

Из цикла лекций «Internet-технологии разработки приложений» для студентов 4-го курса  
кафедры Компьютерных технологий физического факультета Донецкого национального университета  
проф. В. К. Толстых

# WSF-службы

## Windows Communication Foundation

предназначены для создания сервис-ориентированных,  
распределенных приложений.

# История технологий программирования для борьбы с повторением кода и для структурирования программ

1. Функции
2. Объектно-ориентированное программирование
3. Компонентно-ориентированное программирование (COM-компоненты - .lib, .dll)
4. Сервис-ориентированное программирование (SOA, Service Oriented Architecture) — модульный подход к разработке программного обеспечения, основанный на использовании сервисов (служб) со стандартизированными интерфейсами. Сервис-ориентированное приложение представляет собой результат агрегирования служб в одно, логически завершенное, связанное приложение.

Сервисы являются естественным результатом эволюции компонентов, как компоненты были естественным результатом эволюции объектов. Клиентом сервиса может быть всё, что угодно – класс Windows Forms, страница ASP.NET, другой сервис. В WCF все сообщения передаются в формате SOAP.

WCF поддерживает следующие транспортные схемы (**Адреса**):

1. HTTP: `http://localhost:8001/MyService` (в глобальной сети)
2. TCP: `net.tcp://localhost:8002/MyService` (в лок. сети)
3. IPC (именованные каналы): `net.pipe://localhost/MyService` (на одном компьютере)
4. MSMQ (механизм очередей): `net.msmq://localhost/MyService`
5. Одноранговые сети: `net.p2p:` (например, узлы GRID)

# WAS: реализация не HTTP протоколов

Именно WAS (Windows process Activation Service) при IIS 7 поддерживает для WCF отличные от HTTP протоколы (net.tcp, net.pipe...). Он позволяет для не HTTP-запросов реализовать их обработку аналогично IIS: активировать WCF-сервисы по требованию, создавать для них пулы и запускать рабочие процессы, наблюдать за работоспособностью процесса, управлять приложениями уровня предприятия (TCP), обеспечивать быструю защиту от сбоев.

Web-служба IIS (**Svchost.exe**) сохраняет роль прослушивателя HTTP, но компоненты, ответственные за настройку и активацию процесса, были перенесены в WAS, которая имеет три части: диспетчер настройки, диспетчер обработки и интерфейс адаптера прослушивателя.

Диспетчер настройки считывает настройки приложения и пула приложений из файла **applicationhost.config**. Диспетчер обработки сопоставляет пулы приложений существующим рабочим процессам и ответственен за запуск новых экземпляров **w3wp.exe** для размещения новых пулов приложений в ответ на запросы на активацию. Интерфейс адаптера прослушивателя используется WCF для передачи принятых запросов на активацию по протоколам, отличным от HTTP (а именно, TCP, именованные каналы и MSMQ).

# Сервисы, посредники и операции (методы)

Каждая служба WCF может содержать несколько независимых операций – методов, к которым может обращаться клиент. Клиент взаимодействует с операциями службы через своего посредника – экземпляр прокси.

Подключение к каждой службе происходит через своего посредника, более того, подключение по разным каналам, к разным точкам одной и той же службы организуется разными посредниками. Классы посредников (прокси-классы), создаются клиентом при предварительной настройке его взаимодействия со службой на основе метаданных службы, описанных в виде контрактов службы и операций.

Пример организации вызова операции `GetData()` службы `MyService` через посредника `MyProxi` на основе заранее созданного прокси-класса `MyServiceClient`:

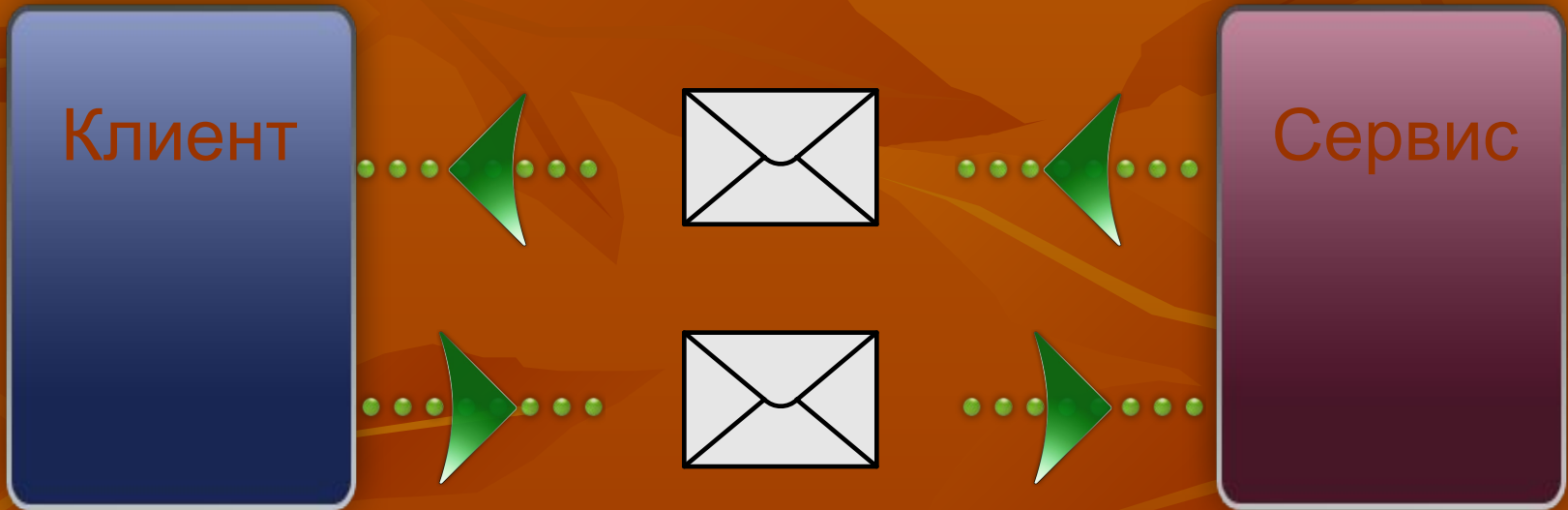
```
MyServiceClient MyProxi = new MyServiceClient();  
MyProxi.GetData();
```

