МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Информационные технологии»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ
ПО ДИСЦИПЛИНЕ
«СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ»
ДЛЯ СТУДЕНТОВ ЗАОЧНОЙ И ОЧНО-ЗАОЧНОЙ ФОРМ ОБУЧЕНИЯ
(Семестр 2)

Ростов-на-Дону
ДГТУ
2023

УДК 372.8:004
Составители: М. В. Привалов, Е. В. Рашидова

Методические указания для выполнения лабораторных работ по дисциплине «Современные технологии программирования» для студентов заочной и очно-заочной форм обучения. (Семестр 2). - Ростов-на-Дону: Донской гос. техн. ун-т, 2021. – 35 с.

Рассматриваются современные подходы и технологии разработки программного обеспечения для Web-ориентированных и распределённых информационных систем.

Предназначены для студентов направления 09.04.02 «Информационные технологии» заочной и очно-заочной форм обучения.

УДК 372.8:004

Печатается по решению редакционно-издательского совета
Донского государственного технического университета

Ответственный за выпуск зав. кафедрой «Информационные технологии»,
д-р техн. наук, профессор Б.В. Соболь

В печать ___.___. 20___г.
Формат 60×84/16. Объем___усл.п.л.
Тираж ___экз. Заказ №___.

Издательский центр ДГТУ
Адрес университета и полиграфического предприятия:
344000, г. Ростов-на-Дону, пл. Гагарина,1

Оглавление

Лабораторная работа №1. Обработка иерархических данных. Язык запросов для формирования выборок по иерархиям.

Цель: научиться представлять информацию в виде XML-документов, формировать их в приложениях C# и Java и выполнять выборку интересующих данных с помощью языка XPath

Теоретические сведения.

Расширяемый язык разметки (eXtensible Markup Language — XML)

Первая технология, представляющая последовательный, результативный и расширяемый механизм для описания данных и метаданных в одном и том же документе. Эта технология позволяет получать самоописывающиеся данные, упрощает реализацию интероперабельности и предоставляет возможность хранения самых разнообразных данных, например, текстовые документы, реляционные данные, объектно-ориентированные структуры и иерархическую информацию.

В настоящее время XML может использоваться в любых приложениях, которым нужна структурированная информация — от сложных геоинформационных систем с гигантскими объемами передаваемой информации до простейших программ, использующих этот язык для описания служебной информации.

XML-документ представляет собой обычный текстовый файл, в котором при помощи специальных маркеров создаются элементы данных, последовательность и вложенность которых определяет структуру документа и его содержимое. Структура XML-документа (рис. 1) сходна со структурами из многих других иерархических, структурных и метаинформационных технологий.

```
<Корневой элемент>
     <Элемент_1 Атрибут_1=« »> </Элемент_1>
     <Элемент_2> </Элемент_2>
     <Элемент_3>
          <Элемент_4> </Элемент_4>
          <Элемент_5 Атрибут_1=« » Атрибут_2=« »> </Элемент_5>
     </Элемент_3>
</Корневой элемент>
```

Рисунок 1. - Структура XML-документа.

Элементом верхнего уровня XML-документа является корневой элемент,

существующий в документе в единственном числе. Элементы, помещенные внутрь другого элемента, называются вложенными или дочерними элементами. Содержащий их элемент — родительским элементом.

Описание на языке XML представляет собой операторы, написанные с соблюдением строго определенного синтаксиса. Каждый элемент XML-документа должен содержать начальный и конечный тег (либо специальный пустой тег). Любой вложенный элемент должен быть полностью определен внутри элемента, в состав которого он входит. Это обусловлено тем, что структура XML-документов должна быть понятной для программы, которая выполняет обработку и отображение информации, содержащейся в этих документах. Строгий синтаксис придает XML-документу предсказуемую форму и облегчает написание программы обработки.

При создании XML-документа часто используется возможность создания собственных элементов и присваивания им необходимых имен — именно поэтому язык XML является расширяемым (extensible). Например, на рис. 2 приведено описание перечня книг.

```
<?xml version=«1.0» encoding=«utf-8» ?>
<inventory>
     <book>
          <title>The Adventures of Huckleberry Finn</title>
          <author>Mark Twain</author>
          <binding>mass market paperback</binding>
          <pages>298</pages>
          <price>$5.49</price>
     </book>
     <book>
          <title>XML для проектировщиков</title>
          <author>Джеймс Бин</author>
          <binding>mass market paperback</binding>
          <pages>255</pages>
          <price>$5.00</price>
     </book>
     <book>
          <title>Создание корпоративных систем на основе JAVA 2 Enterprise
Edition</title>
          <author>Пол Дж. Перроун</author>
          <binding>trade paperback</binding>
          <pages>1184/pages>
          <price>$24.38</price>
     </book>
<inventory>
```

Рисунок 2. - Пример XML-документа с описанием перечня книг.

Основным достоинством XML-документов является то, что при относительно простом способе создания и обработки (обычный текст может редактироваться

любым тестовым процессором и обрабатываться стандартными XML-анализаторами), они позволяют создавать структурированную информацию, которую удобно использовать при компьютерной обработке данных.

Чтобы обработать XML-документ, прикладная программа должна иметь возможность ориентироваться в его иерархической структуре и извлекать, при необходимости, содержимое контейнеров. Такую функциональность прикладному ПО предоставляют синтаксические анализаторы (парсеры). Упрощенно это выглядит так: прикладная программа вызывает парсер, который разбирает XML-документ, идентифицирует каждый контейнер и передает значения данных, содержащихся в каждом контейнере, в прикладное ПО. Как правило, синтаксические анализаторы могут сравнивать исходный XML -документ с набором правил из некоторых метаданных и извещать прикладное ПО при обнаружении противоречивости или ошибки — верифицировать документ. Набор правил и ограничений для XML-документа определяет так называемая схема.

Существует два основных типа синтаксических анализаторов: парсеры, создающие объектную модель документа (Document Object Model — DOM), и парсеры, предоставляющие простой API для работы с XML (simple API for XML — SAX). Главным достоинством DOM-парсеров является то, что они передают прикладному ПО всю структуру XML-документа полностью. При использовании SAX-парсеров такая структура XML-документа не строится. SAX-парсеры обычно требуют меньше памяти в отличие от DOM-парсеров и являются достаточно производительными при обходе иерархических структур. Однако их использование утяжеляет логику прикладного ПО и усложняет навигацию по документу.

