Игорь Шапошников

bhv

Web-сервисы Microsoft .NET

- XML Web-сервисы
- Visual Studio .NET
- Протокол SOAP и язык WSDL
- Взаимодействие с базами данных
- Создание и использование Web-сервисов

МАСТЕР

Игорь Шапошников

Web-сервисы Microsoft .NET

Санкт-Петербург
«БХВ-Петербург»
2002

УДК 681.3.06
ББК 32.973
Ш24

Шапошников И. В.

Web-сервисы Microsoft .NET. — СПб.: БХВ-Петербург, 2002. — 336 с.: ил.

ISBN 5-94157-199-2

Книга посвящена одной из самых интересных частей новой технологии Microsoft .NET — разработке XML Web-сервисов. Рассматриваются основные приемы создания Web-сервисов, вопросы интеграции их с серверами баз данных на основе технологии ADO.NET, процедуры создания распределенных приложений на базе Web-сервисов, а также применение XML Web-сервисов на практике. Приводятся сведения о языках WSDL и SOAP, с помощью которых осуществляется разработка сервисов и клиентских приложений. Книга насыщена большим количеством авторских примеров действующих Web-сервисов с подробными комментариями.

Для программистов, разработчиков и администраторов Web-сайтов

УДК 681.3.06
ББК 32.973

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Анна Кузьмина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Елена Дудко*
Корректор	*Сергей Минин*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 17.07.02.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 27,09.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9-я линия, 12.

ISBN 5-94157-199-2

Содержание

Даниленко Ольге

"Сентябрьских яблок
Тонкая кислинка
И губ твоих"

Введение

Эта книга посвящена одной из самых интересных частей новой технологии Microsoft .NET. Как вы уже поняли из наименования книги, речь пойдет о Web-сервисах, создаваемых при помощи этой технологии. Сама технология Microsoft .NET является несомненным новшеством, а Web-сервисы можно назвать ее изюминкой, так называемой режущей гранью (cutting edge). Более подробно о них будет рассказано в первой главе, и потом на протяжении всей книги мы будем узнавать о них все больше и больше, но уже сейчас их можно охарактеризовать вкратце. Сервисы — это приложения, которые функционируют на серверах с установленной поддержкой Microsoft .NET и осуществляют коммуникацию с клиентскими приложениями при помощи языка XML и протокола HTTP. За счет того, что используются платформенно-независимые язык XML и протокол HTTP, достигается возможность создавать клиентские приложения на любой компьютерной платформе.

Естественно, область применения подобных Web-сервисов достаточно широка, и они отлично вписываются в парадигму Сети, согласно которой доступ к документам и сервисам может осуществляться с любой компьютерной платформы. Но в отличие от статических HTML-документов, Web-сервисы позволяют гораздо эффективнее оперировать информацией и предоставляют пользователю достаточно высокую функциональность. Именно такая смесь межплатформенности (только в части клиентских приложений, естественно) и высокой функциональности позволяет создавать приложения, действующие через Сеть. Web-сервисы на базе Microsoft .NET позволяют разбивать приложение на серверную и клиентскую части, переходя к новой парадигме распространения программного обеспечения, когда пользователю передается не само приложение, а всего лишь доступ к нему.

Но перейдем к рассмотрению структуры книги. *Первая глава* посвящена постановке задачи. Мы узнаем, что же представляет собой концепция Microsoft .NET, и что такое Web-сервисы. Во *второй главе* мы рассмотрим создание простейшего Web-сервиса и средства доступа к нему с помощью двух

достаточно простых клиентских приложений. В *третьей главе* рассматривается язык WDSL, при помощи которого описывается конфигурация Web-сервисов. Это очень важный аспект разработки Web-сервисов, так как только на основе описания конфигурации сервиса клиентское приложение сможет получить доступ к функциям "незнакомого" сервиса. *Четвертая глава* посвящена рассмотрению языка SOAP, при помощи которого сервисы и клиенты обмениваются данными. В *пятой главе* мы разберем механизм разработки клиентских приложений, которые используют язык SOAP. Из *шестой главы* мы узнаем, как Web-сервисы могут интегрироваться с серверами баз данных. А *седьмая глава* посвящена изучению некоторых приемов разработки Web-сервисов.

Следующие три главы демонстрируют возможности применения Web-сервисов для решения различных задач. Так, в *восьмой главе* показано, как Web-сервис может функционировать в роли основы решения класса B2B. В *девятой главе* Web-сервис будет использоваться для координирования работы пиринговой сети. А *десятая глава* предназначена для рассмотрения процедуры создания на основе Web-сервисов распределенных приложений.

Итак, мы уже знаем, про что эта книга. Мы уже знаем, какова ее структура. Самое время перейти к работе.

Глава 1

Основы Web-сервисов

Microsoft .NET

Что же такое Microsoft .NET? Это новая технология Microsoft, которая предоставляет разработчику возможность достаточно легкого решения стандартных проблем, преследующих его в Интернете. Если же говорить более точно, то Microsoft .NET — это присоединяемая среда периода выполнения (runtime), функционирующая в операционных системах Windows 2000 и Windows XP. Основой этой технологии является .NET Framework, та самая среда, в которой и выполняются программы .NET. Эта среда выполнения является некоей надстройкой над операционной системой и предоставляет разработчику некоторые дополнительные возможности и удобства. Среди наиболее известных дополнительных возможностей — автоматический сборщик "мусора" и упрощенное развертывание как самой среды, так и приложений. Плюс ко всему эта среда предоставляет новый усовершенствованный способ доступа к базам данных — ADO.NET.

К среде, естественно, прилагается SDK.NET (Software Development Kit, средство для разработки программного обеспечения, не включающее в себя интегрированную среду разработки), позволяющее разрабатывать приложения, функционирующие в .NET Framework. Впрочем, гораздо удобнее разрабатывать эти приложения в оболочке Visual Studio .NET. Тем более, что при работе в этом средстве разработки программист может использовать один из трех языков — Visual Basic .NET, Visual C++ .NET и Visual C# .NET.