WCF по-умолчанию поддерживает классический вызов методов (клиент выдаёт вызов, блокируется ожидая ответ с тайм-аутом 1 минуту и продолжает работу после ответа). Кроме того он поддерживает *односторонние операции* («вызвал и забыл» без возвращаемых сообщений), *операции обратного вызова* (для оповещения клиентов о событиях на стороне сервера) и *потокосые операции* (обработка данных во время их приёма).

WCF могут поддерживать *сеансы* между клиентом и определенным экземпляром службы, могут поддерживать *транзакции* и *очереди*, а также *параллельную обработку*.

Все настройки служб и операций осуществляются в их контрактах (`Contract`), а также в привязках (`Binding`) и поведении (`behaviors`) точек взаимодействия (`Endpoint`).

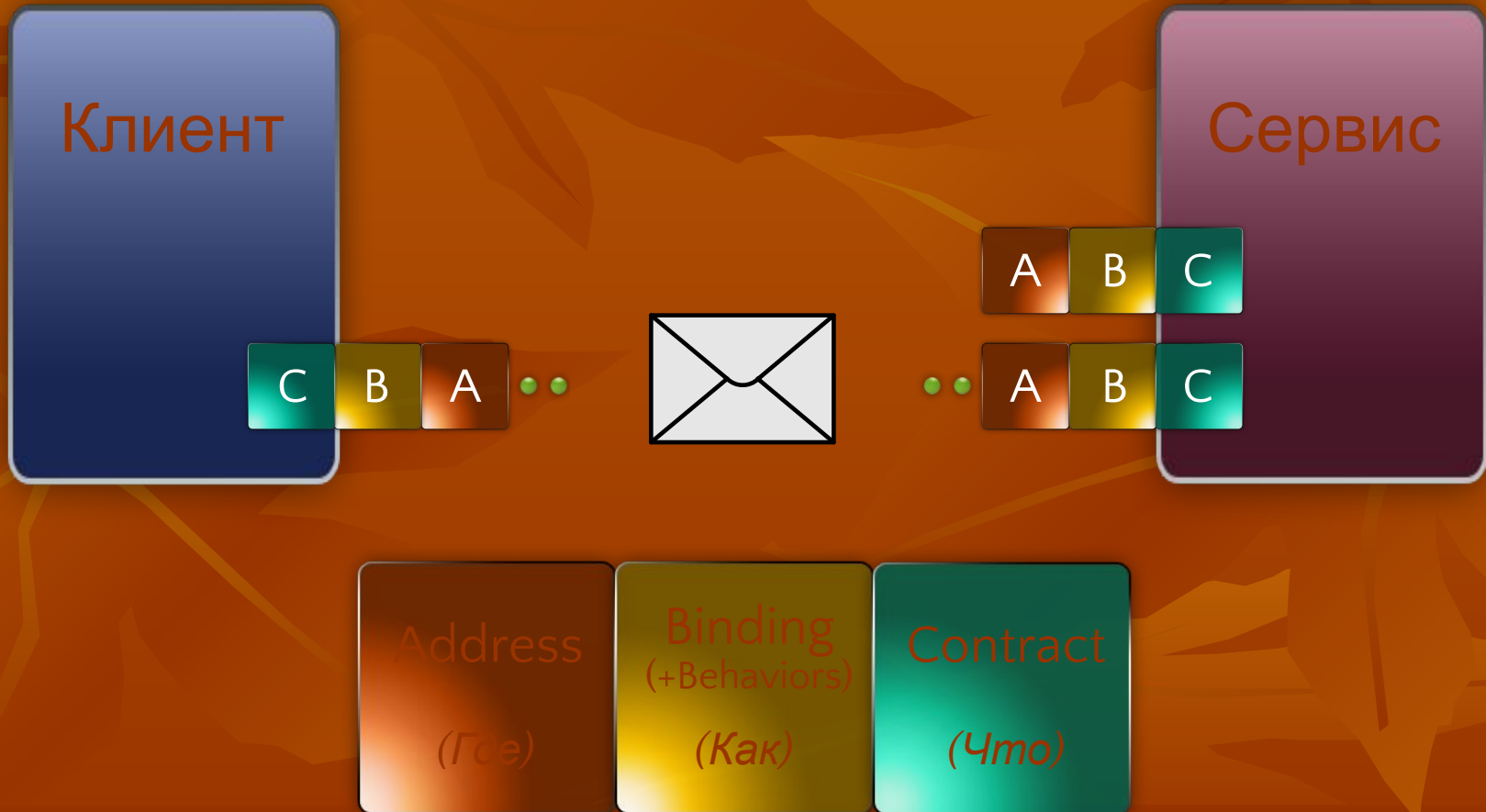
# Обмен сообщениями в SOAP-конвертах



# Конечные точки



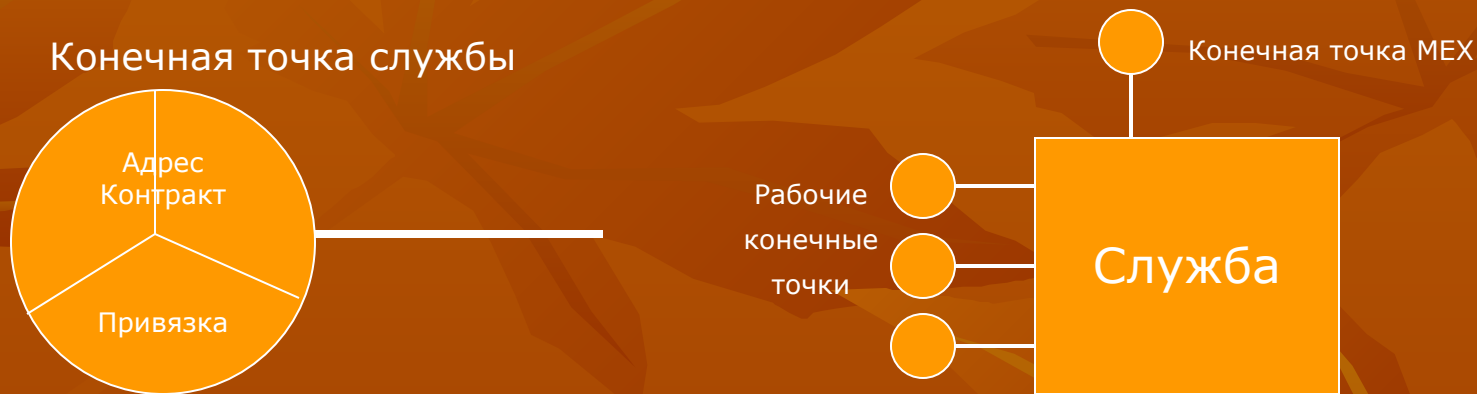
# Address, Binding, Contract





# Конечные точки

Каждая служба связывается с **адресом**, определяющим местоположение службы, с **привязкой**, определяющей способ взаимодействия со службой, и с **контрактом**, который указывает, что делает служба, её параметры, ошибки и сообщения.



Каждая служба должна предоставлять как минимум одну рабочую конечную точку, и каждая конечная точка должна иметь один контракт. Все конечные точки службы имеют уникальные адреса. Одна служба может предоставлять несколько конечных точек.

Конечная точка MEX предоставляет метаданные, описывающие функциональность WCF и способы взаимодействия с ней. Она используется на этапе проектирования клиента службы для настройки его прокси-класса.

Конечные точки могут использовать одинаковые или разные привязки, а также предоставлять одинаковые или разные контракты.

Разные конечные точки никак не связаны друг с другом, они не представлены в коде службы. Конечные точки настраиваются на административном уровне (через конфигурационный файл) или на программном уровне.



# Привязки

- фиксированный набор настроек, относящихся к транспортному протоколу, кодированию сообщений, коммуникационной схеме, надёжности, безопасности, распространению транзакций и совместимости.

Можно настраивать свойства стандартных привязок, можно писать собственные привязки. Служба публикует свои привязки в метаданных (в формате WSDL). Клиент должен использовать точно такие же параметры привязки, что и служба. Одна служба может поддерживать несколько привязок по разным адресам.

