eBook Gratuit

APPRENEZ wcf

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#wcf

Table des matières

À propos

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.

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

Chapitre 1: Démarrer avec wcf

Remarques

Cette section fournit une vue d'ensemble de ce que wcf est et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tout sujet important dans wcf, et établir un lien avec les sujets connexes. La documentation de wcf étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Démonstration de WCF Restful Service

svc

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

Interface

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

svc Markup (Clic droit sur svc file & click view MarkUp)

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

Configuration 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>
```

Maintenant, exécutez simplement le service ou l'hôte dans un port. Et accédez au service à l'aide de " http: // nom_hôte / WCFRestfulService / GetServiceName? Id = 1 "

Service WCF simple

Les exigences minimales pour le service WCF sont un contrat de service avec un contrat de fonctionnement.

Contrat de service:

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

Mise en place du contrat de service:

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

Fichier de configuration:

```
<?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; }
}
```

Lire Démarrer avec wcf en ligne: https://riptutorial.com/fr/wcf/topic/992/demarrer-avec-wcf

Chapitre 2: Comment utiliser un conteneur d'injection de dépendance avec un service WCF

Examples

Comment configurer un service WCF pour utiliser un conteneur d'injection de dépendance (Castle Windsor)

Cet exemple comprend deux parties: des étapes standard pour ajouter Castle Windsor à votre service WCF, puis un exemple simple et concret pour montrer comment configurer et utiliser le conteneur de Windsor.

Cela rend l'exemple un peu long. Si vous avez déjà compris l'utilisation d'un conteneur DI, vous ne vous souciez probablement que des étapes standard. Si l'utilisation d'un conteneur de DI n'est pas familière, cela prend un peu plus de temps - voir le tout fonctionner de bout en bout - avant que cela ne soit logique.

Etapes de la chaudière

1. Ajoutez le package Nuget de l' installation d'intégration Castle Windsor WCF à votre application de service WCF. Cela ajoutera des références à Castle Windsor ainsi que des composants spécifiques aux services WCF.

2. Ajoutez une classe d'application globale (global.asax) à votre projet: Ajouter> Nouvel élément> Visual C #> Web> Classe d'application globale.
   Le code qui configure votre conteneur doit être appelé à partir de la méthode `Application_Start` . Pour le garder organisé, nous pouvons placer toute notre configuration de conteneur dans une classe distincte. Nous n'avons pas besoin Cela n'a pas d'importance. Vous verrez cela se faire différemment dans différents exemples. Ce qui compte, c'est qu'il soit appelé à partir de la méthode `Application_Start` car c'est votre *racine de composition* - où l'application démarre. L'idée est de configurer votre conteneur au démarrage de l'application et de ne plus jamais le toucher directement. Il reste juste en arrière-plan en faisant son travail.

3. Créez une classe pour configurer le conteneur. Cela fait deux choses:

   - `ContainerInstance.AddFacility<WcfFacility>()` indique au code WCF de Windsor d'utiliser ce conteneur particulier lors de la création d'instances de vos services WCF.
   - `ContainerInstance.Install(FromAssembly.This())` indique à Windsor d'analyser `This` assembly (en d'autres termes, votre projet WCF) à la recherche des classes qui implémentent `IWindsorInstaller` . Ces classes fourniront des instructions pour indiquer à votre conteneur comment résoudre des dépendances. (Nous en créerons quelques unes plus tard.)

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

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

4. Appelez cette méthode depuis `Application_Start` dans votre global.asax:

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

5. Créer un installateur Ceci est juste une classe qui implémente `IWindsorInstaller` . Celui-ci est vide. Ça ne fait rien. Nous allons ajouter à cette classe en quelques étapes. Lorsque nous appelons `ContainerInstance.Install(FromAssembly.This())` , `ContainerInstance` est transmis à la méthode `Install` afin que nous puissions enregistrer des dépendances avec le conteneur.

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

6. Dans le balisage de tout service WCF que vous créez, ajoutez cette directive. Cela indique que l'application utilisera la fonction WCF de Windsor pour créer des instances du service, ce qui signifie que in peut injecter des dépendances lorsqu'une instance est créée.

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

Cela met fin aux étapes standard pour l'installation du service afin d'utiliser un conteneur d'injection de dépendance de Castle Windsor.

Mais l'exemple n'est pas complet à moins de configurer au moins un service WCF pour l'injection de dépendance. Le reste n'est pas un exemple, mais un exemple simple et concret.

1. Créez un nouveau service WCF appelé `GreetingService.svc` .
2. Modifier le balisage Ça devrait ressembler à ça:

```
<%@ ServiceHost Language="C#" Debug="true"
```

```
    Service="WcfWindsorDocumentation.GreetingService"
    CodeBehind="GreetingService.svc.cs"
    Factory="Castle.Facilities.WcfIntegration.DefaultServiceHostFactory,
Castle.Facilities.WcfIntegration"
 %>
```

3. Remplacez `IGreetingService` (le contrat de service) par ceci:

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

4. Remplacez `GreetingService` (dans GreetingService.svc.cs) par ce code. Notez que le constructeur requiert une instance de `IGreetingProvider` dont nous avons besoin de notre conteneur pour injecter.

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

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

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

5. Ajoutez cette implémentation de `IGreetingProvider` . Il a également quelques dépendances dont nous avons besoin du conteneur. Les spécificités de ces classes ne sont pas trop importantes. Ils sont juste pour créer quelque chose de facile à suivre.

```
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. Maintenant, nous avons toutes les classes dont nous avons besoin. Il ne reste plus qu'à enregistrer les dépendances avec notre conteneur. En d'autres termes, nous allons dire au conteneur quelles classes il doit créer pour qu'il sache comment "construire" une instance de `GreetingService` . Ce code entre dans notre implémentation de `IWindsorInstaller` (étape 5 du code standard).

```
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>());
    }
}
```

Cela indique au conteneur:

- Il est responsable de la création de `GreetingService` cas de besoin.
- Quand il essaiera de créer `GreetingService` il aura besoin d'un `IGreetingProvider` . Lorsque cela est nécessaire, il doit créer un `ComputerNameGreetingProvider` .
- Lorsqu'il essaie de créer `ComputerNameGreetingProvider` , cette classe requiert un `IComputerNameProvider` . Il doit créer une instance de `EnvironmentComputerNameProvider` pour répondre à ce besoin.

Si, quelque part au cours de la création de `GreetingService` ou de l’une de ses dépendances, il est nécessaire de disposer d’une dépendance que nous n’avons pas enregistrée, elle nous le fera savoir avec une exception utile, comme celle-ci.

> Dépendance manquante
> Le composant WcfWindsorDocumentation.ComputerNameGreetingProvider a une dépendance à WcfWindsorDocumentation.IComputerNameProvider, qui n'a pas pu être résolue.
> Assurez-vous que la dépendance est correctement enregistrée dans le conteneur en tant que service ou fournie en tant qu'argument en ligne.

Cela signifie que quelque chose dépend de `IComputerNameProvider` mais rien n’a été enregistré pour

satisfaire cette dépendance.

Cela ne fait que commencer. Il y a beaucoup plus à configurer correctement et à utiliser un conteneur d'injection de dépendance pour des scénarios réels. Cet exemple ne couvre que ce qui est spécifique à l'ajout de Windsor à une application de service WCF. Si vous utilisez un conteneur différent comme Autofac ou Unity, vous constaterez que, même si la syntaxe et les détails varient, ils font en principe la même chose et vous pouvez facilement identifier les similitudes.

Lire Comment utiliser un conteneur d'injection de dépendance avec un service WCF en ligne: https://riptutorial.com/fr/wcf/topic/5509/comment-utiliser-un-conteneur-d-injection-de-dependance-avec-un-service-wcf

Chapitre 3: Comment: désactiver / activer le suivi WCF dans le code d'application C

Examples

Un moyen: utiliser un écouteur personnalisé défini dans votre code C

Il m'a fallu du temps pour bien faire les choses, alors j'ai décidé de partager une solution car cela pourrait faire épargner plusieurs jours d'essais et d'erreurs à quelqu'un d'autre.