Объединение трех различных языков программирования под одной оболочкой стало возможным благодаря еще нескольким концепциям Microsoft .NET. Все эти концепции опираются на понятие *управляемого кода* (managed code). Приложения, написанные при помощи этого управляемого кода, исполняются не сразу в операционной системе, а в среде CLR (Common Language Runtime). Эта среда, как мы уже отмечали, является некоторой надстройкой над операционной системой и предоставляет приложениям несколько большие возможности, нежели сама Windows.

Однако, как мы помним, чтобы программы выполнялись в среде CLR, необходимо, чтобы они были написаны на стандартном языке MSIL (Microsoft Intermediate Language). Любое средство разработки приложений в среде Microsoft .NET обязано компилировать приложения именно в формат MSIL, а не только в исполняемый код операционной системы. Именно поэтому в рамках Visual Studio .NET объединено три языка.

Но необходимо понимать, что MSIL-код не может выполняться операционной системой напрямую. Его необходимо еще перевести в машинный код. Эту операцию назвали компиляцией "по требованию" (just-in-time). Соответственно, компилятор, переводящий MSIL-код в машинный код для специфичной операционной системы, называется *джиттером* (just-in-time compiler, JITter). Основное достоинство этой концепции состоит в том, что подобные компиляторы-джиттеры можно сделать практически для любой платформы. Таким образом, создав код приложения один раз в среде Microsoft .NET, его при помощи джиттеров можно безболезненно портировать под все платформы, для которых разработаны джиттеры.

Вам ничего эта модель не напоминает? Да, конечно, то же самое декларировала компания Sun, но только применительно к Java. Однако, несмотря на все усилия апологетов Java, она не получила столь широкого распространения, на которое они рассчитывали. И теперь межплатформенность заявлена в Microsoft .NET. На самом деле, на данный момент джиттеры созданы лишь для операционных систем семейства Windows, да и то не для всех. Кстати, Microsoft пока даже и не объявила о планах создания джиттеров для систем, не входящих в семейство Windows. Интересно, почему?

Так как все программы в концепции Microsoft .NET из исходных кодов компилируются в MSIL-код, то теоретически все языки .NET должны предоставлять разработчику одинаковый набор системных функций. Таким образом, унифицируется и технология разработки программ. Действительно, доселе различные языки программирования имели разные возможности, например, для доступа к COM-объектам, которые являются действительно хорошим средством расширяемости и модульности приложений. В языках .NET набор системных функций одинаков, поэтому доступ к тем же COM-объектам будет производиться одинаково как из кода Visual Basic, так и из кода, написанного на C++.

Естественно, подобная функциональность CLR реализуется при помощи дополнительного пространства имен System. Соответственно, разработчик получает несколько более высокую структурированность используемых системных функций и избавление от конфликтов имен. Идеология пространств имен, получившая широкое распространение еще со времен распространения XML, действительно позволяет избавиться от конфликтов имен функций, что дает возможность назначать им достаточно короткие и осмысленные имена.

К остальным достоинствам технологии Microsoft .NET можно отнести прозрачное взаимодействие с объектами COM, улучшенное управление версиями, разграничение доступа к системным функциям для программ (а не только для пользователей), автоматическую сборку "мусора" в куче памяти, и многое-многое другое. Уже просто знакомясь со списком нововведений Microsoft .NET, хочется начать работать в этой системе. Ну, так и не отказывайте себе в удовольствии.

Web-сервисы

Web-сервисы являются одной из самых интересных возможностей, которые предоставляет среда Microsoft .NET. Если говорить точнее, то сама идея сервисов в Сети не нова, но никогда их создание не было таким простым, как с использованием средства разработки Visual Studio .NET. Однако вернемся к самим Web-сервисам.

Давайте вспомним основной принцип работы WWW, когда документы создаются в едином формате HTML, а клиентские приложения-браузеры, функционирующие на самых различных компьютерных платформах, почти одинаково отображают эти документы по заранее оговоренным правилам. Однако это были всего лишь статические документы. Для придания им динамики требовалось использование различных средств программирования, действующих как на стороне клиента, так и на стороне сервера.

Если разработчик применял скриптовые языки, функционирующие на стороне клиента, то область их действия была ограничена отображаемым документом, а в качестве входных данных использовались либо действия пользователя, либо информация, хранящаяся на его машине. Связывалось это ограничение с тем, что скриптовые языки не получали какой-либо дополнительной информации от WWW-сервера, с которого был загружен документ, вместе с которым и были получены скрипты.

Если на сайте требовалось использовать несколько более серьезные способы управления контентом, приходилось применять исполняемые модули, функционирующие на самом сервере. Только таким образом можно было осуществлять доступ к базам данных, и на основе полученной информации формировать страницы, передаваемые затем удаленному пользователю. Только на сервере удавалось обеспечить поддержку сеансов работы и идентификацию пользователей, столь необходимые во всех приложениях электронной коммерции.

Естественно, технологий Web-программирования было немало. Для этой цели использовались и классические языки, такие как C++, и технологии, специально ориентированные на Web, такие как ASP. Однако все они в своем разнообразии имели одну общую черту — конечный вывод документов производился в формате HTML.

Если учитывать общую скорость прогресса в компьютерной индустрии, то можно признать, что долгожитель HTML достаточно долго использовался, и обещанное вытеснение его XML произойдет еще не скоро. Однако нельзя не видеть, что XML все сильнее и сильнее проникает в мир WWW. И основное ограничение заключается в том, что стандартные браузеры не в состоянии адекватно отображать содержимое XML-документов.

Итак, проблема локализована. Было бы неплохо заменить HTML более функциональным языком XML, облегчить взаимодействие между пользователем и сервером. Однако одна из основных проблем — стандартные браузеры, которые отображают только HTML и не в состоянии пересылать на WWW-сервер что-либо, кроме стандартных HTTP-пакетов. Следовательно, для получения большей функциональности необходимо отойти от стандартных браузеров.

