**Лекция 36. Разработка службы WCF**

В лекции описана реализация простой службы с помощью WCF. Говоря «простая», мы имеем в виду, что транспортным протоколом является HTTP, а сообщение представлено в виде XML-документа. Предполагается, что безопасность каким-то образом обеспечивает само приложение. Будем считать, что коммуникация представляет собой синхронный диалог типа запрос-ответ и что наша служба поддерживает только одну операцию, которая принимает на входе строку, а возвращает число типа double. В следующих лекциях все эти допущения будут подвергнуты ревизии, но пока обойдемся без лишних сложностей.

**Всего лишь АПК**

Чтобы определить оконечную точку службы, нам понадобится АПК: адрес, привязка и контракт. В листингах с 1.1 по 1.3 описываются следующие аспекты:

* «А» – адрес или куда. Наша служба будет ожидать входящие запросы по адресу http://localhost:8000/EssentialWCF;
* «П» – привязка или как. В данном случае воспользуемся привязкой basicHttpBinding, которая заставит WCF реализовать спецификацию WSI Basic Profile 1.1 – самый распространенный набор протоколов для взаимодействия с Web-службами;
* «К» – контракт или что. Это синтаксическое описание операций, на которые отвечает служба, и форматов входных и выходных сообщений. В этом примере контракт определен классом StockService.

Мы реализуем эту службу дважды. Сначала покажем, как это можно сделать программно, когда АПК целиком определен в исходном коде. Тем самым мы избегнем каких-либо внешних зависимостей. Затем продемонстрируем, как то же самое можно сделать с помощью конфигурационных файлов. Кода при этом потребуется писать меньше, зато сложность службы возрастет из-за наличия зависимостей между кодом и конфигурацией. На практике вы, скорее, предпочтете второй подход, так как некоторое усложнение с лихвой окупается приобретаемой гибкостью. Гибкость проистекает из того факта, что часть функциональности выносится в конфигурационные файлы, которые может изменять администратор, а часть остается в коде, так что модифицировать ее сможет только программист.

**Полностью программная реализация службы**

На некоем сверхвысоком уровне написание WCF-службы аналогично написанию любой другой службы и не зависит от ее семантики. Сначала пишется код, реализующий какую-то функцию, затем этот код размещается в процессе операционной системы, а этот процесс ожидает поступления запросов и отвечает на них. WCF в какой-то мере формализует эти шаги, помогая разработчику не допустить ошибок на каждом этапе. Например, с помощью готовых привязок и кодировщиков WCF-службы могут обмениваться стандартными SOAP-сообщениями.

Механизмы многопоточности, параллелизма и порождения экземпляров по умолчанию имеют реализации, отличающиеся вполне предсказуемым поведением. Для реализации WCF-службы требуется написать класс на одном из языков .NET, а затем снабдить его атрибутами из пространства имен System.ServiceModel.

Это пространство имен устанавливается вместе с .NET 3.0 и содержит большую часть кода WCF. Во время компиляции вашего класса CLR интерпретирует атрибуты, подставляя вместо них исполняемый код. Атрибуты – не новость в .NET; они существуют еще со времен .NET 1.0. В WCF, как и в ASMX в .NET 1.0, 1.1 и 2.0, атрибуты применяются ради упрощения и ускорения процесса написания служб.  
В листинге 1.1 приведен полный код WCF-службы, размещаемой в консоль  
ном приложении. Вот что мы делаем:

* Определяем контракт. Пишется класс, который делает нечто полезное, после чего он снабжается атрибутами WCF. Атрибут [ServiceContract] помечает класс, как контракт. В терминах языка WSDL [ServiceContract] определяет тип порта PortType. Атрибут [OperationContract] определяет методы класса, которые можно вызывать через интерфейс службы. Одновременно он определяет, какие сообщения можно передать этим методам и получить от них. С точки зрения WSDL, этот атрибут соответствует разделам Operations и Messages. В листингах 1.1–1.3 определен класс StockService, содержащий единственный метод GetPrice.

**Примечание*.*** В примерах приведены упрощенные интерфейсы, ограниченные обычно приемом или возвратом единственной строки или числа. На практике операции служб, скорее всего, должны будут принимать и возвращать данные составных типов. При сетевом обмене следует стремиться к минимизации трафика и задержек, а для этого при каждом обращении нужно передавать как можно больше информации, из за чего и возникает потребность в составных типах.

* Определяем оконечную точку. В определении оконечной точки адрес, привязка и контракт задаются с помощью метода AddServiceEndpoint класса ServiceHost. Адрес мы оставляем пустым, это означает, что адрес оконечной точки такой же, как адрес самой службы. В качестве привязки указывается basicHttpBinding, совместимая со спецификацией WSI BP 1.1 и обеспечивающая интероперабельность с большинством систем, в которых реализованы Web-службы на базе XML. Спецификация WSI, или Web Services Interop. WSI – это не стандарт, а руководство и комплект инструментов для определения того, насколько ПО отвечает существующим стандартам, например HTTP и XML.
* Размещаем службу в процессе, чтобы она могла прослушивать входящие запросы. В листингах 1.1–1.3 служба размещается в консольном приложении с помощью класса ServiceHost. Служба ожидает поступления запросов на адрес  http://localhost:8000/EssentialWCF.