## Стандартные привязки

**BasicHttpBinding** (HTTP, HTTPS) – предоставляет WCF-клиентам доступ к старым Web-службам .asmx

**NetTcpBinding** (TCP) – для интрасетей, поддерживает надёжность, транзакции, безопасность, оптимизирована для взаимодействия WCF-WCF

**NetPeerTcpBinding** (P2P) – для одноранговых сетей типа GRID

**NetNamedPipeBinding** (IPC) – именованные каналы в пределах одного компьютера. Наиболее защищённые (не принимают вызовы от TCP), поддерживают все функции NetTcpBinding

**wsHttpBinding** (HTTP, HTTPS) – для интернет сетей с поддержкой надёжности, транзакций, безопасности

**wsDualHttpBinding** (HTTP, HTTPS) – в дополнение к предыдущей поддерживает двухстороннее взаимодействие между службой и клиентом (поддерживается второй HTTP-канал для обратного вызова от службы к клиенту)

**NetMsmqBinding** (MSMQ) – для поддержки очередей автономных вызовов в интрасетях

# Контракты

- стандартный, платформенно-независимый способ описания того, что делает данная служба.

Существуют четыре разновидности контрактов:

- 1. Контракты служб [ServiceContract]** описывают операции (методы), которые могут выполняться клиентом с помощью службы. Включает контракты необходимых операций [OperationContract] ;
- 2. Контракты данных [DataContract]** определяют, какие типы данных принимаются и передаются службой. При передаче объекта или структурного типа в параметре операции, в действительности, надо передать лишь его состояние, а принимающая сторона должна преобразовать его обратно к своему родному представлению. Это называется *маршаллинг по значению*. Он реализуется посредством *сериализации*, когда пользовательские типы переводятся из CLR представлений в XML содержимое SOAP-конвертов. При приёме параметров происходит *десериализация*, т.е. набор XML преобразуется в объект CLR и дальше передаётся для обработки;
- 3. Контракты ошибок [FaultContract]** определяют, какие исключения инициируются службой, как служба обрабатывает их и передаёт своим клиентам;
- 4. Контракты сообщений [MessageContract], [MessageHeader] [MessageBodyHeader]** позволяют службам напрямую взаимодействовать с сообщениями и моделировать структуру всего конверта SOAP.

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.Serialization;
5 using System.ServiceModel;
6 using System.Text;
7 // NOTE: If you change the interface name "IService" here, you must also update the reference to "IServ
8 [ServiceContract]
9 public interface IService
10 {
11     [OperationContract]
12     string GetData(int value);
13
14     [OperationContract]
15     CompositeType GetDataUsingDataContract(CompositeType composite);
16
17     // TODO: Add your service operations here
18 }
19 // Use a data contract as illustrated in the sample below to add composite types to service operations.
20 [DataContract]
21 public class CompositeType
22 {
23     bool boolValue = true;
24     string stringValue = "Hello ";
25     [DataMember]
26     public bool BoolValue
27     {
28         get { return boolValue; }
29         set { boolValue = value; }
30     }
31     [DataMember]
32     public string StringValue
33     {
34         get { return stringValue; }
35         set { stringValue = value; }
36     }
37 }
38

```

## Пример контрактов (интерфейс службы .svc из примера Visual Studio 2008)

### Контракт службы

Контракты методов (операций).  
Другие методы интерфейса без этого атрибута в  
контракт не включаются.

### Контракт данных

Этот атрибут означает необходимость  
осуществления *маршаллинга по значению* для  
этого типа данных

Этот атрибут означает необходимость  
*сериализации* указанных полей данных, чтобы  
они участвовали в маршаллинге по значению  
текущего типа данных

# Примеры контрактов

1. Классический вызов метода с ожиданием ответа поддерживается всеми привязками (кроме **NetPeerTcpBinding** и **NetMsmqBinding**):

```
[OperationContract(IsOneWay = false)]  
string MyMethod(out int n1, int n2);
```

Возвращаемые параметры должны стоять в начале списка параметров.

Режим двусторонней операции,  
включается по умолчанию  
(можно не указывать)

2. Односторонний вызов метода поддерживается всеми привязками:

```
[OperationContract(IsOneWay = true)]  
void MyMethod();
```

3. Двухсторонний (обратный) вызов поддерживается привязками **NetTcpBinding**, **NetNamedPipeBinding**, **wsDualHttpBinding**:

```
[ServiceContract(CallbackContract = typeof(ISomeBackContract)]
```

Обратный вызов должен специально организовываться и у клиента.

4. Поддержка сеанса в контракте для привязок TCP, IPC и WS :

```
[ServiceContract(SessionMode = SessionMode.Allowed)]  
и в поведении службы: [ServiceBehavior(InstanceContextMode =  
InstanceContextMode.PerSession)]
```

Принято по  
умолчанию  
(можно не  
указывать)

# Структура файла конфигурации служб - **Web.config**

**<system.serviceModel>** - раздел WCF

**<services>** - раздел настроек всех служб

**<service name="MyNamespace.MyService1">** Описание первой службы

**behaviorConfiguration="SrvBehavior">** - имя её поведения в **<behaviors>**

**<endpoint ...>** конечная рабочая точка 1

**<endpoint ...>** конечная рабочая точка 2

...

**</service>**

**<service name="MyNamespace.MyService2">** Описание второй службы

**<endpoint ...>** конечная рабочая точка 1

**<endpoint ...>** конечная рабочая точка 2

...

**</service>**

**</services>**

**<bindings>** - раздел конфигурации привязок (при необходимости)...

**</bindings>**

**<behaviors>** - раздел настроек поведения (доступ к метаданным, исключения...)

**</behaviors>** для различных служб и конечных точек

**</system.serviceModel>**



## Конфигурация конечных точек (адрес, привязка, контракт) для первой службы

