
    }
}
```

6. En el marcado para cualquier servicio WCF que cree, agregue esta directiva. Esto indica que la aplicación utilizará la función WCF de Windsor para crear instancias del servicio, lo que a su vez significa que puede inyectar dependencias cuando se crea una instancia.

```
Factory="Castle.Facilities.WcfIntegration.DefaultServiceHostFactory,
Castle.Facilities.WcfIntegration"
```

Eso termina los pasos para configurar el servicio para usar un contenedor de inyección de dependencias de Castle Windsor.

Pero el ejemplo no está completo a menos que configuremos al menos un servicio WCF para la inyección de dependencia. El resto de esto no es repetitivo, solo un ejemplo simple y concreto.

1. Cree un nuevo servicio WCF llamado `GreetingService.svc` .
2. Editar el marcado. Debe tener un aspecto como este:

```
<%@ ServiceHost Language="C#" Debug="true"
    Service="WcfWindsorDocumentation.GreetingService"
    CodeBehind="GreetingService.svc.cs"
    Factory="Castle.Facilities.WcfIntegration.DefaultServiceHostFactory,
```

```
Castle.Facilities.WcfIntegration"
 %>
```

3. Reemplace `IGreetingService` (el contrato de servicio) con esto:

```
[ServiceContract]
public interface IGreetingService
{
    [OperationContract]
    string GetGreeting();
}
```

4. Reemplace `GreetingService` (en GreetingService.svc.cs) con este código. Observe que el constructor requiere una instancia de `IGreetingProvider` que necesitaremos nuestro contenedor para inyectar.

```
public class GreetingService : IGreetingService
{
    private readonly IGreetingProvider _greetingProvider;

    public GreetingService(IGreetingProvider greetingProvider)
    {
        _greetingProvider = greetingProvider;
    }

    public string GetGreeting()
    {
        return _greetingProvider.GetGreeting();
    }
}
```

5. Agregue esta implementación de `IGreetingProvider` . También tiene algunas dependencias propias que necesitaremos el contenedor para suministrar. Los detalles de estas clases no son demasiado importantes. Solo son para crear algo fácil de seguir.

```
public interface IGreetingProvider
{
    string GetGreeting();
}

public interface IComputerNameProvider
{
    string GetComputerName();
}

public class ComputerNameGreetingProvider : IGreetingProvider
{
    private readonly IComputerNameProvider _computerNameProvider;

    public ComputerNameGreetingProvider(IComputerNameProvider computerNameProvider)
    {
        _computerNameProvider = computerNameProvider;
    }

    public string GetGreeting()
    {
```

```
        return string.Concat("Hello from ", _computerNameProvider.GetComputerName());
    }
}

public class EnvironmentComputerNameProvider : IComputerNameProvider
{
    public string GetComputerName()
    {
        return System.Environment.MachineName;
    }
}
```

6. Ahora tenemos todas las clases que necesitamos. Todo lo que queda es registrar las dependencias en nuestro contenedor. En otras palabras, vamos a decirle al contenedor qué clases necesita crear para que sepa cómo "construir" una instancia de `GreetingService` . Este código se `IWindsorInstaller` en nuestra implementación de `IWindsorInstaller` (paso 5 del código repetitivo).

```
public class WindsorInstaller : IWindsorInstaller
{
    public void Install(IWindsorContainer container, IConfigurationStore store)
    {
        container.Register(
            Component.For<IGreetingService, GreetingService>(),
            Component.For<IGreetingProvider, ComputerNameGreetingProvider>(),
            Component.For<IComputerNameProvider, EnvironmentComputerNameProvider>());
    }
}
```

Esto le dice al contenedor:

- Es responsable de crear `GreetingService` cuando sea necesario.
- Cuando intente crear `GreetingService` necesitará un `IGreetingProvider` . Cuando sea necesario, debe crear un `ComputerNameGreetingProvider` .
- Cuando intenta crear `ComputerNameGreetingProvider` , esa clase requiere un `IComputerNameProvider` . Debe crear una instancia de `EnvironmentComputerNameProvider` para satisfacer esa necesidad.

Si, en algún momento del proceso de creación de `GreetingService` o una de sus dependencias, se encuentra con un requisito para una dependencia que no hemos registrado, nos avisará con una excepción útil, como esta.

> Falta dependencia.
> El componente WcfWindsorDocumentation.ComputerNameGreetingProvider tiene una dependencia en WcfWindsorDocumentation.IComputerNameProvider, que no se pudo resolver.
> Asegúrese de que la dependencia esté correctamente registrada en el contenedor como un servicio, o provista como un argumento en línea.

Eso significa que algo depende de `IComputerNameProvider` pero no había nada registrado para cumplir esa dependencia.

Esto acaba de poner la bola en marcha. Hay mucho más para configurar y usar correctamente un contenedor de inyección de dependencias para situaciones reales. Este ejemplo solo cubre lo que es específico para agregar Windsor a una aplicación de servicio WCF. Si usas un contenedor diferente como Autofac o Unity, encontrarás que aunque la sintaxis y los detalles varían, en principio hacen lo mismo y podrás detectar fácilmente las similitudes.

Lea Cómo utilizar un contenedor de inyección de dependencia con un servicio WCF en línea: https://riptutorial.com/es/wcf/topic/5509/como-utilizar-un-contenedor-de-inyeccion-de-dependencia-con-un-servicio-wcf

Capítulo 3: Cómo: Deshabilitar / Habilitar el rastreo de WCF en el código de aplicación C

Examples

De una manera: use un escucha personalizado definido en su código C

Me tomó un tiempo hacerlo bien, así que decidí compartir una solución porque podría ahorrarle a alguien más varios días de prueba y error.

El problema: quiero poder habilitar / deshabilitar el seguimiento de WCF en mi aplicación C # .NET y elegir el nombre de archivo de salida de seguimiento. No quiero que los usuarios editen el archivo .config, hay demasiado espacio para el error allí.

Aquí hay una solución.

El archivo .config de la aplicación:

```
<?xml version="1.0"?>
<configuration>
  <system.diagnostics>
    <trace autoflush="true"/>
    <sources>
      <source name="System.ServiceModel" switchValue="All">
        <listeners>
          <add name="MyListener"/>
        </listeners>
      </source>
      <source name="System.ServiceModel.MessageLogging" switchValue="All">
        <listeners>
          <add name="MyListener"/>
        </listeners>
      </source>
      <source name="System.ServiceModel.Activation" switchValue="All">
        <listeners>
          <add name="MyListener"/>
        </listeners>
      </source>
      <source name="System.IdentityModel" switchValue="All">
        <listeners>
          <add name="MyListener"/>
        </listeners>
      </source>
    </sources>
    <sharedListeners>
      <add name="MyListener" type="MyNamespace.MyXmlListener, MyAssembly"/>
    </sharedListeners>
  </system.diagnostics>
  <system.serviceModel>
    <diagnostics wmiProviderEnabled="true">
```

```
      <messageLogging
        logEntireMessage="true"
        logMalformedMessages="true"
        logMessagesAtServiceLevel="true"
        logMessagesAtTransportLevel="true"
        maxMessagesToLog="1000"
        maxSizeOfMessageToLog="8192"/>
    </diagnostics>
  </system.serviceModel>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

Y el código C #:

```
using System;
using System.IO;
using System.Diagnostics;
namespace MyNamespace
{
    public class MyXmlListener : XmlWriterTraceListener
    {
        public static String TraceOutputFilename = String.Empty;

        public static Stream MakeOutputStream()
        {
            if (String.IsNullOrWhiteSpace(TraceOutputFilename))
                return Stream.Null;

            return new FileStream(TraceOutputFilename, FileMode.Create);
        }

        public MyXmlListener ()
            : base(MakeOutputStream())
        { }
    }
}
```

Para habilitar el seguimiento de WCF en un archivo, establezca TraceOutputFilename antes de crear el objeto WCF:

```
MyXmlListener.TraceOutputFilename = "trace.svclog";
```

He recibido grandes beneficios de este foro, espero que esta publicación pague un poco más adelante.

Obtener el "tipo" correcto en el archivo .config fue mucho más desafiante de lo que debería haber sido, vea Especificar nombres de tipo completamente calificados para configurar el "tipo" correctamente en un archivo .config.

Lea Cómo: Deshabilitar / Habilitar el rastreo de WCF en el código de aplicación C # en línea: https://riptutorial.com/es/wcf/topic/5559/como--deshabilitar---habilitar-el-rastreo-de-wcf-en-el-codigo-de-aplicacion-c-sharp

Capítulo 4: DataContractSerializer es un serializador Opt-In y Opt-Out.

Introducción

en realidad es tan simple: el enfoque Opt-In dice que las propiedades que se consideran parte de DataContract deben marcarse explícitamente; de lo contrario, se ignorarán, mientras que Opt-Out significa que todas las propiedades se asumirán como parte de DataContract a menos que se marquen explícitamente.

Examples

¿Qué es optar en serializador?

```
/// <summary>
/// Defines a student.
/// </summary>
[DataContract]
public class Student
{
    /// <summary>
    /// Gets or sets the student number.
    /// </summary>
    [DataMember]
    public string StudentNumber { get; set; }

    /// <summary>
    /// Gets or sets the first name.
    /// </summary>
    [DataMember]
    public string FirstName { get; set; }

    /// <summary>
    /// Gets or sets the last name.
    /// </summary>
    [DataMember]
    public string LastName { get; set; }

    /// <summary>
    /// Gets or sets the marks obtained.
    /// </summary>
    public int MarksObtained { get; set; }
}

/// <summary>
/// A service that provides the operations for students.
/// </summary>
[ServiceContract]
public interface IStudentService
{
    //Service contract code here.
}
```

En el código anterior `StudentNumber` , `FirstName` , `LastName` propiedades del `Student` clase están marcados explícitamente `DataMember` atributo como se oponen a `MarksObtained` , por lo `MarksObtained` será ignorado. De ignorado significa que `MarksObtained` no será la parte de los datos que pasan por el cable hacia / desde este servicio.

¿Qué es el serializador?

El código a continuación representa un ejemplo de enfoque de `NonSerialized` mediante el `NonSerialized` atributos `Serializable` y no `Serializable` .

```
/// <summary>
/// Represents a student.
/// </summary>
[Serializable]
public class Student
{
    /// <summary>
    /// Gets or sets student number.
    /// </summary>
    public string StudentNumber { get; set; }

    /// <summary>
    /// Gets or sets first name.
    /// </summary>
    public string FirstName { get; set; }

    /// <summary>
    /// Gets or sets last name.
    /// </summary>
    public string LastName { get; set; }

    /// <summary>
    /// Gets or sets marks obtained.
    /// </summary>
    [NonSerialized]
    public string MarksObtained { get; set; }
}

/// <summary>
/// A service that provides the operations for student.
/// </summary>
[ServiceContract]
public interface IStudentService
{
    //Service contract code here. Example given.

    /// <summary>
    /// Adds a student into the system.
    /// </summary>
    /// <param name="student">Student to be added.</param>
    [OperationContract]
    void AddStudent(Student student);
}
```

En el ejemplo anterior, marcamos explícitamente la propiedad `MarksObtained` como atributo `[NonSerialized]` , por lo que se ignorará excepto los demás. ¡Espero que esto ayude!

