**Лекція13. Повна розробка служби WCF**

У лекції описана реалізація простий служби за допомогою WCF. Говорячи «проста», ми маємо на увазі, що транспортним протоколом є HTTP, а повідомлення представлено у вигляді XML-документа. Передбачається, що безпека якимось чином забезпечує сам додаток. Будемо вважати, що комунікація є синхронний діалог типу запит-відповідь і що наша служба підтримує тільки одне операцію, яка приймає на вході рядок, а повертає число типу double. У наступних лекціях всі ці припущення будуть піддані ревізії, але поки обійдемося без зайвих складнощів.

Всього лише АПК

Щоб визначити оконечную точку служби, нам знадобиться АПК: адреса, прив'язка і контракт. У лістингах з 1.1 по 1.3 описуються наступні аспекти:

• «А» - адресу або куди. Наша служба буде очікувати вхідні запити за адресою http: // localhost: 8000 / EssentialWCF;

• «П» - прив'язка або як. В даному випадку скористаємося прив'язкою basicHttpBinding, яка змусить WCF реалізувати специфікацію WSI Basic Profile 1.1 - найпоширеніший набір протоколів для взаємодії з Web-службами;

• «К» - контракт або що. Це синтаксичне опис операцій, на які відповідає служба, і форматів вхідних і вихідних повідомлень. У цьому прикладі контракт визначений класом StockService.

Ми реалізуємо цю службу двічі. Спочатку покажемо, як це можна зробити програмно, коли АПК цілком визначений у вихідному коді. Тим самим ми уникнемо будь-яких зовнішніх залежностей. Потім продемонструємо, як те ж саме можна зробити за допомогою конфігураційних файлів. Коду при цьому буде потрібно писати менше, зате складність служби зросте через наявність залежностей між кодом і конфігурацією. На практиці ви, швидше, віддасте перевагу другий підхід, оскільки деяке ускладнення з лишком окупається купується гнучкістю. Гнучкість виникає з того факту, що частина функціональності виноситься в конфігураційні файли, які може змінювати адміністратор, а частина залишається в коді, так що модифікувати її зможе тільки програміст.

**Повністю програмна реалізація служби**

На якомусь надвисокому рівні написання WCF-служби аналогічно написання будь-який інший служби і не залежить від її семантики. Спочатку пишеться код, який реалізує якусь функцію, потім цей код розміщується в процесі операційної системи, а цей процес очікує надходження запитів і відповідає на них. WCF в якійсь мірі формалізує ці кроки, допомагаючи розробнику не допустити помилок на кожному етапі. Наприклад, за допомогою готових прив'язок і кодировщиков WCF-служби можуть обмінюватися стандартними SOAP-повідомленнями.

Механізми многопоточности, паралелізму і породження примірників за замовчуванням мають реалізації, що відрізняються цілком передбачуваним поведінкою. Для реалізації WCF-служби потрібно написати клас на одній з мов .NET, а потім забезпечити його атрибутами з простору імен System.ServiceModel.

Це простір імен встановлюється разом з .NET 3.0 і містить велику частину коду WCF. Під час компіляції вашого класу CLR інтерпретує атрибути, підставляючи замість них виконуваний код. Атрибути - не новина в .NET; вони існують ще з часів .NET 1.0. У WCF, як і в ASMX в .NET 1.0, 1.1 і 2.0, атрибути застосовуються задля спрощення і прискорення процесу написання служб.

У лістингу 1.1 приведений повний код WCF-служби, що розміщується в консоль

ном додатку. Ось що ми робимо:

• Визначаємо контракт. Пишеться клас, який робить щось корисне, після чого він забезпечується атрибутами WCF. Атрибут [ServiceContract] позначає клас, як контракт. У термінах мови WSDL [ServiceContract] визначає тип порту PortType. Атрибут [OperationContract] визначає методи класу, які можна викликати через інтерфейс служби. Одночасно він визначає, які повідомлення можна передати цим методам і отримати від них. З точки зору WSDL, цей атрибут відповідає розділах Operations і Messages. У лістингах 1.1-1.3 визначено клас StockService, що містить єдиний метод GetPrice.

