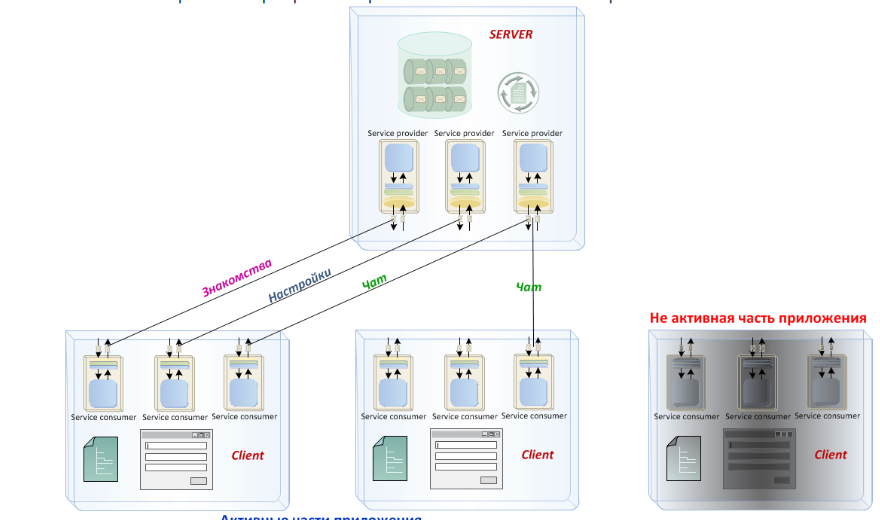
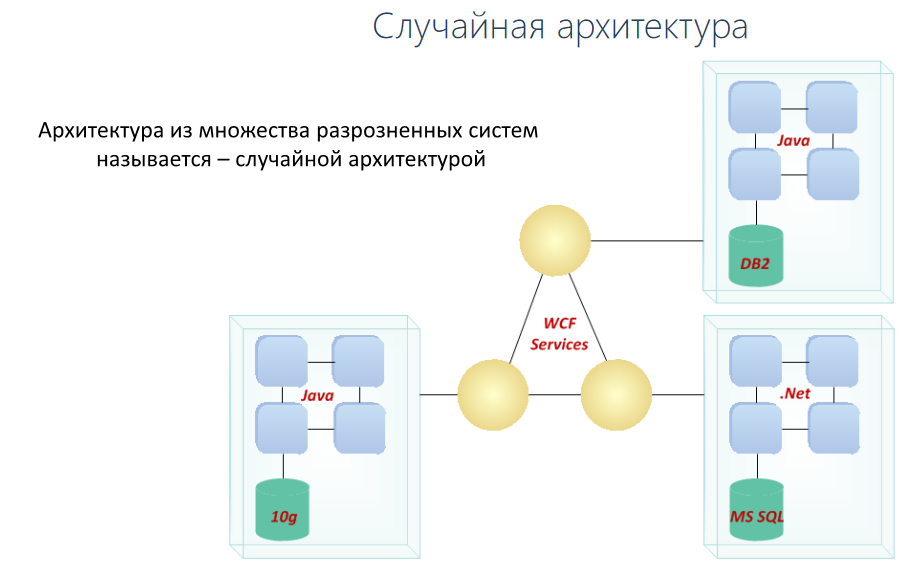
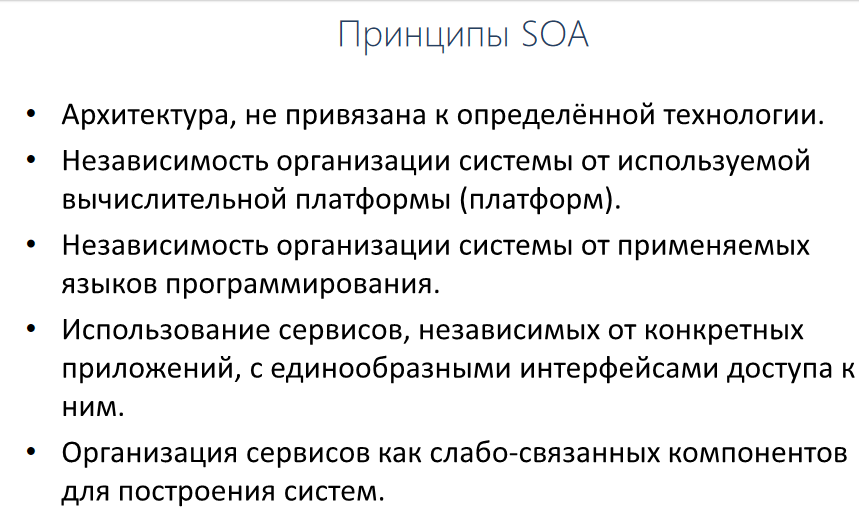
***WCF*** – это технология, предназначенная для проектирования, построения, сопровождения и модификации распределенных приложений.



