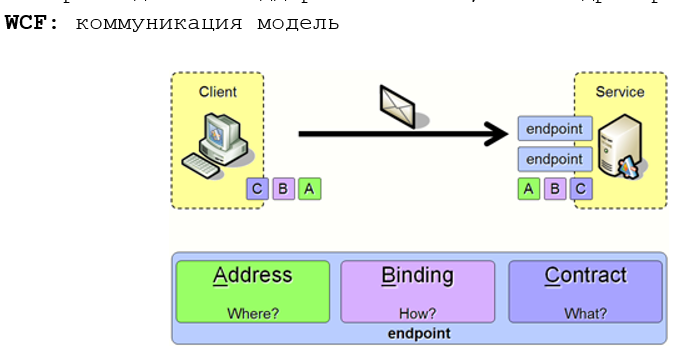
1. **WCF:** определение WCF-сервиса, коммуникационная модель, WCF-контракты, WCF-хостинг, конечные точки, стандартные привязки, основные отличия от ASMX-сервисов, поведение и безопасность WCF-сервиса, порядок разработки WCF/RPC и WCF/REST-сервисов, разработка WCF-сервиса с несколькими конечными точками.

**WCF**: Windows Communication Foundation; технология основанная на .NET FRAMEWORK для разработки приложений SOA-архитектуры, первая версия 2006 (Indigo).

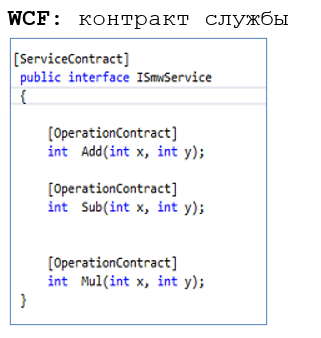
**WCF**: основные принципы

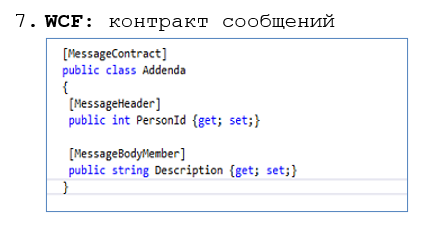
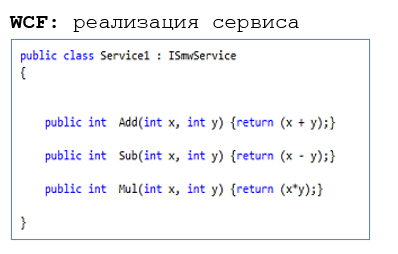
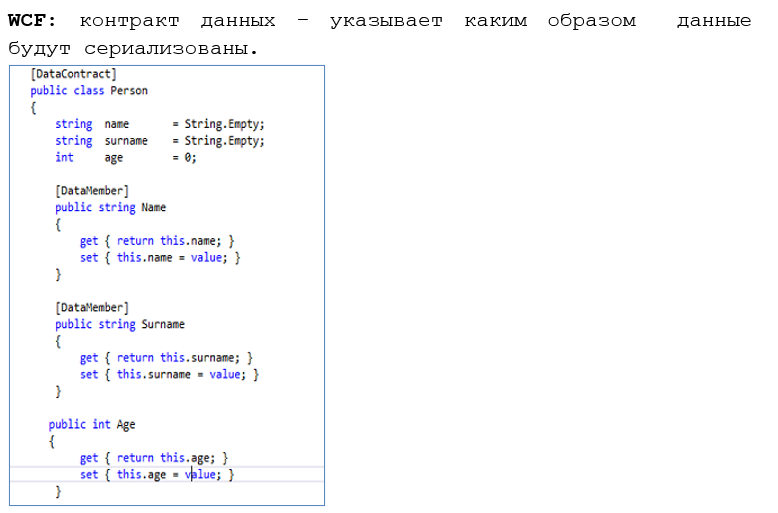
* разработка сервиса должна быть простой и иметь способность к расширению его функциональных возможностей;
* один API для всех коммуникационных протоколов;
* сервис должен быть интероперабельным (функционировать по отрытым телекоммуникационным стандартам);
* сервис должен поддерживать стандарты WS-\*;
* сервис должен поддерживать REST, RPC и др. архитектуры;