```
<endpoint address="http://localhost:8000/MyService1/" - конечная рабочая точка 1  
binding="wsHttpBinding"  
contract="MyNamespace.IMyContract"  
name="MyPoint1"  
    bindingConfiguration="MyConfigNetTCP" – имя для конфигурации привязки в <bindings>  
    behaviorConfiguration="PointBehavior"> – имя для поведения точки в <behavior>
```

```
<endpoint address=net.tcp://localhost:8001/MyService1/ - конечная рабочая точка 2  
binding="netTcpBinding"  
contract="MyNamespace.IMyContract"  
name="MyPoint2">
```

# Обмен метаданными

Метаданные необходимы для создания прокси-класса у клиента через который он будет взаимодействовать со службой. Метаданные можно опубликовать двумя способами:

1. по протоколу HTTP-GET,
2. через конечную точку MEX

Оба варианта автоматически генерируются VS в файле конфигурации **.config**:

Доступ к метаданным через  
конечную **mex**-точку  
(относительно базового адреса)

```
<!-- Metadata Endpoints -->
```

```
<!-- This endpoint does not use a secure binding and should be secured or removed before deployment -->
```

```
<endpoint
```

```
address="mex"
```

```
binding="mexHttpBinding"
```

```
contract="IMetadataExchange" />
```



# Настройка поведения behaviors

Настройка поведения всех служб

```
<behaviors>
```

```
<serviceBehaviors>
```

```
<behavior name="SrvBehavior">
```

```
<!-- To avoid disclosing metadata information, set the value below to false and remove  
the metadata endpoint above before deployment -->
```

```
<serviceMetadata httpGetEnabled="true" />
```

```
<!-- To receive exception details in faults for debugging purposes, set the value below to true.
```

```
Set to false before deployment to avoid disclosing exception information -->
```

```
<serviceDebug includeExceptionDetailInFaults="false" />
```

```
<serviceThrottling maxConcurrentSessions="20" />
```

```
</behavior>
```

```
</serviceBehaviors>
```

Настройка поведения для службы с  
behaviorConfiguration="SrvBehavior">

HTTP-GET-доступ к метаданным.  
**true** - можно просмотреть  
метаинформацию в браузере

Не отправлять клиенту сообщения  
об исключениях, возникающих  
внутри методов

Настройка поведения конечных рабочих точек служб