Lea DataContractSerializer es un serializador Opt-In y Opt-Out. en línea:
https://riptutorial.com/es/wcf/topic/9588/datacontractserializer-es-un-serializador-opt-in-y-opt-out-

Capítulo 5: Manejo de excepciones

Observaciones

Otras lecturas

Más sobre FaultException: MSDN FaultException

Examples

Mostrando información adicional cuando ocurre una excepción

Es importante manejar las excepciones en su servicio. Al desarrollar el servicio, puede configurar el WCF para proporcionar información más detallada, agregando esta etiqueta al archivo de configuración, generalmente Web.config:

```
 <serviceDebug includeExceptionDetailInFaults="true"/>
```

Esta etiqueta debe colocarse dentro de la etiqueta serviceBehavior, normalmente así:

```
<system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
</system.serviceModel>
```

Ejemplo de información detallada:

> Rastreo de la pila del servidor: en System.ServiceModel.Channels.ServiceChannel.HandleReply (Operación ProxyOperationRuntime, ProxyRpc & rpc) en System.ServiceModel.Channels.ServiceChannel.Call (String action, Boolean oneway, ProxyOperationazucación) , TimeSpan timeout) en System.ServiceModel.Channels.ServiceChannelProxy.InvokeService (IMethodCallMessage methodCall, ProxyOperationRuntime operation) en System.ServiceModel.Channels.ServiceChannelProxy.Invoke (IMessage message)
>
> Excepción rethrown en [0]: em System.Runtime.Remoting.Proxies.RealProxy.HandleReturnMessage (IMessage reqMsg, IMessage retMsg) en System.Runtime.gasas de los animales de la que se trata las personas de la que se trata las personas de la que se trata. RequestObterBeneficiario request) en MyServiceClient.GetDataOpration (RequestData request)

Esto devolverá al cliente información detallada. **Durante el desarrollo,** esto puede ayudar, pero cuando su servicio **entra en producción** , ya no lo mantendrá porque su servicio puede enviar datos confidenciales, como el nombre o la configuración de su base de datos.

Lanzar un mensaje fácil de usar con FaultException

Puede manejar las excepciones y lanzar un mensaje más amigable como 'Servicio no disponible' lanzando una FaultException:

```
try
{
    // your service logic here
}
catch (Exception ex)
{
    // You can do something here, like log the original exception
    throw new FaultException("There was a problem processing your request");
}
```

En su cliente, puede obtener fácilmente el mensaje:

```
try
{
    // call the service
}
catch (FaultException faultEx)
{
   var errorMessage = faultEx.Message;
   // do something with error message
}
```

Puede distinguir la excepción manejada de otras excepciones, como un error de red, agregando otras capturas a su código:

```
try
{
    // call the service
}
catch (FaultException faultEx)
{
   var errorMessage = faultEx.Message;
   // do something with error message
}
catch (CommunicationException commProblem)
{
    // Handle the communication error, like trying another endpoint service or logging
}
```

Lanzar una FaultException con un código de falla

La FaultException también puede incluir un **FaultCode** , que es una cadena de datos que puede usar para pasar información adicional, de modo que el cliente pueda distinguir diferentes excepciones:

```
try
{
    // your service logic here
}
catch (Exception ex)
{
   throw new FaultException("There was a problem processing your request",
      new FaultCode(("01"));
}
```

Obteniendo el FaultCode:

```
try
{
    // call the service
}
catch (FaultException faultEx)
{
   switch (faultEx.Code.Name)
   {
    case "01":
       // do something
       break;
    case "02":
        // do another something
        break
   }
}
```

Desacoplamiento ErrorHandlerAttribute al registrarse desde la implementación del servicio

Para desacoplar y reutilizar el mismo código de registro de error para **todos** sus servicios, necesita dos clases repetitivas y guardarlas en una biblioteca en algún lugar.

ErrorhandlerAttribute implementando IServiceBehavior. FaultErrorhandler implementa IErrorhandler que registra todas las excepciones.

```
[AttributeUsage(AttributeTargets.Class)]
public class ErrorHandlerAttribute : Attribute, IServiceBehavior
{
    Type mErrorType;
    public ErrorHandlerAttribute(Type t)
    {
        if (!typeof(IErrorHandler).IsAssignableFrom(t))
            throw new ArgumentException("Type passed to ErrorHandlerAttribute constructor must
inherit from IErrorHandler");
        mErrorType = t;
    }

    public void AddBindingParameters(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase, Collection<ServiceEndpoint> endpoints, BindingParameterCollection
bindingParameters)
    {
    }
```

```
    public void ApplyDispatchBehavior(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase)
    {
        foreach (ChannelDispatcher dispatcher in serviceHostBase.ChannelDispatchers)
        {
            dispatcher.ErrorHandlers.Add((IErrorHandler)Activator.CreateInstance(mErrorType));
        }
    }

    public void Validate(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase)
    {
    }
}


class FaultErrorHandler : IErrorHandler
{
    public bool HandleError(Exception error)
    {
        // LOG ERROR
        return false; // false so the session gets aborted
    }

    public void ProvideFault(Exception error, MessageVersion version, ref Message fault)
    {
    }
}
```

