3) Веб-службы WCF // Структура приложения ASP.NET MVC

Веб-службы WCF

(про вебслужбы в общем виде в вопросе веб службы Web API)

Windows Communication Foundation (далее *WCF*) - единая программная модель, предназначенная для создания распределённых и сервис-ориентированных приложений. В свою очередь, *сервис-ориентированная архитектура* (*service-oriented architecture*, *SOA*) – это модульный подход к разработке программного обеспечения, основанный на использовании сервисов (служб) со стандартизированными интерфейсами.

В основе функционирования WCF лежат так называемые конечные точки (endpoints), состоящие из адреса, привязки и контракта (address-binding-contract, ABC). Адрес задает местоположение конечной точки. Привязка определяет протокол, на основе которого будет происходить взаимодействие. Контракт регламентирует, какие операции умеет выполнять сервис.

Любой адрес, используемый в WCF, имеет формат:

```
[базовый адрес]/[опциональный URI]
```

В свою очередь, базовый адрес состоит из следующих частей:

```
[транспорт]://[машина или домен][:порт(опционально)]
```

Здесь `[транспорт]` указывает на протокол коммуникации:

- `net.tcp` – для протокола TCP;
- `http` (или `https`) – для протокола HTTP;
- `net.pipe` – при использовании именованных каналов.

Для протокола TCP в адресе обычно указывается и порт. Если этого не сделать, используется порт 808. Для протокола HTTP по умолчанию используется порт 80. Если в качестве транспорта используются именованные каналы, то адрес имеет форму `net.pipe://localhost/MyPipe`. Необходимо указывать или имя машины, или `localhost` (коммуникация между машинами запрещена). Последняя часть адреса рассматривается как имя канала.

Как было указано выше, в WCF привязка определяет транспортный протокол для взаимодействия клиента и службы. Кроме этого, привязка задаёт формат передаваемых данных. От выбора привязки зависит поддержка транзакций, сессий, возможность дуплексного обмена.

Основные привязки WCF

Имя привязки	**Описание**	**Транспорт**

basicHttpBinding	Привязка для веб-служб, совместимых с WS-I Basic Profile 1.1 (в частности, asmx-службы)	HTTP/HTTPS
netTcpBinding	Привязка для коммуникации между двумя .NET приложениями по протоколу TCP	TCP
netPeerTcpBinding	Привязка для построения пиринговых сетей	P2P
netNamedPipeBinding	Привязка, использующая именованные каналы. Применяется для взаимодействия двух .NET приложений на **одной** машине	IPC
wsHttpBinding	Привязка для веб-служб с дополнительными возможностями (безопасность, транзакции)	HTTP/HTTPS
wsFederationHttpBinding	Привязка для продвинутых веб-служб с интегрированной идентификацией	HTTP/HTTPS
wsDualHttpBinding	Аналог wsHttpBinding, но поддерживает двухсторонние коммуникации на основе дуплексных контрактов	HTTP
netMsmqBinding	Привязка, использующая в качестве транспорта Microsoft Message Queue	MSMQ
msmqIntegrationBinding	Аналог netMsmqBinding, но для работы с унаследованным ПО на основе MSMQ	MSMQ

Любые службы WCF реализуют сервисные контракты. *Сервисный контракт* – это платформенно-независимый стандартизированный способ описания того, что может делать служба. Кроме этого, на основе контракта на клиенте строится прокси-класс для взаимодействия со службой[1].

Если в случае веб-служб их средой выполнения (или *хостингом*) было веб-приложение ASP.NET, то для хостинга WCF-служб может использоваться несколько вариантов:

[1] Кроме сервисных контрактов, в WCF существуют *контракты данных* (определяют типы данных, которые можно передавать между клиентом и службой) и *контракты исключений* (описывают, какие исключения может генерировать служба, и как информация об этом передается клиенту).

1. WCF-служба входит в состав сайта ASP.NET (подобно веб-службам).
2. Хостингом WCF-службы является сервер IIS (это, по сути, ASP.NET-сайт, состоящий из одной WCF-службы).
3. WCF-служба может размещаться в Windows-приложении .NET (включая системные сервисы).