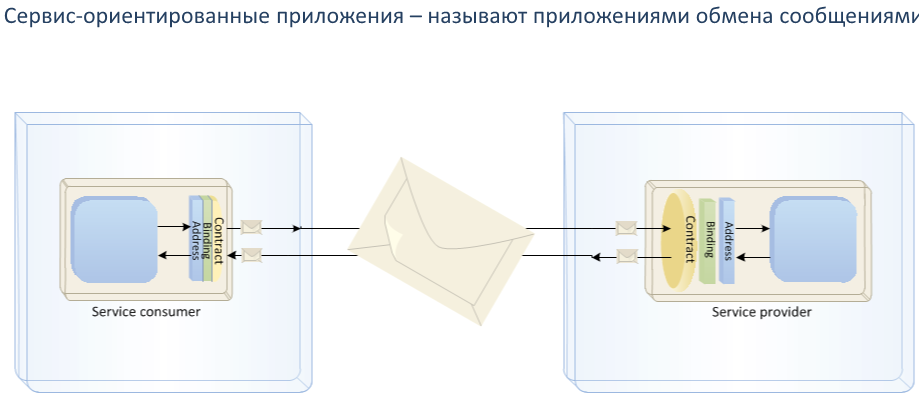


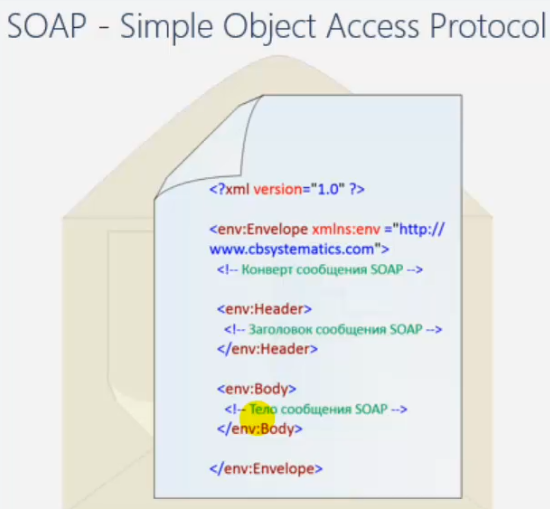




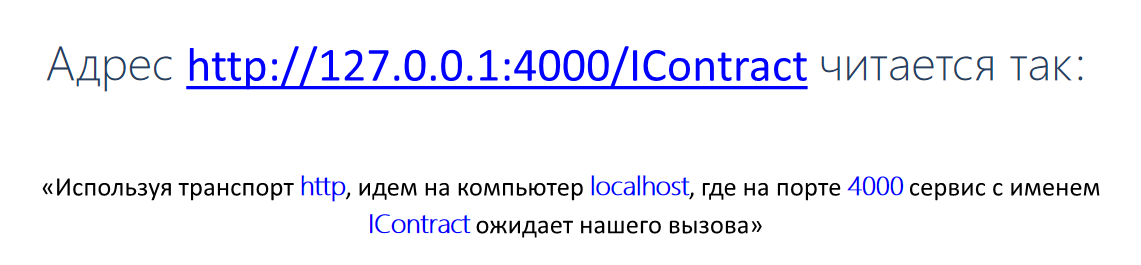


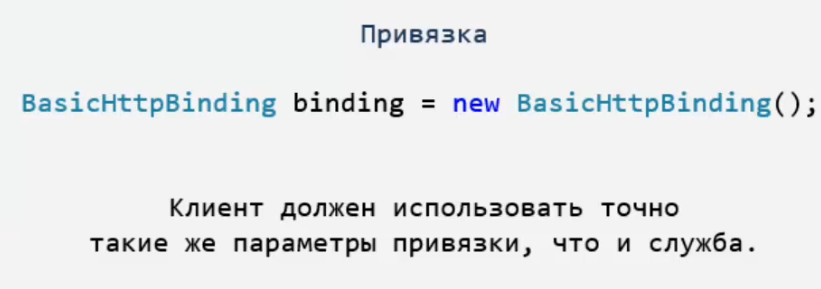




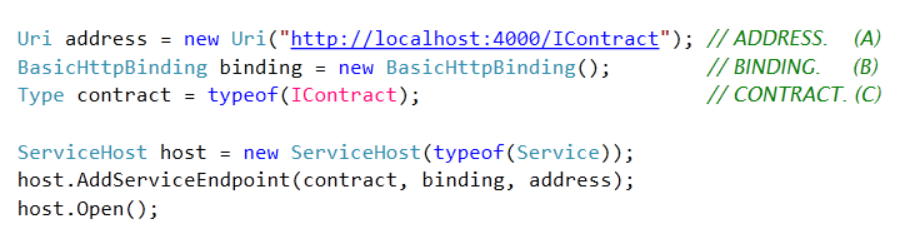


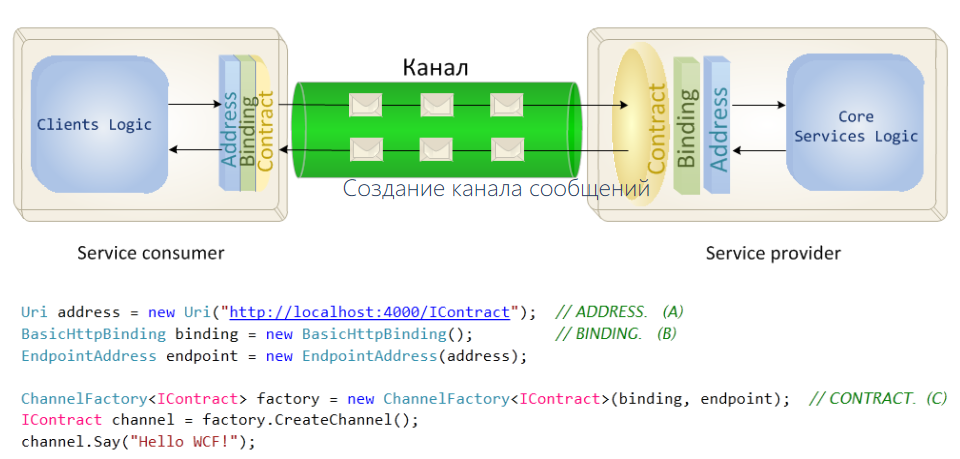


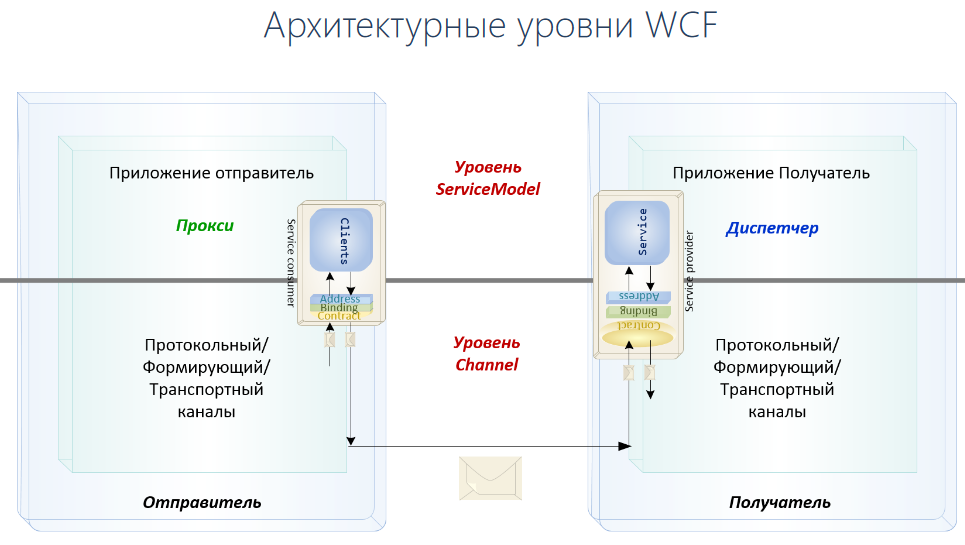












\*\*\*

Клиент

namespace Client

{

[ServiceContract]

interface IContract

{

[OperationContract]

void Say(string name, string input);

}

}

class Client

{

static void Main(string[] args)

{

Console.Title = "CLIENT";

Console.WriteLine("Введите имя пользоветеля");

string user = Console.ReadLine();

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract"); // ADDRESS. (A)

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding(); // BINDING. (B)

// Создание Конечной Точки.

EndpointAddress endpoint = new EndpointAddress(address);

// Создание фабрики каналов.

ChannelFactory<IContract> factory = new ChannelFactory<IContract>(binding, endpoint); // CONTRACT. (C)

// Использование factory для создания канала (прокси).

IContract channel = factory.CreateChannel();

// Использование канала для отправки сообщения получателю.

while (true)

{

Console.WriteLine("Введите сообщение");

string message = Console.ReadLine();

channel.Say(user,message);

}

// Задержка.

Console.ReadKey();

}

}

Сервер

[ServiceContract]

interface IContract

{

[OperationContract]

void Say(string name, string input);

}

class Service : IContract

{

// Реализация контракта - IContract.

public void Say(string name,string input)

{

Console.WriteLine("{0}: {1}", name,input);

}

}

class Server

{

static void Main()

{

Console.Title = "SERVER";

// Указание адреса, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract"); // ADDRESS. (A)

// Указание привязки, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding(); // BINDING. (B)

// Указание контракта.

Type contract = typeof(IContract); // CONTRACT. (C)

// Создание провайдера Хостинга с указанием Сервиса.

ServiceHost host = new ServiceHost(typeof(Service));

// Добавление "Конечной Точки".

host.AddServiceEndpoint(contract, binding, address);

// Начало ожидания прихода сообщений.

host.Open();

Console.WriteLine("Приложение готово к приему сообщений.");

Console.ReadKey();

// Завершение ожидания прихода сообщений.

host.Close();

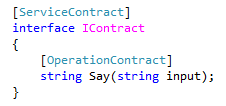
}

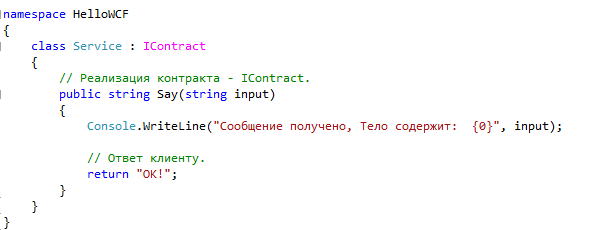
}

\*\*\*

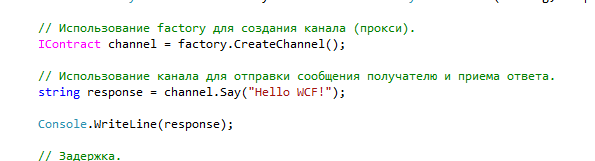
Приложение с возвращаемым значением

Сервер





Клиент



Основная функциональность WCF

разделяется на 10 категорий

1. Независимая модификация
2. Асинхронный однонаправленный обмен сообщениями

**Возможность написания приложений с применением вызовов функций типа «Запрос-Ответ».**

3. Платформенная консолидация

**Объединение возможностей многих предыдущих технологий в единую программную модель.**

4. Безопасность

* •Аутентификация
* •Авторизация
* •Конфиденциальность
* •Целостность
* •Безопасность на уровне транспорта
* •Безопасность на уровне сообщений

5. Надежность

**В распределенных вычислениях, надежная доставка сообщений называется гарантированной.**

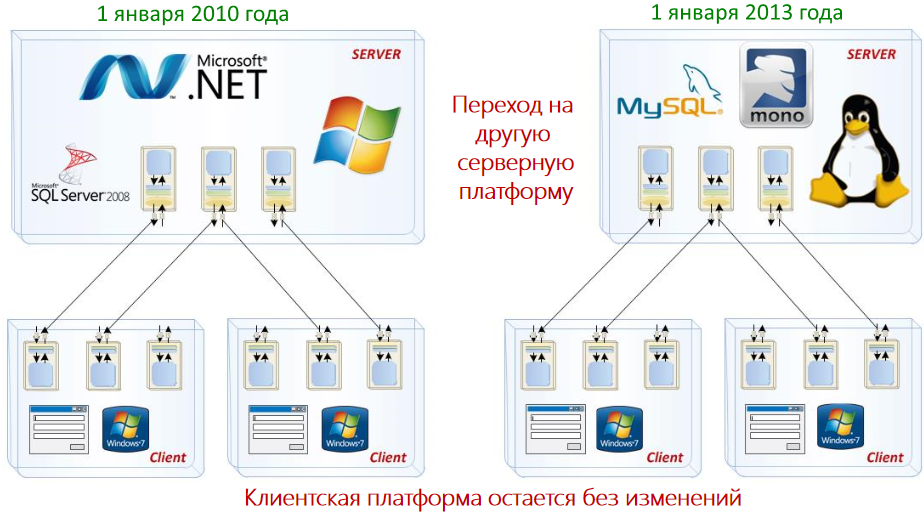
WCF содержит механизмы, обеспечивающие три вида гарантии надежной доставки сообщений:

* 1.Не более одного раза.
* 2.Хотя бы один раз.
* 3.По порядку.