Luego, en su implementación de servicio, agregue el atributo ErrorHandler que pasa en la instancia de Tipo de FaultErrorHandler. ErrorHandler construirá una instancia de ese tipo en el que se llama HandleError.

```
[ServiceBehavior]
[ErrorHandler(typeof(FaultErrorHandler))]
public class MyService : IMyService
{
}
```

Usando un marco de registro de errores personalizado

A veces es útil integrar un marco de registro de errores personalizado para garantizar que se registran todas las excepciones.

```
[ServiceContract]
[ErrorHandler]
public interface IMyService
{

}

[AttributeUsage(AttributeTargets.Interface)]
public class CustomErrorHandler : Attribute, IContractBehavior, IErrorHandler
{
    public bool HandleError(Exception error)
    {
        return false;
```

```
    }

    public void ProvideFault(Exception error, MessageVersion version, ref Message fault)
    {
        if (error == null)
        {
            return;
        }

        //my custom logging framework
    }

    public void ApplyDispatchBehavior(ContractDescription contractDescription, ServiceEndpoint
endpoint, DispatchRuntime dispatchRuntime)
    {
        dispatchRuntime.ChannelDispatcher.ErrorHandlers.Add(this);
    }

    public void ApplyClientBehavior(ContractDescription contractDescription, ServiceEndpoint
endpoint,
        ClientRuntime clientRuntime)
    {

    }

    public void AddBindingParameters(ContractDescription contractDescription, ServiceEndpoint
endpoint,
        BindingParameterCollection bindingParameters)
    {

    }

    public void Validate(ContractDescription contractDescription, ServiceEndpoint endpoint)
    {

    }
}
```

Lea Manejo de excepciones en línea: https://riptutorial.com/es/wcf/topic/5557/manejo-de-excepciones

Capítulo 6: Publicación por entregas

Examples

Serialización en WCF

> La serialización es el proceso de convertir un objeto en un flujo de bytes para almacenar el objeto o transmitirlo a la memoria, una base de datos o un archivo. Serialización de la página de Microsoft

El siguiente ejemplo demuestra la serialización en WCF:

```
[ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples")]
public interface IPerson
{

    [OperationContract]
    void Add(Person person);

    [DataContract]
    public class Person
    {
        private int id;

        [DataMember]
        public int Age{ set; get;}
    }
}
```

1. `[DataContract]` atributo `[DataContract]` se usa con las clases. Aquí está decorado con clase de `Person` .
2. `[OperationContract]` se utiliza para los métodos. Aquí está decorado con el método `Add` .
3. `[DataMember]` se usa con las propiedades. los que están decorados con los `[DataMember]` solo estarán disponibles para que el proxy pueda acceder. Aquí tenemos 2 propiedades en las que no se puede acceder a la `id` se puede acceder a `Age` .
4. `[DataMember]` es útil cuando no quieres mostrar campos privados al mundo exterior y solo quieres mostrar propiedades públicas.
5. Con el `[DataMember]` tienes algunas propiedades que se adhieren a él. son los siguientes

   Propiedades de DataMember

a. `IsRequired` se puede usar de esta manera `[DataMember(IsRequired=true)]`

segundo. `Name` se puede usar así `[DataMember(Name="RegistrationNo")]`

do. `order` se puede utilizar de esta manera `[DataMember(order=1)]`

Sin especificar atributos, no podremos acceder a la clase / método / propiedad en los proyectos con los que trabajamos (por ejemplo, la interfaz de servicio wcf).

La forma en que estos atributos hacen que el código sea accesible a través de proyectos individuales en tiempo de ejecución se denomina "Serialización".

- Con WCF puede comunicarse con otros proyectos, aplicaciones o cualquier otro software mediante la serialización, **sin hacer** todo el trabajo de configurar los puntos finales, crear flujos manualmente y mantenerlos. Sin mencionar la conversión de todos los datos a bytes y viceversa.

Lea Publicación por entregas en línea: https://riptutorial.com/es/wcf/topic/2491/publicacion-por-entregas

Capítulo 7: Rastreo

Examples

Configuración de seguimiento

El rastreo de WCF está construido sobre System.Diagnostics. Para utilizar el seguimiento, debe definir los orígenes de seguimiento en el archivo de configuración o en el código.

El rastreo no está habilitado por defecto. Para activar el seguimiento, debe crear un detector de seguimiento y establecer un nivel de seguimiento que no sea "Desactivado" para la fuente de seguimiento seleccionada en la configuración; De lo contrario, WCF no genera ningún rastro.

El siguiente ejemplo muestra cómo habilitar el registro de mensajes y especificar opciones adicionales:

```
<configuration>
 <system.diagnostics>
  <sources>
    <source name="System.ServiceModel"
            switchValue="All"
            propagateActivity="true" >
      <listeners>
        <add name="wcf_trace"/>
      </listeners>
    </source>
    <source name="System.ServiceModel.MessageLogging">
      <listeners>
        <add name="wcf_trace"/>
      </listeners>
    </source>
  </sources>
  <sharedListeners>
    <add name="wcf_trace"
         type="System.Diagnostics.XmlWriterTraceListener"
         initializeData="c:\temp\wcf_trace\DistanceService.svclog" />
  </sharedListeners>
 </system.diagnostics>

 <system.serviceModel>
  <diagnostics wmiProviderEnabled="true">
      <messageLogging
           logEntireMessage="true"
           logMalformedMessages="true"
           logMessagesAtServiceLevel="true"
           logMessagesAtTransportLevel="true"
           maxMessagesToLog="3000" />
  </diagnostics>
 </system.serviceModel>
</configuration>
```

initializeData especifica el nombre del archivo de salida para ese oyente. Si no especifica un nombre de archivo, se generará un nombre de archivo aleatorio en función del tipo de escucha

utilizado.

