Министерство образования Республики Беларусь
Учреждение образования
«Международный государственный экологический
университет имени А. Д. Сахарова»

Факультет мониторинга окружающей среды
Кафедра автоматизированных систем обработки информации

Б. А. Тонконогов

Проектирование и разработка информационных систем средствами технологии .NET

Учебно-методическое пособие

Минск
2010

УДК 004.4
ББК 32.97
Т57

Рекомендовано к изданию НМС МГЭУ им. А. Д. Сахарова
(протокол № 2 от 24 октября 2007 г.)

Автор:

Б. А. Тонконогов, к. т. н., доцент, доцент кафедры автоматизированных систем обработки информации МГЭУ им. А. Д. Сахарова

Рецензенты:

В. Л. Ланин, д. т. н., профессор, профессор кафедры электронной техники и технологии БГУИР;

Г. М. Левин, д. т. н., с. н. с., зав. лабораторией «Исследования операций» ОИПИ НАН Беларуси

Тонконогов, Б. А.

Т57 Проектирование и разработка информационных систем средствами технологии .NET: учебно-методическое пособие / Б. А. Тонконогов. – Минск : МГЭУ им. А. Д. Сахарова, 2010. – 262 с.

ISBN 978-985-6931-28-7.

Учебно-методическое пособие предназначено для студентов старших курсов специальностей, связанных с информационными ситемами и технологиями. Рассмотрены теоретические и практические аспекты, касающиеся: 1) основ проектирования и разработки информационных систем средствами технологии .NET.; 2) разработки Windows- и Web-приложений с использованием доступа к данным при помощи технологии ADO.NET; 3) принципов разработки Web-приложений с применением технологии ASP.NET; 4) назначения и использования Web-служб; 5) навыков работы в интегрированной среде разработки Microsoft Visual Studio. Пособие может быть также использовано специалистами, занимающимися проектированием и разработкой различного рода информационных систем.

УДК 004.4
ББК 32.97

ISBN 978-985-6931-28-7

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Проектирование и развертывание распределенных систем – сложный процесс. По мере того как система все больше фрагментируется, визуализация ее структуры как единого целого становится все сложнее. Кроме того, со временем на предприятиях обычно скапливаются различные программные системы, в которых могут использоваться самые разные технологии и технические приемы. Организовать обмен данными и взаимодействие между такими системами становится все труднее. Чтобы обеспечить взаимодействие в гетерогенной среде, разработчикам все чаще требуется проектировать интерфейсы и приложения, использующие определенное взаимодействие друг с другом, в частности взаимодействие посредством определенных протоколов, лежащее в основе архитектур, ориентированных на использование различных служб. При этом проектирование таких новых приложений и обеспечение их соответствия имеющимся схемам становится очень важным.

Цель изучения дисциплины. Предметом изучения дисциплины «Разработка информационных систем средствами .NET» являются методы проектирования и разработки информационных систем.

Цель преподавания дисциплины – освоение основ технологии .NET, формирование у студентов представлений об особенностях проектирования и программной реализации информационных систем, навыков работы с интегрированной средой разработки приложений Microsoft Visual Studio и системой управления базами данных Microsoft SQL Server, а также некоторыми вспомогательными программными продуктами для настройки функционирования информационных систем для дальнейшего квалифицированного использования в учебном процессе, научных исследованиях и практической работе.

Изучение данного курса предусматривается учебными планами по специальностям 1-40 01 02-06 «Информационные системы и технологии (в экологии)», 1-33 01 03 «Радиоэкология» и 1-33 01 04 «Экологический мониторинг, менеджмент и аудит».

Задачи дисциплины. В результате усвоения этой дисциплины обучаемый должен:

иметь представление:

- об основах технологии .NET;
- об особенностях проектирования и программной реализации информационных систем с использованием доступа к данным при помощи технологии ADO.NET;
- о принципах разработки Web-приложений с применением технологии ASP.NET;
- о назначении и использовании Web-служб;

знать:

- особенности программных средств, используемых при реализации информационных систем;

владеть:

- методами проектирования и программной реализации информационных систем;
- методами поиска информации с использованием автоматизированных систем;

уметь использовать:

- интегрированную среду разработки Microsoft Visual Studio для создания программных приложений для доступа к базам данных и различного рода Web-приложений;
- систему управления базами данных Microsoft SQL Server;
- службы Internet Information Services и ASP.NET Development Server;

иметь опыт:

- реализации доступа к данным, хранимым в памяти и в файлах систем управления базами данных Microsoft Access и Microsoft SQL Server, на языке программирования Visual C# с использованием технологий ADO.NET и ASP.NET и элементов управления Windows Forms и WebForm;
- администрирования служб Internet Information Services и ASP.NET Development Server;
- разработки пользовательского интерфейса посредством элементов управления WebForm и клиентских скриптов на языках программирования JavaScript и VBScript для серверных скриптов;
- проектирования и разработки локального ASP.NET Web-сайта;
- использования технологии ADO.NET для взаимодействия с локальными и удаленными базами данных при программировании кода для страниц ASP.NET.
- Дисциплины, усвоение которых необходимо для изучения данной темы (с учетом усвоения всех разделов и тем):
- «Информатика и программирование (информационные технологии)»;
- «Информатика и программирование (основы программирования)»;
- «Объектно-ориентированное программирование»;
- «Системы баз данных»;
- «Информационные системы».