Естественно, этот шаг повлечет за собой определенные проблемы, так как для каждого ресурса будет нужно отдельное клиентское приложение, но следует признать, что специализированные клиенты потребуются только для тех немногих ресурсов, которым необходима какая-либо особая функциональность. Для обычных Web-сайтов с небольшой функциональностью стандартных браузеров более чем достаточно.

Система Web-сервисов и обособленных клиентов нужна только для ресурсов с чрезвычайно сложной функциональностью. Первые ласточки среди таких ресурсов уже появляются. Это серверы различных пиринговых сетей, таких, как, например, небезызвестный Napster, или серверы распределенных вычислений (знаменитая программа CETI@Home является отличным примером). Однако написание подобных проектов являлось до недавнего времени достаточно нетривиальной и трудоемкой задачей. Заметили? Мы сказали: "До недавнего времени".

Технология Microsoft .NET позволяет писать сервисы достаточно легко. По сравнению с тем, что было раньше — чрезвычайно легко. Конечно, написание клиентских приложений для этих Web-сервисов будет несколько более сложной задачей, по сравнению с разработкой самих сервисов, но тоже не такой уж и трудной. Вы увидите.

Сервисы .NET обладают весьма высокой функциональностью и теоретически позволяют разбивать обычные приложения на две части — клиентскую и серверную. Если взять, например Microsoft Excel, то клиентская часть может включать в себя только средства отображения информации, подобно некоему специализированному табличному браузеру, а вся логика приложения может находиться в серверной части.

Конечно, для функционирования подобных приложений пользователю потребуется иметь весьма "толстый" канал связи с сервером, но в условиях локальной сети эта проблема может быть легко решена. А использование модели "разделенных приложений" позволяет достаточно легко изменить

политику лицензирования программного обеспечения и начать новый виток борьбы с пиратами (впрочем, судя по всему, эта борьба в ближайшее время не закончится).

В общем случае, Web-сервисы принимают и передают информацию на языке XML. Этот язык, естественно, является открытым, а не проприетарным стандартом. Также следует учитывать, что XML, как и его предшественник HTML, не зависит от платформы и операционной системы. Сочетание этих двух факторов позволяет разработчикам свободно создавать самые различные клиентские приложения, функционирующие на разных компьютерных платформах. То есть, одному сервису может соответствовать несколько клиентских приложений.

XML-документы, пересылающиеся от сервера к клиенту и обратно, передаются по протоколу HTTP, который пропускается практически всеми брандмауэрами, что также прибавляет привлекательности идее Web-сервисов.

Следует отметить и тот факт, что Web-сервисы являются самодокументируемыми. То есть любое клиентское приложение может получить информацию о структуре сервиса, о его функциональности и правилах вызова функций, поддерживаемых Web-сервисом.

Web-сервисы способны передавать и получать информацию тремя способами: с применением методов GET и POST стандартного протокола HTTP или при помощи языка SOAP, который является производным от XML. Первые два варианта разрешают использовать в качестве клиентского приложения стандартный браузер, но необходимо отдавать себе отчет, что это далеко не идеальный вариант. Во-первых, при помощи браузера весьма трудно организовать вызов всех функций достаточно сложного сервиса. Разработчику необходимо исследовать структуру сервиса заранее, перед тем, как создавать Web-страницы для доступа к сервису. Так что в этом случае мы теряем преимущества самодокументируемости. Во-вторых, следует помнить, что вывод информации все равно будет идти в "чистом" XML, который браузер не сможет адекватно отобразить. Поэтому для работы с Web-сервисами, обладающими достаточно серьезной функциональностью, следует все же ориентироваться на язык SOAP. Его структуру мы разберем в *главе 4*.

Однако пришло время перейти от общих слов к детальному рассмотрению концепции и структуры Web-сервисов.

Структура сервиса

Рассмотрим сначала структуру взаимодействия Web-сервиса .NET с клиентским приложением. При первом обращении клиентского приложения, которое "не знает" структуры сервиса, тот передает клиенту информацию о поддерживаемых им функциях и параметрах, необходимых для вызова этих функций. Передача этой информации о структуре называется "контрактом".

Подобная информация извлекается из описания сервиса, которое для каждого сервиса создается на языке WSDL. Естественно, языку WSDL и структуре описания сервиса мы уделим особое внимание в *главе 3*.

После того как клиент получает этот "контракт", он анализирует полученную информацию и на ее основе формирует запросы к сервису, используя вызовы его функций. А затем идет просто обмен запросами к сервису и ответами на них, которые формируются на языке XML. Естественно, чтобы Web-сервисы могли получать и отправлять информацию, для чего, как мы знаем, используется протокол HTTP, на серверной машине должен быть установлен WWW-сервер IIS. Но об этом будет сказано в *разд. "Технические требования"* данной главы.

Теперь перейдем к рассмотрению внутренней структуры сервиса. Web-сервисы действуют на платформе Microsoft .NET, следовательно, они вполне могут использовать все возможности этой платформы, такие, как доступ к базам данных, отслеживание своего состояния и прочие системные функции.

Следует обратить внимание, что до появления платформы Microsoft .NET разработка подобных Web-сервисов тоже была возможна, но требовала куда больших затрат сил и времени разработчика. Средство разработки Visual Studio .NET позволяет создавать Web-сервисы чрезвычайно легко. Дело в том, что при создании сервисов в этом средстве разработки среда берет на себя управление модулем, который ответственен за связь с клиентским приложением, форматирование отправляемых данных по правилам XML, документирование сервиса и многие другие обязательные, но рутинные вещи. Таким образом, разработчик фокусируется на программировании именно логики и функциональности сервиса, а рутину передает среде разработки. Подобная концепция весьма напоминает ситуацию, когда появились первые визуальные средства разработки приложений RAD (Rapid Application Development), которые избавили программистов от разработки стандартных интерфейсов, предоставив им визуальные средства проектирования. Теперь то же самое произошло и при создании Web-сервисов, что позволит разрабатывать их намного быстрее. Однако перед тем, как мы перейдем к созданию своего первого сервиса, следует уточнить, какие системные требования предъявляются к той машине, на которой будет функционировать Web-сервис.

