Rozproszone Systemy Informatyczne

Raport - Ćwiczenie 2a

Paweł Kluska, 260391

Katsiaryna Ziatsikava, 245891

System składa się z 3 modułów:

- WcfServiceLibrary

- WcfServiceHost

- WcfClient

WcfServiceLibrary umożliwia tworzenie usług WCF, zawiera kontrakty usług, implementacje i konfiguracje.

WcfServiceHost - aplikacja konsolowa hostująca usługę WCF.

WcfClient - aplikacja klienta usługi - korzysta ona z metod zdefiniowanych w kontrakcie.

WcfServiceLibrary

Zawiera interfejs ICalculator, klasę MyCalculator, która implementuje interfejs i App.config, w którym jest zdefiniowana konfiguracja własności i dostępności implementacji kontaktu.

Interfejs **ICalculator** zawiera metody **iAdd, iSub, iMul, iDiv, iMod** i **iCountPrimes.**

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Net.Security;
5   using System.Runtime.Serialization;
6   using System.ServiceModel;
7   using System.Text;
8   using System.Threading.Tasks;
9
10  namespace WcfServiceLibrary
11  {
12      [ServiceContract(ProtectionLevel = ProtectionLevel.None)]
        2 references
13      public interface ICalculator
14      {
15
16          [OperationContract]
            1 reference
17          int iAdd(int n1, int n2);
18          [OperationContract]
            1 reference
19          int iSub(int n1, int n2);
20          [OperationContract]
            1 reference
21          int iMul(int n1, int n2);
22          [OperationContract]
            1 reference
23          int iDiv(int n1, int n2);
24          [OperationContract]
            1 reference
25          int iMod(int n1, int n2);
26
27          [OperationContract]
            1 reference
28          Task<(int count,int maxPrimer)> iCountPrimes(int n1, int n2);
29      }
30  }
```

Klasa **MyCalculator** zawiera implementację metod interfejsu **ICalculator**. Metody zawierają instrukcję checked, która rzuca wyjątek gdy nastąpi przepełnienie podczas wykonywania operacji arytmetycznych.

```
1 reference
public class MyCalculator : ICalculator
{
    1 reference
    public int iAdd(int n1, int n2)
    {
        checked // kontrola przepełnienia
        {
            return n1 + n2;
        }
    }

    1 reference
    public int iSub(int n1, int n2)
    {
        checked // kontrola przepełnienia
        {
            return n1 - n2;
        }
    }

    1 reference
    public int iMul(int n1, int n2)
    {
        checked // kontrola przepełnienia
        {
            return n1 * n2;
        }
    }

    1 reference
    public int iDiv(int n1, int n2)
    {
        if (n2 == 0)
        {
            throw new FaultException<DivideByZeroException>(
                new DivideByZeroException(), "Cannot divide by zero");
        }
        return n1 / n2;
    }
```

1 reference
```
public int iMod(int n1, int n2)
{
    if (n2 == 0)
    {
        throw new ArgumentException("Cannot modulo by zero");
    }
    return n1 % n2;
}
```

1 reference
```
async public Task<(int count, int maxPrimer)> iCountPrimes(int n1, int n2)
{
    int count = 0;
    int maxPrime = 0;

    for (int i = n1; i <= n2; i++)
    {
        bool isPrime = true;

        for (int j = 2; j <= Math.Sqrt(i); j++)
        {
            if (i % j == 0)
            {
                isPrime = false;
                break;
            }
        }

        if (isPrime)
        {
            count++;
            if (i> maxPrime)
            {
                maxPrime = i;

            }
        }
    }

    int s1 =  count;
    int s2 =  maxPrime;

    return (s1,s2);
}
```

WcfServiceHost

Na początku została dodana referencja do projektu kontraktu usługi, czyli **WcfServiceLibrary.**

WcfServiceHost zawiera klasę **Program**, w której jest zaimplementowane utworzenie URI, instancji serwisu, dodanie punktu końcowego serwisu, ustawienie metadanych oraz uruchomienie serwisu.

```
0 references
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        MyData.info();
        // Krok 1 URI dla bazowego adresu serwisu
        Uri baseAddress = new Uri("http://localhost:8080/ServiceBaseName");
        // Krok 2 Instancja serwisu
        ServiceHost myHost = new
        ServiceHost(typeof(MyCalculator), baseAddress);
        // Krok 3 Endpoint serwisu
        BasicHttpBinding myBinding = new BasicHttpBinding();
        ServiceEndpoint endpoint1 = myHost.AddServiceEndpoint(
        typeof(ICalculator), myBinding,"endpoint1");

        WSHttpBinding binding2 = new WSHttpBinding();
        binding2.Security.Mode = SecurityMode.None;
        ServiceEndpoint endpoint2 = myHost.AddServiceEndpoint(
        typeof(ICalculator),
        binding2, "endpoint2");