Введение в язык XPath

XPath - это набор синтаксических правил для адресации частей XML-документа

XPath - это синтаксис для адресации частей XML-документа

XPath использует пути для адресации элементов XML

XPath является важнейшей частью стандарта XSLT

XPath не является XML-форматом

XPath является стандартом W3C

Рассмотрим фрагмент XML-документа:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country=”USA”>
    <title>Empire Burlesque</title>
```

```
    <artist>Bob Dylan</artist>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
</catalog>
```

Пусть имеется выражение XPath:

```
catalog/cd/price
```

Оно определяет набор элементов price, дочерних относительно элементов cd, подэлементов элемента catalog.

XPath используется в приложениях, в которых есть необходимость обрабатывать данные, представленные в формате XML, а также он является неотъемлемой частью стандарта XSLT, который используется при необходимости отображения информации в заданном формате.

Для данного примера XML-документа возможны следующие варианты извлечения данных:

Запрос	Действие
`/catalog`	Выбирает корневой элемент
`/catalog/cd`	Выбирает все элементы cd элемента catalog
`/catalog/cd/price`	Выбирает все элементы price элементов cd элемента catalog
`/catalog/cd[price>10.0]`	Выбирает все элементы cd элемента catalog с price>10
`Выражение, начинающееся с /`	Представляет абсолютный путь к элементу
`Выражение, начинающееся с //`	Выделяет все элементы документа, удовлетворяющие критерию
`//cd`	Выделяет все элементы cd в документе
`/catalog/cd/title | /catalog/cd/artist`	Выделяет все элементы title и artist элементов cd элемента catalog
`//title | //artist`	Выделяет все элементы title и artist в документе

Запрос	Действие
`/catalog/cd/*`	Выделяет все дочерние элементы узла cd элемента catalog
`/catalog/*/price`	Выделяет всех «внуков» элемента catalog
`/*/*/price`	Выделяет все элементы price, имеющие двух предков
`//*`	Все элементы документа
`/catalog/cd[1]`	Первый дочерний элемент cd элемента catalog
`/catalog/cd[last()]`	Последний дочерний элемент cd элемента catalog
`/catalog/cd[price]`	Выделяет все элементы cd элемента catalog, которые имеют дочерний элемент price
`/catalog/cd[price=10.90]/price`	Выделяет все элементы price, дочерние относительно /catalog/cd со значением 10.90
`//@country`	Выделяет все атрибуты country
`//cd[@country]`	Выделяет все элементы cd в документе, имеющие атрибут country
`//cd[@*]`	Выделяет элементы cd с любыми атрибутами
`//cd[@country='UK']`	Выделяет элементы cd, имеющие атрибут country со значением UK

Более подробно спецификацию XPath и все возможные варианты построения путей можно почерпнуть на http://www.w3.org/TR/xpath/

Порядок выполнения работы

При выполнении работы необходимо выполнить следующие требования:

1. Сформировать с помощью любого текстового редактора исходный XML-документ, содержащий не менее 10 записей, согласно индивидуальному заданию.
2. Разработать приложение, которое считывает документ в программе и выполняет его обработку в соответствии с заданием. При обработке обязательно использование выражений XPath для разбора информации и/или отбора узлов, соответствующих заданию.
3. Сформировать на диске результирующий XML-документ.
4. Парсер, используемый в программе определяется вариантом:
- SAX Parser – для нечётных вариантов;
- DOM Parser – для чётных вариантов.

Реализации парсеров допускается брать из сторонних библиотек при условии соответствия типа.

5. Платформа, предлагаемая для реализации приложения: .NET или Java.
6. Для формирования результирующего документа допускается применение сериализации коллекций объектов в XML. **Внимание!** В большинстве вариантов

заданий этот подход применить затруднительно.

Индивидуальные задания

1. В расписании движения самолетов из аэропорта указаны следующие сведения: номер рейса (4 цифры), аэропорт назначения, расстояние в км, стоимость билета (взрослый билет, детский билет), время в часах и мин. (отправление, прибытие в аэропорт назначения). Сформировать документ со сведениями о трех рейсах, имеющих наибольшую продолжительность полета при расстоянии не большем заданного пользователем.

2. В каталоге студии звукозаписи имеются следующие данные: название группы, название альбома, год выпуска альбома, название студии, записавшей альбом. Необходимо сформировать XML-каталог групп, выпустивших альбомы в заданном году и на заданной студии.

3. В журнале успеваемости академгруппы по программированию имеются следующие данные: фамилия студента, оценки по пяти лабораторным работам, количество пропусков занятий. Определить трех студентов, имеющих наибольшее количество пропусков (студентов, сдавших все работы не включать). Записи в XML-документе расположить в алфавитном порядке.

4. В магазине имеются следующие данные о товарах: название, единица измерения, цена, норма отпуска в одни руки. Составить документ, включающий элемент, содержащий список товаров, норма отпуска которых не более двух единиц, а также элемент, содержащий список товаров, стоимость которых превышает указанную пользователем. Списки упорядочить по названию товаров в алфавитном порядке.

5. На заводе радиоэлектроники выпускают звуковоспроизводящую технику и имеются следующие данные: название прибора, назначение (магнитофон, магнитола, проигрыватель), год создания, стоимость, гарантийный срок эксплуатации. Составить XML-документ, включающий список магнитофонов, разработанных в заданном году, а также список проигрывателей, гарантийный срок эксплуатации которых более 3-х лет.

6. В каталоге программного обеспечения имеются следующие данные: имя файла, расширение, размер файла, дата создания. Составить каталог текстовых файлов (расширение TXT, DOC), а также таблицу файлов, размером более 64 Кбайт. Каталоги отсортировать по имени файла в алфавитном порядке.

7. В больнице ведется учет больных по следующим данным: фамилия больного, номер палаты, дата поступления, дата выписки (может отсутствовать), диагноз (название болезни). Необходимо выдать список больных, лежавших в

больнице на заданную дату. Список сортировать по номеру палаты.

8. На станции технического обслуживания автомобилей (СТО) ведется учет автомобилей, прошедших капитальный ремонт, по следующим данным: марка машины, серийный номер, пробег (в км) после предыдущего ремонте, год выпуска автомобиля. Необходимо составить документ, содержащий список машин, имеющих пробег более 100 000 км, а также список автомобилей, выпущенных после заданного года и прошедших ремонт. Списки сортировать по году выпуска машины.

9. В библиотеке имеются следующие данные о книгах: название, фамилия автора, год издания, издательство, количество экземпляров книг в библиотеке. Необходимо сформировать XML-документ, содержащий список книг, изданных в заданном году и список книг, имеющихся в библиотеке в одном экземпляре. Список упорядочить по названию книг в алфавитном порядке.

11. В военкомате ведется учет юношей допризывного и призывного возраста. Имеются следующие данные: фамилия, год рождения, номер личного дела, годность к службе («годен» или «не годен»). Необходимо вывести список юношей, призываемых на службу в заданном году (по достижении 18 лет). Список упорядочить по году рождения.

11. В аптеке ведется учет лекарственных средств. Имеются следующие данные: название лекарства, цена одной упаковки, количество упаковок в аптеке, год выпуска, срок хранения (в годах). Необходимо вывести документ, содержащий список лекарств, не годных к употреблению на заданный год, и и список лекарств, стоимость которых выше заданной пользователем. Упорядочить по названию лекарства в алфавитном порядке.

12. В заводском цеху ведется журнал расхода материалов по следующим данным: название материала, ГОСТ, расход в сутки, количество имеющихся в цеху. Необходимо вывести документ, содержащий список материалов, которые закончатся через заданное количество дней, а также список 5-ти наименее расходуемых материалов. Списки упорядочить по названию материала в алфавитном порядке.

13. За материально ответственным лицом числятся материальные ценности, записанные в журнале: название предмета, количество, дата приобретения (год), срок службы (в годах). Необходимо вывести документ, содержащий список предметов, подлежащих списанию в заданный год, а также список предметов, срок службы которых превышает указанный пользователем. Списки упорядочить по дате приобретения.

14. В заводском цеху ведется учет электроэнергии, расходуемой машинами и

приборами. Имеются следующие данные: название машины или прибора, инвентарный номер, потребляемая мощность, количество таких приборов в цеху. Вывести список десяти наиболее энергоемких приборов, в котором атрибутом корневого узла указать суммарную мощность, потребляемую цехом при всех включенных приборах. Список упорядочить по инвентарному номеру.

15. На АТС ведется учет междугородних разговоров абонентов по следующим данным: фамилия абонента, домашний адрес, номер телефона, сумма междугородних телефонных разговоров за месяц (в рублях). Необходимо вывести список абонентов, тратящих на междугородние звонки больше указанной суммы. Списки упорядочить по номеру телефона.

Лабораторная работа №2. Проектирование и создание Web-сервисов

Цель: научиться определять функциональные границы Web-сервисов, реализовывать, развёртывать и использовать их.

Теоретические сведения.

В настоящее время SOA и Web-сервисы получили достаточно широкое распространение, и применяются в самых разных качествах: от простого предоставления справочных данных в Сети, например, данных о прилете самолетов, курсов валют и драгоценных металлов, до работы с кредитными карточками и переводов текста online. Еще больше существует корпоративных Web-сервисов, которые используются при решении самых разных производственных задач.

Широкому распространению Web-сервисов немало способствовала платформа Microsoft .NET. Дело в том, что создать Web-сервис в той же Microsoft Visual Studio .NET очень просто. Достаточно написать свой класс, унаследованный от класса System.Web.Services.WebService и объявить его методы как Web-методы. Компилируйте, переносите на рабочий сервер и ваш новый Web-сервис готов к работе.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
```

```
namespace WebServicesExample
{
	public class nw : System.Web.Services.WebService
	{
		public nw()
		{
			InitializeComponent();
		}
		#region Component Designer generated code

		//Required by the Web Services Designer
		private IContainer components = null;

		private void InitializeComponent()
		{
		}
		protected override void Dispose( bool disposing )
		{
			if(disposing && components != null)
			{
				components.Dispose();
			}
			base.Dispose(disposing);
		}

		#endregion

		// WEB SERVICE EXAMPLE
		[WebMethod]
		public string HelloWorld()
		{
			return "Hello World";
		}
	}
}
```