Технические требования

Естественно, технические требования к серверу, на котором будут развернуты Web-сервисы, вытекают из основных нужд этих сервисов. Итак, сервер, под которым мы понимаем совокупность аппаратных компонентов и программного обеспечения, должен обеспечивать выполнение следующего набора базовых функций:

- Возможность получать входящие запросы по протоколу HTTP.
- Возможность осуществлять аутентификацию и авторизацию удаленных пользователей.
- Изолировать сервисы друг от друга, чтобы каждый имел свое собственное адресное пространство в оперативной памяти, и ошибка в одном сервисе не повлекла бы за собой сбои в остальных сервисах, выполняющихся одновременно с проблемным сервисом.
- Предоставление администратору средств для развертывания сервисов, а также для контроля и наблюдения за ними.
- Возможность управлять ресурсами, которые выделяются каждому сервису.

Естественно, некоторые сервисы могут предъявлять повышенные и нестандартные требования к машине, на которой они установлены, но базовый перечень требований мы привели. Остальное администратор сервера может делать самостоятельно.

Для программирования сервиса неплохо также иметь средство разработки, которое позволяло бы программисту достаточно прозрачно работать с языком XML, так как иначе ему придется писать свой парсер.

Теперь, когда список требований определен, настало время узнать, какая платформа им удовлетворяет. Так как мы собираемся при разработке сервисов опираться на технологию Microsoft .NET, естественно, нас будут интересовать операционные системы семейства Windows. Поэтому отметим, что перечисленные требования начали поддерживаться в этих операционных системах, начиная с Windows 2000. Само собой, кроме нее установленным критериям удовлетворяют Windows XP и Windows .NET. При этом следует помнить, что во всех этих системах необходимо устанавливать и активировать WWW-сервер IIS, который позволяет принимать и отправлять информацию по протоколу HTTP.

Естественно, все перечисленные системы имеют API, который позволяет приложениям использовать следующие возможности:

- доступ к COM-объектам, которые могут предоставлять функциональность для доступа к базам данных и элементам бизнес-логики;
- ADO, OLE DB и ODBC для доступа к базам данных;
- MSXML для анализа, разбора и создания сообщений на языке XML;
- Возможности технологии ASP для приема запросов по протоколу HTTP.

Мы здесь не учитываем потенциал .NET Framework, которая предоставляет еще больше возможностей, но смысл уже понятен. Для разработки Web-сервисов нам придется использовать именно систему Windows 2000 или старше. Конечно, для более комфортной работы следует установить сре-

ду .NET, а для разработки было бы неплохо использовать Visual Studio .NET. Впрочем, можно обойтись без среды разработки Visual Studio .NET и пользоваться для написания кода стандартным Блокнотом, но честное слово, с ней работается настолько легче, что мы искренне советуем обзавестись этим средством RAD.

В качестве взаимодействующих с Web-сервисами средств перечислим следующие продукты:

- Application Center 2000 для развертывания приложений и управления ими;
- BizTalk Server 2000 для связи Web-сервисов с уже готовыми приложениями, обменивающимися информацией на языке SOAP;
- Commerce Server 2000 для создания сервисов, работающих в сфере электронной коммерции;
- Internet Security and Acceleration Server 2000, позволяющий обезопасить работу в Сети при помощи встроенного брандмауэра и управляющий кэшированием данных на сервере;
- Mobile Information Server 2001, позволяющий предоставлять доступ к данным с мобильных телефонов и других устройств, воспринимающих WML;
- SQL Server 2000 для связи сервисов с базами данных.

Совершенно необязательно устанавливать на сервер все эти продукты. Каждый администратор сам выберет необходимые компоненты из этого списка или добавит еще некоторые средства. Чаще всего для работы Web-сервисов ограничиваются SQL-сервером SQL Server 2000 и Commerce Server 2000.

Конечно, подобные средства разработки и операционные системы выставляют некоторые требования к аппаратному обеспечению, на котором они установлены. В том случае, если используется весь вышеперечисленный набор, необходим будет действительно мощный компьютер с быстрым дисковым массивом, объемом оперативной памяти не менее гигабайта и процессором уровня Pentium 4. Но это крайний случай. Обычно требуется установить операционную систему, .NET Framework и SQL Server 2000. Для этих целей вполне можно обойтись достаточно средней по сегодняшним меркам платформой. А именно: 256 Мбайт оперативной памяти, обычный качественный жесткий диск и процессор с частотой около 700 МГц для обработки запросов к базам данных вполне подойдут для сервера, на котором будут базироваться искомые Web-сервисы. Впрочем, все зависит не только от внутренних задач, но и от популярности сервиса, а значит, и от количества одновременных подключений пользователей.

Конечно же, не стоит даже особого упоминания, что к любой конфигурации сервера необходима хорошая сетевая плата известного производителя на 100 Мбит. Сама связь с Сетью ни в коем случае не должна быть "узким местом" в производительности сервера.

Теперь, после того как мы рассмотрели основные требования к системе, пришла пора рассмотреть пример создания самого первого и самого простого Web-сервиса. Этакий "Hello, World", но с учетом нашей специфики.

Глава 2

Создание простейшего сервиса

Сервис

Для начала попробуем создать самый простой Web-сервис, который по запросу пользователя будет выдавать текущую дату или комбинацию даты и времени. Для этого в среде разработки Visual Studio .NET следует выполнить команду меню **File | New | Project** и в появившемся диалоговом окне **New Project** в наборе **Templates** выделить значок **ASP.NET Web Service**. В поле **Name** следует указать наименование создаваемого проекта. Укажем для начала имя `MyService`. После того, как все необходимые приготовления средой разработки будут сделаны, на основном рабочем поле появятся две новые страницы. Одна будет предназначена для разработки визуального дизайна, а на второй будет располагаться код нашего сервиса. Естественно, у создаваемого сервиса не может быть внешнего вида как такового, ему нечего отображать, поэтому страницу для дизайна можно спокойно закрыть.