6. Поддержка транзакций

7. Сетевое взаимодействие (Interoperability)

**Способность системы менять программную или аппаратную платформу, не затрагивая других участников сценария распределенных вычислений.**



8. Производительность

**Производительность при взаимодействии между двумя WCF приложениями выше,**

**чем между WCF-приложением и Web-сервисом на базе Java.)**

9. Расширяемость

**WCF - предоставляет возможности**

**для расширения своей архитектуры.**

WCF позволяет создавать свои собственные:

* •Транспорт
* •Каналы
* •Привязки
* •Кодировки
* •Архитектурные парадигмы

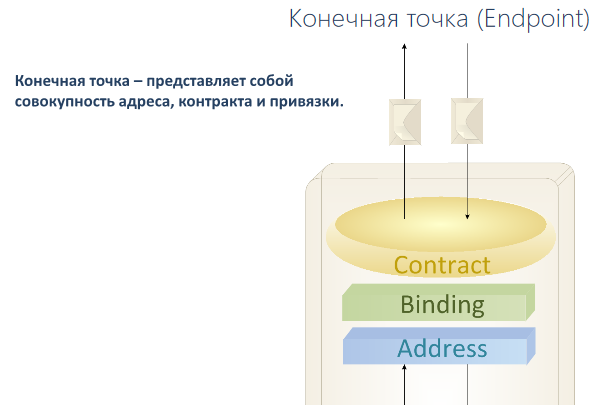
10. Возможности конфигурирования

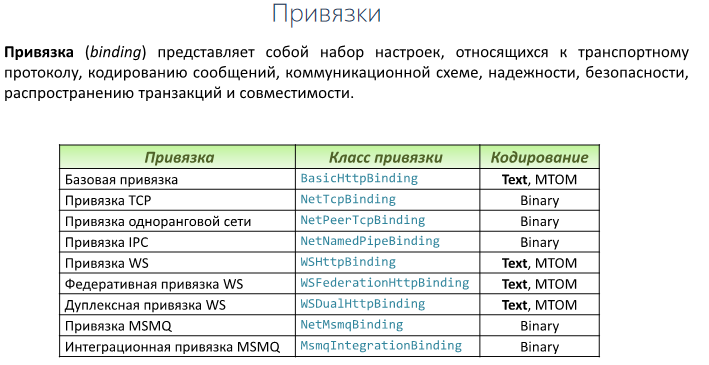
* **WCF - предоставляет возможности конфигурирования посредством**
* **конфигурационных файлов на базе XML.**
* **App.config**
* Возможность конфигурирования позволяет сотрудникам, не являющимся разработчиками, сопровождать приложение и изменять порядок его действий без привлечения команды разработчиков.

Выводы

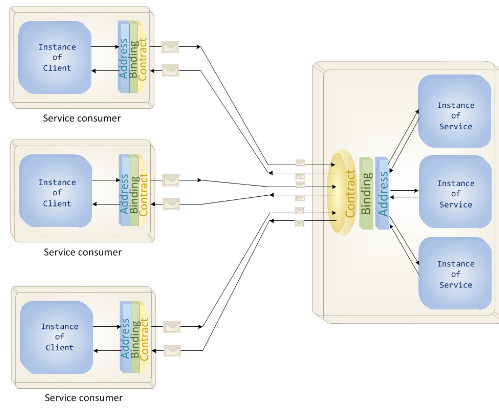
* ***WCF*** позволяет проектировать, строить, отлаживать и сопровождать распределенные системы быстрее и более функционально, чем это было раньше.
* ***WCF*** объединяет множество технологий: RPC, COM+, Remoting, ASMX, WSE и MSMQ.
* ***WCF*** расширяема.

# Конфигурирование и хостинг сервисов WCF









Варианты хостинга

* 1.Автохостинг *(Self-Hosting)*
* 2.Хостинг в службах Windows
* 3.Хостинг WAS *(Windows Activation Services)*
* 4.Хостинг IIS *(Internet Information Services)*

## Client

namespace ClientWCF

{

// КОНТРАКТ.

[ServiceContract]

interface IContractService

{

[OperationContract]

double Method(string s);

}

// КЛИЕНТ.

class Client

{

static void Main()

{

Console.Title = "CLIENT";

ChannelFactory<IContractService> channelFactory =

new ChannelFactory<IContractService>(new BasicHttpBinding(), new EndpointAddress("http://localhost:8000/ServiceWCF"));

IContractService service = channelFactory.CreateChannel();

double digit = service.Method("double");

Console.WriteLine("Double: {0}", digit);

// Задержка.

Console.WriteLine("Для завершения нажмите <Any Key>.");

Console.ReadKey();

}

}

}

## Service

namespace ServiceWCF

{

// КОНТРАКТ.

[ServiceContract]

interface IContractService

{

[OperationContract]

double Method(string s);

}

// СЕРВИС.

class MyService : IContractService

{

public double Method(string s)

{

Console.WriteLine("Обработан запрос. " + s);

if (s == "double")

return 777.77;

else

return 0;

}

}

// ХОСТ.

class Service

{

static void Main(string[] args)

{

Console.Title = "SERVER";

ServiceHost serviceHost = new ServiceHost(typeof(MyService), new Uri("http://localhost:8000/ServiceWCF"));

serviceHost.AddServiceEndpoint(typeof(IContractService), new BasicHttpBinding(), "");

serviceHost.Open();

Console.WriteLine("Для завершения нажмите <Any Key>.");

Console.ReadKey();

serviceHost.Close();

}

}

}

## Config

namespace ClientWCF

{

// КОНТРАКТ.

[ServiceContract]

interface IContractService

{

[OperationContract]

double Method(string s);

}

// КЛИЕНТ.

class Client

{

static void Main()

{

Console.Title = "CLIENT";

ChannelFactory<IContractService> channelFactory =

new ChannelFactory<IContractService>(new BasicHttpBinding(), new EndpointAddress("http://localhost:8000/ServiceWCF"));

IContractService proxy = channelFactory.CreateChannel();

double digit = proxy.Method("double");

Console.WriteLine("Double: {0}", digit);

// Задержка.

Console.WriteLine("Для завершения нажмите <Any Key>.");

Console.ReadKey();

}

}

}

Service

// Использование конфигурационного файла - App.config

namespace ServiceWCF

{

// КОНТРАКТ.

[ServiceContract]

interface IContractService

{

[OperationContract]

double Method(string s);

}

// СЕРВИС.

class MyService : IContractService

{

public double Method(string s)

{

Console.WriteLine("Обработан запрос. " + s);

if (s == "double")

return 777.77;

else

return 0;

}

}

// ХОСТ.

class Service

{

static void Main(string[] args)

{

Console.Title = "SERVER";

ServiceHost serviceHost = new ServiceHost(typeof(MyService));

serviceHost.Open();

Console.WriteLine("Для завершения нажмите <Any Key>.");

Console.ReadKey();

serviceHost.Close();

}

}

}

App config

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

<configuration>

<system.serviceModel>

<services>

<service name ="ServiceWCF.MyService">

<host>

<baseAddresses>

<add baseAddress="http://localhost:8000/ServiceWCF"/>

</baseAddresses>

</host>

<!-- Указание Адреса, Привязки, Контракта -->

<endpoint address=""

binding="basicHttpBinding"

contract="ServiceWCF.IContractService" />

</service>

</services>

</system.serviceModel>

</configuration>

## Wpf

using System.ServiceModel;

// КОНТРАКТ.

namespace Client

{

[ServiceContract]

interface IContract

{

[OperationContract]

string Say(string input);

}

}

namespace Client

{

public partial class Window1 : Window

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экземпляр ChannelFactory<T>, где Т - Контракт.

ChannelFactory<IContract> factory;

// Ссылка на канал (прокси).

IContract channel;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (factory == null)

{

// Создание экземпляра ChannelFactory<T>, где Т - Контракт.

factory = new ChannelFactory<IContract>(binding, new EndpointAddress(address));

// Использование factory для создания канала (прокси).

channel = factory.CreateChannel();

}

if (factory != null && channel != null)

{

textBox1.Text += "From Me: " + textBox2.Text + Environment.NewLine;

// Использование channel (прокси) для отправки сообщения получателю.

textBox1.Text += channel.Say(textBox2.Text) + Environment.NewLine;

textBox2.Text = "";

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

// Реакция на изменение textBox1.

private void textBox1\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

textBox1.ScrollToEnd();

}

}

}

Server

// КОНТРАКТ.

namespace Server

{

[ServiceContract]

interface IContract

{

[OperationContract]

string Say(string input);

}

}

// СЕРВИС.

namespace Server

{

class Service : IContract

{

public string Say(string input)

{

return "From Server: Вы сказали - " + input;

}

}

}

namespace Server

{

public partial class Window1 : Window

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экзкмпляр ServiceHost.