Opciones y niveles de registro

- Nivel de servicio

Los mensajes registrados en esta capa están a punto de ingresar (al recibir) o dejar (al enviar) el código de usuario. Si se han definido filtros, solo se registran los mensajes que coinciden con los filtros. De lo contrario, todos los mensajes en el nivel de servicio se registran. Los mensajes de infraestructura (transacciones, canal de igual y seguridad) también se registran en este nivel, excepto los mensajes de mensajería confiable. En los mensajes transmitidos, solo se registran los encabezados. Además, los mensajes seguros se registran descifrados en este nivel.

- Nivel de transporte

Los mensajes registrados en esta capa están listos para ser codificados o decodificados para o después del transporte en el cable. Si se han definido filtros, solo se registran los mensajes que coinciden con los filtros. De lo contrario, todos los mensajes en la capa de transporte se registran. Todos los mensajes de infraestructura se registran en esta capa, incluidos los mensajes de mensajes confiables. En los mensajes transmitidos, solo se registran los encabezados. Además, los mensajes seguros se registran como cifrados en este nivel, excepto si se utiliza un transporte seguro como HTTPS.

- Nivel malformado

Los mensajes mal formados son mensajes que son rechazados por la pila WCF en cualquier etapa del procesamiento. Los mensajes mal formados se registran tal como están: cifrados si lo son, con XML no apropiado, y así sucesivamente. maxSizeOfMessageToLog definió el tamaño del mensaje que se registrará como CDATA. De forma predeterminada, maxSizeOfMessageToLog es igual a 256K. Para obtener más información sobre este atributo, consulte la sección Otras opciones.

Si desea deshabilitar el origen de seguimiento, debe usar los atributos logMessagesAtServiceLevel, logMalformedMessages y logMessagesAtTransportLevel del elemento messageLogging en su lugar. Debes establecer todos estos atributos en falso.

Lea Rastreo en línea: https://riptutorial.com/es/wcf/topic/1931/rastreo

Capítulo 8: Seguridad WCF

Examples

Seguridad WCF

La seguridad es una pieza fundamental de cualquier tecnología de programación o marco para implementar aplicaciones orientadas a servicios

WCF se ha creado desde cero para proporcionar la infraestructura de seguridad necesaria a nivel de mensaje y servicio.

En las siguientes secciones, verá cómo usar muchas de las configuraciones de seguridad disponibles en WCF y algunos escenarios de implementación comunes.

Para la protección de mensajes, WCF admite los dos modelos de seguridad tradicionales, seguridad de transporte y seguridad de mensajes.

Los enlaces, además de especificar el protocolo de comunicación y la codificación de los servicios, también le permitirán configurar las configuraciones de protección de mensajes y el esquema de autenticación.

Configuración de seguridad predeterminada en WCF:

UNIÓN	AJUSTES
WsHttpBinding	Seguridad de mensajes con autenticación de Windows
BasicHttpBinding	Sin seguridad
WsFederationHttpBinding	Seguridad de mensajes con autenticación federada
NetTcpBinding	Seguridad de transporte con autenticación de Windows
NetNamedPipeBinding	Seguridad de transporte con autenticación de Windows
NetMsmqBinding	Seguridad de transporte con autenticación de Windows

Considere el siguiente ejemplo:

```
<wsHttpBinding >
  <binding name=”UsernameBinding” >
   <security mode=”Message” >
     <message clientCredentialType=”UserName”/ >
   </security >
  </binding >
</wsHttpBinding >
```

En este ejemplo, el servicio se ha configurado con seguridad de mensajes y el perfil de token de seguridad de nombre de usuario. El resto de las configuraciones de seguridad para el enlace toman los valores predeterminados.

modo de seguridad

La configuración del modo de seguridad determina dos aspectos de seguridad fundamentales para cualquier servicio: el modelo de seguridad para la protección de mensajes y el esquema de autenticación de cliente compatible.

Modo de seguridad	Descripción
Ninguna	El servicio está disponible para todos, y los mensajes no están protegidos a medida que pasan por el transporte. Cuando se utiliza este modo, el servicio es vulnerable a cualquier tipo de ataque.
Transporte	Utiliza el modelo de seguridad de transporte para autenticar clientes y proteger los mensajes. Este modo proporciona las ventajas y desventajas discutidas en la seguridad del transporte.
Mensaje	Utiliza el modelo de seguridad de mensajes para autenticar clientes y proteger los mensajes. Este modo proporciona las ventajas y desventajas discutidas en la seguridad del mensaje.
Ambos	Utiliza los modelos de seguridad de transporte y seguridad de mensajes al mismo tiempo para autenticar a los consumidores del servicio y proteger los mensajes. Este modo solo es compatible con los enlaces de MSMQ y requiere las mismas credenciales en ambos niveles.
TransportWithMessageCredentials	La protección del mensaje es proporcionada por el transporte, y las credenciales para autenticar el servicio que los consumidores viajan como parte del mensaje. Este modo proporciona la flexibilidad de usar cualquiera de las credenciales o tipos de token admitidos en la autenticación de mensajes, mientras que la autenticación del servicio y la protección de mensajes se realizan a nivel de transporte.
TransportCredentialOnly	Utiliza la seguridad de transporte para autenticar clientes. El servicio no está autenticado, y los mensajes, incluidas las credenciales del cliente, van como texto sin formato a través del transporte. Este modo de seguridad puede ser útil para escenarios donde el tipo

Modo de seguridad	Descripción
	de información transmitida entre el cliente y el servicio no es sensible, aunque las credenciales también se exponen a cualquier persona.