Le problème: je veux pouvoir activer / désactiver le suivi WCF dans mon application C # .NET et choisir le nom du fichier de sortie de trace. Je ne veux pas que les utilisateurs éditent le fichier .config, il y a trop de place pour les erreurs.

Voici une solution.

Le fichier .config de l'application:

```
<?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>
```

Et le code 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())
        { }
    }
}
```

Pour activer le suivi WCF dans un fichier, définissez TraceOutputFilename avant que l'objet WCF ne soit créé:

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

J'ai eu de grands avantages de ce forum, j'espère que ce post paye une partie de ce projet.

Obtenir le "type" droit dans le fichier .config était beaucoup plus difficile que cela aurait dû être, consultez Spécifier des noms de type entièrement qualifiés pour définir le "type" correctement dans un fichier .config.

Lire Comment: désactiver / activer le suivi WCF dans le code d'application C # en ligne: https://riptutorial.com/fr/wcf/topic/5559/comment--desactiver---activer-le-suivi-wcf-dans-le-code-d-application-c-sharp

Chapitre 4: DataContractSerializer est un sérialiseur opt-in et opt-out.

Introduction

En fait, c'est si simple: l'approche Opt-In indique que les propriétés considérées comme faisant partie de DataContract doivent être explicitement marquées, sinon elles seront ignorées, tandis que l'option Opt-Out signifie que toutes les propriétés feront partie du contrat sauf indication explicite.

Examples

Qu'est-ce que opt in sérialiseur

```
/// <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.
}
```

Dans le code ci-dessus `StudentNumber` , `FirstName` , les propriétés `LastName` de `Student` classe `Student` sont explicitement marquées avec l'attribut `DataMember` comme opposé à `MarksObtained` , donc `MarksObtained` sera ignoré. De ignoré, cela signifie que `MarksObtained` ne sera pas la partie des données traversant le réseau vers / depuis ce service.

Qu'est-ce que le sérialiseur opt-out?

Le code ci-dessous représente un exemple d'approche opt-out utilisant des attributs `Serializable` et `NonSerialized` .

```
/// <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);
}
```

Dans l'exemple ci-dessus, nous avons explicitement marqué la propriété `MarksObtained` comme attribut `[NonSerialized]` , il sera donc ignoré à l'exception des autres. J'espère que cela t'aides!

Lire DataContractSerializer est un sérialiseur opt-in et opt-out. en ligne: https://riptutorial.com/fr/wcf/topic/9588/datacontractserializer-est-un-serialiseur-opt-in-et-opt-out-

Chapitre 5: Gestion des exceptions

Remarques

Lectures complémentaires

En savoir plus sur FaultException: MSDN FaultException

Examples

Affichage d'informations supplémentaires lorsqu'une exception se produit

Il est important de gérer les exceptions dans votre service. Lors du développement du service, vous pouvez définir WCF pour fournir des informations plus détaillées, en ajoutant cette balise au fichier de configuration, généralement Web.config:

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

Cette balise doit être placée dans la balise serviceBehavior, généralement comme ceci:

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

Exemple d'information détaillée:

> Trace de la pile du serveur: em System.ServiceModel.Channels.ServiceChannel.HandleReply (opération ProxyOperationRuntime, ProxyRpc & rpc) em System.ServiceModel.Channels.ServiceChannel.Call (action String, Boolean oneway, opération ProxyOperationRuntime, Object [] ins, Object [] outs , TimeSpan timeout) em System.ServiceModel.Channels.ServiceChannelProxy.InvokeService (méthode IMethodCallMessageCall, opération ProxyOperationRuntime) em System.ServiceModel.Channels.ServiceChannelProxy.Invoke (message IMessage)
>
> Exception repoussée à [0]: em System.Runtime.Remoting.Proxies.RealProxy.HandleReturnMessage (IMessage reqMsg, IMessage retMsg) em System.Runtime.Remoting.Proxies.RealProxy.PrivateInvoke (MessageData & msgData, type Int32) em IMyService.GetDataOperation (Requête RequestObterBeneficiario) dans MyServiceClient.GetDataOpration (requête

RequestData)

Cela retournera au client des informations détaillées. **Pendant le développement,** cela peut vous aider, mais lorsque votre service **est mis en production** , vous ne le conserverez plus car votre service peut envoyer des données sensibles, comme le nom ou la configuration de votre base de données.

Lancer un message convivial avec FaultException

Vous pouvez gérer les exceptions et lancer un message le plus convivial, tel que "Service indisponible", générant une exception 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");
}
```