**WCF:** контракты

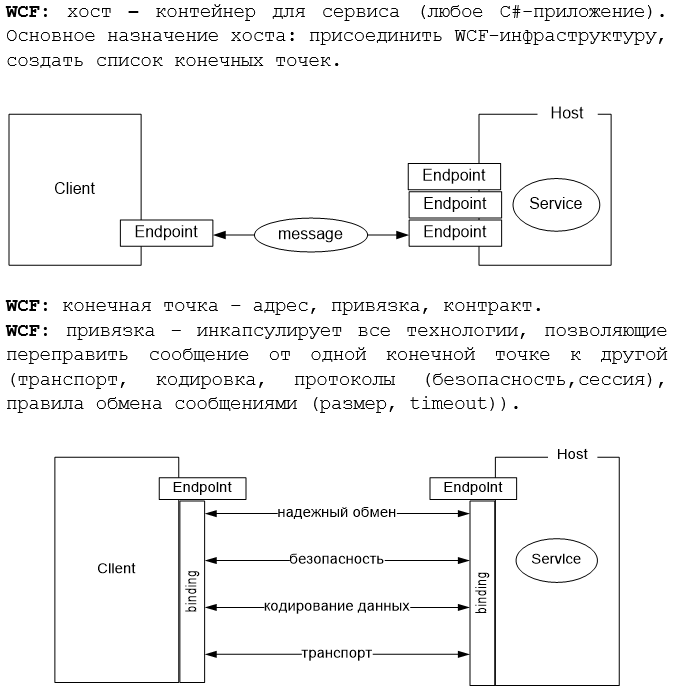
* контракт службы;
* контракт данных;
* контракт сообщений.



****

Контракт данных описывает структуру данных, которые будут сериализоваться и передаваться между клиентом и службой. Это основной механизм обмена данными, который используется для работы с объектами высокого уровня (POCO-классы).

Контракт сообщений используется для более низкоуровневого контроля над структурой SOAP-сообщения. Он позволяет задавать, как данные будут упакованы в сообщение, включая заголовки, тело, и атрибуты.



**WCF:** привязка: безопасность:WS-ReliableMessaging;

**WCF:** привязка: безопасность:WS-Security

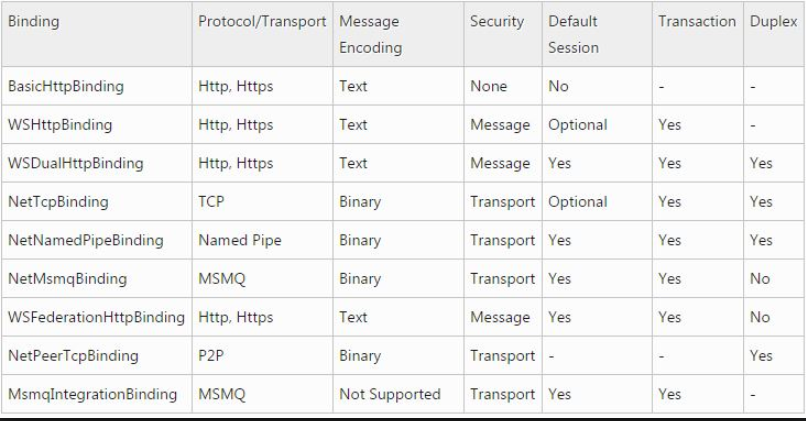
**WCF:** привязка: кодирование:

* текст (ASCII, UTF-8, UTF-16);
* двоичная (проприетарные алгоритмы);
* МТОМ.

**WCF:** привязка: транспорт:

* HTTP;
* TCP;
* IPC Named Pipes;
* MSMQ;
* Custom.

**WCF:** привязка: стандартные:



Основные отличия от ASMX-сервисов:

1. Протоколы: ASMX-сервисы используют протоколы SOAP и HTTP, в то время как WCF-сервисы могут использовать различные протоколы, такие как SOAP, HTTP, TCP, и т. д.