Примітка. У прикладах наведені спрощені інтерфейси, обмежені зазвичай прийомом або поверненням єдиного рядка або числа. На практиці операції служб, швидше за все, повинні будуть приймати і повертати дані складових типів. При мережевому обміні слід прагнути до мінімізації трафіку і затримок, а для цього при кожному зверненні потрібно передавати якомога більше інформації, із за чого і виникає потреба в складових типах.

• Визначаємо кінцеву точку. У визначенні кінцевої точки адреса, прив'язка і контракт задаються за допомогою методу AddServiceEndpoint класу ServiceHost. Адреса ми залишаємо порожнім, це означає, що адреса кінцевої точки такий же, як адресу самої служби. В якості прив'язки вказується basicHttpBinding, сумісна зі специфікацією WSI BP 1.1 і забезпечує інтероперабельність з більшістю систем, в яких реалізовані Web-служби на базі XML. Специфікація WSI, або Web Services Interop. WSI - це не стандарт, а керівництво та комплект інструментів для визначення того, наскільки ПО відповідає існуючим стандартам, наприклад HTTP і XML.

• Розміщуємо службу у процесі, щоб вона могла прослуховувати вхідні запити. У лістингах 1.1-1.3 служба розміщується в консольному додатку за допомогою класу ServiceHost. Служба очікує надходження запитів на адресу http: // localhost: 8000 / EssentialWCF.

Лістинг 1.1. Повністю програмна реалізація служби

using System;

using System.ServiceModel;

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));

serviceHost.Open ();

Console.WriteLine ("Для завершення натисніть <ENTER>. \ N \ n");

Console.ReadLine ();

serviceHost.Close ();

}

}

}

У лістингу 1.2 приведений конфігураційний файл, використовуваний спільно з програмою У секції <serviceModel> визначена крайова точка. Для кожної кінцевої точки задаються адреса, прив'язка і контракт. Адреса в даному випадку порожній, це означає, що потрібно використовувати ту ж адресу, що для самої служби. Якщо у служби кілька кінцевих точок, то у кожної повинен бути унікальний адресу. В якості прив'язки ми вказали basicHttpBinding, а в якості імені контракту - ім'я класу, визначеного в коді, - EssentialWCF. StockService.

Лістинг 1.2. Конфігураційний файл для служби, реалізованої в лістингу 1.2

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

<configuration>

<system.serviceModel>

<services>

<service name = "EssentialWCF.StockService">

<host>

<baseAddresses>

<add baseAddress = "http: // localhost: 8080 / EssentialWCF" />

</ baseAddresses>

</ host>

<endpoint address = "" binding = "basicHttpBinding" contract = "EssentialWCF.IStockService" />

</ service>

</ services>

</system.serviceModel>

</ configuration>

**Ще про конфігураційні файли**

Конфігураційний файл служби, web.config або app.config залежно від того, де служба розміщена, повинен містити секцію <system.serviceModel>. Всередині неї визначаються служби, прив'язки, поведінки, клієнти, способи діагностики, розширення, параметри розміщення та взаємодії з COM +. Як мінімум, повинна існувати підсекція <services>, в якій описані кінцеві точки, а в ній хоча б один вузол <endpoint>, описує не інфраструктурну точку. АПК самої кінцевої точки визначаються всередині цього вузла