ServiceHost service;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service == null)

{

// Создание экзкмпляра ServiceHost.

service = new ServiceHost(typeof(Service));

// Добавление "Конечной Точки".

service.AddServiceEndpoint(typeof(IContract), binding, address);

// Начало ожидания прихода сообщений.

service.Open();

textBox1.Text += "Сервер запущен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service != null)

{

// Завершение ожидания прихода сообщений.

service.Close();

service = null;

textBox1.Text += "Сервер остановлен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

}

Manifest for privileges

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

<asmv1:assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1" xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns:asmv2="urn:schemas-microsoft-com:asm.v2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>

<trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">

<security>

<requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">

<!--UAC Manifest Options

If you want to change the Windows User Account Control level replace the

requestedExecutionLevel node with one of the following.-->

<!--<requestedExecutionLevel level="asInvoker" uiAccess="false" />-->

<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />

<!--<requestedExecutionLevel level="highestAvailable" uiAccess="false" />-->

<!--Specifying requestedExecutionLevel node will disable file and registry virtualization.

If you want to utilize File and Registry Virtualization for backward

compatibility then delete the requestedExecutionLevel node.-->

<!--<requestedExecutionLevel level="asInvoker" uiAccess="false" />-->

</requestedPrivileges>

</security>

</trustInfo>

<compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">

<application>

<!-- A list of all Windows versions that this application is designed to work with.

Windows will automatically select the most compatible environment.-->

<!-- If your application is designed to work with Windows Vista, uncomment the following supportedOS node-->

<!--<supportedOS Id="{e2011457-1546-43c5-a5fe-008deee3d3f0}"></supportedOS>-->

<!-- If your application is designed to work with Windows 7, uncomment the following supportedOS node-->

<!--<supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}"/>-->

<!-- If your application is designed to work with Windows 8, uncomment the following supportedOS node-->

<!--<supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}"></supportedOS>-->

</application>

</compatibility>

<!-- Enable themes for Windows common controls and dialogs (Windows XP and later) -->

<!-- <dependency>

<dependentAssembly>

<assemblyIdentity

type="win32"

name="Microsoft.Windows.Common-Controls"

version="6.0.0.0"

processorArchitecture="\*"

publicKeyToken="6595b64144ccf1df"

language="\*"

/>

</dependentAssembly>

</dependency>-->

</asmv1:assembly>

## Wpf host form

using System.ServiceModel;

// КОНТРАКТ.

namespace Server

{

[ServiceContract]

interface IContract

{

[OperationContract]

string Say(string input);

}

}

// КЛИЕНТ.

namespace Client

{

public partial class Window1 : Window

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экземпляр ChannelFactory<T>, где Т - Контракт.

ChannelFactory<IContract> factory;

// Ссылка на канал (прокси).

IContract channel;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (factory == null)

{

// Создание экземпляра ChannelFactory<T>, где Т - Контракт.

factory = new ChannelFactory<IContract>(binding, new EndpointAddress(address));

// Использование factory для создания канала (прокси).

channel = factory.CreateChannel();

}

if (factory != null && channel != null)

{

textBox1.Text += "From Me: " + textBox2.Text + Environment.NewLine;

// Использование channel (прокси) для отправки сообщения получателю.

textBox1.Text += "From Server: " + channel.Say(textBox2.Text) + Environment.NewLine;

textBox2.Text = "";

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

// Реакция на изменение textBox1.

private void textBox1\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

textBox1.ScrollToEnd();

}

}

}

Server

// КОНТРАКТ.

namespace Server

{

[ServiceContract]

interface IContract

{

[OperationContract]

string Say(string input);

}

}

// ХОСТ.

namespace Server

{

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]

public partial class Window1 : Window, IContract

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экзкмпляр ServiceHost.

ServiceHost service;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service == null)

{

// Создание экзкмпляра ServiceHost.

service = new ServiceHost(this);

// Добавление "Конечной Точки".

service.AddServiceEndpoint(typeof(IContract), binding, address);

// Начало ожидания прихода сообщений.

service.Open();

textBox1.Text += "Сервер запущен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service != null)

{

// Завершение ожидания прихода сообщений.

service.Close();

service = null;

textBox1.Text += "Сервер остановлен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// СЕРВИС.

public string Say(string input)

{

textBox1.Text += "From Client: " + input + Environment.NewLine;

return "Вы сказали - " + input;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

}

## UIConnection

// ХОСТ.

namespace Server

{

public partial class Window1

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экзкмпляр ServiceHost.

ServiceHost service;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service == null)

{

// Создание экзкмпляра ServiceHost.

service = new ServiceHost(typeof(Service));

// Добавление "Конечной Точки".

service.AddServiceEndpoint(typeof(IContract), binding, address);

// Начало ожидания прихода сообщений.

service.Open();

textBox1.Text += "Сервер запущен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service != null)

{

// Завершение ожидания прихода сообщений.

service.Close();

service = null;

textBox1.Text += "Сервер остановлен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

// Сервис

class Service : IContract

{

public string Say(string input)

{

UiConnection.UpdateText("From Client: " + input + Environment.NewLine);

return "Вы сказали - " + input;

}

}

internal static class UiConnection

{

// Получаем ссылку на текущую форму

static readonly Window1 Window1 =

Application.Current.Windows

.Cast<Window>()

.FirstOrDefault(window => window is Window1) as Window1;

// Изменяем состояние контрола

internal static void UpdateText(string @string)

{

Window1.textBox1.Text += Environment.NewLine + @string;

}

}

}

NT Services

namespace WindowsServiceNT

{

[RunInstaller(true)]

public class ProjectInstaller : Installer

{

private ServiceProcessInstaller process;

private ServiceInstaller service;

public ProjectInstaller()

{

process = new ServiceProcessInstaller();

process.Account = ServiceAccount.LocalSystem;

service = new ServiceInstaller();

service.ServiceName = "lllll ServiceNT lllll";

service.Description = "My Service !!!!!!!!!!";

service.StartType = ServiceStartMode.Automatic;

Installers.Add(process);

Installers.Add(service);

}

}

public partial class Service1 : ServiceBase

{

// Ссылка на экземпляр ServiceHost.

ServiceHost service = null;

public Service1()

{

InitializeComponent();

}

protected override void OnStart(string[] args)

{

if (service == null)

{

// Создание экземпляра ServiceHost.

service = new ServiceHost(typeof(Service));

// Начало ожидания прихода сообщений.

service.Open();

}

}

protected override void OnStop()

{

if (service != null)

{

// Завершение ожидания прихода сообщений.

service.Close();

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[ServiceContract]

interface IContract // КОНТРАКТ.

{

[OperationContract]

string Say(string input);

}

class Service : IContract // СЕРВИС.

{

public string Say(string input)

{

return "From Server: Вы сказали - " + input;

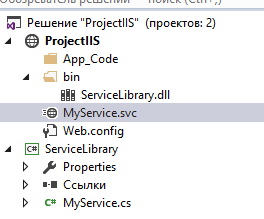
}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

## WAS host



<?xml version="1.0" encoding="UTF-8"?>

<!--

Note: As an alternative to hand editing this file you can use the

web admin tool to configure settings for your application. Use

the Website->Asp.Net Configuration option in Visual Studio.

A full list of settings and comments can be found in

machine.config.comments usually located in

\Windows\Microsoft.Net\Framework\v2.x\Config

-->

<configuration>

<configSections>

<sectionGroup name="system.web.extensions" type="System.Web.Configuration.SystemWebExtensionsSectionGroup, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">

<sectionGroup name="scripting" type="System.Web.Configuration.ScriptingSectionGroup, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">

<!--<section name="scriptResourceHandler" type="System.Web.Configuration.ScriptingScriptResourceHandlerSection, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false" allowDefinition="MachineToApplication" />-->

<sectionGroup name="webServices" type="System.Web.Configuration.ScriptingWebServicesSectionGroup, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">

<!--<section name="jsonSerialization" type="System.Web.Configuration.ScriptingJsonSerializationSection, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false" allowDefinition="Everywhere" />-->

<!--<section name="profileService" type="System.Web.Configuration.ScriptingProfileServiceSection, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false" allowDefinition="MachineToApplication" />-->

<!--<section name="authenticationService" type="System.Web.Configuration.ScriptingAuthenticationServiceSection, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false" allowDefinition="MachineToApplication" />-->

<!--<section name="roleService" type="System.Web.Configuration.ScriptingRoleServiceSection, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false" allowDefinition="MachineToApplication" />-->

</sectionGroup>

</sectionGroup>

</sectionGroup>

</configSections>

<!--ВСТАВКА-->

<system.serviceModel>

<services>

<service behaviorConfiguration="MEXServiceTypeBehavior" name="ServiceLibrary.MyService">

<endpoint address="" binding="basicHttpBinding" contract="ServiceLibrary.IContract" />

<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />

<endpoint address="net.tcp://localhost:808/My/MyService.svc" binding="netTcpBinding" contract="ServiceLibrary.IContract" />

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="MEXServiceTypeBehavior">

<serviceMetadata httpGetEnabled="true" />

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

<!--Конец вставки-->

<appSettings />

<connectionStrings />

<system.web>

<!--

Set compilation debug="true" to insert debugging

symbols into the compiled page. Because this

affects performance, set this value to true only

during development.

-->

<compilation debug="true">

<assemblies>

<add assembly="System.Core, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />

<add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

<add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />

<add assembly="System.Data.DataSetExtensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />

</assemblies>

</compilation>

<!--

The <authentication> section enables configuration

of the security authentication mode used by

ASP.NET to identify an incoming user.