В состав одного проекта может входить несколько отдельных сервисов. Для нашего примера потребуется всего один Web-сервис, заготовка для которого создается средой разработки Visual Studio. NET автоматически, поэтому ничего изменять не потребуется, и мы можем сразу перейти на страницу с наименованием Service1.asmx.vb. На этой странице мы и разместим код нашего Web-сервиса. Он приведен в листинге 2.1.

Листинг 2.1

```
Imports System.Web.Services

Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "
```

```
    Public Sub New()
        MyBase.New()
        InitializeComponent()
    End Sub

    Private components As System.ComponentModel.Container

    <System.Diagnostics.DebuggerStepThrough()> Private
Sub InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    End Sub

#End Region

    <WebMethod()> Public Function MyDate(ByVal ShowTime As Boolean) As
String
        Dim MD As DateTime
        If ShowTime Then
            MyDate = MD.Now
        Else
            MyDate = MD.Today
        End If

    End Function

End Class
```

Из всего этого кода лишь малая часть написана разработчиком. Это функция `MyDate`, расположенная в конце листинга. Ее нам и следует рассмотреть.

Из листинга видно, что объявление функции предваряется тегом `<WebMethod()>`. Он и сигнализирует компилятору, что следующая за ним функция является частью создаваемого сервиса, и результаты ее работы будут передаваться клиенту. В объявлении функции мы указали, что она будет воспринимать параметр `ShowTime` логического типа `Boolean`. В том случае, если функции передано значение `True`, клиенту будет возвращено значение, в которое входит и текущая дата и время вызова функции. Для этого мы используем свойство `Now`, входящее в состав типа `DateTime`. Если же функция

в качестве параметра получает значение `False`, то клиентскому приложению возвращается только дата, что производится при помощи свойства `Today`.

Создание Web-сервиса не явилось сложной задачей. Теперь рассмотрим, как он действует. После компиляции к нему можно обратиться даже из браузера. На рис. 2.1 приведен внешний вид Web-страницы, которая генерируется Web-сервером при попытке обращения к Web-сервису.

Рис. 2.1. Внешний вид Web-страницы, соответствующей Web-сервису

На этой Web-странице расположена ссылка на формальное описание структуры Web-сервиса, а также перечислены все функции, поддерживаемые Web-сервисом. В нашем случае, естественно, указана лишь одна функция `Mydate`. Наименование каждой функции также является гиперссылкой, нажав на которую можно перейти к Web-странице, позволяющей воспользоваться этой функцией. Так мы и поступим. После щелчка мышью на ссылке, указывающей на функцию `MyDate`, в браузере будет отображена Web-страница, предоставляющая доступ к этой функции. Внешний вид этой Web-страницы показан на рис. 2.2.

На этой странице расположено поле ввода, в котором пользователь может указать значение параметра, передаваемого функции, и кнопка **Invoke**, при нажатии на которую этот параметр будет передан функции сервиса, и в браузер будет отправлен результат работы сервиса.

На самом деле, содержимое рассматриваемой Web-страницы не ограничивается органами управления, показанными на рисунке. На ней также показаны блоки кода, которые следует использовать клиентским приложениям для связи с сервисом. В силу того, что эти блоки кода занимают достаточно большое пространство, они не приведены на иллюстрации. Но после того как вы создадите Web-сервис и обратитесь к нему через браузер, можно будет увидеть эти инструкции. Следует отметить, что на Web-странице приве-

дены инструкции для создания клиентов на основе методов GET и POST, а также при помощи языка SOAP. Забота о разработчике налицо. Например, текст запроса к Web-сервису на языке SOAP выглядит следующим образом:

```
POST /MyService/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/MyDate"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <MyDate xmlns="http://tempuri.org/">
      <ShowTime>boolean</ShowTime>
    </MyDate>
  </soap:Body>
</soap:Envelope>
```

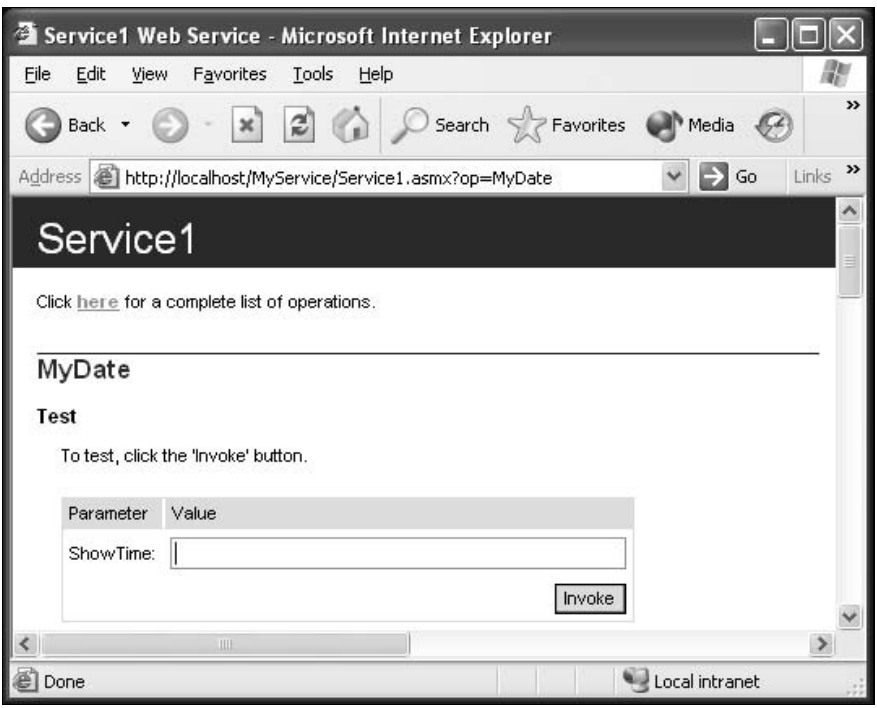

Рис. 2.2. Web-страница, предоставляющая пользователю доступ к функции MyDate

Но вернемся к тестированию созданного Web-сервиса. Как уже было сказано, в поле текстового ввода необходимо внести значение параметра `ShowTime`. Кстати, наименование параметра также указано на странице, рядом с соответствующим полем ввода. Тип параметра указан в блоках кода для клиентских приложений.