Configure el WsHttpBinding para usar la seguridad de transporte con la autenticación básica

```
<bindings >
  <wsHttpBinding >
   <binding name="mybinding" >
    <security mode="Transport" >
     <transport clientCredentialType="Basic"/ >
    </security >
   </binding >
  </wsHttpBinding >
</bindings >
```

Lea Seguridad WCF en línea: https://riptutorial.com/es/wcf/topic/6021/seguridad-wcf

Capítulo 9: Tu primer servicio

Examples

1. Tu primer servicio y host.

Cree una interfaz decorada con un atributo ServiceContract y funciones miembro decoradas con el atributo OperationContract.

```
namespace Service
{
    [ServiceContract]
    interface IExample
    {
        [OperationContract]
        string Echo(string s);
    }
 }
```

Cree una clase concreta implementando esta interfaz y tendrá el servicio.

```
namespace Service
{
    public class Example : IExample
    {
        public string Echo(string s)
        {
            return s;
        }
    }
}
```

Luego crea el host desde donde se ejecutará el servicio. Esto puede ser cualquier tipo de aplicación. Consola, servicio, aplicación GUI o servidor web.

```
namespace Console
{
    using Service;

    class Console
    {
        Servicehost mHost;

        public Console()
        {
            mHost = new ServiceHost(typeof(Example));
        }

        public void Open()
        {
            mHost.Open();
        }
```

```
        public void Close()
        {
            mHost.Close();
        }

        public static void Main(string[] args)
        {
            Console host = new Console();

            host.Open();

            Console.Readline();

            host.Close();
        }
    }
}
```

La clase ServiceHost leerá el archivo de configuración para inicializar el servicio.

```
<system.serviceModel>
  <services>
    <service name="Service.Example">
      <endpoint name="netTcpExample" contract="Service.IExample" binding="netTcpBinding" />
      <host>
        <baseAddresses>
          <add baseAddress="net.tcp://localhost:9000/Example" />
        </baseAddresses>
      </host>
    </service>
  </services>
</system.serviceModel>
```

Primer cliente de servicio

Supongamos que tiene un servicio igual al que se define en el ejemplo "Primer servicio y host".

Para crear un cliente, defina la sección de configuración del cliente en la sección system.serviceModel de su archivo de configuración del cliente.

```
<system.serviceModel>
  <services>
    <client name="Service.Example">
      <endpoint name="netTcpExample" contract="Service.IExample" binding="netTcpBinding"
address="net.tcp://localhost:9000/Example" />
    </client>
  </services>
</system.serviceModel>
```

Luego copie la definición de contrato de servicio del servicio:

```
namespace Service
{
    [ServiceContract]
    interface IExample
    {
```

```
        [OperationContract]
        string Echo(string s);
    }
}
```

(NOTA: También puede consumir esto al agregar una referencia binaria al ensamblaje que contiene el contrato de servicio).

Luego, puede crear el cliente real utilizando `ChannelFactory<T>` y llamar a la operación en el servicio:

```
namespace Console
{
    using Service;

    class Console
    {
        public static void Main(string[] args)
        {
            var client = new
System.ServiceModel.ChannelFactory<IExample>("Service.Example").CreateChannel();
            var s = client.Echo("Hello World");
            Console.Write(s);
            Console.Read();
        }
    }
}
```

Agregar un punto final de metadatos a su servicio

Los servicios SOAP pueden publicar metadatos que describen los métodos que pueden invocar los clientes. Los clientes pueden usar herramientas como *Visual Studio* para generar código automáticamente (conocidos como *proxies de cliente*). Los proxies ocultan la complejidad de invocar un servicio. Para invocar un servicio, uno simplemente invoca un método en un proxy de cliente.

Primero debe agregar un punto final de metadatos a su servicio. Suponiendo que su servicio tenga el mismo aspecto que el definido en el ejemplo "Primer servicio y host", puede realizar los siguientes cambios en el archivo de configuración.

