**Лекція 14. Триланкова архітектура на базі технології WCF з визначенням MEX -точки**

**Включение оконечной точки обмена метаданными (MEX)**

Метаданные в WCF содержат информацию, точно описывающую, как следует обращаться к службе. Запросив у работающей службы метаданные, клиент может узнать о ее оконечных точках и требуемых форматах сообщений. На этапе проектирования клиенты посылают такой запрос в виде сообщения, определенного в стандарте WS\_MetadataExchange, и получают в ответ WSDL\_документ. Этот документ клиент может использовать для генерации прокси\_класса и конфигурационного файла, которые впоследствии будут использоваться для доступа к службе во время выполнения. Эта схема взаимодействия представлена на рис. 1.4. По умолчанию WCF\_службы не раскрывают оконечную точку MEX. Это означает, что никто не сможет узнать у службы, как с ней взаимодействовать. Не

зная адреса, привязок и контракта, очень трудно обратиться к службе, которая не занесена в реестр. Но к счастью WCF позволяет очень просто раскрыть оконеч ную точку MEX, чтобы клиенты могли корректно общаться со службой. Это можно сделать как в коде, так и в конфигурационном файле. В листинге 1.4 приведен код, необходимый для раскрытия оконечной точки MEX службы. Он является продолжением листинга 1.1 в нескольких отношениях. Во\_первых, к службе добавлено поведение, которое заставляет WCF включить контракт для точки MEX – IMetaDataExchange. Во\_вторых, в описание службы добавлена оконечная точка, для которой указан контракт IMetaDataExchange, протокол HTTP и адрес "mex". Поскольку адрес относительный, то в его начало дописывается базовый адрес службы, так что полный адрес будет равен http://localhost:8000/EssentialWCF/mex. Отметим, что поведение также модифицировано с целью разрешить запросы HTTP GET. Это не обязательно, но позволяет пользователям напрямую обращаться к оконечной точке MEX из браузера.

**Листинг 1.4. Служба, раскрывающая оконечную точку MEX в коде**

using System;

using System.ServiceModel;

**using System.ServiceModel.Description;**

namespace EssentialWCF;

{

[ServiceContract]

public interface IStockService

{

[OperationContract]

double GetPrice(string ticker);

}

public class StockService : IStockService

{

public double GetPrice(string ticker)

{

return 94.85;

}

}

public class Service

{

public static void Main()

{

ServiceHost serviceHost = new

ServiceHost(typeof(StockService)),

new Uri("http://localhost:8000/EssentialWCF"));

serviceHost.AddServiceEndPoint(typeof(IStockService),new BasicHttpBinding(),"");

**ServiceMetadataBehavior behavior = new ServiceMetadataBehavior();**

**behavior.HttpGetEnabled = true;**

****

*Рис. 1.4.* Получение метаданных от оконечной точки MEX

**Реализация клиента целиком в коде**

Если оконечная точка должна определить свои АПК, чтобы WCF могла раскрыть ее возможности при запросах из сети, то клиент должен знать АПК, если хочет этими возможностями воспользоваться. Поэтому при написании кода, обращающегося к оконечным точкам службы, АПК включаются в клиентское приложение. С *адресом* оконечной точки все просто – это сетевой адрес, на который отправляются сообщения. Его формат определен транспортным протоколом, заданным в привязке. *Привязка* оконечной точки точно определяет механизм коммуникации, который использует данная точка. В комплект поставки WCF входит ряд заранее сконфигурированных привязок, например: netTcpBinding, wsHttppBinding и basicHttppBinding. *Контракт* определяет точный формат XML, распознаваемый службой. Обычно он задается с помощью атрибутов [ServiceContract] и [DataContract] в определении класса и/или интерфейса, а WCF сериализует структуру класса в виде XML для передачи по сети.

В листинге 1.6 приведен код для вызова операции службы. В него «зашиты» АПК оконечной точки, позволяющие воспользоваться ее возможностями. Прежде всего, клиент определяет интерфейс, к которому собирается обратиться. Определение интерфейса представляет собой точное описание того, как обращаться к службе, и включает имя операции и ее параметры. Далее клиент создает экземпляр класса ChannelFactory, передавая его конструктору АПК оконечной точки. В данном случае мы указываем адрес сервера IIS, в котором размещена служба, в качестве привязки задаем

BasicHttpBinding, а в качестве контракта – интерфейс IStockService. Наконец, клиент получает от фабрики канал для установления связи со службой и «вызывает метод» службы.