```
<endpointBehaviors>
```

```
<behavior name="PointBehavior">
```

```
...
```

```
</behavior>
```

```
</endpointBehaviors>
```

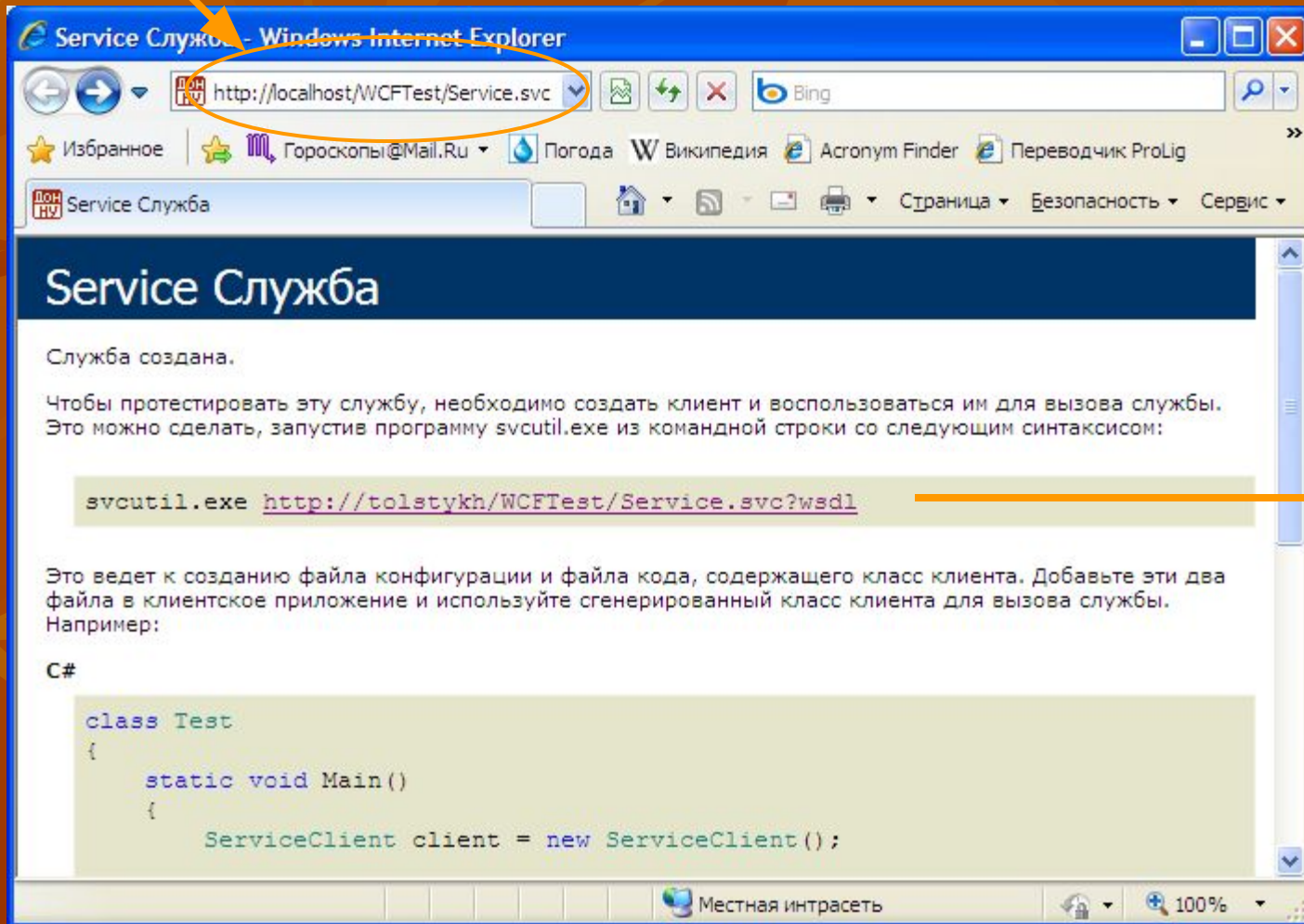
```
</behaviors>
```

Настройка поведения для конечной точки с  
behaviorConfiguration="PointBehavior"

Поддерживать до 20  
параллельных сеансов.  
По умолчанию - 10

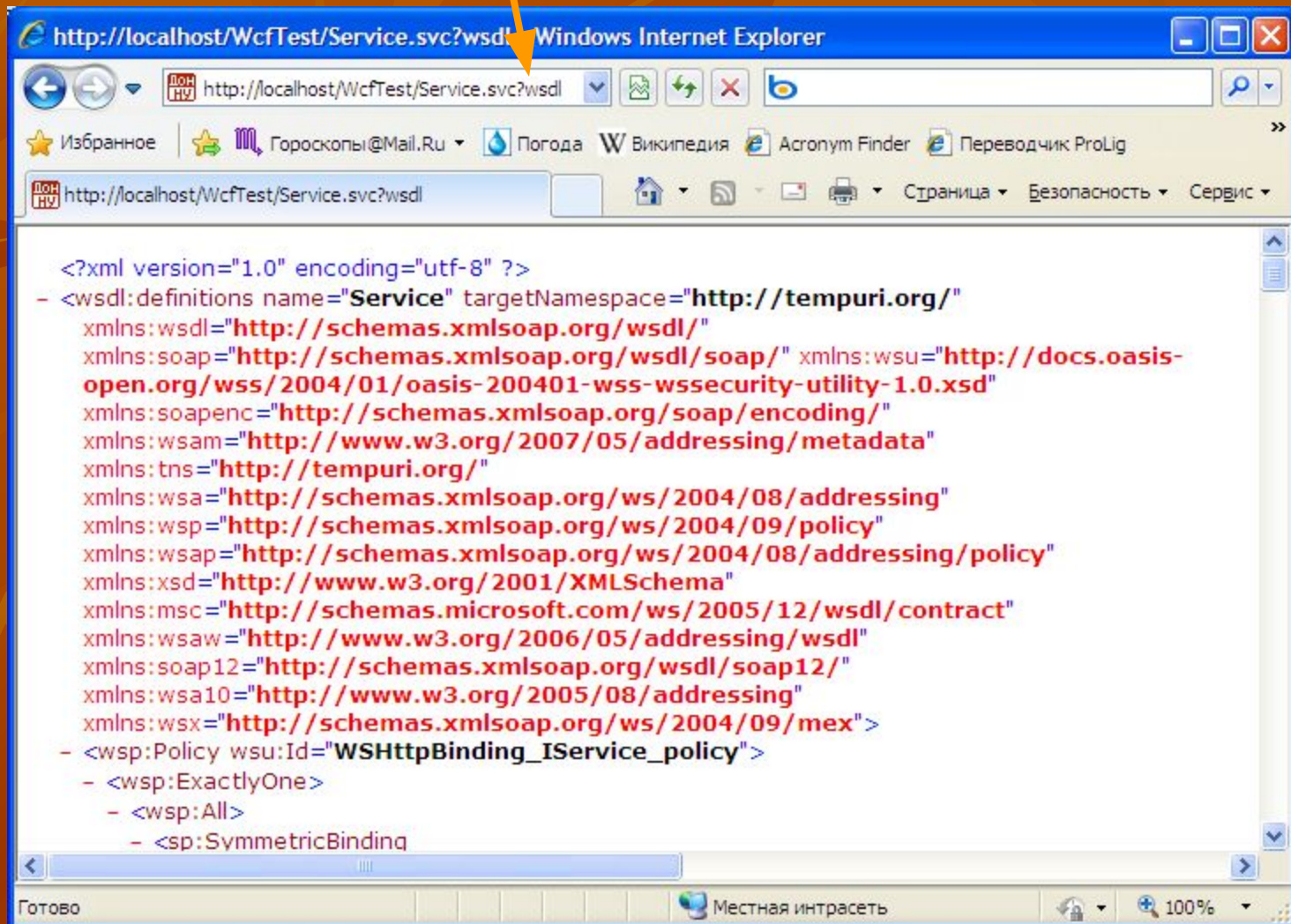
# Тестирование службы

(Проверка доступа к метаданным службы)



Это окно означает, что хостинг службы организован успешно, имеется **Get**-доступ к метаданным.

## Метаданные службы (GET-доступ – ?wsdl)



Windows Internet Explorer

http://localhost/WcfTest/Service.svc?wsdl

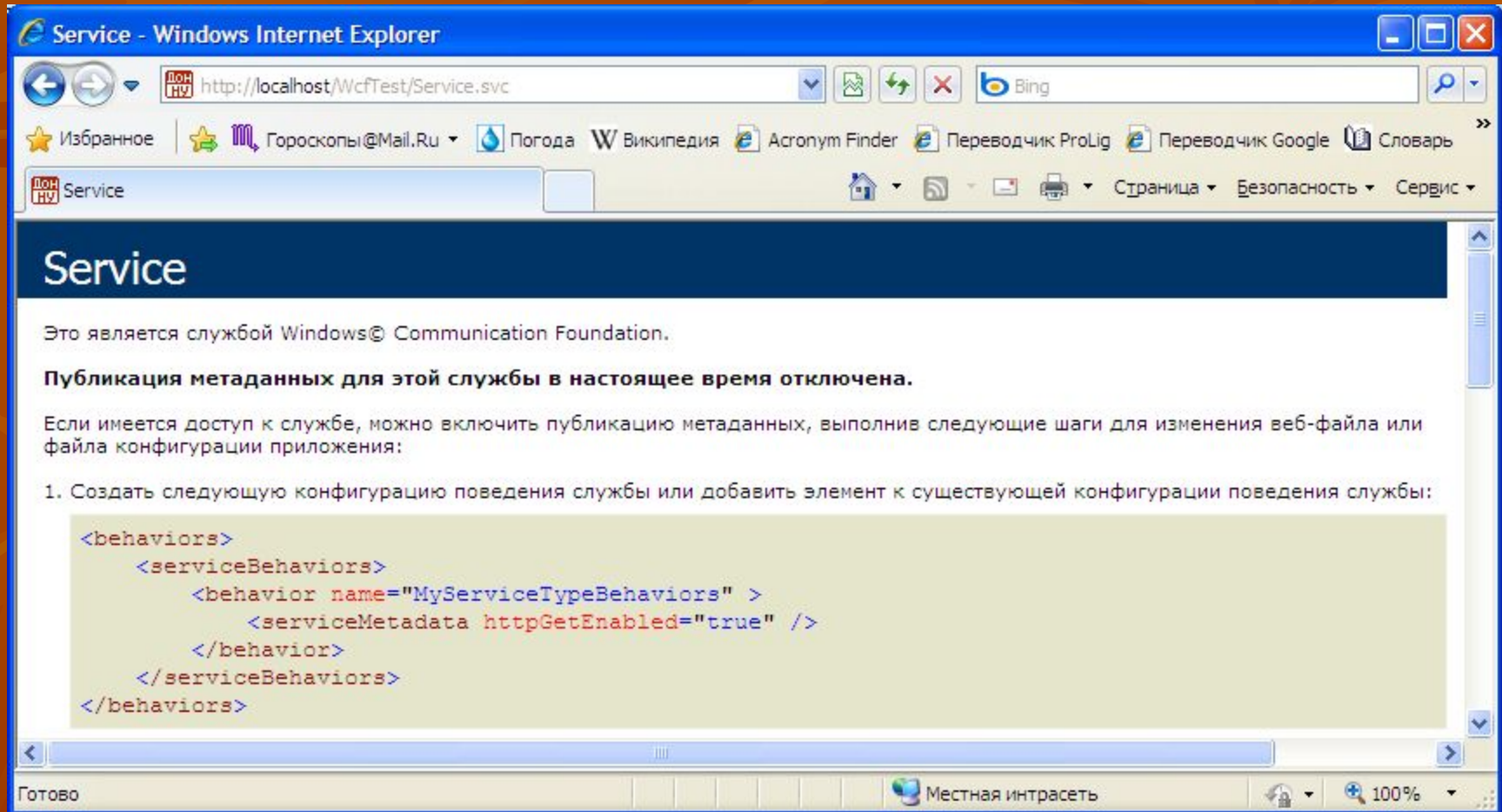
```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions name="Service" targetNamespace="http://tempuri.org/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:tns="http://tempuri.org/"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:microsoft="http://schemas.microsoft.com/ws/2005/12/wsdl/contract"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:wsa10="http://www.w3.org/2005/08/addressing"
  xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex">
- <wsp:Policy wsu:Id="WSHttpBinding_IService_policy">
- <wsp:ExactlyOne>
- <wsp>All>
- <sp:SymmetricBinding
```

Готово Местная интрасеть 100%