**Листинг 1.1.** Полностью программная реализация службы

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 завершения нажмите <ENTER>.\n\n");

                    Console.ReadLine();

                    serviceHost.Close();

               }

      }

}

В листинге 1.3 приведен конфигурационный файл, используемый совместно с программой из листинга 1.2. В секции <serviceModel> определена оконечная точка. Для каждой оконечной точки задаются адрес, привязка и контракт. Адрес в данном случае пуст, это означает, что нужно использовать тот же адрес, что для самой службы. Если у службы несколько оконечных точек, то у каждой должен быть уникальный адрес. В качестве привязки мы указали basicHttpBinding, а в качестве имени контракта – имя класса, определенного в коде, – EssentialWCF. StockService.

**Листинг 1.3.** Конфигурационный файл для службы, реализованной в листинге 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>

**Еще о конфигурационных файлах**

Конфигурационный файл службы, web.config или app.config в зависимости от того, где служба размещена, должен содержать секцию <system.serviceModel>. Внутри нее определяются службы, привязки, поведения, клиенты, способы диагностики, расширения, параметры размещения и взаимодействия с COM+. Как минимум, должна существовать подсекция <services>, в которой описаны оконечные точки, а в ней хотя бы один узел <endpoint>, описывающий не инфраструктурную точку. АПК самой оконечной точки определяются внутри этого узла

Атрибут address определяет URI, на который клиенты будут посылать сообщения оконечной точке. Например, если для службы определена привязка к протоколу HTTP basicHttpBinding, то URI может выглядеть как http://www.myserver.com:8080/ MyService/. Если задан абсолютный адрес (то есть не пустой и содержащий не только путь), то он замещает базовый адрес, заданный владельцем службы в момент ее создания. Когда владелец запускает службу, WCF начинает прослушивать указанный  адрес в ожидании входящих запросов. Если в качестве владельца выступает IIS, то, скорее всего, прослушиватель уже запущен, поэтому WCF сообщает ему о себе – регистрируется, чтобы запросы на этот URI направлялись WCF-службе. Атрибут binding описывает коммуникационные детали, необходимые для соединения со службой. В частности, определяется весь стек каналов, который должен как минимум включать канал сетевого адаптера. Помимо этого, можно  включить в стек каналы шифрования, сжатия и другие. В комплекте с WCF поставляется ряд уже сконфигурированных привязок, например: BasicHttpBinding, совместимая с ASMX; WSHttpBinding, реализующая более развитые Web-службы, для которых требуется безопасность на уровне сообщений, поддержка транзакций и другие возможности, и NetTcpBinding, реализующая быстрый и безопасный формат передачи сообщений, аналогичный .NET Remoting и DCOM.  
Атрибут contract ссылается на тип, определенный для оконечной точки службы. WCF инспектирует этот тип и раскрывает его метаданные в виде оконечной точки MEX, если таковая входит в состав службы. Указанный тип WCF ищет сначала в папке \bin, а затем в глобальном кэше сборок (GAC) на данной машине. Если найти тип не удается, то служба возвращает информацию об ошибке, когда вы щелкаете по узлу Add Service Reference в Visual Studio или запускаете утилиту svcutil.exe для генерации WSDL-документа. Если оконечная точка MEX не задана, то служба будет работать нормально, но клиенты не смогут опрашивать ее АПК.

**Еще о размещении служб**

WCF позволяет размещать службы в любом процессе операционной системы. В большинстве случаев наиболее подходящим владельцем является IIS, который обеспечивает высокую производительность, удобство управления и безопасность. Если в вашем окружении IIS работает, то инфраструктура обеспечения безопасности уже присутствует.В небольших организациях обычно довольствуются параметрами безопасности, применяемыми в IIS и Windows 2003 по умолчанию. Так или иначе, к WCF-службам, размещенным в IIS, применяются уже определенные правила.  
Но иногда использовать IIS в качестве владельца службы нежелательно. Возможно, вас не устраивает протокол HTTP. Или вы хотите явно управлять запуском и остановом службы. Или требуется предоставить специальный административный интерфейс, отличный от того, что имеется в IIS. Что ж, нет проблем. WCF позволяет очень легко и гибко осуществлять авторазмещение (self-hosting). Этим термином описывается метод размещения, при котором разработчик сам создает экземпляр владельца службы, не полагаясь ни на IIS, ни на Windows Process Activation Services (WAS).  
Простейший способ разместить службу – это написать консольное приложение, как было показано в листинге 1.1. Для промышленной эксплуатации это не очень полезно, так как открывать окно команд на сервере не рекомендуется. Но для первоначального тестирования службы вполне годится, так как при этом устраняются все зависимости от инфраструктуры IIS. Главная программа создает новый экземпляр класса ServiceHost, который, как следует из самого названия, призван стать владельцем службы. Затем программа вызывает метод Open этого объекта, и продолжает заниматься своим делом. В данном случае она просто ждет, пока кто-нибудь нажмет клавишу Enter, после чего вызывает метод Close объекта ServiceHost. После вызова Open объект ServiceHost начинает прослушивать адреса, указанные в описаниях оконечных точек. Когда поступает сообщение, ServiceHost выполняет несколько действий. Во-первых, в соответствии с описанием стека каналов в привязке, он выполняет дешифрирование, распаковку и применяет правила безопасности. Во-вторых, обращается к контракту для десериализации сообщения, в результате чего создается новый объект. Затем вызывается затребованная операция этого объекта.