        // Krok 4 Ustawienie metadanych
        ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
        smb.HttpGetEnabled = true;
        myHost.Description.Behaviors.Add(smb);
        try
        {
            // Krok 5 Uruchomienie serwisu.
            myHost.Open();
            Console.WriteLine("Service is started and running.");
            Console.WriteLine("\n---> Endpoints:");
            Console.WriteLine("\nService endpoint {0}:", endpoint1.Name);
            Console.WriteLine("Binding: {0}", endpoint1.Binding.ToString());
            Console.WriteLine("ListenUri: {0}", endpoint1.ListenUri.ToString());
            Console.WriteLine("Press <ENTER> to STOP service...");
            Console.WriteLine();
            Console.ReadLine(); // aby nie kończyć natychmiast:
            myHost.Close();
        }
        catch (CommunicationException ce)
        {
            Console.WriteLine("Exception occured: {0}", ce.Message);
            myHost.Abort();
        }
    }
```

WcfClient

Została tutaj dodana referencja serwisowa do zdefiniowanej wcześniej usługi. Klient łączy się z usługą przy pomocy url określonego przy budowie usługi hostującej serwis.

Został również zaimplementowany interfejs kontaktu usługi, czyli **ICalculator.**

```
namespace WcfClient
{
    [ServiceContract(ProtectionLevel = ProtectionLevel.None)]
    4 references
    public interface ICalculator
    {

        [OperationContract]
        1 reference
        int iAdd(int n1, int n2);
        [OperationContract]
        1 reference
        int iSub(int n1, int n2);
        [OperationContract]
        0 references
        int iMul(int n1, int n2);
        [OperationContract]
        1 reference
        int iDiv(int n1, int n2);
        [OperationContract]
        1 reference
        int iMod(int n1, int n2);

        [OperationContract]
        1 reference
        Task<int> iCountPrimes(int n1, int n2);
    }
}
```

W klasie **Program** jest zaimplmentowane utworzenie obiektu Uri adresu bazowego usługi, binding'u, punktu końcowego, klienta proxy oraz proste menu.

```
namespace WcfClient
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            MyData.info();
            Console.WriteLine("... The client is started");
            // Step 1: Create client proxy based on communication channel.
            // base address:
            Uri baseAddress;
            // binding, address, endpoint address:
            BasicHttpBinding myBinding = new BasicHttpBinding();
            baseAddress = new
            Uri("http://192.168.43.30:8080/ServiceBaseName/endpoint1");
            EndpointAddress eAddress = new EndpointAddress(baseAddress);
            // channel factory:
            ChannelFactory<ICalculator> myCF = new
            ChannelFactory<ICalculator>(myBinding, eAddress);
            // client proxy (here myClient) based on channel
            ICalculator myClient = myCF.CreateChannel();



            bool exit = false;
            while (!exit)
            {
                Console.WriteLine("Calculator Menu:");
                Console.WriteLine("1. Add");
                Console.WriteLine("2. Subtract");
                Console.WriteLine("3. Multiply");
                Console.WriteLine("4. Divide");
                Console.WriteLine("5. Modulo");
                Console.WriteLine("6. Prime");
                Console.WriteLine("7. Exit");
                Console.Write("Enter your choice: ");
```

```
try
{
    string choice = Console.ReadLine();
    switch (choice)
    {
        case "1":
            Console.Write("...calling Add (for entpoint1) ");
            Console.Write("Enter first number: ");
            int x = int.Parse(Console.ReadLine());
            Console.Write("Enter second number: ");
            int y = int.Parse(Console.ReadLine());
            Console.WriteLine("Result: {0}", myClient.iAdd(x, y));
            break;
        case "2":
            Console.Write("Enter first number: ");
            int x1 = int.Parse(Console.ReadLine());
            Console.Write("Enter second number: ");
            int y1 = int.Parse(Console.ReadLine());
            Console.WriteLine("Result: {0}", myClient.iSub(x1, y1));
            break;
        case "3":
            CalculatorClient myClient2 = new CalculatorClient("WSHttpBinding_ICalculator");
            Console.Write("...calling Multiply (for endpoint2) - ");
            Console.Write("Enter first number: ");
            int x2 = int.Parse(Console.ReadLine());
            Console.Write("Enter second number: ");
            int y2 = int.Parse(Console.ReadLine());
            Console.WriteLine("Result: {0}", myClient2.iMul(x2, y2));
            break;
        case "4":
            Console.Write("Enter first number: ");
            int x3 = int.Parse(Console.ReadLine());
            Console.Write("Enter second number: ");
            int y3 = int.Parse(Console.ReadLine());
            Console.WriteLine("Result: {0}", myClient.iDiv(x3, y3));
            break;
        case "5":
            Console.Write("Enter first number: ");
            int x4 = int.Parse(Console.ReadLine());
            Console.Write("Enter second number: ");
            int y4 = int.Parse(Console.ReadLine());
            Console.WriteLine("Result: {0}", myClient.iMod(x4, y4));
            break;
```