# Проверка доступа к метаданным службы при отключённом доступе к метаданным

После настройки прокси-класса клиента надо удалить **mex**-точку службы и задать `HttpGetEnabled="false"` для предотвращения несанкционированного подключения к службе. В этом случае попытка доступа к метаданным:



При этом служба и её клиенты продолжают работать.

# Хостинг служб WCF

Каждая служба WCF должна находиться под управлением некоторого процесса Windows, называемого **хостовым** процессом. Существуют 4 типа хостинга:

1. *Резидентное размещение в управляемом приложении .NET (со своим экземпляром класса ServiceHost);*
2. *Размещение в виде управляемой службы Windows;*
3. *Размещение в IIS;*
4. *Размещение в службе активации Windows – WAS (Windows Server 2008, Vista)*

## Понятие базового адреса

Базовый адрес — это корневой адрес для резидентного хостинга службы, реализующего работу класса ServiceHost, указывается в файле конфигурации в ветке `<host><baseAddresses>...`

Базовый адрес эквивалентен виртуальному каталогу в ASP.NET. При хостинге в службах IIS базовый адрес — это всегда адрес службы, указанный в её файле SVC. При размещении службы в IIS создается один базовый адрес в виртуальном каталоге, содержащем приложение. Следовательно, для конечных точек служб, размещенных в IIS, следует использовать относительные адреса. Указание полного адреса конечной точки может привести к ошибкам при развертывании службы.

# Построение клиентов для служб WCF

Клиент должен знать, где находится служба, использовать ту же привязку, что и служба и импортировать определение контракта службы (по протоколу WSDL), т.е., клиентское приложение должно содержать информацию о конечных точках службы. Visual Studio, при добавлении ссылки на службы, автоматически добавляет необходимую информацию о конечных точках службы в раздел

**<system.serviceModel>** своего файла конфигурации **web.config**. Данный раздел может находиться и в корневом файле конфигурации сайта и в файле конфигурации каталога где находится клиент.

Для вызова операций службы клиент должен сначала импортировать контракт службы в родное представление своей среды и создать посредника – прокси-класс для общения с WCF-службой. Посредник предоставляет те же операции, что и контракт службы, но при этом содержит дополнительные методы для управления жизненным циклом и подключением к службе.

Visual Studio позволяет просто генерировать посредника и импортировать метаданные службы в папку ссылок проекта **App\_WebReferences** и в файл конфигурации **web.config**. В файле конфигурации автоматически появляется узел **<client>** - рабочая точка и её привязка - **<bindings>**.

После построения посредника клиент может прямо обращаться к операциям (методам) службы.

# Конфигурация конечных точек на стороне клиента

...