Так же просто написать и клиент для любого Web-сервиса. В состав средств разработки любых платформ входит утилита wsdl.exe. Достаточно ей указать адрес WSDL документа, желаемый язык разработки (C#, VB.NET) и она сгенерирует вам код класса, который является прокси-классом для указанного Web-сервиса, то есть его клиентом. Вы получаете готовый к использованию класс с тем же набором

методов, что и Web-сервис. Любое обращение к этим методам автоматически транслируется Web-сервису и ответ Web-сервиса возвращается как результат. Другой способ — добавление в проект ссылки на сервис. В этом случае среда разработки сама выполнит генерацию прокси-класса.

Аналоги wsdl.exe присутствуют, как и было сказано, и в составе Java SE/EE SDK. Однако, для данной платформы приходится использовать две утилилы, которые покрывают функциональнось wsdl.exe: это утилиты wsimport и wsgen. Для создания клиента как раз применяется wsimport.

Web-сервисы .NET могут отвечать не только на запросы SOAP, но и на обычные запросы POST и GET. То же самое касается WSDL сервисов, написанных для других платформ: параметр запроса wsdl, переданный развёрнутой службе позволит получить WSDL-документ, который её описывает.

Написание любого Web-сервиса начинается с изучения WSDL документа – описания самого Web-сервиса, его методов, параметров методов и типов данных.

Создание службы может производиться по одному из сценариев:

- code first;
- contract first.

В первом случае создается код сервиса, а метаданные генерируются автоматически. Во втором — сначала описывается контракт службы в виде WSDL-документа, затем по нему генерируется код без реализации.

