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

**Включення кінцевої точки обміну метаданими (MEX)**

Метадані в WCF містять інформацію, точно описує, як слід звертатися до служби. Запитавши у працюючої служби метадані, клієнт може дізнатися про її кінцевих точках і потрібних форматах повідомлень. На етапі проектування клієнти посилають такий запит у вигляді повідомлення, визначеного в стандарті WS\_MetadataExchange, і отримують у відповідь WSDL\_документ. Цей документ клієнт може використовувати для генерації проксі\_класса і конфігураційного файлу, які згодом будуть використовуватися для доступу до служби під час виконання. Ця схема взаємодії представлена ​​на рис. 1.4. За замовчуванням WCF\_служби не розкривають оконечную точку MEX. Це означає, що ніхто не зможе дізнатися у служби, як з нею взаємодіяти. Чи не

знаючи адреси, прив'язок і контракту, дуже важко звернутися до служби, що не занесена в реєстр. Але на щастя WCF дозволяє дуже просто розкрити оконеч ву точку MEX, щоб клієнти могли коректно спілкуватися зі службою. Це можна зробити як в коді, так і у файлі конфігурації. У лістингу 1.4 приведений код, необхідний для розкриття кінцевої точки MEX служби. Він є продовженням лістингу 1.1 в декількох аспектах. Во\_первих, до служби додано поведінку, яке змушує WCF включити контракт для точки MEX - IMetaDataExchange. Во\_вторих, в опис служби додана крайова точка, для якої зазначений контракт IMetaDataExchange, протокол HTTP і адреса "mex". Оскільки адреса відносний, то в його початок дописується базовий адресу служби, так що повна адреса буде дорівнює http: // localhost: 8000 / EssentialWCF / mex. Відзначимо, що поведінка також модифіковано з метою дозволити запити HTTP GET. Це не обов'язково, але дозволяє користувачам безпосередньо звертатися до кінцевої точки MEX з браузера.

**Лістинг 1.4. Служба, яка розкриває кінцеву точку MEX в коді**

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;

. Параметр не застосовується до розподілених запитам, які виконують процедури, на пов'язаних серверах, визначених при використанні процедури sp\_addlinkedserver.

**2. Виконання розподілених запитів до пов'язаних серверів або виконання віддаленого виклику процедур на віддалених серверах.**

3. Виклик стандартних інструкцій Transact-SQL COMMIT TRANSACTION, COMMIT WORK, ROLLBACK TRANSACTION або ROLLBACK WORK для завершення транзакції.

Для будь-якої розподіленої транзакції Transact-SQL екземпляр Database Engine, що обробляє з'єднання або сценарій Transact-SQL, автоматично викликає MS DTC для координації її фіксації або відкоту.

**4. Приклади**

Приклад розподіленого запиту

Нехай є пов'язаний сервер SEATLESALES в якому обслуговується база даних AdventureWorks2008R2 в якій створені дві схеми

схема Production з таблицею Product і схема Sales з таблицею SalesOrderDetail

Тоді розподілений запит може мати вигляд:

SELECT p.Name, sod.SalesOrderID

FROM

   SEATLESALES. AdventureWorks2008R2. Production.Product p INNER JOIN

  SEATLESALES. AdventureWorks2008R2. Sales. SalesOrderDetail sod

ON p.ProductID = sod.ProductID ORDER BY p.Name

Приклад розподіленої транзакції видалення

У прикладі видаляється кандидат з бази даних База даних AdventureWorks2008R2 як на локальному екземплярі компонента Database Engine, так і на віддаленому сервері. І локальна, і віддалена база даних зафіксує транзакцію або виконає її відкат.

 Примітка. Цей приклад наводить до повідомлення про помилку, якщо координатор MS DTC не встановлено на комп'ютері, де виконується екземпляр компонента Database Engine.

USE AdventureWorks2008R2;

GO

BEGIN DISTRIBUTED TRANSACTION;

- Delete candidate from local instance.

DELETE AdventureWorks2008R2.HumanResources.JobCandidate WHERE JobCandidateID = 13;

- Delete candidate from remote instance.

DELETE RemoteServer.AdventureWorks2008R2.HumanResources.JobCandidate

WHERE JobCandidateID = 13;

COMMIT TRANSACTION;

GO

Приклад розподіленої транзакції на оновлення

BEGIN DISTRIBUTED TRANSACTION;

- Update in local instance.

UPDATE Person

SET LastName = 'JONSON'

WHERE PersonID = 10;

- Update inremote instance.

UPDATE [NOLI / SQL]. Family.dbo.Person

SET LastName = 'JONSON'

WHERE PersonID = 10;

COMMIT TRANSACTION;

GO

5. Моніторинг виконання розподіленої транзакції

Роботу координатора розподілених транзакцій можна переглянути засобами операційної системи.

Ланцюжок дій Start-> Control Panel-> Administrative Tools-> Component services-> Distributed Transaction Controller

Виводить дерево функцій з можливістю перегляду журналу розподілених транзакцій і відповідної статистики

****

*Рис. 1.4.* Отримання метаданих от кінцевої точки MEX

**Реалізація клієнта цілком в коді**

Якщо крайова точка повинна визначити свої АПК, щоб WCF могла розкрити її можливості при запитах з мережі, то клієнт повинен знати АПК, якщо хоче цими можливостями скористатися. Тому при написанні коду, що звертається до крайовим точкам служби, АПК включаються в клієнтську програму. З адресою кінцевої точки все просто - це мережева адреса, на який відправляються повідомлення. Його формат визначений транспортним протоколом, заданим в прив'язці. Прив'язка кінцевої точки точно визначає механізм комунікації, який використовує дана точка. У комплект поставки WCF входить ряд заздалегідь сконфігурованих прив'язок, наприклад: netTcpBinding, wsHttppBinding і basicHttppBinding. Контракт визначає точний формат XML, розпізнається службою. Зазвичай він задається за допомогою атрибутів [ServiceContract] і [DataContract] у визначенні класу і / або інтерфейсу, а WCF серіалізуются структуру класу у вигляді XML для передачі по мережі.