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="serviceBehaviour">
        <serviceMetadata httpGetEnabled="true" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <services>
    <service name="Service.Example" behaviorConfiguration="serviceBehaviour">
      <endpoint address="mex" binding="mexHttpBinding" name="mexExampleService"
contract="IMetadataExchange" />
      <endpoint name="netTcpExample" contract="Service.IExample" binding="netTcpBinding" />
      <host>
```

```
        <baseAddresses>
          <add baseAddress="net.tcp://localhost:9000/Example" />
          <add baseAddress="http://localhost:8000/Example" />
        </baseAddresses>
      </host>
    </service>
  </services>
</system.serviceModel>
```

El mexHttpbinding expone la interfaz a través de http, por lo que ahora puede ir con un navegador web para

http: // localhost: 8000 / Example http: // localhost: 8000 / Example? wsdl

y mostrará el servicio y sus metadatos.

Crear un ServiceHost programáticamente

Creando un ServiceHost programáticamente (**sin** archivo de configuración) en su forma más básica:

```
namespace ConsoleHost
{
  class ConsoleHost
  {
    ServiceHost mHost;

    public Console()
    {
      mHost = new ServiceHost(typeof(Example), new Uri("net.tcp://localhost:9000/Example"));

      NetTcpBinding tcp = new NetTcpBinding();

      mHost.AddServiceEndpoint(typeof(IExample),tcp,"net.tcp://localhost:9000/Example");

    }

    public void Open()
    {
        mHost.Open();
    }

    public void Close()
    {
        mHost.Close();
    }

    public static void Main(string[] args)
    {
        ConsoleHost host = new ConsoleHost();

        host.Open();

        Console.ReadLine();

        host.Close();
    }
```

```
    }
}
```

1. Cree una instancia de ServiceHost que pase el tipo de clase concreto y cero o más nombres básicos de Uri's.
2. Construya el enlace deseado, NetTcpBinding en este caso.
3. llame a AddServiceEndpoint pasando la dirección **A** , **B** inding y **C** ontract. (ABC mnemotécnica para los puntos finales de WCF).
4. Abre el host.
5. Mantenga el host abierto hasta que el usuario presione la tecla en la consola.
6. Cierra el host.

Agregar un punto final de metadatos a un servicio mediante programación

Cuando también desea exponer los metadatos sin un archivo de configuración, puede construir en el ejemplo creando un ServiceHost mediante programación:

```
public ConsoleHost()
{
    mHost = new ServiceHost(typeof(Example), new Uri("http://localhost:8000/Example"), new
Uri("net.tcp://9000/Example"));

    NetTcpBinding tcp = new NetTcpBinding();

    mHost.AddServiceEndpoint(typeof(IExample), tcp, "net.tcp://localhost:9000/Example");


    ServiceMetadataBehavior metaBehavior =
mHost.Description.Behaviors.Find<ServiceMetadataBehavior>();

    if (metaBehavior == null)
    {
        metaBehavior = new ServiceMetadataBehavior();
        metaBehavior.MetadataExporter.PolicyVersion = PolicyVersion.Policy15;
        metaBehavior.HttpGetEnabled = true;

        mHost.Description.Behaviors.Add(metaBehavior);
        mHost.AddServiceEndpoint(ServiceMetadataBehavior.MexContractName,
MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
    }

    mHost.Open();
}
```

1. Cree una instancia de ServiceHost que pase el tipo de clase concreto y cero o más nombres básicos de Uri's.
2. Cuando use mexHttpBinding, debe agregar http: // localhost: 8000 / Example baseaddress
3. Construya el enlace deseado, NetTcpBinding en este caso.
4. llame a AddServiceEndpoint pasando la dirección, el enlace y el contrato. (A B C).
5. Construir un comportamiento de ServiceMetadata
6. Establecer HttpGetEnabled en true
7. Agregue el comportamiento de los metadatos a la colección de comportamientos.
8. llame a AddServiceEndpoint pasando las constantes para el intercambio de metadatos

9. Abre el host.

Lea Tu primer servicio en línea: https://riptutorial.com/es/wcf/topic/2333/tu-primer-servicio

Capítulo 10: WCF - Http y Https en SOAP y REST

Examples

Muestra web.config

Aquí hay un ejemplo de `web.config` para tener compatibilidad con `http` y `https` .

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

  <appSettings>
    <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
  </appSettings>

  <system.web>
    <compilation debug="true" targetFramework="4.5.2" />
    <httpRuntime targetFramework="4.5.2" />
  </system.web>

  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="SoapBinding" />
      </basicHttpBinding>
      <basicHttpsBinding>
        <binding name="SecureSoapBinding" />
      </basicHttpsBinding>

      <webHttpBinding>
        <binding name="RestBinding" />
        <binding name="SecureRestBinding">
          <security mode="Transport" />
        </binding>
      </webHttpBinding>

      <mexHttpBinding>
        <binding name="MexBinding" />
      </mexHttpBinding>
      <mexHttpsBinding>
        <binding name="SecureMexBinding" />
      </mexHttpsBinding>
    </bindings>

    <client />

    <services>
      <service behaviorConfiguration="ServiceBehavior" name="Interface.Core">
        <endpoint address="soap" binding="basicHttpBinding" bindingConfiguration="SoapBinding"
name="Soap" contract="Interface.ICore" />
        <endpoint address="soap" binding="basicHttpsBinding"
bindingConfiguration="SecureSoapBinding" name="SecureSoap" contract="Interface.ICore" />
        <endpoint address="" behaviorConfiguration="Web" binding="webHttpBinding"
bindingConfiguration="RestBinding" name="Rest" contract="Interface.ICore" />
```

```
        <endpoint address="" behaviorConfiguration="Web" binding="webHttpBinding"
bindingConfiguration="SecureRestBinding" name="SecureRest" contract="Interface.ICore" />
        <endpoint address="mex" binding="mexHttpBinding" bindingConfiguration="MexBinding"
name="Mex" contract="IMetadataExchange" />
        <endpoint address="mex" binding="mexHttpsBinding"
bindingConfiguration="SecureMexBinding" name="SecureMex" contract="IMetadataExchange" />
      </service>
    </services>

    <behaviors>
      <endpointBehaviors>
        <behavior name="Web">
          <webHttp helpEnabled="true" defaultBodyStyle="Bare"
defaultOutgoingResponseFormat="Json" automaticFormatSelectionEnabled="true" />
        </behavior>
      </endpointBehaviors>

      <serviceBehaviors>
        <behavior name="ServiceBehavior">
          <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>

    <protocolMapping>
      <add binding="basicHttpsBinding" scheme="https" />
    </protocolMapping>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
multipleSiteBindingsEnabled="true" />
  </system.serviceModel>

  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true" />
    <directoryBrowse enabled="false" />
  </system.webServer>

</configuration>
```