WSDL(Web Services Description Language) - это XML-ориентированный язык, предназначенный для определения web-сервисов и доступа к ним.

Документ WSDL является XML-документом, описывающим web-сервис. Он определяет расположение сервиса и операции (или методы), предоставляемые им.

Структура документа WSDL

Документ WSDL - это всего лишь простой документ XML, который содержит набор выражений, определяющих web-сервис.

В документе WSDL определяется web-сервис с помощью следующих основных элементов:

Элемент	**Определяет**
<portType>	Методы, предоставляемые web-сервисом
<message>	Сообщения, используемые web-сервисом
<types>	Типы данных, используемые web-сервисом
<binding>	Протоколы связи, используемые web-сервисом

Основная структура документа WSDL выглядит следующим образом:

```
<definitions>
<types>
  определение типов.....
</types>

<message>
  определение сообщения....
</message>

<portType>
  определение порта.....
</portType>

<binding>
  определение связей.....
</binding>

</definitions>
```

Документ WSDL может также содержать другие элементы, например, элементы расширения и элемент service, который позволяет объединить вместе в одном отдельном документе WSDL определения нескольких web-сервисов.

Порты WSDL

Элемент **<portType>** является наиболее важным элементом в WSDL.

Он определяет сам web-сервис, предоставляемые им операции и используемые сообщения. Порт определяет точку монтирования к web-сервису. Его можно сравнить с библиотекой функций (модулем, классом) в традиционном языке программирования, а каждую операцию можно сравнить с функцией в традиционном языке программирования.

Элемент <portType> можно сравнить с библиотекой функций (модулем, классом) в традиционном языке программирования.

Сообщения WSDL

Элемент **<message>** определяет элементы данных операции.

Каждое сообщение может содержать одну или несколько частей. Эти части можно сравнить с параметрами вызываемых функций в традиционном языке программирования.

Типы WSDL

Элемент **<types>** определяет тип данных, используемых web-сервисом.

Для максимальной платформо-независимости WSDL использует синтаксис XML Schema для определения типов данных.

Связи WSDL

Элемент **<binding>** определяет формат сообщения и детали протокола для каждого порта.

Пример WSDL

Это простейшая фрагмент WSDL-документа:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>
<portType name="glossaryTerms">
 <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
 </operation>
</portType>
```

В этом примере элемент **portType** определяет "glossaryTerms" как имя **порта**, и "getTerm" как имя **опрерации**.

Операция "getTerm" имеет **входящее сообщение**, называемое "getTermRequest", и **исходящее сообщение** --- "getTermResponse".

Элементы **message** определяют **части** каждого сообщения и ассоциированные типы данных.

Если сравнивать с традиционным программированием, то glossaryTerms --- библиотека функций, а "getTerm" --- функция с параметром "getTermRequest", которая возвращает getTermResponse после выполнения.

Порты WSDL описывают интерфейс (список допустимых операций) работы с web-сервисом.

Типы Операций

Запрос-ответ (request-response) --- самый распространенный тип операций. В WSDL определены такие типы операций:

Тип	**Описание**
Однонаправленный (One-way)	Операция может принимать сообщение, но не будет возвращать ответ
Запрос-ответ (Request-response)	Операция может принимать запрос и возвратит ответ
Вопрос-ответ (Solicit-response)	Операция может послать запрос и будет ждать ответ

Однонаправленная (One-Way) Операция

Пример однонаправленной операции:

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
```

```
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

В этом примере порт "glossaryTerms" определяет однонаправленную операцию под названием "setTerm".

Операция "SetTerm" позволяет ввод новых словарных термов с помощью сообщения "newTermValues" со входными параметрами "term" и "value". Однако, данная операция не предусматривает какого-либо выходного сообщения.

Операция типа "Запрос-Ответ"

Пример операции типа "запрос-ответ":

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>
<portType name="glossaryTerms">
 <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
 </operation>
</portType>
```

В этом примере порт "glossaryTerms" определяет операцию типа "запрос-ответ" с именем "getTerm".

Операция "getTerm" принимает на вход сообщение с именем "getTermRequest" и параметром "term" и возвращает сообщение с именем "getTermResponse" и параметром "value".

Связи WSDL определяют формат сообщения и подробности протокола для каждого порта.

Связь с SOAP

Пример операции запроса-ответа:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
```