Dans votre client, vous pouvez facilement obtenir le message:

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

Vous pouvez distinguer l'exception gérée des autres exceptions, comme une erreur réseau, en ajoutant d'autres captures à votre code:

```
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
}
```

Lancer une exception FaultException avec un code d'erreur

L'exception FaultException peut également inclure un **FaultCode** , c'est-à-dire une chaîne de données que vous pouvez utiliser pour transmettre des informations supplémentaires, afin que le client puisse distinguer différentes exceptions:

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

Obtenir le code d'erreur:

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

Découplage de ErrorHandlerAttribute enregistré à partir de l'implémentation du service

Pour découpler et réutiliser le même code de journalisation des erreurs pour **tous** vos services, vous avez besoin de deux classes standard et de les ranger quelque part dans une bibliothèque.

ErrorhandlerAttribute implémentant IServiceBehavior. FaultErrorhandler implémentant IErrorhandler qui consigne toutes les exceptions.

```
[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)
    {
    }
}
```

Ensuite, dans votre implémentation de service, ajoutez l'attribut ErrorHandler qui passe dans l'instance de type FaultErrorHandler. ErrorHandler va construire une instance à partir de ce type sur lequel HandleError est appelé.

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

Utilisation d'un cadre de journalisation des erreurs personnalisé

Il est parfois utile d'intégrer une structure de journalisation des erreurs personnalisée pour garantir que toutes les exceptions sont consignées.

```
[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)
    {

    }
}
```

Lire Gestion des exceptions en ligne: https://riptutorial.com/fr/wcf/topic/5557/gestion-des-exceptions

Chapitre 6: La sérialisation

Examples

Sérialisation en WCF

> La sérialisation est le processus de conversion d'un objet en flux d'octets afin de stocker l'objet ou de le transmettre en mémoire, dans une base de données ou dans un fichier. Sérialisation de la page Microsoft

L'exemple suivant illustre la sérialisation dans 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]` attribut est utilisé avec les classes. Ici, il est décoré avec la classe `Person` .
2. `[OperationContract]` est utilisé pour les méthodes. Ici, il est décoré avec la méthode `Add` .
3. `[DataMember]` attribut est utilisé avec les propriétés. ceux qui sont décorés avec les attributs `[DataMember]` uniquement ceux qui seront accessibles au proxy. Ici, nous avons 2 propriétés dans cet `id` n'est pas accessible et `Age` est accessible.
4. `[DataMember]` attribut est pratique lorsque vous ne souhaitez pas afficher les champs privés vers l'extérieur et que vous souhaitez uniquement afficher les propriétés publiques.
5. Avec l'attribut `[DataMember]` vous avez des propriétés qui s'y rattachent. ils sont comme suit

   Propriétés de DataMember

une. `IsRequired` peut être utilisé comme ceci `[DataMember(IsRequired=true)]`

b. `Name` peut être utilisé comme ceci `[DataMember(Name="RegistrationNo")]`

c. `order` peut être utilisé comme ceci `[DataMember(order=1)]`

Sans spécifier d'attributs, nous ne pourrons pas accéder à la classe / méthode / propriété dans les

projets avec lesquels nous travaillons (cette interface de service wcf unique).

La manière dont ces attributs rendent le code accessible via des projets individuels au moment de l'exécution est appelée "Sérialisation".

- Avec WCF, vous pouvez communiquer avec d'autres projets, applications ou tout autre logiciel à l'aide de la sérialisation, **sans** avoir à configurer les terminaux, à créer des flux manuellement et à les gérer. Sans oublier la conversion de toutes les données en octets et inversement.