Пример реализации WCF/клиента целиком в коде

using System; using System.Collections.Generic; using System.Linq;

using System.Text;

using System.ServiceModel;

namespace WCFClient

{

[ServiceContract]

public interface IStockService

{

[OperationContract]

double GetPrice(string ticket);

}

class Client

{

static void Main()

{

ChannelFactory<IStockService> myChannelFactory = new ChannelFactory<IStockService>

(new BasicHttpBinding(), new EndpointAddress("http://localhost:8000/EssentialWCF"));

IStockService wcfClient = myChannelFactory.CreateChannel();

string ticket;

ticket = "1";

while (ticket != "")

{

Console.WriteLine("Введите тип билета");

ticket = Console.ReadLine();

double p = wcfClient.GetPrice(ticket);

Console.WriteLine("Price is {0}", p);

}

}

}

}

**Реализация клиента с помощью кода**

**и конфигурационного файла**

. В Visual Studio 2008 операция Add Web Reference по\_прежнему поддерживается ради совместимости с ASMX\_файлами и другими Web\_службами, но добавилась еще и операция Add Service Reference (ASR) (Добавить ссылку на службу) для поддержки WCF. Поскольку WCF не зависит от протокола и поддерживает различные механизмы сериализации, кодирования и обеспечения безопасности, то ASR оказывается более гибкой с точки зрения удобства управления, производительности и безопасности. Операция ASR в Visual Studio применяется для получения метаданных от WCF\_службы и генерации прокси\_класса и конфигурационного файла, как показано на рис. 1.4. Невидимо для вас ASR вызывает программу svcutil.exe, которая запрашивает у оконечной точки MEX ее интерфейсы и генерирует проксикласс и конфигурационный файл. Прокси\_класс позволяет клиенту обращаться к операциям службы так, будто они являются методами локального класса. Прокси\_класс пользуется классами WCF для конструирования и интерпретации SOAP\_сообщений согласно контракту, определенному в оконечной точке службы. В конфигурационном файле хранятся АПК службы.Для программирования клиента, обращающегося к службе, необходимо, во первых, сгенерировать конфигурационный файл и прокси\_класс, а, во\_вторых, написать код, который будет с помощью прокси\_класса обращаться к службе. Чтобы воспользоваться операцией ASR в Visual Studio 2008, щелкните правой кнопкой мыши по узлу Service References (Ссылки на службы) в окне Solution Explorer и выберите из контекстного меню пункт Add Service Reference. В результате появится диалоговое окно, изображенное на рис. 1.5. Из этого окна производится обращение к утилите svcutil с целью создания исходного файла прокси\_класса на языке проекта. Также генерируется файл app.config с секцией <system.serviceModel>, в которой хранится информация об адресе, привязке и контракте, необходимая для вызова оконечных точек. Вместо ASR можно напрямую вызвать утилиту svcutil.exe, которая находится в папке C:\Program Files\Microsoft SDKs\ Windows\v6.0\Bin. У нее много флагов, полное описание которых можно получить, задав флаг -h. Эта утилита принимает на входе метаданные, а на выходе может создавать различные файлы. Метаданные могут читаться из DLL, в которой находится реализация класса, из WSDL\_файла или из WSDL\_документа, возвращаемого в результате обращения к работающей службе с запросом WS\_Metadata. В листинге 1.7 показано, как вызвать svcutil.exe для генерации метаданных для службы, представленной в листингах 1.4 и 1.5.

**Листинг 1.7. Генерация прокси/класса и конфигурационного файла для клиента с помощью svcutil.exe**

svcutil http://localhost:8080/EssentialWCF/mex/

-config:app.config

-out:generatedProxy.cs

Какой бы способ ни выбрать, генерируются одинаковые прокси\_класс и конфигурационный файл. В листинге 1.8 приведен конфигурационный файл. Обратите внимание, что конфигурационный файл для клиента гораздо подробнее, чем для соответствующей службы (см. листинг 1.3). Это позволяет клиенту гибко переопределять некоторые атрибуты, например, величины таймаутов, размеры буферов и предъявляемые верительные грамоты.