-->

<authentication mode="Windows" />

<!--

The <customErrors> section enables configuration

of what to do if/when an unhandled error occurs

during the execution of a request. Specifically,

it enables developers to configure html error pages

to be displayed in place of a error stack trace.

<customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">

<error statusCode="403" redirect="NoAccess.htm" />

<error statusCode="404" redirect="FileNotFound.htm" />

</customErrors>

-->

<pages>

<controls>

<add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

<add tagPrefix="asp" namespace="System.Web.UI.WebControls" assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

</controls>

</pages>

<httpHandlers>

<remove verb="\*" path="\*.asmx" />

<add verb="\*" path="\*.asmx" validate="false" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

<add verb="\*" path="\*\_AppService.axd" validate="false" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

<add verb="GET,HEAD" path="ScriptResource.axd" validate="false" type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

</httpHandlers>

<httpModules>

<add name="ScriptModule" type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

</httpModules>

</system.web>

<system.codedom>

<compilers>

<compiler language="c#;cs;csharp" extension=".cs" type="Microsoft.CSharp.CSharpCodeProvider,System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" warningLevel="4">

<providerOption name="CompilerVersion" value="v3.5" />

<providerOption name="WarnAsError" value="false" />

</compiler>

<compiler language="vb;vbs;visualbasic;vbscript" extension=".vb" type="Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" warningLevel="4">

<providerOption name="CompilerVersion" value="v3.5" />

<providerOption name="OptionInfer" value="true" />

<providerOption name="WarnAsError" value="false" />

</compiler>

</compilers>

</system.codedom>

<system.webServer>

<validation validateIntegratedModeConfiguration="false" />

<modules>

<remove name="ScriptModule" />

<add name="ScriptModule" preCondition="managedHandler" type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

</modules>

<handlers>

<remove name="WebServiceHandlerFactory-Integrated" />

<remove name="ScriptHandlerFactory" />

<remove name="ScriptHandlerFactoryAppServices" />

<remove name="ScriptResource" />

<add name="ScriptHandlerFactory" verb="\*" path="\*.asmx" preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

<add name="ScriptHandlerFactoryAppServices" verb="\*" path="\*\_AppService.axd" preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

<add name="ScriptResource" verb="GET,HEAD" path="ScriptResource.axd" preCondition="integratedMode" type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

</handlers>

<directoryBrowse enabled="true" showFlags="Date, Time, Size, Extension, LongDate" />

</system.webServer>

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="System.Web.Extensions" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Extensions.Design" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0" />

</dependentAssembly>

</assemblyBinding>

</runtime>

</configuration>

namespace ServiceLibrary

{

[ServiceContract]

interface IContract

{

[OperationContract]

string Method(string s);

}

public class MyService : IContract

{

public string Method(string s)

{

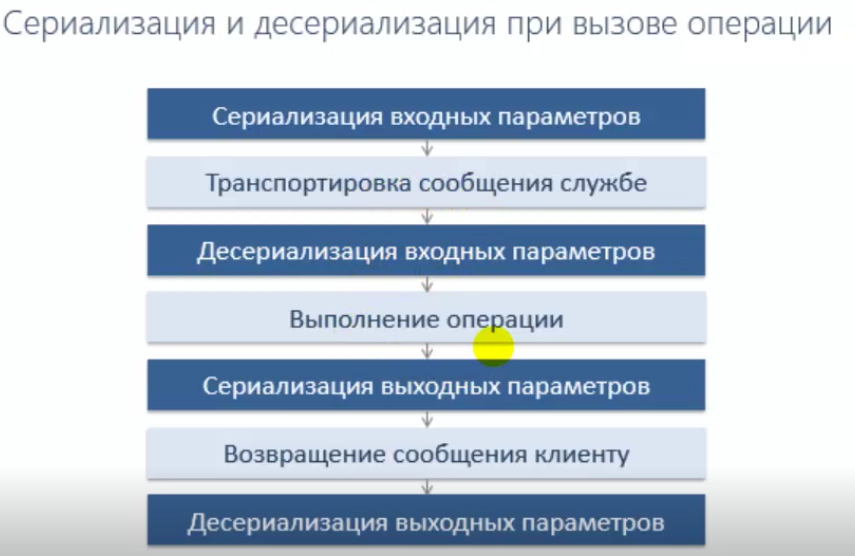
return s + " WAS/IIS7";

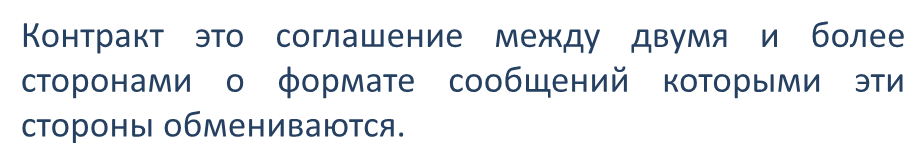
}

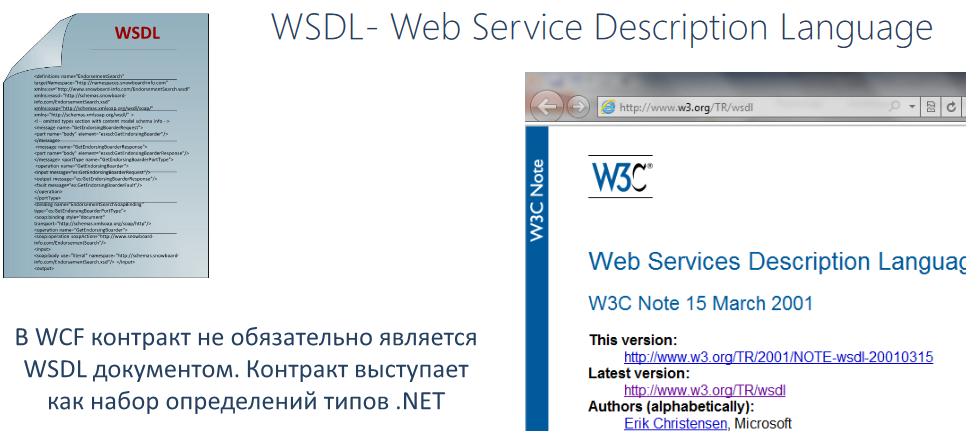
}

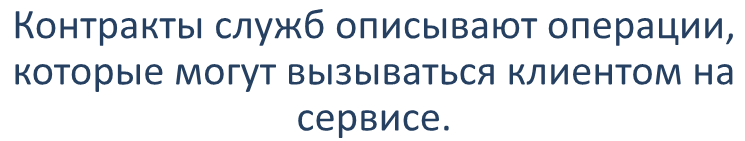
# Проектирование и определение контрактов

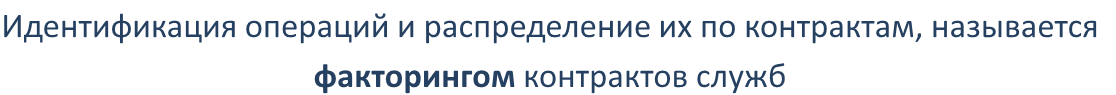
Маршалинг и Демаршалиг











namespace CientService

{

// КОНТРАКТ.

[ServiceContract]

interface IInterface

{

[OperationContract(Name = "AddInt")]

int Add(int a, int b);

[OperationContract(Name = "AddDouble")]

double Add(double a, double b);

}

// СЕРВИС.

class MyService : IInterface

{

public int Add(int a, int b)

{

Console.WriteLine(OperationContext.Current.RequestContext.RequestMessage.ToString() + "\n");

return a + b;

}

public double Add(double a, double b)

{

Console.WriteLine(OperationContext.Current.RequestContext.RequestMessage.ToString() + "\n");

return a + b;

}

}

}

// Тема: Перегрузка Методов при работе с сервисами.

namespace CientService

{

class Program

{

static void Main()

{

Console.Title = "SERVER";

ServiceHost host = new ServiceHost(typeof(MyService), new Uri("http://localhost"));

host.AddServiceEndpoint(typeof(IInterface), new BasicHttpBinding(), "");

host.Open();

Console.WriteLine("Хост открыт...");

// Задержка.

Console.ReadKey();

host.Close();

}

}

}

Client

// Тема: Перегрузка Методов при работе с сервисами.

namespace Client

{

// КОНТРАКТ.

[ServiceContract]

interface IInterface

{

[OperationContract(Name = "AddInt")]

int Add(int a, int b);

[OperationContract(Name = "AddDouble")]

double Add(double a, double b);

}

class Program

{

static void Main()

{

Console.Title = "CLIENT";

IInterface proxy = ChannelFactory<IInterface>.CreateChannel(

new BasicHttpBinding(), new EndpointAddress("http://localhost"));

using (proxy as IDisposable)

{

int i = proxy.Add(2, 3);

Console.WriteLine(i);

double d = proxy.Add(2.2, 3.3);

Console.WriteLine(d);

}

// Задержка.

Console.ReadKey();

}

}

}

## Contract inherit

namespace Client

{

// КОНТРАКТЫ.