У лістингу 1.6 приведений код для виклику операції служби. У нього «зашиті» АПК кінцевої точки, що дозволяють скористатися її можливостями. Перш за все, клієнт визначає інтерфейс, до якого збирається звернутися. Визначення інтерфейсу являє собою точний опис того, як звертатися до служби, і включає ім'я операції і її параметри. Далі клієнт створює екземпляр класу ChannelFactory, передаючи його конструктору АПК кінцевої точки. В даному випадку ми вказуємо адресу сервера IIS, в якому розміщена служба, як прив'язку задаємо

BasicHttpBinding, а в якості контракту - інтерфейс IStockService. Нарешті, клієнт отримує від фабрики канал для встановлення зв'язку зі службою і «викликає метод» служби.

Приклад реалізації WCF / клієнта цілком в кодіusing 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);

}

}

}

}

Реалізація клієнта за допомогою коду

і конфігураційного файлу

. У Visual Studio 2008 операція Add Web Reference по\_прежнему підтримується заради сумісності з ASMX\_файламі і іншими Web\_службамі, але додалася ще й операція Add Service Reference (ASR) (Додати посилання на службу) для підтримки WCF. Оскільки WCF не залежить від протоколу і підтримує різні механізми серіалізації, кодування і забезпечення безпеки, то ASR виявляється більш гнучкою з точки зору зручності управління, продуктивності і безпеки. Операція ASR в Visual Studio застосовується для отримання метаданих від WCF\_служби і генерації проксі\_класса і конфігураційного файлу, як показано на рис. 1.4. Невидимо для вас ASR викликає програму svcutil.exe, яка запитує у прикінцевій точки MEX її інтерфейси і генерує проксікласс і конфігураційний файл. Проксі\_класс дозволяє клієнту звертатися до операцій служби так, ніби вони є методами локального класу. Проксі\_класс користується класами WCF для конструювання та інтерпретації SOAP\_сообщеній згідно з контрактом, визначеному в кінцевої точки служби. У файлі конфігурації зберігаються АПК служби.Для програмування клієнта, що звертається до служби, необхідно, по-перше, згенерувати конфігураційний файл і проксі\_класс, а, во\_вторих, написати код, який буде за допомогою проксі\_класса звертатися до служби. Щоб скористатися операцією ASR в Visual Studio 2008, клацніть правою кнопкою миші по вузлу Service References (Посилання на служби) у вікні Solution Explorer і виберіть з контекстного меню пункт Add Service Reference. В результаті з'явиться діалогове вікно, зображене на рис. 1.5. З цього вікна проводиться звернення до утиліти svcutil з метою створення вихідного файлу проксі\_класса на мові проекту. Також генерується файл app.config з секцією <system.serviceModel>, в якій зберігається інформація про адресу, прив'язці і контракті, необхідна для виклику кінцевих точок. Замість ASR можна безпосередньо викликати утиліту svcutil.exe, яка знаходиться в папці C: \ Program Files \ Microsoft SDKs \ Windows \ v6.0 \ Bin. У неї багато прапорів, повний опис яких можна отримати, задавши прапор -h. Ця утиліта приймає на вході метадані, а на виході може створювати різні файли. Метадані можуть читатися з DLL, в якій знаходиться реалізація класу, з WSDL\_файла або з WSDL\_документа, що повертається в результаті звернення до працюючої службі із запитом WS\_Metadata. У лістингу 1.7 показано, як викликати svcutil.exe для генерації метаданих для служби, представленої в лістингах 1.4 і 1.5.

Лістинг 1.7. Генерація проксі / класу і конфігураційного файлу для клієнта за допомогою svcutil.exe

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

-config: app.config

-out: generatedProxy.cs

Який би спосіб не вибрати, генеруються однакові проксі\_класс і конфігураційний файл. У лістингу 1.8 приведений конфігураційний файл. Зверніть увагу, що конфігураційний файл для клієнта набагато докладніше, ніж для відповідної служби (див. Лістинг 1.3). Це дозволяє клієнту гнучко перевизначити деякі атрибути, наприклад, величини таймаутів, розміри буферів і пропоновані вірчі грамоти.

Лістинг 1.8. Файл app.config, згенерований 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>

Після того як конфігураційний файл і проксі\_класс згенеровані, звернутися до операції, що має на увазі діалог виду запрос\_ответ, зовсім просто. Ім'я проксі\_класса утворюється з імені контракту про службу, в кінець якого дописано слово Client. Для служби, представленої в лістингах 1.4 і 1.5, проксікласс буде називатися StockServiceClient. У клієнтському коді створюється екземпляр проксі\_класса, а потім викликається його метод. Див. Лістинг 1.9

Лістинг 1.9. Код клієнта, що викликає операцію служби

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 сервері WCF для передачі таблиці 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 запит", con, trans);

OleDbDataAdapter adapter = new OleDbDataAdapter (cmd);

adapter.Fill (dt);

trans.Commit ();

con.Close ();

return dt;

}

}

Код на клієнті WCF для передачі таблиці 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 з app.config сервера");

IImportdata s = scf.CreateChannel ();

dt = s.getPersonControlPoint (dt1, dt2);

scf.Close ();