Клиентские WCF-приложения применяют для работы со службами прокси-классы. Работа прокси-классов основывается на использовании каналов. *Канал* – это клиентский объект, агрегирующий информацию о сервисном контракте и одной концевой точке вызываемой службы.

6.3. ДЕМОНСТРАЦИОННОЕ WCF-ПРИЛОЖЕНИЕ

Наша служба будет содержать два метода. Метод `GetCustomer()` возвращает строку с информаций о покупателе по идентификатору покупателя. Метод `GetCustomersCount()` возвращает общее число покупателей.

1. Создание сервисного контракта.

Начнём разработку службы с создания сервисного контракта. На платформе .NET сервисным контрактом становится класс или интерфейс, помеченный атрибутом `[ServiceContract]`. Обычно атрибут `[ServiceContract]` применяется к интерфейсу – это позволяет разделить описание контракта между службой и клиентом. Если же этот атрибут применяется к классу, то класс должен иметь `public`-конструктор без параметров. Методы сервисного контракта, доступные клиенту, должны быть помечены атрибутом `[OperationContract]`. Помечены могут быть только методы – свойства или поля пометить нельзя.

```
using System.ServiceModel;

namespace ServiceInterface
{
    [ServiceContract]
    public interface ICustomer
    {
        [OperationContract]
        string GetCustomer(int customerId);


        [OperationContract]
        int GetCustomersCount();
    }
}
```

Сервисный контракт разместим в сборке `ServiceInterface.dll`. В дальнейшем эта сборка будет использоваться как сервером, так и клиентом.

2. Реализация сервисного контракта.

Следующим шагом будет создание класса, реализующего сервисный контракт на стороне сервера. Опишем класс `Customer` в консольном приложении. В дальнейшем это консольное приложение исполнит роль хостинга WCF-службы.

```
using ServiceInterface;

namespace Host

{

    public class Customer : ICustomer

    {

        public string GetCustomer(int customerId)

        {

            return "Customer " + customerId;

        }


        public int GetCustomersCount()

        {

            return 10;

        }

    }


    public class Program

    {

        private static void Main() { }

    }

}
```

3. Хостинг WCF-службы.

Организуем хостинг WCF-службы. Для этого нужно описать концевую точку службы (или несколько концевых точек) и запустить службу.

Описание концевой точки выполняется или программно, или декларативно (в файле конфигурации). При программном подходе задаётся адрес (можно использовать класс `System.Uri` или строку), объект класса-привязки, и используется класс `ServiceHost` для связывания всех элементов и запуска службы:

```
// Реализация метода Main() из предыдущего фрагмента кода

private static void Main()
```

```
{
    // 1. Создаем объект, описывающий адрес службы
    var uri = new Uri("net.tcp://localhost:8228/Customer");

    // 2. Создаем объект, описывающий привязку
    var binding = new NetTcpBinding();

    // 3. Создаём хостинг для службы
    var host = new ServiceHost(typeof(Customer));
    host.AddServiceEndpoint(typeof(ICustomer), binding, uri);

    // 4. Запускаем хостинг
    host.Open();
    Console.ReadLine();
}
```

Если применяется декларативная настройка концевой точки, то используется секция конфигурационного файла <system.serviceModel>. В подсекцию <services> помещается описание служб.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="Host.Customer">
        <endpoint address="net.tcp://localhost:8228/Customer"
                  binding="netTcpBinding"
                  contract="ServiceInterface.ICustomer" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

При декларативной настройке концевой точки достаточно создать в программе объект класса `ServiceHost` и запустить хостинг.

```
private static void Main()
{
    var host = new ServiceHost(typeof(Customer));
    host.Open();
    Console.ReadLine();
}
```

4. Построение клиентского приложения.