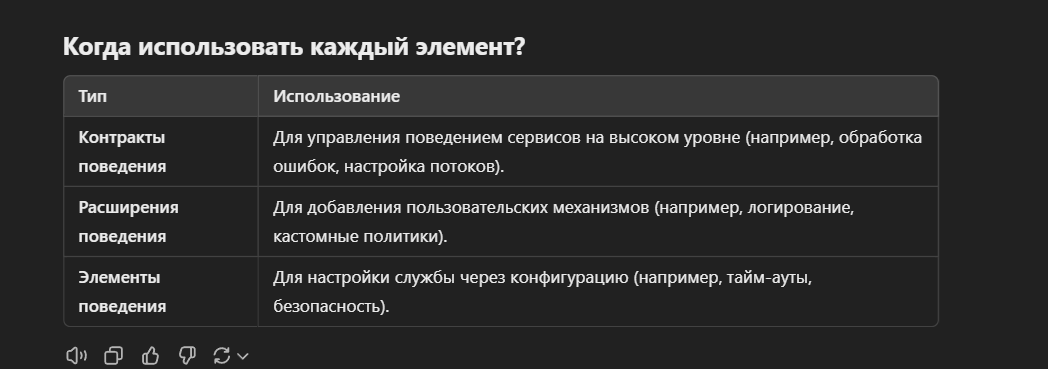
2. Гибкость настройки: WCF предоставляет большую гибкость настройки и конфигурирования, чем ASMX. Например, в WCF можно настроить различные протоколы, форматы сообщений, безопасность, маршрутизацию и т. д.

3. Поддержка различных стандартов: WCF поддерживает различные стандарты, такие как WS-\*, WS-Security, WS-ReliableMessaging и другие, что делает его более гибким для разработки сложных интеграционных сценариев.

Поведение и безопасность WCF-сервиса:

WCF позволяет разработчикам настроить поведение своих сервисов с использованием таких элементов, как контракты поведения (behavior contracts), расширения поведения (behavior extensions) и элементы поведения (behavior elements). Поведение может включать в себя различные настройки, такие как протоколы, сериализация, механизмы безопасности и другие аспекты работы сервиса.

В отношении безопасности, WCF предоставляет различные возможности для обеспечения безопасности взаимодействия между клиентом и сервисом. Это включает аутентификацию, авторизацию, шифрование, подпись сообщений и другие механизмы безопасности. Разработчик может выбрать подходящие механизмы безопасности и настроить их в соответствии с требованиями своего приложения.



Порядок разработки WCF/RPC и WCF/REST-сервисов:

При разработке WCF/RPC-сервисов, разработчик определяет контракт WCF-сервиса с использованием атрибутов, конфигурирует протоколы обмена сообщениями (например, HTTP или TCP) и настраивает поведения и безопасность. Затем реализуется логика операций сервиса.

При разработке WCF/REST-сервисов, разработчик определяет контракт WCF-сервиса с использованием атрибута [WebGet] или [WebInvoke], который указывает на REST-операции, и настраивает протоколы и поведения для работы с REST-запросами. Затем реализуется логика операций сервиса.

Разработка WCF-сервиса с несколькими конечными точками:

WCF позволяет определить несколько конечных точек (endpoints) для одного и того же сервиса. Каждая конечная точка может использовать разный протокол, формат сообщений, поведение и безопасность. Это позволяет сервису предоставлять различные интерфейсы и протоколы для разных клиентов. Для этого используется конфигурация WCF-сервиса, где задаются различные конечные точки и их свойства.

Архитектура WCF

****

### 1. **Приложение**

Это верхний уровень, где находится ваше приложение.  
**Задача**: Определяет, какие данные и сообщения отправлять и как вести себя при взаимодействии с другими приложениями.

### 2. **Контракты**

Контракты задают правила взаимодействия.

* **Контракт данных**: Определяет структуру данных, которые передаются.
* **Контракт сообщения**: Определяет, как выглядит сообщение (заголовки, тело).
* **Контракт службы**: Определяет, какие операции (методы) доступны.
* **Политика и привязка**: Настраивает способы доставки данных (например, безопасность, транспорт).

### 3. **Среда выполнения службы**

Это правила, которые определяют, как службы работают.

* **Поведение регулирования**: Как ограничить доступ или управлять нагрузкой.
* **Поведение при ошибке**: Что делать, если произошла ошибка.
* **Поведение метаданных**: Определяет, какую информацию о службе можно показать клиенту.
* **Поведение экземпляра**: Определяет, как создаются объекты (например, один на клиента или на запрос).
* **Проверка сообщений**: Убедиться, что сообщения корректны и безопасны.
* **Поведение транзакций**: Управляет согласованностью операций.
* **Поведение диспетчера**: Управляет обработкой запросов.
* **Поведение параллелизма**: Определяет, сколько запросов можно обрабатывать одновременно.
* **Фильтрация параметров**: Проверяет входные параметры запросов.