Lire La sérialisation en ligne: https://riptutorial.com/fr/wcf/topic/2491/la-serialisation

Chapitre 7: Service de repos WCF

Examples

Service Reserv WCF

```
[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);
}
```

Implémentation du service

Maintenant, la partie d'implémentation de service utilisera le contexte et les entités générés par la structure d'entité pour effectuer toutes les opérations respectives.

```
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");
        }
    }
}
```

Configuration du service WCF reposant

À présent, du point de vue de ServiceContract, le service est prêt à répondre à la demande REST, mais pour accéder à ce service, nous devons également modifier le comportement et la liaison du

service.

Pour rendre le service disponible via le protocole REST, la liaison à utiliser est webHttpBinding. Nous devons également définir la configuration du comportement du noeud final et définir le paramètre webHttp dans endpointBehavior. Donc, notre configuration résultante ressemblera à quelque chose comme:

```
<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>
```

Lire Service de repos WCF en ligne: https://riptutorial.com/fr/wcf/topic/3041/service-de-repos-wcf

Chapitre 8: Tracé

Examples

Paramètres de suivi

Le suivi WCF est construit sur System.Diagnostics. Pour utiliser le traçage, vous devez définir des sources de trace dans le fichier de configuration ou dans le code.

Le suivi n'est pas activé par défaut. Pour activer le traçage, vous devez créer un écouteur de trace et définir un niveau de trace autre que "Off" pour la source de trace sélectionnée dans la configuration. sinon, WCF ne génère aucune trace.

L'exemple suivant montre comment activer la journalisation des messages et spécifier des options supplémentaires:

```
<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 spécifie le nom du fichier de sortie pour cet écouteur. Si vous ne spécifiez pas de nom de fichier, un nom de fichier aléatoire est généré en fonction du type d'écouteur utilisé.

Niveaux de journalisation et options

- Niveau de service

Les messages enregistrés sur cette couche sont sur le point d'entrer (lors de la réception) ou de laisser (lors de l'envoi) le code utilisateur. Si des filtres ont été définis, seuls les messages correspondant aux filtres sont consignés. Sinon, tous les messages au niveau du service sont consignés. Les messages d'infrastructure (transactions, canal homologue et sécurité) sont également consignés à ce niveau, à l'exception des messages de messagerie fiable. Sur les messages en streaming, seuls les en-têtes sont enregistrés. De plus, les messages sécurisés sont consignés à ce niveau.

- Niveau de transport

Les messages enregistrés sur cette couche sont prêts à être encodés ou décodés pour ou après le transport sur le câble. Si des filtres ont été définis, seuls les messages correspondant aux filtres sont consignés. Sinon, tous les messages de la couche de transport sont consignés. Tous les messages d'infrastructure sont enregistrés sur cette couche, y compris les messages de messagerie fiables. Sur les messages en streaming, seuls les en-têtes sont enregistrés. De plus, les messages sécurisés sont consignés sous forme chiffrée à ce niveau, sauf si un transport sécurisé tel que HTTPS est utilisé.

- Niveau mal formé

Les messages mal formés sont des messages rejetés par la pile WCF à n'importe quel stade du traitement. Les messages mal formés sont consignés tels quels: chiffrés s'ils le sont, avec du XML non approprié, etc. maxSizeOfMessageToLog a défini la taille du message à consigner en tant que CDATA. Par défaut, maxSizeOfMessageToLog est égal à 256K. Pour plus d'informations sur cet attribut, consultez la section Autres options.

Si vous souhaitez désactiver la source de trace, vous devez utiliser les attributs logMessagesAtServiceLevel, logMalformedMessages et logMessagesAtTransportLevel de l'élément messageLogging. Vous devez définir tous ces attributs sur false.

Lire Tracé en ligne: https://riptutorial.com/fr/wcf/topic/1931/trace

Chapitre 9: Votre premier service

Examples

1. Votre premier service et hôte

Créez une interface décorée avec un attribut ServiceContract et des fonctions membres décorées avec l'attribut OperationContract.

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

Créez une classe concrète implémentant cette interface et vous avez le service.

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

Ensuite, créez l'hôte à partir duquel le service sera exécuté. Cela peut être n'importe quel type d'application. Console, service, application graphique ou serveur 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 classe ServiceHost lit le fichier de configuration pour initialiser le service.