[ServiceContract]

interface IInterface1

{

[OperationContract]

string Method1(string s);

}

[ServiceContract]

interface IInterface2 : IInterface1

{

[OperationContract]

string Method2(string s);

}

class Program

{

static void Main()

{

Console.Title = "CLIENT";

Thread.Sleep(2000);

IInterface2 proxy = ChannelFactory<IInterface2>.CreateChannel(

new BasicHttpBinding(), new EndpointAddress("http://localhost"));

using (proxy as IDisposable)

{

Console.WriteLine(proxy.Method1("Ping "));

Console.WriteLine(proxy.Method2("Ping "));

}

// Задержка.

Console.ReadKey();

}

}

}

Service

namespace CientService

{

// КОНТРАКТЫ.

[ServiceContract]

interface IInterface1

{

[OperationContract]

string Method1(string s);

}

[ServiceContract]

interface IInterface2 : IInterface1

{

[OperationContract]

string Method2(string s);

}

// СЕРВИС.

class MyService : IInterface1, IInterface2

{

// Реализация IInterface1

public string Method1(string s)

{

return s + " Контракт - IInterface1";

}

// Реализация IInterface2

public string Method2(string s)

{

return s + " Контракт - IInterface2";

}

}

}

namespace CientService

{

class Program

{

static void Main()

{

Console.Title = "SERVER";

ServiceHost host = new ServiceHost(typeof(MyService), new Uri("http://localhost"));

host.AddServiceEndpoint(typeof(IInterface2), new BasicHttpBinding(), "");

host.Open();

Console.WriteLine("Хост открыт...");

// Задержка.

Console.ReadKey();

host.Close();

}

}

}

## Contract inherit

namespace CientService

{

// КОНТРАКТЫ.

[ServiceContract]

interface IInterface1

{

[OperationContract]

string Method1(string s);

}

[ServiceContract]

interface IInterface2 : IInterface1

{

[OperationContract]

string Method2(string s);

}

// СЕРВИС.

class MyService : IInterface2

{

// Реализация IInterface1

public string Method1(string s)

{

Console.WriteLine(OperationContext.Current.RequestContext.RequestMessage.ToString() + "\n");

return s + " Контракт - IInterface1";

}

// Реализация IInterface2

public string Method2(string s)

{

Console.WriteLine(OperationContext.Current.RequestContext.RequestMessage.ToString() + "\n");

return s + " Контракт - IInterface2";

}

}

}

## Data contract

// КОНТРАКТЫ.

namespace Client

{

// СЕРВИСНЫЙ КОНТРАКТ.

[ServiceContract]

public interface IContract

{

[OperationContract]

Point Add(Point a, Point b);

}

// КОНТРАКТ ДАННЫХ.

[DataContract(Namespace = "OtherNamespace")]

public class Point

{

[DataMember]

public double x;

[DataMember]

public double y;

public Point(double x, double y)

{

this.x = x;

this.y = y;

}

}

}

// КЛИЕНТ.

namespace Client

{

public partial class Window1 : Window

{

Uri address = new Uri("http://localhost:4000/IContract");

BasicHttpBinding binding = new BasicHttpBinding();

ChannelFactory<IContract> factory = null;

IContract channel = null;

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (factory == null)

{

factory = new ChannelFactory<IContract>(binding, new EndpointAddress(address));

channel = factory.CreateChannel();

}

if (factory != null && channel != null)

{

Point A = new Point(1, 1);

Point B = new Point(2, 2);

Point Res = channel.Add(A, B);

textBox1.Text = string.Format("Add({0} + {1}, {2} + {3}) = {4} {5}", A.x, A.y, B.x, B.y, Res.x, Res.y);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

// Реакция на изменение textBox1.

private void textBox1\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

textBox1.ScrollToEnd();

}

}

}

Server

// КОНТРАКТЫ.

namespace Server

{

// СЕРВИСНЫЙ КОНТРАКТ.

[ServiceContract]

public interface IContract

{

[OperationContract]

Point Add(Point a, Point b);

}

// КОНТРАКТ ДАННЫХ.

[DataContract(Namespace = "OtherNamespace")]

public class Point

{

[DataMember]

public double x;

[DataMember]

public double y;

public Point(double x, double y)

{

this.x = x;

this.y = y;

}

}

}

// СЕРВИС.

namespace Server

{

public class Service : IContract

{

public Point Add(Point a, Point b)

{

//MessageBox.Show(OperationContext.Current.RequestContext.RequestMessage.ToString());

return new Point(a.y + a.x, b.y + b.x);

}

}

}

namespace Server

{

public partial class Window1 : Window

{

Uri address = new Uri("http://localhost:4000/IContract");

BasicHttpBinding binding = new BasicHttpBinding();

ServiceHost service = null;

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service == null)

{

service = new ServiceHost(typeof(Service));

service.AddServiceEndpoint(typeof(IContract), binding, address);

service.Open();

textBox1.Text += "Сервер запущен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service != null)

{

service.Close();

service = null;

textBox1.Text += "Сервер остановлен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

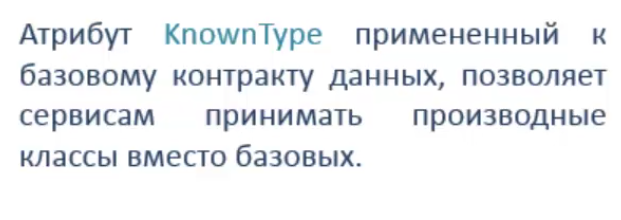
}

}

}

}

## KnownType



Client

// КОНТРАКТЫ.

namespace Client

{

// СЕРВИСНЫЙ КОНТРАКТ.

[ServiceContract]

public interface IContactManager

{

[OperationContract]

void AddCustomer(Customer customer);

[OperationContract]

void AddContact(Contact contact);

[OperationContract]

Contact[] GetContacts();

}

// КОНТРАКТЫ ДАННЫХ.

[DataContract(Namespace = "OtherNamespace")]

[KnownType(typeof(Customer))]

public class Contact

{

[DataMember]

public string FirstName;

[DataMember]

public string LastName;

}

[DataContract(Namespace = "OtherNamespace")]

public class Customer : Contact

{

[DataMember]

public int OrderNumber;

}

}

// КЛИЕНТ.

namespace Client

{

public partial class Window1 : Window

{

Uri address = new Uri("http://localhost:4000/IContract");

WSHttpBinding binding = new WSHttpBinding();

ChannelFactory<IContactManager> factory = null;

IContactManager channel = null;

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (factory == null)

{

factory = new ChannelFactory<IContactManager>(

binding, new EndpointAddress(address));

channel = factory.CreateChannel();

}

if (factory != null && channel != null)

{

Customer customer = new Customer();

customer.FirstName = "Ivan";

customer.LastName = "Petrov";

customer.OrderNumber = 123;

channel.AddContact(customer as Contact);

Contact[] contacts = channel.GetContacts();

for (int i = 0; i < contacts.Length; i++)

{

textBox1.Text += "->" + contacts[i].FirstName

+ " " + contacts[i].LastName + Environment.NewLine; ;

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "CLIENT");

}

}

}

}

Server

// КОНТРАКТЫ.

namespace Server

{

// СЕРВИСНЫЙ КОНТРАКТ.

[ServiceContract]

public interface IContactManager

{

[OperationContract]

void AddCustomer(Customer customer);

[OperationContract]

void AddContact(Contact contact);

[OperationContract]

Contact[] GetContacts();

}

// КОНТРАКТЫ ДАННЫХ.

[DataContract(Namespace = "OtherNamespace")]

[KnownType(typeof(Customer))]

public class Contact

{

[DataMember]

public string FirstName;

[DataMember]

public string LastName;

}

[DataContract(Namespace = "OtherNamespace")]

public class Customer : Contact

{

[DataMember]

public int OrderNumber;

}

}

// СЕРВИС.

namespace Server

{

public class Service : IContactManager

{

List<Customer> m\_Customers = new List<Customer>();

public void AddCustomer(Customer customer)

{

m\_Customers.Add(customer);

//MessageBox.Show(OperationContext.Current.RequestContext.RequestMessage.ToString(),

// "SERVER AddCustomer()" + " " + this.GetHashCode().ToString());

}

public void AddContact(Contact contact)

{

m\_Customers.Add(contact as Customer);

//MessageBox.Show(OperationContext.Current.RequestContext.RequestMessage.ToString(),

// "SERVER AddContact()" + " " + this.GetHashCode().ToString());

}

public Contact[] GetContacts()

{

//MessageBox.Show(OperationContext.Current.RequestContext.RequestMessage.ToString(),

// "SERVER GetContacts()" + " " + this.GetHashCode().ToString());

return m\_Customers.ToArray();

}

}

}

public partial class Window1 : Window

{

Uri address = new Uri("http://localhost:4000/IContract");

WSHttpBinding binding = new WSHttpBinding();

ServiceHost service = null;

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service == null)

{

service = new ServiceHost(typeof(Service));

service.AddServiceEndpoint(typeof(IContactManager), binding, address);

service.Open();

textBox1.Text += "Сервер запущен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service != null)

{

service.Close();

service = null;

textBox1.Text += "Сервер остановлен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

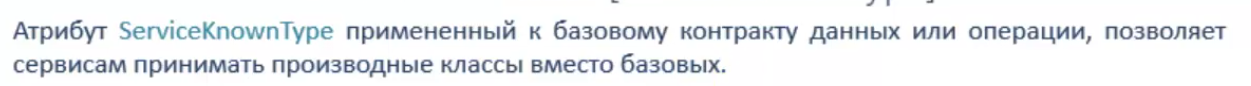
MessageBox.Show(ex.Message);

}

}

}

## ServiceKnownType



// КОНТРАКТЫ.

namespace Server

{

// СЕРВИСНЫЙ КОНТРАКТ.