```
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>

<binding type="glossaryTerms" name="b1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
 <operation>
   <soap:operation
    soapAction="http://example.com/getTerm"/>
   <input>
     <soap:body use="literal"/>
   </input>
   <output>
     <soap:body use="literal"/>
   </output>
 </operation>
</binding>
```

Элемент **binding** имеет два атрибута --- атрибут name и атрибут type.
Атрибут name определяет название (можно использовать любое) связи, а атрибут type --- точку монтирования для связи, в данном случае это порт "golossaryTerms".
Элемент **soap:binding** имеет два атрибута --- атрибут style и атрибут transport.
Элемент style может быть либо "rpc", либо "document". В данном случае используется значение document. Атрибут transport определяет используемый SOAP протокол. В данном случае это HTTP.
Элемент **operation** описывает все операции, с которыми порт может работать.
Для каждой операции должно быть определено в соответствие действие SOAP. Также необходимо указать кодировку ввода и вывода. В этом случае используется "literal".

Развертывание сервиса в MSVS.

Microsoft Visual Studio предоставляет возможность работы со службой при отладке без её развёртывания. Для этого запускается легковесный Web-сервер, в котором автоматически развернута только что созданная служба. После окончания разработки службу всё же будет необходимо развернуть в Web-контейнере. Таким контейнером может выступать платформа Microsoft IIS (Internet Information Services). Для развертывания службы можно использовать сервер по умолчанию, но лучше для служб иметь отдельный виртуальный сервер или хотя бы каталог.

Таким образом, для развертывания службы в Web-контейнере требуется выполнить следующие шаги:

1. Создать виртуальный сервер или хотя бы виртуальный каталог в IIS

2. Создать в каталоге папку bin
3. Поместить службу в виртуальный каталог, а соответствующую ей сборку .NET - в каталог bin.

Развёртывание сервиса в Java

Аналогично, для развёртывания SOAP-сервиса в Java нам понадобится либо контейнер сервлетов, либо хост-приложение. Во втором случае развёртывание происходит за счёт запуска хост-приложения, а в первом понадобятся следующие шаги:

1. Создать дескриптор службы, файл sun-jaxws.xml, и описать в нём конечные точки вашей службы.
2. Добавить сервлет и слушатель в конфигурацию Web-приложения (файл web.xml)
3. Развернуть WAR-файл с приложением в контейнере сервлетов.

Порядок выполнения работы

В соответствии с индивидуальным заданием

1. Определить функциональные границы веб-сервиса.
2. Описание контракта сервиса и задокументировать его.
3. Реализовать сервис на выбранной платформе (допускается использование .NET, Java или Node.js) и выполнить его развертывание.
4. Реализовать клиентское приложение и с помощью вызовов созданного сервиса из его кода продемонстрировать работу методов (вызовите объявленные методы, продемонстрируйте получение результатов).

Индивидуальное задание

1. В соответствии с темой заданиея (согласовывается с преподавателем) выполнить описание бизнес-процесса в виде одной Activity-диаграммы или их набора.
2. Определите функциональные границы двух-трех сервисов, которые могли бы быть использованы для представления первых шагов бизнес-процесса.
3. Выделите функциональность для организации первого из сервисов таким образом, чтобы можно было создать операции: однонаправленные и запрос-ответ.
4. Реализуйте первую службу (первый шаг процесса).
5. Создайте клиентское приложение, с помощью которого продемонстрируйте работу запросов. Основные требования к клиентскому приложению:
    - приложение должно использовать разработанную веб-службу или сервис
    - приложение должно быть оснащено графическим пользовательским

интерфейсом, который должен обеспечивать демонстрацию работы всех функций, а также ввод пользователем всех основных параметров и отображение результатов

Темы для индивидуального задания

Индивидуальное задание согласовывается с преподавателем. Рекомендуется в качестве индивидуального задания использовать тему дипломного проекта или одно из перечисленных ниже заданий.

1. Спроектировать компьютерную подсистему учета Интернет-услуг
2. Спроектировать подсистему учета ресурса воздушных судов
3. Спроектировать компьютерную подсистему начисления и учета страховых выплат клиентам в условиях страховой компании
4. Спроектировать подсистему учета и анализа экономической деятельности в условиях агентства недвижимости
5. Спроектировать подсистему формирования и обработки договоров добровольного страхования наземного автотранспорта
6. Спроектировать подсистему обработки торговых агентов
7. Спроектировать подсистему планирования и учета чартеров
8. Спроектировать подсистему формирования и учета счетов-заказов в для туристических агентств
9. Спроектировать подсистему бронирования билетов в кинотеатр
10. Спроектировать подсистему расчета себестоимости для рыбного хозяйства
11. Спроектировать подсистему учета потребительских кредитов
12. Спроектировать подсистему реализации и движения горюче-смазочных материалов в условиях сети АЗС
13. Спроектировать подсистему учета продаж товаров
14. Спроектировать подсистему учета сбора и реализации зерновых культур

15. Спроектировать подсистему учета убытков автогражданской ответственности
16. Спроектировать подсистему учета выплат за услуги газоснабжения для населения
17. Спроектировать подсистему расчета арендной платы в условиях коммунального предприятия
18. Спроектировать подсистему учета движения грузового подвижного железнодорожного транспорта в условиях металлургического завода
19. Спроектировать подсистему учета сырья и материалов
20. Спроектировать подсистему приема электронных коммунальных платежей в условиях банка
21. Спроектировать подсистему учета и планирования ремонтных работ
22. Спроектировать подсистему учета технического состояния компьютерного оборудования

Лабораторная работа №3. Проектирование и создание Web-сервисов с использованием Windows Communication Foundation (WCF), Node.js, JAX-WS и асинхронных коммуникаций.

Цель: научиться создавать сервисы с асинхронными коммуникациями на основе WCF, Node.js, JAX-WS.

Теоретические сведения.
WCF

WCF - это единая программная модель от Microsoft, предназначенная для создания сервис-ориентированных приложений. WCF позволяет разработчикам строить безопасные, надежные решения с поддержкой транзакций, которые могут взаимодействовать с различными платформами и уже существующими решениями.

Вообще, WCF предназначен для создания как сервис-ориентированных приложений, так и распределенных приложений. Никто не мешает нам создать одноранговую сеть с дуплексными связями на WCF. Но все-таки основной ориентацией WCF являются сервисы.

WCF поддерживает основные существующие на данный момент протоколы и технологии для передачи данных: HTTP/HTTPS, TCP, именованные каналы, MSMQ, и так далее. Взаимодействовать можно как угодно и с кем угодно. Модель обладает рядом положительных характеристик.

Во-первых, это надежные сеансы. Между клиентом и сервером устанавливается сеанс (session, почти как в ASP.NET) и WCF может гарантировать нам 100%-ую доставку сообщений (конечно, за счет производительности). Здесь есть 2 варианта:

1. Неупорядоченный сеанс - если сообщение потерялось, то после обнаружения потери оно будет запрошено заново и доставлено получателю. Сообщения, отправленные после потерянного сообщения, но до обнаружения потери, будут доставлены получателю до доставки потерянного сообщения.
2. Упорядоченный сеанс - как только обнаруживается пропажа, входящие сообщения ставятся в очередь на принимающей стороне и "замораживаются". Они не будут переданы получателю до передачи потерянного сообщения. Это гарантирует нам, что получатель примет сообщения именно в том порядке, в каком их послал отправитель. Это очень удобно, если нужно передавать большие объемы данных побуферно (например, мы хотим их еще шифровать, при этом скорость уже не так

важна, поточность уже не поддерживается, а вот надежность и безопасность - на первом месте).

Во-вторых - безопасность. WCF поддерживает различные механизмы аутентификации и авторизации (Windows-авторизация, сертификаты, ASP.NET membership). WCF также предлагает цифровую подпись сообщений (SHA-хэши) для гарантии целостности передаваемых данных и шифрование сообщений (TripleDes, Basic, RSA как "key wrap"-алгоритм) для защиты наших данных. Вообще, безопасность в WCF делится на два вида - уровня транспорта и уровня сообщений, но об этом в одной из следующих статей.

Основные концепции WCF. Рассмотрим сначала понятие **конечной точки** (это буквальный, но довольно распространенный перевод английского слова endpoint, дальше будем называть это **"точка соединения"**) - это абстрактный объект, от которого "уходят" данные, и к которому "приходят" данные (некий высокоуровневый аналог TCP-сокетов). Точка соединения объединяет в себе три "столпа WCF" - адрес, привязку и контракт (address, binding, connection - ABC, "азбука WCF").

Адрес. У точки соединения может быть только один адрес, у разных точек соединения может быть один базовый адрес (тогда адреса самих точек соединения будут задаваться относительно этого базового адреса). С помощью точки соединения адрес однозначно связывается с привязкой и контрактом **Привязка**, в свою очередь, определяет то, как наша точка соединения общается с окружающим миром. Она позволяет управлять такими вещами, как: сеанс, безопасность, поточность, транзакции, транспорт, кодировка сообщений.

Контракт же, по сути, является **интерфейсом службы**, помеченным специальным атрибутом *ServiceContractAttribute*. Он определяет **требования** службы к безопасности, сеансу, задает параметры операций (начало сеанса, завершение сеанса, является ли операция односторонней, асинхронность операций на сервере).

WCF позволяет передавать любые объекты. Единственное требование - классы, которые мы опишем для использования при взаимодействии клиента со службой, должны быть помечены атрибутом *DataContractAttribute*, их поля и свойства - *DataMemberAttribute*, а члены перечислений - *EnumMemberAttribute*.

WCF предполагает несколько вариантов хостинга сервисов:

1. Хостинг в управляемом приложении.
2. Хостинг в управляемом сервисе Windows.
3. Хостинг с использованием Web-контейнера IIS.
4. Хостинг с использованием механизма Windows Activation Service (WAS).

Рассмотрим создание простого сервиса WCF, размещаемого в управляемом приложении и возвращающего приветствие на основании ввода пользователя.

Для этого, в первую очередь, создадим консольное приложение и определим контракт сервиса.