```
<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>
```

Premier client de service

Disons que vous avez un service identique à celui défini dans l'exemple "Premier service et hôte".

Pour créer un client, définissez la section de configuration du client dans la section system.serviceModel de votre fichier de configuration client.

```
<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>
```

Ensuite, copiez la définition du contrat de service à partir du service:

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

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

(REMARQUE: vous pouvez également utiliser ceci en ajoutant une référence binaire à l'assembly contenant le contrat de service à la place.)

Vous pouvez ensuite créer le client réel à l'aide de `ChannelFactory<T>` et appeler l'opération sur le service:

```
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();
        }
    }
}
```

Ajout d'un point de terminaison de métadonnées à votre service

Les services SOAP peuvent publier des métadonnées décrivant les méthodes pouvant être appelées par les clients. Les clients peuvent utiliser des outils tels que *Visual Studio* pour générer automatiquement du code (appelé *proxy client*). Les proxys masquent la complexité de l'appel d'un service. Pour appeler un service, on appelle simplement une méthode sur un proxy client.

Vous devez d'abord ajouter un point de terminaison de métadonnées à votre service. En supposant que votre service ressemble à celui défini dans l'exemple «Premier service et hôte», vous pouvez apporter les modifications suivantes au fichier de configuration.

```
<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>
```

Le mexHttpbinding expose l'interface sur http, vous pouvez donc maintenant utiliser un navigateur Web pour

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

et il affichera le service et ses métadonnées.

Créer un ServiceHost par programme

Créer un ServiceHost par programme (**sans** fichier de configuration) dans sa forme la plus élémentaire:

```
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. Créez une instance de ServiceHost en transmettant le type de classe Concrete et zéro ou plus de Uead.
2. Construisez la liaison souhaitée, NetTcpBinding dans ce cas.
3. appelez AddServiceEndpoint en passant les **A** dresse, **B** inding et **C** ontract. (Mnémonique ABC pour les points de terminaison WCF).
4. Ouvrez l'hôte.
5. Gardez l'hôte ouvert jusqu'à ce que l'utilisateur appuie sur la touche sur la console.
6. Fermez l'hôte.

Ajout par programme d'un point de terminaison de métadonnées à un service

Lorsque vous souhaitez également exposer des métadonnées sans fichier de configuration, vous pouvez utiliser l'exemple pour créer un ServiceHost par programmation:

```
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. Créez une instance de ServiceHost en transmettant le type de classe Concrete et zéro ou plus de Uead.
2. Lorsque vous utilisez mexHttpBinding, vous devez ajouter http: // localhost: 8000 / Example baseaddress
3. Construisez la liaison souhaitée, NetTcpBinding dans ce cas.
4. appelez AddServiceEndpoint en passant l'adresse, la liaison et le contrat. (ABC).
5. Construire un serviceMetadataBehavior
6. Définissez HttpGetEnabled sur true
7. Ajoutez le comportement des métadonnées à la collection de comportements.
8. appeler AddServiceEndpoint en passant les constantes pour l'échange de métadonnées
9. Ouvrez l'hôte.

Lire Votre premier service en ligne: https://riptutorial.com/fr/wcf/topic/2333/votre-premier-service

Chapitre 10: WCF - Http et Https sur SOAP et REST

Examples

Échantillon web.config

Voici un exemple `web.config` afin d'avoir à la fois le support `http` et `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>
```

Lire WCF - Http et Https sur SOAP et REST en ligne: https://riptutorial.com/fr/wcf/topic/9604/wcf---http-et-https-sur-soap-et-rest

Chapitre 11: WCF Sécurité

Examples

WCF Sécurité

La sécurité est un élément essentiel de toute technologie de programmation ou de tout cadre pour la mise en œuvre d'applications orientées services.

WCF a été conçu dès le départ pour fournir l'infrastructure de sécurité nécessaire au niveau des messages et des services.