Итак, мы знаем, что наша функция воспринимает логический параметр типа `Boolean`. Поэтому в поле ввода можно ввести значение `True`, что мы и сделаем. После того как искомое значение будет введено в текстовое поле и нажата кнопка **Invoke**, браузер отобразит Web-страницу с результатом работы функции сервиса. Внешний вид этой страницы показан на рис. 2.3.

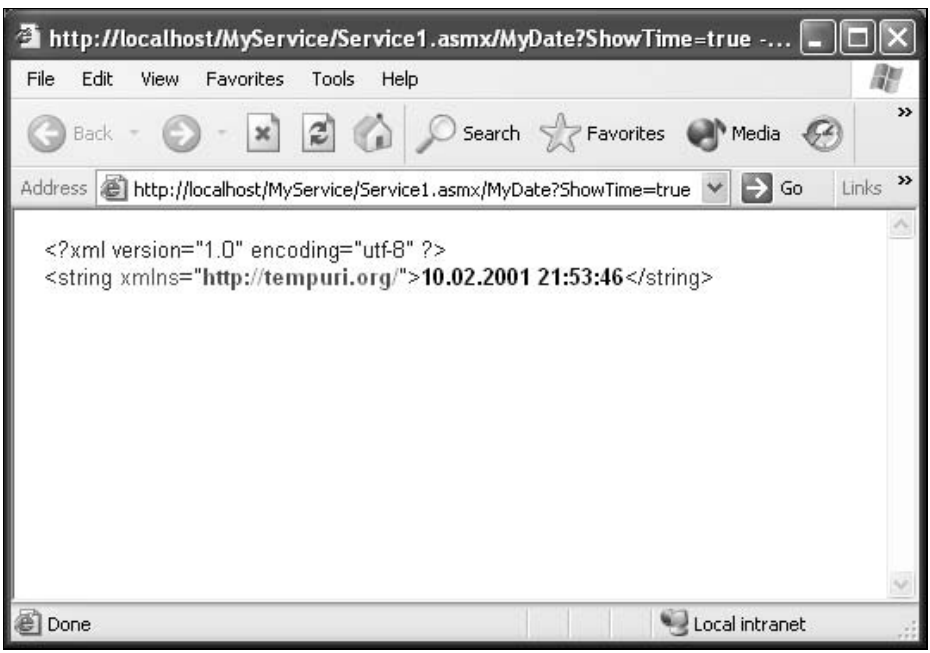

Рис. 2.3. Web-страница с результатом работы функции

Легко заметить, что на последней Web-странице отображен код XML-документа. Этот XML-документ и является результатом работы созданного нами Web-сервиса. Теперь клиентское приложение получит этот XML-документ, само обработает его и выдаст результат пользователю или использует этот результат для собственных целей. Таким образом, практически без всяких усилий с нашей стороны, был создан сервис, который сам документирует себя, принимает данные тремя различными способами по протоколу HTTP и возвращает результаты работы на языке XML. Это действительно удобно.

Теперь обратимся к вопросу самодокументированности Web-сервисов. Дело в том, что при создании Web-сервиса компилятор одновременно создает WSDL-файл, в котором описана структура сервиса. Именно на основе этого файла клиенты получают информацию о функциональности сервиса.

Структуру языка WSDL мы будем рассматривать в *главе 3*, поэтому листинг пока не приводится.

А теперь, когда мы знаем, как создавать сервисы, следует рассмотреть приемы создания клиентских приложений. Но об этом — в следующем разделе.

Клиентская часть

Как мы знаем, клиентские приложения могут использовать три варианта передачи запросов Web-сервису. Это методы `GET` и `POST` протокола HTTP и язык SOAP. Здесь мы рассмотрим создание клиентских приложений, действующих по первым двум вариантам.

Если обращаться к методам `GET` и `POST`, то следует признать, что нет нужды создавать отдельные приложения, которые бы формировали HTTP-запросы по правилам этих методов. У нас уже есть такое приложение — обычный браузер. Достаточно лишь создать для него соответствующие Web-страницы. Это будет намного быстрее и проще, нежели разрабатывать отдельные приложения, которые должны будут самостоятельно создавать HTTP-запросы, отправлять их на сервер, получать ответ на языке XML и извлекать из него данные. Поэтому начнем с создания обычных Web-страниц.

Первый созданный нами клиент для подключения к Web-сервису будет применять метод `GET`. Как известно, при использовании этого метода вся информация, пересылаемая на сервер, передается через строку запроса вместе с URL. То есть на клиентской Web-странице достаточно будет создать ссылку на сервис и в URL поместить необходимую для вызова функции информацию. Так как наш сервис воспринимает один параметр логического типа, нам потребуется создать на клиентской Web-странице две ссылки: одну со значением параметра `True`, а другую — с `False`. Код этой Web-страницы приведен в листинге 2.2.