Перейдем к созданию клиентского приложения, использующего WCF-службу. В пространстве имён `System.ServiceModel` содержится абстрактный универсальный класс `ClientBase<T>`, который можно использовать как базовый класс для клиентского прокси. Параметр `T` задаёт тип клиентского канала.

```
public class CustomerClient : ClientBase<ICustomer>, ICustomer
{
    public CustomerClient() { }

    public CustomerClient(string endpointConfigurationName) :
            base(endpointConfigurationName) { }

    public CustomerClient(string endpointConfigurationName,
                          string remoteAddress) :
            base(endpointConfigurationName, remoteAddress) { }

    public CustomerClient(string endpointConfigurationName,
                          EndpointAddress remoteAddress) :
            base(endpointConfigurationName, remoteAddress) { }
    public CustomerClient(
                       System.ServiceModel.Channels.Binding
binding,
                          EndpointAddress remoteAddress) :
            base(binding, remoteAddress) { }
```

```
        public string GetCustomer(int customerId)
        {
            return base.Channel.GetCustomer(customerId);
        }

        public int GetCustomersCount()
        {
            return base.Channel.GetCustomersCount();
        }
    }
```

Обратите внимание на следующие аспекты. Чтобы определить тип канала, указан интерфейс `ICustomer` - сервисный контракт. Для удобства настройки канала класс `CustomerClient` определяет несколько конструкторов, просто вызывающих конструкторы базового класса. Класс `CustomerClient` реализует интерфейс `ICustomer`. В соответствующих методах вызовы делегируются внутреннему объекту `Channel`.

Объект прокси-класса будет нуждаться в дополнительной настройке перед использованием. Как и в случае WCF-службы, настройку можно выполнить или программно, или в файле конфигурации.

В следующем примере кода настройка выполняется программно:

```
    private static void Main(string[] args)
    {
        // 1. Создаем объект, описывающий адрес концевой точки
        var address =
                new
EndpointAddress("net.tcp://localhost:8228/Customer");

        // 2. Создаем объект, описывающий привязку
        var binding = new NetTcpBinding();

        // 3. Создаем прокси-класс
        var client = new CustomerClient(binding, address);
```

```
        // 4. Работаем с сервисом через прокси
        Console.WriteLine(client.GetCustomer(23););
    }
```

Если настройка выполняется в конфигурационном файле, то указываются адрес, привязка, контракт и, дополнительно, имя настраиваемой точки:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="Customer"
                address="net.tcp://localhost:8228/Customer"
                binding="netTcpBinding"
                contract="ServiceInterface.ICustomer" />
    </client>
  </system.serviceModel>
</configuration>
```

Имя точки используется при создании объекта прокси-класса.

```
var client = new CustomerClient("Customer");
```

Заметим, что обычно прокси-классы не создаются вручную - используется специальная утилита `svcutil.exe`. Данная утилита может создать прокси-класс, даже если клиенту не известен сервисный контракт.

6.4. ДОПОЛНИТЕЛЬНЫЕ АСПЕКТЫ ИСПОЛЬЗОВАНИЯ WCF

1. Хостинг WCF-служб в приложениях ASP.NET.

Чтобы поместить WCF-службу в веб-приложение ASP.NET, добавьте в состав сайта элемент `WCF service`. Сделав это для службы с именем `Service`, мы получим файлы `Service.svc`, `Service.svc.cs` и `IService.cs`. Последний файл – это сервисный контракт. Содержимое первых двух файлов представлено ниже.

Service.svc:

```
<%@ ServiceHost Language="C#" Debug="true"
Service="WebApp.Service" CodeBehind="Service.svc.cs" %>
```

Service.svc.cs:

```
namespace WebApp
{
    public class Service : IService
```

```
    {
        public void DoWork() { }
    }
}
```

Два файла просто реализуют подход Code Behind. При необходимости программный код можно разместить непосредственно в файле `Service.svc` сразу после директивы `@ServiceHost`.

Как и для веб-служб, можно тестировать WCF-службу из браузера. Для создания клиентского прокси используется команда Visual Studio `Add Service Reference` или утилита `svcutil.exe` в форме:

```
svcutil.exe http://localhost:2349/Service.svc?wsdl
```

2. Контракты.

Атрибуты `[ServiceContract]` и `[OperationContract]` обладают набором свойств, позволяющих более «тонко» настроить сервисный контракт.

Таблица 4

Свойства атрибута `[ServiceContract]`

Имя	Описание
`ConfigurationName`	Определяет имя службы в конфигурационном файле. По умолчанию используется полное имя класса, реализующего службу
`CallbackContract`	Когда служба используется для дуплексного обмена сообщениями, это свойство определяет контракт, реализованный клиентом
`Name` и `Namespace`	Задают имя и пространство имён для элемента `<portType>` в документе WSDL. `Namespace` рекомендуется указывать
`SessionMode`	Позволяет организовать в WCF-службе поддержку состояния сеанса. Возможные значения: `Allowed`, `NotAllowed` и `Required`; все они определены в перечислении `SessionMode`
`ProtectionLevel`	Определяет, должна ли привязка поддерживать защиту коммуникаций. Возможные значения определены перечислением `ProtectionLevel`: `None`, `Sign`, `EncryptAndSign`

Таблица 5

Свойства атрибута `[OperationContract]`

Имя	Описание

`Action`	WCF использует Action из запроса SOAP для отображения его на соответствующий метод. По умолчанию это комбинация из пространства имен XML контракта, имени контракта и имени операции. Вы можете переопределить значение Action, специфицируя свойство `Action`
`ReplyAction`	Устанавливает SOAP-имя Action ответного сообщения
`AsyncPattern`	Если операция реализуется с использованием асинхронного шаблона, установите свойство в `true`
`IsInitiating` `IsTerminating`	Если контракт состоит из последовательности операций, то инициирующей операции назначается свойство `IsInitiating`, а последней операции последовательности требуется свойство `IsTerminating`. Инициирующая операция запускает новый сеанс; а в операции завершения сервер закрывает сеанс
`IsOneWay`	Если `IsOneWay = true`, клиент не ожидает ответного сообщения. Инициатор однонаправленного сообщения не имеет прямого способа обнаружить сбой после отправки своего запроса
`Name`	Имя операции по умолчанию - это имя метода, которому назначена операция контрактом. Вы можете изменить имя операции, применив свойство `Name`
`ProtectionLevel`	С помощью свойства `ProtectionLevel` вы определяете, должно ли сообщение быть снабжено подписью или зашифровано

Контракт данных – это класс или структура со специальными атрибутами, устанавливающими соответствие между элементами типа и элементами XML документа. Контракты данных следует применять в том случае, если методы WCF-службы получают или возвращают сложные типы данных.

Чтобы превратить тип в контракт данных, требуется применить к типу атрибут `[DataContract]`, а к полям или свойствам, подлежащим сериализации, – атрибут `[DataMember]`.

3. Поведения.

Класс, реализующий WCF-службу, может быть помечен атрибутом `[ServiceBehavior]`. Этот атрибут используется для описания *поведения* службы.

Важные свойства поведения - `InstanceContextMode` и `ConcurrencyMode`. Их значениями являются элементы одноимённых перечислений. Первое свойство управляет режимом создания объектов службы. Второе – режимом параллелизма при обработке запросов клиентов. Сочетания значений свойств описаны в таблице

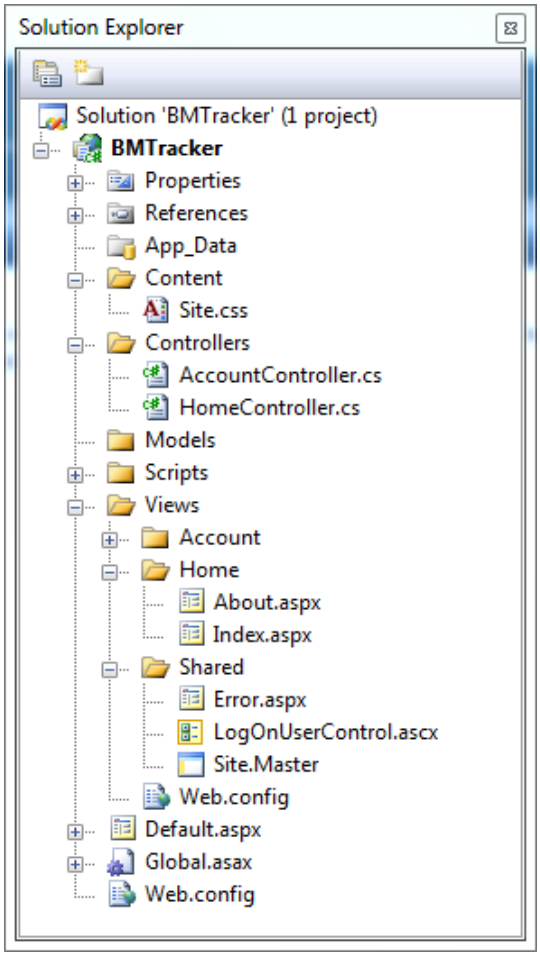