[ServiceContract]

[ServiceKnownType(typeof(Customer))]

public interface IContactManager

{

[OperationContract]

void AddContact(Contact contact);

[OperationContract]

Contact[] GetContacts();

}

// КОНТРАКТЫ ДАННЫХ.

[DataContract(Namespace = "OtherNamespace")]

//[KnownType(typeof(Customer))]

public class Contact

{

[DataMember]

public string FirstName;

[DataMember]

public string LastName;

}

[DataContract(Namespace = "OtherNamespace")]

public class Customer : Contact

{

[DataMember]

public int OrderNumber;

}

}

# Message contract



Client

// КОНТРАКТЫ.

namespace Client

{

// СЕРВИСНЫЙ КОНТРАКТ.

[ServiceContract]

public interface ICalculator

{

[OperationContract]

MyMessage Calculate(MyMessage request);

}

// КОНТРАКТ СООБЩЕНИЯ.

[MessageContract]

public class MyMessage

{

private string operation;

private double n1;

private double n2;

private double result;

// Constructor - создает пустое сообщение.

public MyMessage() { }

// Constructor - создает сообщение и заполняет его.

public MyMessage(double n1, double n2, string operation, double result)

{

this.n1 = n1;

this.n2 = n2;

this.operation = operation;

this.result = result;

}

// Constructor - создает сообщение на основе другого сообщения.

public MyMessage(MyMessage message)

{

this.n1 = message.n1;

this.n2 = message.n2;

this.operation = message.operation;

this.result = message.result;

}

[MessageHeader]

public string Operation

{

get { return operation; }

set { operation = value; }

}

[MessageBodyMember]

public double N1

{

get { return n1; }

set { n1 = value; }

}

[MessageBodyMember]

public double N2

{

get { return n2; }

set { n2 = value; }

}

[MessageBodyMember]

public double Result

{

get { return result; }

set { result = value; }

}

}

}

// КЛИЕНТ.

namespace Client

{

public partial class Window1 : Window

{

Uri address = new Uri("http://localhost:4000/IContract");

WSHttpBinding binding = new WSHttpBinding();

ChannelFactory<ICalculator> factory = null;

ICalculator channel = null;

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (factory == null)

{

factory = new ChannelFactory<ICalculator>(binding, new EndpointAddress(address));

channel = factory.CreateChannel();

}

if (factory != null && channel != null)

{

// Сложение.

MyMessage request = new MyMessage();

request.N1 = 100D;

request.N2 = 15.99D;

request.Operation = "+";

MyMessage response = channel.Calculate(request);

textBox1.Text += string.Format("Add({0},{1}) = {2}",

request.N1, request.N2, response.Result) + Environment.NewLine;

// Вычитание.

request = new MyMessage();

request.N1 = 145D;

request.N2 = 76.54D;

request.Operation = "-";

response = channel.Calculate(request);

textBox1.Text += string.Format("Subtract({0},{1}) = {2}",

request.N1, request.N2, response.Result) + Environment.NewLine;

// Умножение.

request = new MyMessage();

request.N1 = 9D;

request.N2 = 81.25D;

request.Operation = "\*";

response = channel.Calculate(request);

textBox1.Text += string.Format("Multiply({0},{1}) = {2}",

request.N1, request.N2, response.Result) + Environment.NewLine;

// Деление.

request = new MyMessage();

request.N1 = 22D;

request.N2 = 7D;

request.Operation = "/";

response = channel.Calculate(request);

textBox1.Text += string.Format("Divide({0},{1}) = {2}",

request.N1, request.N2, response.Result) + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "CLIENT");

}

}

}

}

Server

// КОНТРАКТЫ.

namespace Server

{

// СЕРВИСНЫЙ КОНТРАКТ.

[ServiceContract]

public interface ICalculator

{

[OperationContract]

MyMessage Calculate(MyMessage request);

}

// КОНТРАКТ СООБЩЕНИЯ.

[MessageContract]

public class MyMessage

{

private string operation;

private double n1;

private double n2;

private double result;

// Constructor - создает пустое сообщение.

public MyMessage() { }

// Constructor - создает сообщение и заполняет его.

public MyMessage(double n1, double n2, string operation, double result)

{

this.n1 = n1;

this.n2 = n2;

this.operation = operation;

this.result = result;

}

// Constructor - создает сообщение на основе другого сообщения.

public MyMessage(MyMessage message)

{

this.n1 = message.n1;

this.n2 = message.n2;

this.operation = message.operation;

this.result = message.result;

}

[MessageHeader]

public string Operation

{

get { return operation; }

set { operation = value; }

}

[MessageBodyMember]

public double N1

{

get { return n1; }

set { n1 = value; }

}

[MessageBodyMember]

public double N2

{

get { return n2; }

set { n2 = value; }

}

[MessageBodyMember]

public double Result

{

get { return result; }

set { result = value; }

}

}

}

// СЕРВИС.

namespace Server

{

public class CalculatorService : ICalculator

{

public MyMessage Calculate(MyMessage request)

{

MyMessage response = new MyMessage(request);

switch (request.Operation)

{

case "+":

response.Result = request.N1 + request.N2;

break;

case "-":

response.Result = request.N1 - request.N2;

break;

case "\*":

response.Result = request.N1 \* request.N2;

break;

case "/":

response.Result = request.N1 / request.N2;

break;

default:

response.Result = 0.0D;

break;

}

return response;

}

}

}

namespace Server

{

public partial class Window1 : Window

{

Uri address = new Uri("http://localhost:4000/IContract");

WSHttpBinding binding = new WSHttpBinding();

ServiceHost service = null;

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service == null)

{

service = new ServiceHost(typeof(CalculatorService));

service.AddServiceEndpoint(typeof(ICalculator), binding, address);

service.Open();

textBox1.Text += "Сервер запущен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service != null)

{

service.Close();

service = null;

textBox1.Text += "Сервер остановлен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

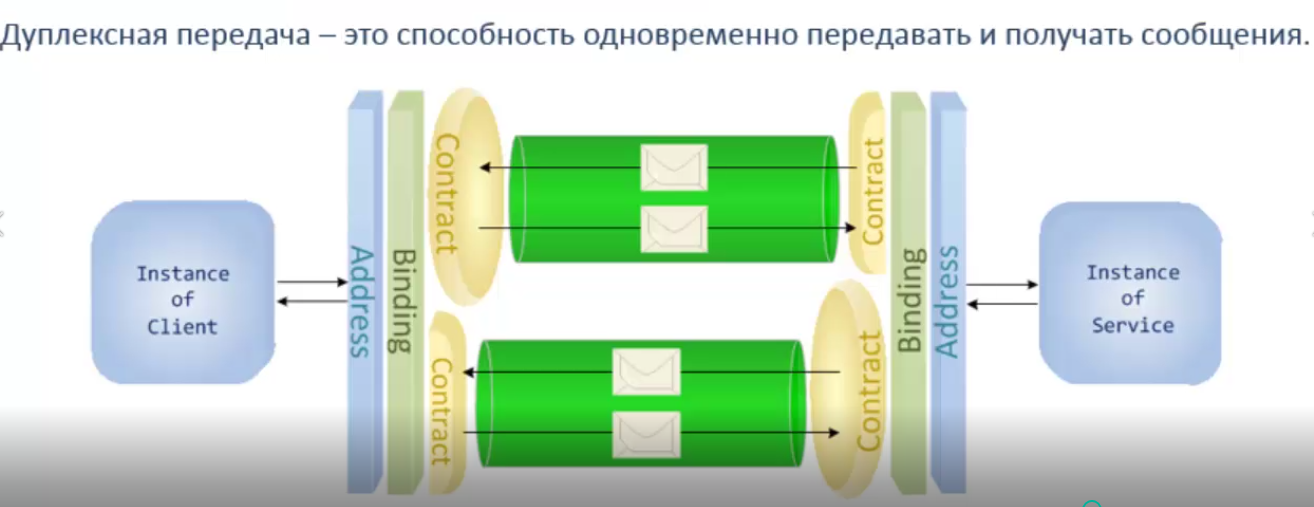
}

}

}

}

## Duplex channel factory



Client

namespace Client

{

class Context : IClientContract

{

public void ClientMethod(string message)

{

Console.WriteLine(message);

}

}

}

namespace Client

{

// КОНТРАКТ (Для клиента)

// Интерфейс для реализации метода на стороне клиента.

public interface IClientContract

{

// IsOneWay = true - клиент не ожидает успешного завершения метода на сервисе.