```
<system.serviceModel>
```

```
<bindings>
```

```
<wsHttpBinding>
```

```
<binding name="MyPoint" closeTimeout="00:01:00"
```

```
openTimeout="00:01:00" receiveTimeout="00:10:00"
```

```
sendTimeout="00:01:00"
```

```
bypassProxyOnLocal="false" transactionFlow="false"
```

```
hostNameComparisonMode="StrongWildcard"
```

```
maxBufferSize="524288" maxReceivedMessageSize="65536"
```

```
messageEncoding="Text" textEncoding="utf-8" useDefaultWebProxy="true"
```

```
allowCookies="false">
```

```
</wsHttpBinding>
```

```
...
```

```
</bindings>
```

```
<client>
```

```
<endpoint name="MyPoint"
```

```
address="http://localhost:8000/MyService1/" binding="wsHttpBinding"
```

```
contract="MyNamespace.IMyPoint"
```

```
bindingConfiguration="MyPoint" >
```

```
</endpoint>
```

```
</client>
```

```
</system.serviceModel>
```

Уточнение настроек для привязок типа **wsHttpBinding**

Настройка привязки конечной точки с  
bindingConfiguration="**MyPoint**"



# Источники

- Джувел Лёве. Создание служб Windows Communication Foundation. – СПб.: Питер, 2008 . – 592 с.: ил.
- <http://msdn.microsoft.com>
- Доминик Байер, Кристиан Вейер, Стив Майн. Расширение служб WCF за пределы HTTP с помощью WAS / <http://msdn.microsoft.com/ru-ru/magazine/cc163357.aspx>