Dans les sections suivantes, vous verrez comment utiliser de nombreux paramètres de sécurité disponibles dans WCF et certains scénarios de déploiement courants.

Pour la protection des messages, WCF prend en charge les deux modèles de sécurité traditionnels, la sécurité du transport et la sécurité des messages.

Les liaisons, en plus de spécifier le protocole de communication et le codage des services, vous permettront également de configurer les paramètres de protection des messages et le schéma d'authentification.

Paramètres de sécurité par défaut dans WCF:

CONTRAIGNANT	PARAMÈTRES
WsHttpBinding	Sécurité des messages avec authentification Windows
BasicHttpBinding	Pas de sécurité
WsFederationHttpBinding	Sécurité des messages avec authentification fédérée
NetTcpBinding	Transport Security avec Windows Authenticatio
NetNamedPipeBinding	Sécurité de transport avec authentification Windows
NetMsmqBinding	Sécurité de transport avec authentification Windows

Prenons l'exemple suivant:

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

Dans cet exemple, le service a été configuré avec la sécurité des messages et le profil de jeton de sécurité du nom d'utilisateur. Les autres paramètres de sécurité pour la liaison prennent les valeurs par défaut.

Mode sécurité

Le paramètre de mode de sécurité détermine deux aspects de sécurité fondamentaux pour tout service: le modèle de sécurité pour la protection des messages et le schéma d'authentification du client pris en charge.

Mode sécurité	La description
Aucun	Le service est disponible pour tous et les messages ne sont pas protégés pendant le transport. Lorsque ce mode est utilisé, le service est vulnérable à tout type d'attaque.
Transport	Utilise le modèle de sécurité de transport pour authentifier les clients et protéger les messages. Ce mode fournit les avantages et les inconvénients de la sécurité des transports.
Message	Utilise le modèle de sécurité des messages pour authentifier les clients et protéger les messages. Ce mode fournit les avantages et les inconvénients de la sécurité des messages.
Tous les deux	Utilise les modèles de sécurité de transport et de sécurité des messages en même temps pour authentifier les consommateurs de services et protéger les messages. Ce mode est uniquement pris en charge par les liaisons MSMQ et requiert les mêmes informations d'identification aux deux niveaux.
TransportWithMessageCredentials	La protection des messages est assurée par le transport et les informations d'identification pour authentifier les consommateurs de services font partie du message. Ce mode offre la possibilité d'utiliser les informations d'identification ou les types de jetons pris en charge dans l'authentification des messages lorsque l'authentification du service et la protection des messages sont effectuées au niveau du transport.
TransportCredentialOnly	Utilise la sécurité de transport pour l'authentification des clients Le service n'est pas authentifié et les messages, y compris les informations d'identification du client, sont envoyés en texte brut via le transport. Ce mode de sécurité peut être utile pour des scénarios où le type

Mode sécurité	La description
	d'informations transmises entre le client et le service n'est pas sensible, bien que les informations d'identification soient également exposées à quiconque.

Configurez le WsHttpBinding pour utiliser la sécurité du transport avec l'authentification de base

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

Lire WCF Sécurité en ligne: https://riptutorial.com/fr/wcf/topic/6021/wcf-securite

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec wcf	Community, MickyD, Sathish Prabhakaran, Uriil
2	Comment utiliser un conteneur d'injection de dépendance avec un service WCF	Scott Hannen
3	Comment: désactiver / activer le suivi WCF dans le code d'application C #	MikeZ
4	DataContractSerializer est un sérialiseur opt-in et opt-out.	David, Yawar Murtaza
5	Gestion des exceptions	Kye, Laurijssen, Ricardo Pontual
6	La sérialisation	Ameya Deshpande, Bardia, Gal Yedidovich
7	Service de repos WCF	Nirav Mehta
8	Tracé	esiprogrammer, Eugene S.
9	Votre premier service	Laurijssen, MickyD, Piyush Parashar, tom redfern
10	WCF - Http et Https sur SOAP et REST	David
11	WCF Sécurité	esiprogrammer