[OperationContract(IsOneWay = true)]

void ClientMethod(string message);

}

//=======================================================================

// КОНТРАКТ (Для сервиса)

// Когда служба используется для дуплексного обмена сообщениями,

// свойство CallbackContract определяет контракт, реализованный клиентом.

[ServiceContract(CallbackContract = typeof(IClientContract))]

public interface IServiceContract

{

// IsOneWay = true - сервис не ожидает успешного завершения метода на клиенте.

[OperationContract(IsOneWay = true)]

void ServiceMethod();

}

}

namespace Client

{

class Program

{

static void Main()

{

Console.Title = "Client";

// InstanceContext - представляет метод вызываемый сервисом на клиенте.

InstanceContext context = new InstanceContext(new Context());

// Создаем фабрику дуплексных каналов на клиенте.

DuplexChannelFactory<IServiceContract> factory =

new DuplexChannelFactory<IServiceContract>(context, new NetTcpBinding(),

"net.tcp://localhost:9000/MyService");

// Создаем конкретный канал.

IServiceContract proxy = factory.CreateChannel();

Console.WriteLine("Нажмите Enter для вызова метода на сервере.");

Console.ReadKey();

// Вызов метода на сервере для получения канала для общения с клиентом.

proxy.ServiceMethod();

// Задержка.

Console.ReadKey();

}

}

}

Service

namespace Duplex

{

// КОНТРАКТ (Для клиента)

//Интерфейс для реализации метода на стороне клиента.

public interface IClientContract

{

// IsOneWay = true - клиент не ожидает успешного завершения метода на сервисе.

[OperationContract(IsOneWay = true)]

void ClientMethod(string message);

}

//=======================================================================

// КОНТРАКТ (Для сервиса)

// Когда служба используется для дуплексного обмена сообщениями,

// свойство CallbackContract определяет контракт, реализованный клиентом.

[ServiceContract(CallbackContract = typeof(IClientContract))]

public interface IServiceContract

{

// IsOneWay = true - сервис не ожидает успешного завершения метода на клиенте.

[OperationContract(IsOneWay = true)]

void ServiceMethod();

}

}

namespace Duplex

{

// СЕРВИС.

class Service : IServiceContract

{

public void ServiceMethod()

{

Console.WriteLine("ServiceMethod");

Program.callback =

OperationContext.Current.GetCallbackChannel<IClientContract>();

}

}

}

namespace Duplex

{

class Program

{

public static IClientContract callback;

static void Main()

{

Console.Title = "Server";

// Создание хостинга для сервиса.

ServiceHost host = new ServiceHost(typeof(Service));

// Добавление конечной точки.

host.AddServiceEndpoint(typeof(IServiceContract), new NetTcpBinding(),

"net.tcp://localhost:9000/MyService");

// Запуск хоста.

host.Open();

Console.WriteLine("Нажмите Enter для вызова метода на клиенте.");

Console.ReadKey();

// Вызов метода на клиенте.

Program.callback.ClientMethod("Hello world!");

// Задержка.

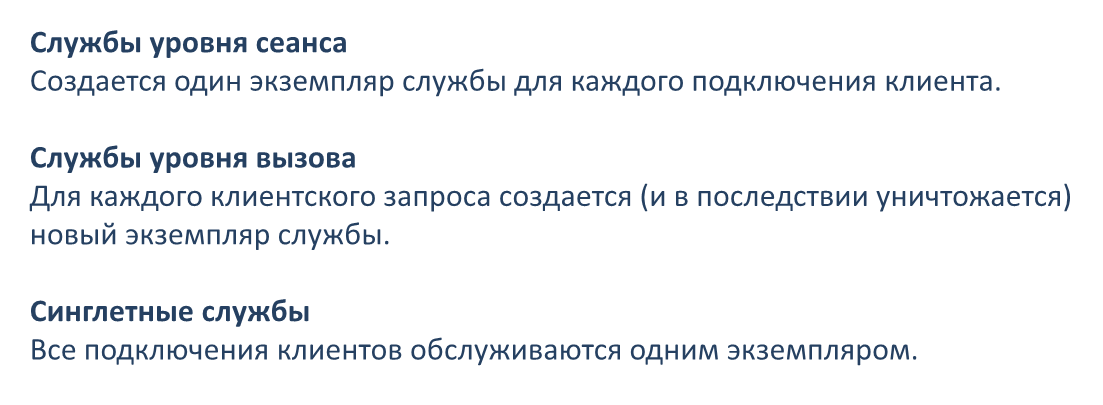
Console.ReadKey();

}

}

}

# Instance





## Call

// Служба УРОВНЯ ВЫЗОВА должна быть с КОНТРОЛЕМ СОСТОЯНИЯ (state-aware).

// Состояние службы бывает ВЫСОКОЗАТРАТНЫМ, поэтому

// экземпляр службы требуется уничтожать после каждого вызова!

namespace Client

{

class Program

{

static void Main(string[] args)

{

Console.Title = "CLIENT";

IContract Proxy = ChannelFactory<IContract>.CreateChannel(

new WSHttpBinding(), new EndpointAddress("http://localhost"));

try

{

for (int i = 0; i < 10; i++)

{

string hash = Proxy.MyMethod();

Console.WriteLine(hash);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

// Задержка.

Console.ReadKey();

}

}

}

// Служба УРОВНЯ ВЫЗОВА должна быть с КОНТРОЛЕМ СОСТОЯНИЯ (state-aware).

// Состояние службы бывает ВЫСОКОЗАТРАТНЫМ, поэтому

// экземпляр службы требуется уничтожать после каждого вызова!

namespace Contract

{

[ServiceContract]

public interface IContract

{

[OperationContract]

string MyMethod();

}

}

namespace Service

{

class State

{

int[] array = new int[1000000];

public State()

{

for (int i = 0; i < 1000000; i++)

array[i] = i;

}

public string GetState()

{

return array.Length.ToString();

}

}

}

namespace Service

{

// Применяя атрибут ServiceBehavior, со свойством InstanceContextMode равным PerCall, -

// служба помечается как служба УРОВНЯ ВЫЗОВА.

[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]

public class MyService : IContract, IDisposable

{

public string MyMethod()

{

// Отображение размера используемого состояния.

Console.WriteLine("Size of state: " + WorkWithState());

return this.GetHashCode().ToString();

}

string WorkWithState()

{

// Возврат размера состояния.

return new State().GetState();

}

public void Dispose()

{

Console.WriteLine("Closed object: " + this.GetHashCode().ToString() + "\n");

}

}

}

namespace DivideService

{

class Program

{

static void Main()

{

Console.Title = "SERVER";

ServiceHost host = new ServiceHost(typeof(MyService));

host.AddServiceEndpoint(typeof(IContract), new WSHttpBinding(), "http://localhost");

host.Open();

Console.ReadKey();

host.Close();

}

}

}

## Single

namespace Service

{

// Применяя атрибут ServiceBehavior, со свойством InstanceContextMode равным Single, -

// служба помечается как служба УРОВНЯ SINGLETON.

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]

public class MyService : IContract, IDisposable

{

public string MyMethod()

{

return this.GetHashCode().ToString();

}

public void Dispose()

{

Console.WriteLine("Closed object: " + this.GetHashCode().ToString());

}

}

}

## Session

// Служба УРОВНЯ СЕАНСА.

namespace Client

{

class Program

{

static void Main()

{

Console.Title = "CLIENT";

NetTcpBinding binding = new NetTcpBinding(SecurityMode.None);

binding.ReliableSession.Enabled = true;

binding.ReliableSession.InactivityTimeout = TimeSpan.FromMinutes(10);

IContract channel = ChannelFactory<IContract>.CreateChannel(

binding, new EndpointAddress("net.tcp://localhost:808/MyService"));

try

{

for (int i = 0; i < 10; i++)

{

string hash = channel.MyMethod();

Console.WriteLine(hash);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

// Задержка.

Console.ReadKey();

}

}

}

namespace Contract

{

[ServiceContract]

public interface IContract

{

[OperationContract]

string MyMethod();

}

}

namespace Service

{

// Применяя атрибут ServiceBehavior, со свойством InstanceContextMode равным PerSession, -

// служба помечается как служба УРОВНЯ СЕАНСА.

// PerSession - является значением по умолчанию.

[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]

public class MyService : IContract, IDisposable

{

public string MyMethod()

{

Console.WriteLine("Current object: " + this.GetHashCode().ToString());

return this.GetHashCode().ToString();

}

public void Dispose()

{

Console.WriteLine("Closed object: " + this.GetHashCode().ToString());

}

}

}

namespace DivideService

{

class Program

{

static void Main()

{

Console.Title = "SERVER";

ServiceHost host = new ServiceHost(typeof(MyService));

NetTcpBinding binding = new NetTcpBinding(SecurityMode.None);

binding.ReliableSession.Enabled = true;

binding.ReliableSession.InactivityTimeout = TimeSpan.FromMinutes(10);

host.AddServiceEndpoint(typeof(IContract), binding, "net.tcp://localhost:808/MyService");

host.Open();

Console.ReadKey();

host.Close();

}

}

}