**Листинг 1.8. Файл app.config, сгенерированный svcutil.exe**

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

<configuration>

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="BasicHttpBinding\_StockService">

closeTimeout="00:01:00" openTimeout="00:01:00"

receiveTimeout="00:10:00" sendTimeout="00:01:00"

allowCookies="false" bypassProxyOnLocal="false"

hostNameComparisonMode="StrongWildCard"

maxBufferSize="65536" maxBufferPoolSize="524288"

maxReceivedMessageSize="65536"

messageEncoding="Text" textEncoding="utf-8"

transferMode="Buffered" userDefaultWebProxy="true"

<readerQuotas maxDepth="32">

maxStringContentLength="8192"

maxArrayLength="16384"

maxBytesPerRead="4096"

maxNameTableCharCount="16384" />

<security mode="None">

<transport clientCredentialType="None"

clientCredentialType="None"

realm="" />

<message clientCredentialType="UserName"

algorithmSuite="Default" />

</security>

</binding>

</basicHttpBinding>

</bindings>

<client>

<endpointAddress="http://localhost:8080/EssentialWCF"

binding="basicHttpBinding"

bindingConfiguration="BasicHttpBinding\_StockService"

contract="StockService"

name="BasicHttpBinding\_StockService" />

</client>

</system.serviceModel>

</configuration>

После того как конфигурационный файл и прокси\_класс сгенерированы, обратиться к операции, подразумевающей диалог вида запрос\_ответ, совсем просто. Имя прокси\_класса образуется из имени контракта о службе, в конец которого дописано слово Client. Для службы, представленной в листингах 1.4 и 1.5, проксикласс будет называться StockServiceClient. В клиентском коде создается экземпляр прокси\_класса, а затем вызывается его метод. См. листинг 1.9

**Листинг 1.9. Код клиента, вызывающего операцию службы**

using System;

using System.ServiceModel;

namespace EssentialWCF;

{

class Client

{

static void Main()

{

StockServiceClient proxy = new StockServiceClient();

double p = wcfClient.GetPrice("msft");

Console.WriteLine("Price:{0}", p);

proxy.Close();

} } }



**Код на сервере WCF для передачи таблицы БД**

 [ServiceContract]

    public interface IImportdata

    {

        [OperationContract]

        DataTable getPersonControlPoint(DateTime S,DateTime Po);

    }

////

 public class ImportSevise : IImportdata

    {

     public DataTable getPersonControlPoint(DateTime S, DateTime Po)

        {

            DataTable dt = new DataTable("PersonInfo");

                OleDbConnection con = new OleDbConnection("Location=Localhost:D:\\\_\_\_\_\_;      Provider=LCPI.IBProvider; User ID=sysdba; Password=\_\_\_\_\_; Ctype=win1251");

                con.Open();

                OleDbTransaction trans = con.BeginTransaction();

                //begin transaction

                OleDbCommand cmd = new OleDbCommand(@"select тут запрос ", con, trans);

                OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);

                adapter.Fill(dt);

                trans.Commit();

                con.Close();

                return dt;

        }

}

**Код на клиенте WCF для передачи таблицы БД**

DateTime dt1 = new DateTime();

                DateTime dt2 = new DateTime();

                DataTable dt = new DataTable("datainf");

// run client

                ChannelFactory<IImportdata> scf = new ChannelFactory<IImportdata>(new WSHttpBinding(), "http://тут все правильно с app.config сервера");

                IImportdata s = scf.CreateChannel();

                dt = s.getPersonControlPoint(dt1, dt2);

                scf.Close();

Клиент - служба Windows. Смотрю по логам, на сервере запрос выполняется данные приходят в dt.  
На клиенте получаю Null. Что делаю не так???  
Если прилепить на сервер еще функцию, которая просто возвращает строку или число, то на клиент все приходит.

Разобрался сам.