```
            case "6":
                Console.Write("Enter first number: ");
                int x5 = int.Parse(Console.ReadLine());
                Console.Write("Enter second number: ");
                int y5 = int.Parse(Console.ReadLine());
                Console.WriteLine("...calling HMultiply ASYNCHRONOUSLY !!!");
                Task<int> asyResult = callHMultiplyAsync(x5, y5, myClient);

                new Thread(() =>
                {
                    Thread.CurrentThread.IsBackground = true;
                    /* run your code here */
                    Console.WriteLine("...HMultiplyAsync Result = " + asyResult.Result);
                }).Start();

                break;
            case "7":
                exit = true;
                break;
            default:
                Console.WriteLine("Invalid choice. Please try again.");
                break;
        }
    }
    catch (FaultException<DivideByZeroException> ex)
    {
        Console.WriteLine(ex.Detail.Message);
    }
    catch (FaultException<OverflowException> ex)
    {
        Console.WriteLine(ex.Detail.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
```

```
        Console.WriteLine("...press <ENTER> to STOP client...");
        Console.WriteLine();
        Console.ReadLine();
        ((IClientChannel)myClient).Close();
        Console.WriteLine("...Client closed - FINISHED");
    }

    1 reference
    static async Task<int> callHMultiplyAsync(int n1, int n2, ICalculator myClient2)
    {
        Console.WriteLine("......called callHMultiplyAsync");
        int reply = await myClient2.iCountPrimes(n1, n2);
        Console.WriteLine("......finished HMultipleAsync");
        return reply;
    }

    0 references
    static void exceptionTest(CalculatorClient client1)
    {
        try
        {
            // int n1 = 2147483647;
            int n1 = 56;
            int n2 = 0;

            int result = client1.iDiv(n1, n2);
            Console.WriteLine($"{n1} / {n2} = {result}");
        }
        catch (FaultException<DivideByZeroException> ex)
        {
            Console.WriteLine(ex.Detail.Message);
        }
        catch (FaultException<OverflowException> ex)
        {
            Console.WriteLine(ex.Detail.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }

    }
}
```

Działanie systemu

System uruchomiliśmy w konfiguracji dwu maszynowej, na jednej maszynie klient a na drugiej serwer. Oba komputery zostały podłączone do tej samej sieci lokalnej. Działanie systemu prezentuje się następująco:

Serwer:

```
D:\RSI Lab\WcfServiceLibrary\WcfServiceHost\bin\Debug>WcfServiceHost.exe
Pawel' Kluska, 260391
Katya Zyatikava, 245891
18 kwietnia 08:57:22
v4.0.30319
DESKTOP-9A54MBP\katya
Microsoft Windows 10.0.19045
192.168.43.30
Service is started and running.

---> Endpoints:

Service endpoint BasicHttpBinding_ICalculator:
Binding: System.ServiceModel.BasicHttpBinding
ListenUri: http://localhost:8080/ServiceBaseName/endpoint1
Press <ENTER> to STOP service...
```

Przykładowe operacje wykonywane przez klienta:

Dodawanie

```
Pawel' Kluska, 260391
Katya Zyatikava, 245891
18 kwietnia 19:26:42
v4.0.30319
DESKTOP-MK2E7R1\pawel
Microsoft Windows 10.0.19045
192.168.43.162
... The client is started
Calculator Menu:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulo
6. Prime
7. Exit
Enter your choice: 1
...calling Add (for entpoint1) Enter first number: 1
Enter second number: 6
Result: 7
Calculator Menu:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulo
6. Prime
7. Exit
Enter your choice: 235
Invalid choice. Please try again.
```

Obsłużenie wyjątka przepełnienia stosu

```
Enter your choice: 1
...calling Add (for entpoint1) Enter first number: 2000000000
Enter second number: 2000000000
Przepelnienie stosu
```

Odejmowanie

```
Enter your choice: 2
Enter first number: 5
Enter second number: 2
Result: 3
```

Mnożenie

```
Enter your choice: 3
...calling Multiply (for endpoint2) - Enter first number: 5
Enter second number: 5
Result: 25
```

Dzielenie

```
Enter your choice: 4
Enter first number: 20
Enter second number: 4
Result: 5
```

Dzielenie modulo

```
Enter your choice: 5
Enter first number: 23
Enter second number: 5
Result: 3
```

Obliczanie asynchroniczne liczb pierwszych (Zostało wykonane dodawanie podczas obliczania liczb pierwszych

```
6
Enter first number: 1
Enter second number: 4000000
...calling HMultiply ASYNCHRONOUSLY !!!
......called callHMultiplyAsync
Calculator Menu:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulo
6. Prime
7. Exit
Enter your choice: 1
...calling Add (for entpoint1) Enter first number: 1
Enter second number: 1
Result: 2
Calculator Menu:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulo
6. Prime
7. Exit
Enter your choice: ......finished HMultipleAsync
Ilosc liczb pierwszych: 283146
Najwieksza liczba pierwsza 3999971
...HMultiplyAsync Result =
(283146, 3999971)
```