Листинг 2.2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <title> GET-клиент</title>
  </head>
```

```
  <body>
  <p>Используйте гиперссылки для работы с web-сервисом</p>
<p><a
href="http://localhost/MyService/Service1.asmx/MyDate?ShowTime=True">
Получить дату и время</a></p>
<p><a
href="http://localhost/MyService/Service1.asmx/MyDate?ShowTime=True">
Получить только дату</a></p>
  </body>
</html>
```

Внешний вид этой Web-страницы при отображении ее в браузере показан на рис. 2.4.

Основное внимание в этом листинге следует уделить гиперссылкам, а точнее, значениям атрибута `href` тегов `<a>`. В URL после указания имени файла Service1.asmx указывается наименование функции `MyDate`. После наименования функции ставится вопросительный знак, отделяющий параметры от основного URL. А уже после символа вопросительного знака указывается наименование параметра и его значение. Именно таким образом передаются данные при помощи метода `GET`.

Так как параметр может принимать два значения, то мы и создаем две ссылки, у которых URL отличаются друг от друга только в последней части, где после знака равенства указывается значение параметра.

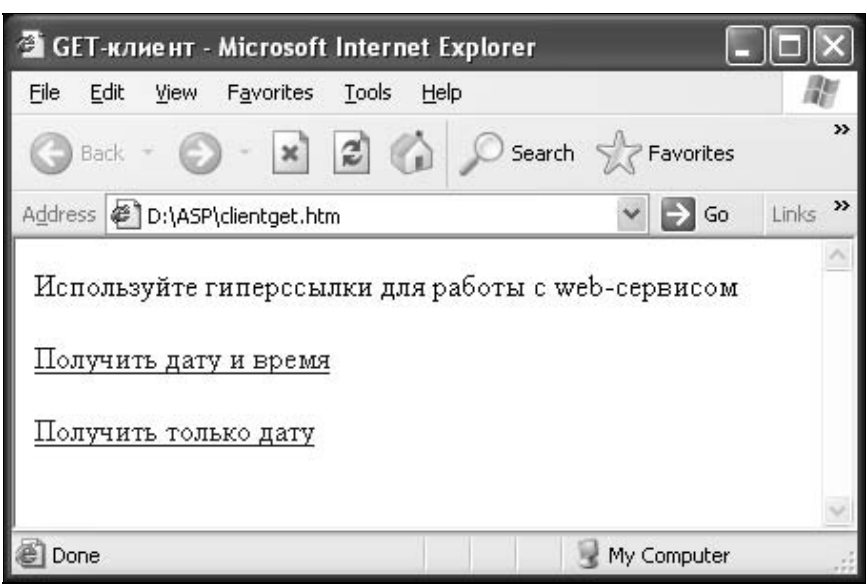

Рис. 2.4. Клиентская Web-страница, работающая с методом `GET`

Теперь перейдем к созданию клиентской Web-страницы, передающей запрос при помощи метода POST. Как известно, метод POST позволяет передавать данные, введенные пользователем в HTML-форме. Следовательно, нам придется на Web-странице создать форму.

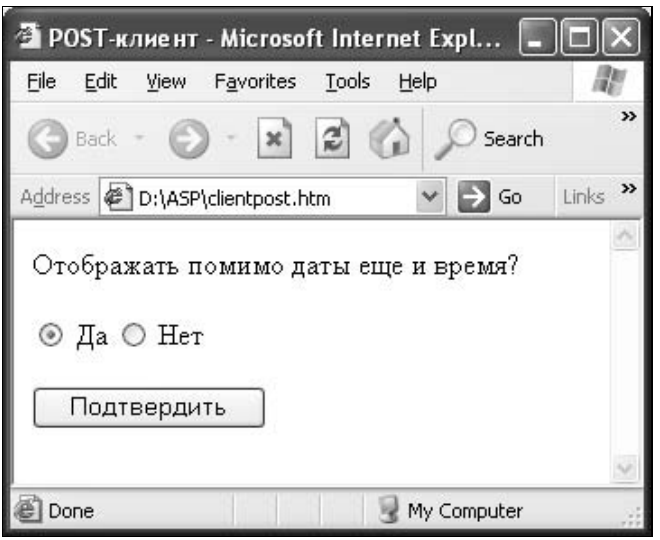

Рис. 2.5. Клиентская Web-страница, работающая с методом POST

Так как параметр имеет логический тип, будет правильно создать группу из двух переключателей. Передаваемое значение может быть или True или False, и именно эти значения будут возвращать переключатель. Однако помимо значения параметра, необходимо передать Web-сервису и его наименование. Как известно, при передаче данных от браузера на сервер, они указываются в виде последовательности пар "имя=значение", где перед знаком равенства указывается наименование органа ввода данных. Следовательно, для передачи правильной пары нам необходимо группу переключателей назвать ShowTime. В результате HTML-код клиентской Web-страницы, передающей информацию при помощи метода POST, будет выглядеть приблизительно так, как это показано в листинге 2.3.

Листинг 2.3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <title> POST-клиент</title>
```

```
  </head>
  <body>
  <form METHOD="POST"
ACTION="http://localhost/MyService/Service1.asmx/MyDate">
  <p>Отображать помимо даты еще и время?</p>
    <p><input TYPE="RADIO" NAME="ShowTime" VALUE="true" CHECKED> Да
    <input TYPE="RADIO" NAME="ShowTime" VALUE="false"> Нет
    </p>
  <input TYPE="SUBMIT" VALUE="Подтвердить">
</form>
 </body>
</html>
```

Внешний вид этой Web-страницы показан на рис. 2.5.

Итак, мы рассмотрели создание клиентов на основе Web-страниц. Если пользоваться языком SOAP для коммуникации между клиентами и сервисом, придется создавать отдельное приложение, что уже выходит за пределы темы данной главы. Поэтому пока мы остановимся на этом.

Глава 3

Язык WSDL

Краткий обзор

Итак, мы знаем, что Web-сервисы должны документировать свою структуру, чтобы клиентские приложения могли обращаться к их функциям. Для этого и используется язык WSDL (Web Services Description Language). Этот язык создан на базе языка XML, знать который следует любому Web-разработчику, так как он предоставляет достаточно много новых возможностей, облегчающих создание в Сети больших и легко управляемых проектов.

На данный момент WWW-консорциумом (W3C) утверждена версия WSDL 1.1. Именно ей мы и займемся в этой главе. Естественно, мы не станем разбирать официальную спецификацию языка, это заняло бы слишком много времени. Достаточно лишь на примерах познакомиться с его синтаксисом.