Атрибут address визначає URI, на який клієнти будуть посилати повідомлення крайовою точці. Наприклад, якщо для служби визначена прив'язка до протоколу HTTP basicHttpBinding, то URI може виглядати як http://www.myserver.com:8080/ MyService /. Якщо заданий абсолютний адресу (тобто не порожній і містить не тільки шлях), то він заміщає базова адреса, заданий власником служби в момент її створення. Коли власник запускає службу, WCF починає прослуховувати вказану адресу в очікуванні вхідних запитів. Якщо в якості власника виступає IIS, то, швидше за все, прослуховувач вже запущений, тому WCF повідомляє йому про себе - реєструється, щоб запити на цей URI прямували WCF-службі. Атрибут binding описує комунікаційні деталі, необхідні для з'єднання зі службою. Зокрема, визначається весь стек каналів, який повинен як мінімум включати канал мережевого адаптера. Крім цього, можна включити в стек канали шифрування, стиснення та інші. У комплекті з WCF поставляється ряд вже сконфигурированних прив'язок, наприклад: BasicHttpBinding, сумісна з ASMX; WSHttpBinding, що реалізує більш розвинені Web-служби, для яких потрібна безпеку на рівні повідомлень, підтримка транзакцій і інші можливості, і NetTcpBinding, що реалізує швидкий і безпечний формат передачі повідомлень, аналогічний .NET Remoting і DCOM.

Атрибут contract посилається на тип, визначений для кінцевої точки служби. WCF інспектує цей тип і розкриває його метадані у вигляді кінцевої точки MEX, якщо така входить до складу служби. Зазначений тип WCF шукає спочатку в папці \ bin, а потім в глобальному кеші збірок (GAC) на даній машині. Якщо знайти тип не вдається, то служба повертає інформацію про помилку, коли ви клацаєте по вузлу Add Service Reference в Visual Studio або запускаєте утиліту svcutil.exe для генерації WSDL-документа. Якщо оконечная точка MEX не задана, то служба працюватиме нормально, але клієнти не зможуть опитувати її АПК.

**Ще про розміщення служб**

WCF дозволяє розміщувати служби в будь-якому процесі операційної системи. У більшості випадків найбільш підходящим власником є IIS, який забезпечує високу продуктивність, зручність управління і безпеку. Якщо у вашому оточенні IIS працює, то інфраструктура забезпечення безпеки вже прісутствует.В невеликих організаціях зазвичай задовольняються параметрами безпеки, застосовуваними в IIS і Windows 2003 за замовчуванням. Так чи інакше, до WCF-службам, розміщеним в IIS, застосовуються вже певні правила.

Але іноді використовувати IIS в якості власника служби небажано. Можливо, вас не влаштовує протокол HTTP. Або ви хочете явно управляти запуском і зупинкою служби. Або потрібно надати спеціальний адміністративний інтерфейс, відмінний від того, що мається на IIS. Що ж, немає проблем. WCF дозволяє дуже легко і гнучко здійснювати авторозміщення (self-hosting). Цим терміном описується метод розміщення, при якому розробник сам створює екземпляр власника служби, не покладаючись ні на IIS, ні на Windows Process Activation Services (WAS).

Найпростіший спосіб розмістити службу - це написати консольний додаток, як було показано в лістингу 1.1. Для промислової експлуатації це не дуже корисно, тому що відкривати вікно команд на сервері не рекомендується. Але для початкового тестування служби цілком годиться, тому що при цьому усуваються всі залежності від інфраструктури IIS. Головна програма створює новий екземпляр класу ServiceHost, який, як випливає з самої назви, покликаний стати власником служби. Потім програма викликає метод Open цього об'єкта, і продовжує займатися своєю справою. В даному випадку вона просто чекає, поки хто-небудь натисне клавішу Enter, після чого викликає метод Close об'єкта ServiceHost. Після виклику Open об'єкт ServiceHost починає прослуховувати адреси, вказані в описах кінцевих точок. Коли надходить повідомлення, ServiceHost виконує декілька дій. По-перше, відповідно до опису стека каналів у прив'язці, він виконує дешифрування, розпакування і застосовує правила безпеки. По-друге, звертається до контракту для десеріалізациі повідомлення, в результаті чого створюється новий об'єкт.

Потім викликається затребувана операція цього об'єкта.