### 4. **Обмен сообщениями**

Это каналы, через которые проходят данные.

* **Канал WS Security**: Для безопасного обмена данными.
* **Канал WS Reliable Messaging**: Для надёжной доставки сообщений.
* **Кодировки**: Определяют, как данные преобразуются (например, текст, бинарный формат).
* **Канал HTTP**: Используется для передачи по веб-протоколу.
* **Канал TCP**: Для быстрой передачи данных внутри сети.
* **Канал потока транзакций**: Управляет транзакциями между приложениями.
* **Именованный канал**: Канал, работающий внутри одной машины.
* **Канал MSMQ**: Для передачи сообщений через очереди.

### 5. **Активация и размещение**

Определяет, где и как запускается ваша служба.

* **Служба активации Windows**: Автоматически активирует службу.
* **.EXE**: Приложение запускается как отдельный процесс.
* **Службы Windows**: Запускается как служба Windows.
* **COM+**: Использует компоненты COM для активации.

Структура address, binding

### **1.** Binding **(Привязка)**

**Binding** определяет транспортный протокол (например, HTTP, TCP, Named Pipes), формат сообщений (текст, двоичные данные) и другие параметры связи. Он определяет **"как"** клиент и сервис будут взаимодействовать.

#### Основные элементы binding:

1. **Протокол**: HTTP, TCP, Named Pipes, MSMQ.
2. **Формат сообщения**: SOAP 1.1, SOAP 1.2, JSON, двоичный формат.
3. **Настройки безопасности**: Тип аутентификации, шифрование, защита сообщений.
4. **Тайм-ауты**: Тайм-ауты для соединения, отправки и получения данных.

### **Примеры WCF-привязок:**

1. **BasicHttpBinding**:
   * Простой SOAP-протокол, совместимый с другими платформами.
   * Используется для взаимодействия с сервисами, поддерживающими HTTP.
2. **NetTcpBinding**:
   * Высокая производительность, двоичный формат сообщений.
   * Используется для связи между WCF-приложениями.
3. **WSHttpBinding**:
   * Поддерживает дополнительные функции безопасности и надежности.
   * Используется в корпоративных приложениях.
4. **CustomBinding**:
   * Для создания привязок с индивидуальными требованиями.

#### Пример binding в конфигурации WCF (app.config/web.config):

xml

Копировать код

<bindings>

<!-- Пример BasicHttpBinding -->

<basicHttpBinding>

<binding name="BasicHttpBindingConfig" maxBufferSize="2147483647" maxReceivedMessageSize="2147483647">

<security mode="None" />

</binding>

</basicHttpBinding>

<!-- Пример NetTcpBinding -->

<netTcpBinding>

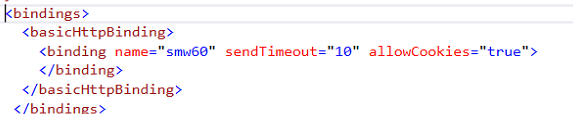
<binding name="NetTcpBindingConfig" maxBufferPoolSize="524288" maxReceivedMessageSize="65536">

<security mode="Transport" />

</binding>

</netTcpBinding>

</bindings>

****

### **2.** Address **(Адрес)**

**Address** указывает местоположение сервиса (URL или URI), чтобы клиент мог подключиться к нему. Он определяет **"где"** находится сервис.

#### Формат адреса:

markdown

Копировать код

[Протокол]://[Имя\_хоста]:[Порт]/[Путь]

##### Примеры:

1. **HTTP**:

arduino

Копировать код

http://localhost:8734/Design\_Time\_Addresses/MyService

1. **TCP**:

arduino

Копировать код

net.tcp://localhost:808/MyTcpService

1. **Named Pipes**:

arduino

Копировать код

net.pipe://localhost/MyPipeService

1. **MSMQ**:

arduino

Копировать код

net.msmq://localhost/private/MyQueue

#### Пример address в конфигурации:

xml

Копировать код

<client>

<endpoint address="http://localhost:8734/MyService"

binding="basicHttpBinding"

bindingConfiguration="BasicHttpBindingConfig"

contract="IMyService" />

</client>

### **3. Связь** binding **и** address **в WCF**

Каждая конечная точка (endpoint) WCF состоит из:

1. **Address (где находится сервис)**.
2. **Binding (как клиент взаимодействует с сервисом)**.
3. **Contract (какие операции предоставляет сервис)**.