**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 WSMetadataExchange, 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.

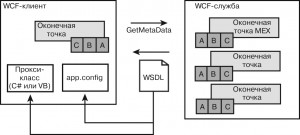
[](http://www.itarchive.ru/wp-content/uploads/2011/02/4.jpg)

Рис. 1.4. Получение метаданных от оконечной точки MEX

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 оконечную 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;

<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 завершения нажмите <ENTER>.\n\n");

             Console.ReadLine();

             serviceHost.Close();

          }

     }

}

Если вы хотите описывать оконечные точки в конфигурационном файле, а не в коде, то там же нужно описать и оконечную точку MEX. В листинге 1.5 показано, как следует модифицировать конфигурационный файл, приведенный в листинге 1.3. К службе добавлена оконечная точка MEX и поведение, позволяющее обращаться к ней по протоколу HTTP.

**Листинг 1.5.** Раскрытие оконечной точки MEX в конфигурационном файле

<?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>

**Реализация Клиента**

WCF предоставляет развитый API для написания клиентов, желающих взаимодействовать со службой. Этот API, находящийся в пространстве имен System. ServiceModel, берет на себя заботу о сериализации типов в XML и отправку сообщений от клиента службе. Вы можете применять API в своих программах напрямую или воспользоваться инструментами для генерации прокси-класса и конфигурационного файла. В этом разделе мы сначала покажем, как обратиться к службе непосредственно, а потом – как сделать то же самое с помощью инструментов. При втором подходе приходится писать меньше кода, а все конфигурационные параметры выносятся во внешние файлы. При первом подходе количество внешних зависимостей меньше, и вы получаете более точный контроль над обращениями. Какой подход предпочесть, зависит от ситуации.

**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 является общим для клиента и службы. Синтаксически определение на языке C# сильно отличается от формата XML или WSDL, но семантически они эквивалентны. То и другое 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.

**Листинг 1.6.** Реализация WCF/клиента целиком в коде

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);

     }

}

}

**Реализация клиента с помощью кода и конфигурационного файла**

Еще в 2001 году в Visual Studio появилась операция Add Web Reference (Добавить Web-ссылку), которая свела распределенные вычисления к щелчку правой кнопкой мыши. Это было совсем неплохо, так как многие профессиональные разработчики получили в свое распоряжение простой способ создания масштабируемых, основанных на стандартах распределенных приложений. Но, предельно облегчив программирование распределенных систем, Microsoft одновременно скрыла многие важные детали. 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 прокси-класс 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.

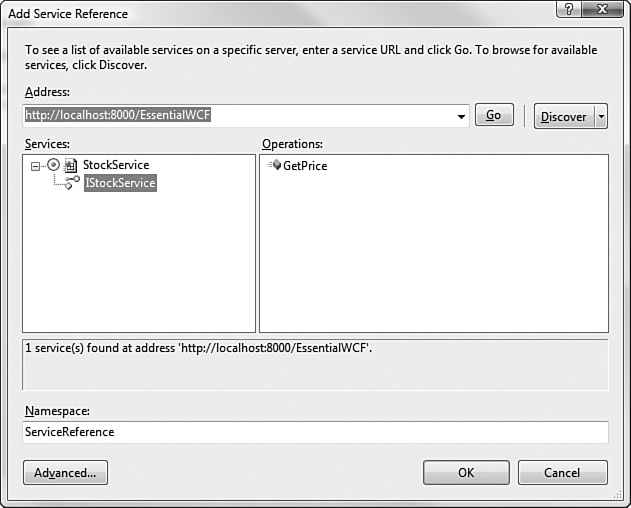
**Листинг 1.7.** Генерация прокси/класса и конфигурационного файла для клиента с помощью svcutil.exe

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 пе реопределять некоторые атрибуты, например, величины таймаутов, размеры буферов и предъявляемые верительные грамоты.

[](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>

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, прокси-класс 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()

       {

          ServiceReference1.StockServiceClient wsfClient = new StockServiceClient();

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

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

          proxy.Close();

       }

    }

}