Включення крайовою точки обміну метаданими (MEX)

Метадані в WCF містять інформацію, точно описує, як слід звертатися до службі. Запитавши у працюючої служби метадані, клієнт може дізнатися про її кінцевих точках і необхідних форматах повідомлень. На етапі проектування клієнти посилають такий запит у вигляді повідомлення, визначеного в стандарті WSMetadataExchange, і отримують у відповідь WSDL-документ. Цей документ клієнт може використовувати для генерації проксі-класу і конфігураційного файлу, які згодом будуть використовуватися для доступу до служби під час виконання. Ця схема взаємодії представлена на рис. 1.4.

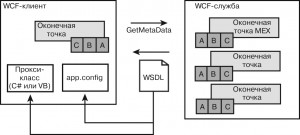
[](http://www.itarchive.ru/wp-content/uploads/2011/02/4.jpg)

Рис. 1.4. Отримання метаданих від кінцевої точки MEX

За замовчуванням WCF-служби не розкривають оконечную точку MEX. Це означає, що ніхто не зможе дізнатися у служби, як з нею взаємодіяти. 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 оконечную 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 крайовою точки MEX служби. Він є продовженням лістингу 1.1 в декількох відносинах. По-перше, до служби додано поведінку, яка змушує WCF включити контракт для точки MEX - IMetaDataExchange. По-друге, в опис служби додана оконечная точка, для якої зазначений контракт IMetaDataExchange, протокол HTTP і адресу «mex». Оскільки адреса відносний, то в його початок дописується базову 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;

<strong> using System.ServiceModel.Description;

</ strong> 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 (), "");

<strong> ServiceMetadataBehavior behavior = new ServiceMetadataBehavior ();

behavior.HttpGetEnabled = true; </ strong>

<strong> serviceHost.Description.Behaviors.Add (behavior);

serviceHost.AddServiceEndpoint (typeof (IMetadataExchange),

MetadataExchangeBindings.CreateMexHttpBinding (), "mex"); </ strong>

serviceHost.Open ();

Console.WriteLine ("Для завершення натисніть <ENTER>. \ N \ n");

Console.ReadLine ();

serviceHost.Close ();

}

}

}

Якщо ви хочете описувати кінцеві точки у файлі конфігурації, а не в коді, то там же потрібно описати і оконечную точку MEX. У лістингу 1.5 показано, як слід модифікувати конфігураційний файл, наведений у лістингу 1.3. До службі додана оконечная точка MEX і поведінку, що дозволяє звертатися до неї по протоколу HTTP.

Лістинг 1.5. Розкриття крайовою точки MEX в конфігураційному файлі

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

<configuration>

<system.serviceModel>

<services>

<service name = "EssentialWCF.StockService" behaviorConfiguration = "myServiceBehavior">

<host>

<baseAddresses>

<add baseAddress = "http: // localhost: 8080 / EssentialWCF" />

</ baseAddresses>

</ host>

<endpoint address = "" binding = "basicHttpBinding" contract = "EssentialWCF.IStockService" />

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

</ service>

</ services>

<behaviors>

<serviceBehaviors>

<behavior name = "myServiceBehavior">

<serviceMetadata htthGetEnabled = "True" />

</ behavior>

</ serviceBehaviors>

</ behaviors>

</system.serviceModel>

</ configuration>

**Реалізація Клієнта**

WCF надає розвинений API для написання клієнтів, що бажають взаємодіяти зі службою. Цей API, що знаходиться в просторі імен System. ServiceModel, бере на себе турботу про серіалізациі типів в XML і відправку повідомлень від клієнта службі. Ви можете застосовувати API в своїх програмах безпосередньо або скористатися інструментами для генерації проксі-класу і конфігураційного файлу. У цьому розділі ми спочатку покажемо, як звернутися до служби безпосередньо, а потім - як зробити те ж саме за допомогою інструментів. При другому підході доводиться писати менше коду, а всі конфігураційні параметри виносяться в зовнішні файли. При першому підході кількість зовнішніх залежностей менше, і ви отримуєте більш точний контроль над зверненнями. Який підхід віддати перевагу, залежить від ситуації.