#### Пример полной конфигурации:

xml

Копировать код

<system.serviceModel>

<services>

<service name="MyNamespace.MyService">

<endpoint address="http://localhost:8734/MyService"

binding="basicHttpBinding"

contract="MyNamespace.IMyService" />

<endpoint address="net.tcp://localhost:808/MyTcpService"

binding="netTcpBinding"

contract="MyNamespace.IMyService" />

</service>

</services>

</system.serviceModel>



### **4. Практический пример**

**Серверная часть:**

csharp

Копировать код

ServiceHost host = new ServiceHost(typeof(MyService));

// HTTP endpoint

host.AddServiceEndpoint(

typeof(IMyService),

new BasicHttpBinding(),

"http://localhost:8734/MyService"

);

// TCP endpoint

host.AddServiceEndpoint(

typeof(IMyService),

new NetTcpBinding(),

"net.tcp://localhost:808/MyTcpService"

);

host.Open();

Console.WriteLine("Service is running...");

**Клиентская часть:**

csharp

Копировать код

ChannelFactory<IMyService> factory = new ChannelFactory<IMyService>(

new BasicHttpBinding(),

new EndpointAddress("http://localhost:8734/MyService"));

IMyService client = factory.CreateChannel();

Console.WriteLine(client.MyOperation());

### **Итог**

* **Binding** определяет способ связи (протокол, формат сообщений, безопасность).
* **Address** указывает место, где сервис доступен.
* Вместе с **Contract** они образуют конечную точку (endpoint) в WCF.

1. **Протокол**:
   * Указывает способ взаимодействия с сервисом.
   * Например, net.tcp для TCP или http для HTTP.
2. **Имя хоста**:
   * Определяет сервер, на котором размещён сервис.
   * Здесь используется localhost, что означает локальную машину.
3. **Порт**:
   * Число, указывающее конкретный порт, на котором слушает сервис.
   * Например, 8734 для TCP и 8733 для HTTP.
4. **Виртуальный путь**:
   * Указывает путь к сервису относительно хоста.
   * Например, /Design\_Time\_Addresses/lab5.WCF/SiplexService/.

**Пример анализа:**

* net.tcp://localhost:8734/Design\_Time\_Addresses/lab5.WCF/SiplexService/
  + Протокол: net.tcp
  + Хост: localhost
  + Порт: 8734
  + Путь: /Design\_Time\_Addresses/lab5.WCF/SiplexService/
* http://localhost:8733/Design\_Time\_Addresses/lab5.WCF/SiplexService/
  + Протокол: http
  + Хост: localhost
  + Порт: 8733
  + Путь: /Design\_Time\_Addresses/lab5.WCF/SiplexService/

**Поведение WCF-сервиса** (Service Behaviors) — это совокупность настроек и конфигураций, которые определяют **способ работы** и **характеристики** сервиса Windows Communication Foundation (WCF). Эти поведения влияют на **время выполнения**, включая управление экземплярами, согласование транзакций, безопасность и мониторинг.

## **Основные виды поведения WCF-сервиса**

1. **Поведение сервиса (ServiceBehaviorAttribute)**  
   Управляет поведением на уровне самого сервиса.
2. **Поведение конечной точки (EndpointBehavior)**  
   Управляет взаимодействием между клиентом и конечной точкой.
3. **Поведение операций (OperationBehavior)**  
   Определяет поведение для конкретных операций сервиса.
4. **Поведение хостинга**  
   Связано с настройками хостинга сервиса.

## **ServiceBehaviorAttribute (атрибут поведения сервиса)**

Этот атрибут применяется к классу сервиса и определяет его **глобальное поведение**. Он предоставляет настройки, такие как жизненный цикл экземпляров, параллелизм, контроль транзакций и др.