Сочетание поведений InstanceContextMode и ConcurrencyMode

		InstanceContextMode		
		Single	PerCall	PerSession
ConcurrencyMode	Single (по умолч.)	Для обработки всех запросов создаётся один объект и один поток. Пока текущий запрос обрабатывается, следующие ставятся в очередь	При каждом обращении создается новый объект. Режим параллелизма не имеет значения	На каждый сеанс связи с клиентом создается один объект и один поток. Асинхронные запросы клиента ставятся в очередь
	Reentrant	Для обработки всех запросов создаётся один объект и один поток. Пока текущий запрос обрабатывается, следующие ставятся в очередь		На каждый сеанс связи с клиентом создается один объект и один поток. Этот поток может покинуть метод, сделать что-то, а потом вернутся (асинхронное кодирование на сервере)
	Multiple	Создается один объект, но с ним могут параллельно работать несколько потоков (клиентов). Нужно позаботиться о защите членов класса с помощью механизмов синхронизации		На каждый сеанс связи с клиентом создается один объект, с которым могут параллельно работать несколько потоков (асинхронные операции). Нужно позаботиться о защите элементов (синхронизация)

Структура приложения ASP.NET MVC

Чтобы начать работу с ASP.NET MVC, следует скачать этот каркас (например, с http://www.asp.net/mvc) и установить его. На компьютере необходимо наличие .NET Framework 3.5 (желательно .NET Framework 3.5 SP1).

После установки ASP.NET MVC в Visual Studio станут доступны шаблоны проектов MVC (*File/New/Project/Web/ASP.NET MVC Web Application*). Новый проект MVC обладает определенной структурой.



Основные элементы проекта ASP.NET MVC

Имя папки или файла	Предполагаемое назначение	Примечания
`/App_Data`	Стандартная папка ASP.NET для размещения файлов с данными приложения (`*.xml`, `*.mdf`, `*.mdb`)	
`/bin`	Стандартная папка ASP.NET для скомпилированных сборок	Сборки, используемые в приложении, могут размещаться в GAC
`/Content`	Папка для статических файлов, связанных с сайтом (изображения, файлы CSS)	Статический контент может располагаться в любом каталоге
`/Controllers`	Папка, используемая по умолчанию для MVC-контроллеров	
`/Models`	Здесь могут размещаться классы, представляющие модель приложения	Часто для описания модели создаётся отдельный проект

/Scripts	Папка для клиентских файлов JavaScript	По умолчанию содержит файлы для поддержки ASP.NET AJAX и файлы библиотеки jQuery
/Views	Здесь содержатся представления (файлы *.aspx) и частичные представления (файлы *.ascx)	По конвенции именования, представления для контроллера XyzController помещаются в папку /Views/Xyz/. Представление для метода DoAction() должно называться DoAction.aspx
/Views/Shared	Здесь содержатся шаблоны представлений (обычно это эталонные страницы *.master) или общие представления	Если MVC не находит представление /Views/Xyz/DoAction.aspx (*.ascx), то производится поиск /Views/Shared/DoAction.aspx
/Views/web.config	Конфигурационный файл, запрещающий прямое обращение к файлам представлений	
/default.aspx	Этот файл не используется в ASP.NET MVC, но нужен для совместимости с IIS 6	
/global.asax	В этом фале происходит регистрация маршрутов вашего MVC-приложения	
/web.config	Конфигурационный файл, по умолчанию содержит настройки инфраструктуры ASP.NET MVC	