**Реалізація клієнта цілком в коді**

Якщо кінцева (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 крайовою точки все просто - це мережевий адресу, на яку відправляються повідомлення. Його формат визначений транспортним протоколом, заданих в прив'язці. Прив'язка крайовою 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, розпізнаваний 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 крайовою точки, що дозволяють скористатися її можливостями. Перш за все, клієнт визначає інтерфейс, до якого збирається звернутися. Визначення інтерфейсу є загальним для клієнта і служби. Синтаксично визначення на мові C # сильно відрізняється від формату XML або WSDL, але семантично вони еквівалентні. Те й інше 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 крайовою 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.

Лістинг 1.6. Реалізація WCF / клієнта цілком в коді

using System;

using System.ServiceModel;

namespace Client;

{

[ServiceContract]

public interface IStockService

{

[OperationContract]

double GetPrice (string ticker);

}

class Client

{

static void Main ()

{

ChannelFactory <IStockService> myChannelFactory =

new ChannelFactory <IStockService> (

new BasicHttpBinding (),

new EndpointAddress

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

IStockService wcfClient =

myChannelFactory.CreateChannel ();

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

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

}

}

}

**Реалізація клієнта за допомогою коду і конфігураційного файлу**

Реалізація клієнта за допомогою коду і конфігураційного файлу

Ще в 2001 році в Visual Studio з'явилася операція Add Web Reference (Додати Web-посилання), яка звела розподілені обчислення до клацанню правою кнопкою миші. Це було зовсім непогано, так як багато професійні розробники отримали в своє розпорядження простий спосіб створення масштабованих, заснованих на стандартах розподілених додатків. Але, гранично полегшивши програмування розподілених систем, Microsoft одночасно приховала багато важливі деталі. У Visual Studio 2008 операція Add Web Reference раніше підтримується заради сумісності з ASMX-файлами та іншими Web-службами, 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 серіалізациі, 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-класу 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 проксі-клас і конфігураційний файл. Проксі-клас дозволяє клієнту звертатися до операцій служби так, ніби вони є методами локального класу. Проксі-клас користується класами WCF для конструювання та інтерпретації SOAP-повідомлень згідно з контрактом, визначеному в крайовою точці служби.

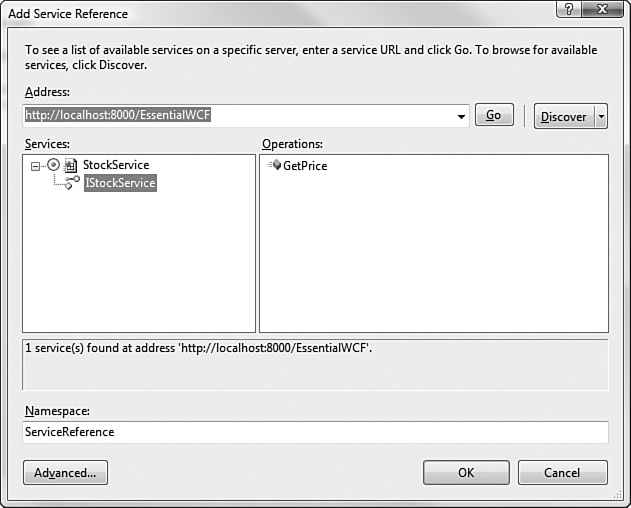
У конфігураційному файлі зберігаються АПК служби. Для програмування клієнта, що звертається до служби, необхідно, по-перше, згенерувати конфігураційний файл і проксі-клас, а, по-друге, написати код, який буде за допомогою проксі-класу звертатися до служби. Щоб скористатися операцією ASR в Visual Studio 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-класу 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 перебуває реалізація класу, з WSDL-файлу 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-клас 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 пе реопределять деякі атрибути, наприклад, величини таймаутів, розміри буферів і пред'являються вірчі грамоти.[](http://www.itarchive.ru/wp-content/uploads/2011/02/5.jpg)

**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>