```
[ServiceContract]
public interface IHelloWorldService
{
    [OperationContract]
    string SayHello(string name);
}

public class HelloWorldService : IHelloWorldService
{
    public string SayHello(string name)
    {
        return string.Format("Hello, {0}", name);
    }
}
```

Создадим экземпляр класса Uri и зададим базовый адрес сервиса.

```
Uri baseAddress = new Uri("http://localhost:8080/hello");
```

Далее создадим экземпляр класса ServiceHost, передавая тип, представляющий тип сервиса и базовый URI ServiceHost. Разрешим передачу метаданных и подготовим сервис к приёму сообщений:

```
using (ServiceHost host = new ServiceHost(typeof(HelloWorldService), baseAddress))
{
    // Разрешим передачу метаданных.
    ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
    smb.HttpGetEnabled = true;
    smb.MetadataExporter.PolicyVersion = PolicyVersion.Policy15;
    host.Description.Behaviors.Add(smb);

    // Слушаем сообщения. Так как нет явно сконфигурированных
    // точек соединения, будет создано по отдельной точке на контракт
    host.Open();

    Console.WriteLine("The service is ready at {0}", baseAddress);
    Console.WriteLine("Press <Enter> to stop the service.");
    Console.ReadLine();

    // Закрываем ServiceHost.
    host.Close();
}
```

Протестировать работу сервиса можно с помощью приложения WCF Test Client.exe

Данный пример продемонстрировал односторонний (синхронный) обмен информацией между сервером и клиентом. Часто возникают ситуации, когда требуется двухсторонняя асинхронная связь, то есть сервис должен отправить сообщение клиенту, когда тот его специально не ожидает. Часто с помощью такого интерфейса реализуются асинхронные уведомления клиента. Модель WCF позволяет реализовать такую связь с использованием двухстороннего контракта.

Двухсторонний контракт моделируется путем использования двух интерфейсов, связанных вместе с помощью ServiceContract. Ниже показан пример сервиса, моделирующего заказ пиццы с отображением прогресса её приготовления и уведомлением о готовности.

```
[ServiceContract(CallbackContract=typeof(IPizzaProgress))]
interface IOrderPizza
{
    [OperationContract]
    void PlaceOrder(string PizzaType);
}

interface IPizzaProgress {

    [OperationContract]
    void TimeRemaining(int minutes);

    [OperationContract]
    void PizzaReady();
}


class PingService : IOrderPizza
{
    IPizzaProgress callback;

    public void PlaceOrder(string PizzaType)  {

        callback = OperationContext.Current.GetCallbackChannel();

        Action preparePizza = PreparePizza;
        preparePizza.BeginInvoke(ar => preparePizza.EndInvoke(ar), null);

    }


    void PreparePizza()    {

        for (int i = 10 - 1; i >= 0; i--) {
            callback.TimeRemaining(i);
```

```
            Thread.Sleep(1000);
        }

        callback.PizzaReady();
    }
}
```

Жирным шрифтом показано получение callback-интерфейса и уведомление клиента.

Если клиенту необходимо иметь возможность получать сообщения от сервиса, он должен предоставить реализацию callback-контракта. Его определение может быть получено из метаданных или из сборки с общим контрактом. При использовании метаданных контракт обратной связи будет именован так же, как и контракт сервиса с суффиксом Callback.