### **Основные свойства** ServiceBehavior

1. **InstanceContextMode**  
   Определяет жизненный цикл экземпляров сервиса:
   * PerCall — новый экземпляр сервиса создается на каждый вызов клиента.
   * PerSession — создается один экземпляр на всю сессию клиента.
   * Single — один экземпляр сервиса обрабатывает все вызовы (singleton).  
     Пример:

csharp

Копировать код

[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]

public class MyService : IMyService { ... }

1. **ConcurrencyMode**  
   Определяет модель параллельного доступа:
   * Single — сервис обрабатывает вызовы последовательно.
   * Reentrant — позволяет повторные вызовы при активной операции.
   * Multiple — разрешает параллельную обработку нескольких вызовов.  
     Пример:

csharp

Копировать код

[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple)]

public class MyService : IMyService { ... }

1. **IncludeExceptionDetailInFaults**  
   Управляет тем, будет ли отправляться подробная информация об исключениях клиенту. Полезно для отладки.  
   Пример:

csharp

Копировать код

[ServiceBehavior(IncludeExceptionDetailInFaults = true)]

public class MyService : IMyService { ... }

1. **UseSynchronizationContext**  
   Определяет, должен ли WCF использовать синхронизацию контекста сервиса.
2. **TransactionAutoCompleteOnSessionClose**  
   Указывает, следует ли автоматически завершать транзакцию при закрытии сессии.

## **EndpointBehavior (Поведение конечной точки)**

Поведение конечной точки настраивается через интерфейс IEndpointBehavior. Оно позволяет расширить функциональность конечной точки сервиса, например:

* Добавление пользовательских обработчиков сообщений.
* Изменение параметров безопасности.
* Включение логирования и диагностики.

Пример:

csharp

Копировать код

public class CustomEndpointBehavior : IEndpointBehavior

{

public void ApplyDispatchBehavior(ServiceEndpoint endpoint, EndpointDispatcher dispatcher)

{

// Настройка обработчика сообщений

dispatcher.DispatchRuntime.MessageInspectors.Add(new CustomMessageInspector());

}

public void AddBindingParameters(ServiceEndpoint endpoint, BindingParameterCollection bindingParameters) { }

public void ApplyClientBehavior(ServiceEndpoint endpoint, ClientRuntime clientRuntime) { }

public void Validate(ServiceEndpoint endpoint) { }

}

## **OperationBehavior (Поведение операций)**

Определяет поведение на уровне отдельных операций сервиса. Используется атрибут OperationBehaviorAttribute.

### **Основные свойства:**

1. **TransactionScopeRequired** — требует транзакции для выполнения операции.
2. **Impersonation** — определяет, как клиентские учетные данные должны использоваться в операции.
3. **AutoDisposeParameters** — автоматически вызывает Dispose() для параметров.

Пример:

csharp

Копировать код

public class MyService : IMyService

{

[OperationBehavior(TransactionScopeRequired = true)]

public void DoWork()

{

// Операция будет работать в рамках транзакции

}

}

## **Поведение хостинга**

Это конфигурации, которые определяются при запуске WCF-сервиса в хост-приложении (например, консоль, IIS или Windows Service).  
Примеры:

* Установка базового адреса (baseAddress).
* Включение ServiceDebugBehavior для детальной отладки.
* Включение ServiceMetadataBehavior для публикации метаданных (WSDL).

Пример:

csharp

Копировать код

ServiceHost host = new ServiceHost(typeof(MyService), new Uri("http://localhost:8000/MyService"));

// Включаем публикацию метаданных

ServiceMetadataBehavior smb = new ServiceMetadataBehavior

{

HttpGetEnabled = true

};

host.Description.Behaviors.Add(smb);

// Включаем подробные исключения

ServiceDebugBehavior sdb = host.Description.Behaviors.Find<ServiceDebugBehavior>();

sdb.IncludeExceptionDetailInFaults = true;

host.Open();

Console.WriteLine("Сервис запущен...");

## **Итог**

Поведение WCF-сервиса (Service Behaviors) позволяет настроить:

* **Жизненный цикл экземпляров** (InstanceContextMode).
* **Параллелизм вызовов** (ConcurrencyMode).
* **Логирование и диагностику** (через IEndpointBehavior).
* **Транзакции и управление операциями** (OperationBehavior).
* **Хостинг и метаданные** (ServiceMetadataBehavior).

Эти настройки гибко управляют работой сервиса и его взаимодействием с клиентами.