В данном разделе мы лишь рассмотрим схему типичного WSDL-документа. Необходимо признать, что в своей логической форме любые XML-документы, к коим, несомненно, следует отнести и WSDL-документы, достаточно сильно отличаются от HTML-документов, так знакомых любому Web-разработчику. Если HTML-документы описывали лишь *отображение* некоего блока информации, то XML описывает прежде всего логическую структуру. Поэтому и WSDL-документы мы будем рассматривать с точки зрения их логической структуры.

Теперь вернемся к первому примеру Web-сервиса, который мы разбирали в предыдущей главе. Так как для разработки было использовано RAD-средство Visual Studio .NET, нам не придется заботиться о собственноручном написании WSDL-файла для этого сервиса. Он был создан автоматически.

Файл WSDL (Web Service Descriptor Language) является обычным XML-файлом, и доступ к нему мы можем получить даже из браузера. Если взглянуть на рис. 2.1, то в верхней части Web-страницы будет видна ссылка **Service Description**, которая отсылает пользователя как раз к упомянутому WSDL-файлу. Код WSDL-файла, соответствующего нашему Web-сервису, приведен в листинге 3.1.

Листинг 3.1

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:s0="http://tempuri.org/" targetNamespace="http://tempuri.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema attributeFormDefault="qualified"
elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <s:element name="MyDate">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="ShowTime"
type="s:boolean" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="MyDateResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="MyDateResult"
nillable="true" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </types>
  <message name="MyDateSoapIn">
    <part name="parameters" element="s0:MyDate" />
  </message>
  <message name="MyDateSoapOut">
    <part name="parameters" element="s0:MyDateResponse" />
  </message>
  <message name="MyDateHttpGetIn">
    <part name="ShowTime" type="s:string" />
  </message>
```

```
  <message name="MyDateHttpGetOut">
    <part name="Body" element="s0:string" />
  </message>
  <message name="MyDateHttpPostIn">
    <part name="ShowTime" type="s:string" />
  </message>
  <message name="MyDateHttpPostOut">
    <part name="Body" element="s0:string" />
  </message>
  <portType name="Service1Soap">
    <operation name="MyDate">
      <input message="s0:MyDateSoapIn" />
      <output message="s0:MyDateSoapOut" />
    </operation>
  </portType>
  <portType name="Service1HttpGet">
    <operation name="MyDate">
      <input message="s0:MyDateHttpGetIn" />
      <output message="s0:MyDateHttpGetOut" />
    </operation>
  </portType>
  <portType name="Service1HttpPost">
    <operation name="MyDate">
      <input message="s0:MyDateHttpPostIn" />
      <output message="s0:MyDateHttpPostOut" />
    </operation>
  </portType>
  <binding name="Service1Soap" type="s0:Service1Soap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <operation name="MyDate">
      <soap:operation soapAction="http://tempuri.org/MyDate"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
```

```
    </operation>
  </binding>
  <binding name="Service1HttpGet" type="s0:Service1HttpGet">
    <http:binding verb="GET" />
    <operation name="MyDate">
      <http:operation location="/MyDate" />
      <input>
        <http:urlEncoded />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
  <binding name="Service1HttpPost" type="s0:Service1HttpPost">
    <http:binding verb="POST" />
    <operation name="MyDate">
      <http:operation location="/MyDate" />
      <input>
        <mime:content type="application/x-www-form-urlencoded" />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
  <service name="Service1">
    <port name="Service1Soap" binding="s0:Service1Soap">
      <soap:address location="http://localhost/MyService/Service1.asmx"
/>
    </port>
    <port name="Service1HttpGet" binding="s0:Service1HttpGet">
      <http:address location="http://localhost/MyService/Service1.asmx"
/>
    </port>
    <port name="Service1HttpPost" binding="s0:Service1HttpPost">
      <http:address location="http://localhost/MyService/Service1.asmx"
/>
    </port>
  </service>
</definitions>
```

Этот файл полностью описывает функциональность созданного Web-сервиса, и именно на него будут опираться клиентские приложения для установки связи с сервисом. Достаточно часто WSDL-файлы, описывающие структуру Web-сервисов, называют *контрактами*.

Теперь попробуем установить его структуру. В WSDL-файлах информация структурируется при помощи всего пяти основных элементов. Если их кратко перечислить, получится следующий список:

- `<types>`. Элемент содержит определения типов данных, используемых в данном Web-сервисе.
- `<message>`. Элемент содержит абстрактные представления функций Web-сервиса вместе с указанием типов входных и выходных данных.
- `<portType>`. Элемент содержит ссылки на искомые функции, описанные в элементе `<message>` с более подробной структурой их входных и выходных данных.
- `<binding>`. Элемент предназначен для связывания содержимого блока `<port>` с конкретными функциями Web-сервиса. В этих разделах определяются протокол и формат данных для операций, на которые указывает конкретный раздел `<port>`.
- `<port>`. Содержат адреса функций, описанных в элементах `<message>`. По сути дела, порты описывают точки вхождения в Web-сервис.
- `<service>`. Основной элемент, в котором перечисляются конкретные порты, указывающие на прямые URL доступа к функциям сервиса.

Таким образом, мы можем сказать, что первые три элемента — `<types>`, `<message>` и `<portType>` — представляют собой несколько более абстрагированный уровень описания структуры Web-сервиса, а последние два — `<binding>` и `<service>` — увязывают абстрактную логику с конкретными механизмами доступа к функциональности сервиса.

Помимо этих элементов, создающих в WSDL-документе целые разделы, есть и отдельный блок, который нельзя отнести ни к какому разделу. Он выглядит следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:s0="http://tempuri.org/" targetNamespace="http://tempuri.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Первая строка, естественно, мгновенно узнается каждым, кто знаком с языком XML. Это стандартное объявление XML-документа. Второй тег содер-