```
class Program {

    static void Main(string[] args) {
        InstanceContext ctx = new InstanceContext(new Callback());

        OrderPizzaClient proxy = new OrderPizzaClient(ctx);
        proxy.PlaceOrder("Pepperoni");
        Console.WriteLine("press enter to exit");
        Console.ReadLine();
    }
}

class Callback : IOrderPizzaCallback {

    public void TimeRemaining(int minutes)  {
        Console.WriteLine("{0} seconds remaining", minutes);
    }

    public void PizzaReady()   {
        Console.WriteLine("Pizza is ready");
    }
}
```

OrderPizzaClient — класс-прокси, сгенерированный с помощью утилиты SysUtil, который будет наследовать DuplexClientBase<T> и контракт.

Node.js

Node или Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода.

Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и Mac OS) и даже программировать микроконтроллеры (например, tessel и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

Создание и запуск HTTP-сервера на Node.js, выдающего Hello, world! :

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

JAX-WS

Java API for XML Web Services (JAX-WS) — это прикладной программный интерфейс языка Java для создания веб-служб, являющийся частью платформы Java EE. JAX-WS является заменой технологии JAX-RPC, предоставляя более документо-ориентированную модель сообщений и упрощая разработку веб-служб за счёт использования аннотаций, впервые появившихся в Java SE 5. Технология JAX-WS является стандартом и описана в JSR 224.

Реализация класса сервиса на сервере.

В этом примере класс реализации, Hello, помечается как веб-служба конечной точки с использованием @WebService аннотации. В классе Hello объявлен один метод с именем SayHello, помеченный @WebMethod аннотацией, которая выставляет аннотированный метод для клиентов веб-служб. Метод SayHello возвращает приветствие клиенту, используя имя, переданное ему. Класс реализации должен также определить конструктор по умолчанию, публичный, без аргументов.

```
package helloservice.endpoint;

import javax.jws.WebService;
import javax.jws.WebMethod;

@WebService
public class Hello {
    private String message = new String("Hello, ");

    public void Hello() {
    }
```

```
    @WebMethod
    public String sayHello(String name) {
        return message + name + ".";
    }
}
```

Реализация клиентского приложения

При вызове удаленных методов по порту, клиент выполняет следующие действия:

1. Использует сгенерированный helloservice.endpoint.HelloService класс, который представляет собой услугу в URI файле WSDL развернутого сервиса:

```
import helloservice.endpoint.HelloService;
import javax.xml.ws.WebServiceRef;

public class HelloAppClient {
    @WebServiceRef(wsdlLocation =
      "META-INF/wsdl/localhost_8080/helloservice/HelloService.wsdl")
    private static HelloService service;
```

2. Извлекает прокси к сервису, также известный как порт, путем вызова getHelloPort из сервиса:

helloservice.endpoint.Hello port = service.getHelloPort();

3. Запускает метод SayHello порта, передавая строку на сервер:

return port.sayHello(arg0);

Ниже приведен полный код HelloAppClient:

```
package appclient;

import helloservice.endpoint.HelloService;
```

```
import javax.xml.ws.WebServiceRef;

public class HelloAppClient {
    @WebServiceRef(wsdlLocation =
      "META-INF/wsdl/localhost_8080/helloservice/HelloService.wsdl")
    private static HelloService service;

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
       System.out.println(sayHello("world"));
    }

    private static String sayHello(java.lang.String arg0) {
        helloservice.endpoint.Hello port = service.getHelloPort();
        return port.sayHello(arg0);
    }
}
```

Асинхронная передача данных

Чтобы реализовать асинхронную связь с WCF необходимо выполнить одно из двух:

- использовать методы с именами *Async, которые уже сгенерированы (асинхронная модель на основе событий);
- изменить сигнатуры методов WCF с целью возврата Task<> (асинхронная модель на основе задач).

Реализация на основе событий:

При условии наличия операции Add(x1, x2) будет сгенерирован метод AddAsync и событие AddOperationCompleted. При этом его параметры будут описаны в классе AddCompletedEventArgs.

```
double value1 = 100.00D;
double value2 = 15.99D;
client.AddCompleted += new EventHandler<AddCompletedEventArgs>(AddCallback);
client.AddAsync(value1, value2);
Console.WriteLine("Add({0},{1})", value1, value2);
```

Результат будет получен локальной callback-функцией:

```
static void AddCallback(object sender, AddCompletedEventArgs e)
{
```

```
    Console.WriteLine("Add Result: {0}", e.Result);
 }
```

Реализация на основе задач.

Контракт и служба.

```
[ServiceContract]
public interface IMessage
{
    [OperationContract]
    Task<string> GetMessages(string msg);
}
public class MessageService : IMessage
{
   async Task<string> IMessage.GetMessages(string msg)
   {
      var task = Task.Factory.StartNew(() =>
                                       {
                                           Thread.Sleep(10000);
                                           return "Return from Server : " + msg;
                                       });
     return await task.ConfigureAwait(false);
   }
}
```

Клиент.

```
class Program
 {
    static void Main(string[] args)
    {
       GetResult();
       Console.ReadLine();
    }

    private async static void GetResult()
    {
       var client = new Proxy("BasicHttpBinding_IMessage");
       var task = Task.Factory.StartNew(() => client.GetMessages("Hello"));
```

```
        var str = await task;
        str.ContinueWith(e =>
        {
           if (e.IsCompleted)
            {
               Console.WriteLine(str.Result);
            }
        });
       Console.WriteLine("Waiting for the result");
    }
 }
```

Полный пример с кодом проекта можно получить здесь: https://www.codeproject.com/Articles/613678/Task-based-Asynchronous-Operation-in-WCF

Пример реализации асинхронного обмена на JAX-WS

Данный пример демонстрирует веб-интерфейс с методами для асинхронных запросов с клиента:

```
@WebService

public interface CreditRatingService {
       // Синхронная операция
       Score getCreditScore(Customer     customer);
       // Асинхронная операция с голосованием
       Response<Score> getCreditScoreAsync(Customer customer);
       // Асинхронная операция с обратным вызовом
       Future<?> getQuoteAsync(Customer customer,
              AsyncHandler<Score> handler);
}
```

Реализация на клиенте:

Использование метода с обратным вызовом

```
CreditRatingService svc = ...;

Future<?> invocation = svc.getCreditScoreAsync(customerTom,
       new AsyncHandler<Score>() {
              public void handleResponse (
                     Response<Score> response)
```

```
                    {
                       score = response.get();
                       // обработка запроса
                    }
    }
);
```

Использование метода опроса

```
CreditRatingService svc = ...;
Response<Score> response = svc.getCreditScoreAsync(customerTom);

while (!response.isDone()) {
       // Делаем что-то пока ожидаем
}

score = response.get();
```

Пример реализации асинхронного обмена на Node.js:

Серверная часть. Допустим у нас есть список пользователей, хранимые в док-ом виде на сервере:

```
{
   "user1" : {
      "name" : "mahesh",
           "password" : "password1",
           "profession" : "teacher",
           "id": 1
   },
   "user2" : {
      "name" : "suresh",
           "password" : "password2",
           "profession" : "librarian",
           "id": 2
   },
   "user3" : {
      "name" : "ramesh",
           "password" : "password3",
           "profession" : "clerk",
           "id": 3
   }
}
```

Напишем запрос для получения всего списка пользователей:

```
var express = require('express');
var app = express();
var fs = require("fs");

// Возвращает весь список
app.get('/users', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
       console.log( data );
       res.end( data );
   });
})
var server = app.listen(8081,’127.0.0.1’);
```

Реализация в клиентском приложении:

```
var xhr = new XMLHttpRequest();
// запрос на получение данных о пользователях
xhr.open('GET', ‘http://127.0.0.1:8081/users’ , true);

xhr.onreadystatechange = function() {
  if (xhr.readyState != 4) return;

  if (xhr.status != 200) {
// в переменную users сохраняем данные, которые получили с сервера
      var users = JSON.parse(xhr.responseText)
  }
}
xhr.send();
```

Порядок выполнения работы

В соответствии с индивидуальным заданием:

1. Определить функциональные границы веб-сервиса.
2. Определить контракт данных и функциональный контракт службы.
3. Создать сервис с контрактом для двухстороннего обмена информацией сервера и клиента.
4. Реализовать сервис с использованием WCF и протестировать его.

Задание

1. В соответствии с согласованным заданием (темой дипломного проекта или индивидуальным заданием из лабораторной работы №2) выполнить описание выполнения функций в виде Activity-диаграмм.
2. Выбрать одну из функций для демонстрации вызова метода с получением уведомления с использованием механизмов обратной связи.
3. Реализовать вышеуказанный сервис с использованием WCF, JAX-WS или Node.js развернуть его, как управляемое клиентское приложение с

интерфейсом.

4. Создать клиентское приложение, с помощью которого продемонстрировать работу с сервисом и асинхронную обработку запросов (для любых технологий).

Лабораторная работа №4. Проектирование и создание Web-сервисов с использованием Windows Communication Foundation (WCF), использующих обратную связь

Цель: научиться создавать сервисы с обратной связью на основе технологии WCF.

Теоретические сведения.

Для организации процессов, выполняющих длительную обработку требуется двухсторонняя асинхронная связь, то есть сервис должен отправить сообщение клиенту, когда тот его специально не ожидает. Модель WCF позволяет реализовать такую связь не только с применением асинхронных коммуникаций, но и с использованием двухстороннего контракта.

Двухсторонний контракт моделируется путем использования двух интерфейсов, связанных вместе с помощью ServiceContract. Ниже показан пример сервиса, моделирующего заказ пиццы с отображением прогресса её приготовления и уведомлением о готовности.

```
[ServiceContract(CallbackContract=typeof(IPizzaProgress))]
interface IOrderPizza
{
    [OperationContract]
    void PlaceOrder(string PizzaType);
}

interface IPizzaProgress {

    [OperationContract]
    void TimeRemaining(int minutes);

    [OperationContract]
    void PizzaReady();
}


class PingService : IOrderPizza
{
    IPizzaProgress callback;

    public void PlaceOrder(string PizzaType)  {
```

```
        callback = OperationContext.Current.GetCallbackChannel();

        Action preparePizza = PreparePizza;
        preparePizza.BeginInvoke(ar => preparePizza.EndInvoke(ar), null);

    }


    void PreparePizza()    {

        for (int i = 10 - 1; i >= 0; i--) {
            callback.TimeRemaining(i);
            Thread.Sleep(1000);
        }

        callback.PizzaReady();
    }
}
```

Жирным шрифтом показано получение callback-интерфейса и уведомление клиента.

Если клиенту необходимо иметь возможность получать сообщения от сервиса, он должен предоставить реализацию callback-контракта. Его определение может быть получено из метаданных или из сборки с общим контрактом. При использовании метаданных контракт обратной связи будет именован так же, как и контракт сервиса с суффиксом Callback.

```
class Program {

    static void Main(string[] args) {
        InstanceContext ctx = new InstanceContext(new Callback());

        OrderPizzaClient proxy = new OrderPizzaClient(ctx);
        proxy.PlaceOrder("Pepperoni");
        Console.WriteLine("press enter to exit");
        Console.ReadLine();
    }
}

class Callback : IOrderPizzaCallback {

    public void TimeRemaining(int minutes)  {
        Console.WriteLine("{0} seconds remaining", minutes);
    }

    public void PizzaReady()   {
        Console.WriteLine("Pizza is ready");
    }
}
```

Важные моменты:

1. Для организации такого взаимодействия необходимы **два** отдельных контракта: контракт службы и контракт обратной связи.
2. Двусторонний контракт подразумевает наличие сессии, которая существует в течение всего периода взаимодействия. Это накладывает следующие ограничения на привязки (binding) службы WCF:
3. Стандартная привязка wsHttpBinding работать не будет!
4. Привязки, которые позволяют реализовать двусторонние коммуникации: wsDualHttpBinding, NetTcpBinding, NetNamedPipeBinding

Таким образом, при реализации службы на уровне конфигурационного файла службы или при конфигурировании хостинга службы в коде требуется указать совместимую привязку!

Порядок выполнения работы

В соответствии с индивидуальным заданием:

1. Определить функциональные границы веб-сервиса.
2. Определить контракт данных и функциональный контракт службы.
3. Создать сервис с контрактом для обмена информацией сервера и клиента с применением обратной связи.
4. Реализовать сервис с использованием WCF и протестировать его.

Задание

1. Выбрать одну из функций для демонстрации вызова метода с получением уведомления с использованием механизмов обратной связи.
2. Реализовать вышеуказанный сервис с использованием WCF и развернуть его, как управляемое клиентское приложение.
3. Создать клиентское приложение, с помощью которого продемонстрировать работу с сервисом и обработку запросов с использованием обратной связи и Callback-контракта.