Lea WCF - Http y Https en SOAP y REST en línea: https://riptutorial.com/es/wcf/topic/9604/wcf---http-y-https-en-soap-y-rest

Capítulo 11: WCF Restful Service

Examples

WCF Resful Service

```
[ServiceContract]
public interface IBookService
{
    [OperationContract]
    [WebGet]
    List<Book> GetBooksList();

    [OperationContract]
    [WebGet(UriTemplate  = "Book/{id}")]
    Book GetBookById(string id);

    [OperationContract]
    [WebInvoke(UriTemplate = "AddBook/{name}")]
    void AddBook(string name);

    [OperationContract]
    [WebInvoke(UriTemplate = "UpdateBook/{id}/{name}")]
    void UpdateBook(string id, string name);

    [OperationContract]
    [WebInvoke(UriTemplate = "DeleteBook/{id}")]
    void DeleteBook(string id);
}
```

Implementando el servicio

Ahora, la parte de implementación del servicio utilizará el contexto de entidad generado y las entidades para realizar todas las operaciones respectivas.

```
public class BookService : IBookService
{
    public List<Book> GetBooksList()
    {
        using (SampleDbEntities entities = new SampleDbEntities())
        {
            return entities.Books.ToList();
        }
    }

    public Book GetBookById(string id)
    {
        try
        {
            int bookId = Convert.ToInt32(id);

            using (SampleDbEntities entities = new SampleDbEntities())
            {
                return entities.Books.SingleOrDefault(book => book.ID == bookId);
            }
```

```
        }
        catch
        {
            throw new FaultException("Something went wrong");
        }
    }

    public void AddBook(string name)
    {
        using (SampleDbEntities entities = new SampleDbEntities())
        {
            Book book = new Book { BookName = name };
            entities.Books.AddObject(book);
            entities.SaveChanges();
        }
    }

    public void UpdateBook(string id, string name)
    {
        try
        {
            int bookId = Convert.ToInt32(id);

            using (SampleDbEntities entities = new SampleDbEntities())
            {
                Book book = entities.Books.SingleOrDefault(b => b.ID == bookId);
                book.BookName = name;
                entities.SaveChanges();
            }
        }
        catch
        {
            throw new FaultException("Something went wrong");
        }
    }

    public void DeleteBook(string id)
    {
        try
        {
            int bookId = Convert.ToInt32(id);

            using (SampleDbEntities entities = new SampleDbEntities())
            {
                Book book = entities.Books.SingleOrDefault(b => b.ID == bookId);
                entities.Books.DeleteObject(book);
                entities.SaveChanges();
            }
        }
        catch
        {
            throw new FaultException("Something went wrong");
        }
    }
}
```

Configuración del servicio WCF de descanso

Ahora, desde la perspectiva de ServiceContract, el servicio está listo para atender la solicitud REST, pero para acceder a este servicio sobre el resto necesitamos hacer algunos cambios en el

comportamiento del servicio y el enlace también.

Para hacer que el servicio esté disponible a través del protocolo REST, el enlace que se debe utilizar es webHttpBinding. Además, debemos establecer la configuración de comportamiento del punto final y definir el parámetro webHttp en el punto final del comportamiento. Así que nuestra configuración resultante se verá algo como:

```
<system.serviceModel>
  <services>
    <service name="WcfRestSample.BookService">
      <endpoint address="" behaviorConfiguration="restfulBehavior"
        binding="webHttpBinding" bindingConfiguration="" contract="WcfRestSample.IBook
      <host>
        <baseAddresses>
          <add baseAddress="http://localhost/bookservice" />
        </baseAddresses>
      </host>
    </service>
  </services>
  <behaviors>
    <endpointBehaviors>
      <behavior name="restfulBehavior">
        <webHttp />
      </behavior>
    </endpointBehaviors>
```

Lea WCF Restful Service en línea: https://riptutorial.com/es/wcf/topic/3041/wcf-restful-service

Creditos

S. No	Capítulos	Contributors
1	Empezando con wcf	Community, MickyD, Sathish Prabhakaran, Uriil
2	Cómo utilizar un contenedor de inyección de dependencia con un servicio WCF	Scott Hannen
3	Cómo: Deshabilitar / Habilitar el rastreo de WCF en el código de aplicación C #	MikeZ
4	DataContractSerializer es un serializador Opt-In y Opt-Out.	David, Yawar Murtaza
5	Manejo de excepciones	Kye, Laurijssen, Ricardo Pontual
6	Publicación por entregas	Ameya Deshpande, Bardia, Gal Yedidovich
7	Rastreo	esiprogrammer, Eugene S.
8	Seguridad WCF	esiprogrammer
9	Tu primer servicio	Laurijssen, MickyD, Piyush Parashar, tom redfern
10	WCF - Http y Https en SOAP y REST	David
11	WCF Restful Service	Nirav Mehta