Изучение дисциплины предусматривает систематическую самостоятельную работу студентов с рекомендуемой литературой, Internet-источниками и т. д., а также использование современных программных и технических средств при выполнении практических занятий.

1. ОСНОВЫ ТЕХНОЛОГИИ .NET

1.1. Обзор существующих технологий и языков программирования. Основополагающие понятия и особенности платформы .NET

Любому современному программисту, который желает идти в ногу с последними веяниями, каждые несколько лет приходится переучиваться. Языки (C++, Visual Basic, Java и др.), библиотеки (MFC (Microsoft Foundation Classes), ATL (Active Template Library), STL (Standard Template Library) и др.), архитектуры (COM (Component Object Model), CORBA (Common Object Request Broker Architecture) и др.), которые стали вехами в развитии программирования за последние годы, постепенно уходят в тень лучших или, по крайней мере, более молодых программных технологий. Вне зависимости от того, нравится этот процесс программистам или нет, он неизбежен. Платформа .NET компании Microsoft – это следующая волна коренных изменений, которая идет из Редмонда.

Основными элементами, на которых основана платформа .NET, являются:

- сборки (Assemblies);
- IL (Intermediate Language) – промежуточный язык;
- JIT (Just-In-Time Compiler) – компилятор синхронного времени (в процессе) выполнения.

Существенное внимание следует уделить рассмотрению взаимосвязи компонентов платформы .NET:

- CLR (Common Language Runtime) – стандартной среды выполнения для языков;
- CTS (Common Type System) – стандартной системы типов;
- CLS (Common Language Specification) – стандартной спецификации для языков,

а также общему обзору возможностей, которые предоставляют библиотеки базовых классов .NET и утилиты, которые помогают в работе с этими библиотеками, и ознакомлению с возможностями компиляции приложений C# с использованием компилятора командной строки и IDE (Integrated Development Environment) – интегрированной среды разработки – Microsoft Visual Studio [1–4].

Обзор существующих технологий и языков программирования. Наряду с известными технологиями, которые имеют свои возможности и огра-

ничения, C# и платформа .NET в целом предоставляют ряд преимуществ, что обуславливает необходимость последних.

Win32 / C. Изначально под программированием под Windows подразумевалось программирование на C с использованием Windows Application Programming Interface (интерфейса (набора функций) прикладного программирования Windows, в 32-разрядных версиях Windows – Win32 API). С использованием этой технологии было создано множество вполне достойных приложений, однако вряд ли кто-нибудь будет спорить с тем, что написание приложения с использованием только Windows API – очень трудоемкая задача.

Еще одна проблема заключается в том, что C – достаточно суровый по отношению к программисту язык. Тем, кто создает на нем свои приложения, приходится вручную заниматься управлением памятью, выполнять расчеты при использовании указателей и работать с совершенно неестественными с точки зрения человеческого языка синтаксическими конструкциями. Кроме того, в C, конечно, недостаточно возможностей для объектно-ориентированного программирования. При связывании тысяч глобальных функций Win32 API в единые гигантские конструкции в подобных приложениях ошибки встречаются очень часто.

C++ / MFC. C++ – это огромный шаг вперед в отношении новых возможностей по сравнению с исходным языком C. Во многих ситуациях C++ вполне допустимо представить как объектно-ориентированную надстройку над C. Такая надстройка позволяет использовать преимущества «столпов» объектно-ориентированного программирования – инкапсуляции, полиморфизма и наследования. Однако программисты, использующие C++, остаются незащищенными от многих и часто опасных особенностей C (теми же самыми низкоуровневыми возможностями работы с памятью и трудными для восприятия синтаксическими конструкциями).

Существует множество библиотек для C++, основное назначение которых – облегчить написание приложений под Windows, предоставив для этой цели уже готовые классы. Одна из наиболее распространенных библиотек базовых классов – это MFC. MFC – это дополнительный уровень над Win32 API, который значительно упрощает работу программиста за счет использования готовых классов, макросов и мастеров. Однако MFC – это лишь частичное решение проблемы. Даже при использовании MFC программисту приходится работать со сложным для чтения кодом, весьма опасным с точки зрения возможных ошибок.

Visual Basic. Люди всегда стремятся сделать свою жизнь проще. Повинуясь этому стремлению, многие программисты на C++ обратили свои взоры к гораздо более простому и дружелюбному языку, каким является Visual Basic. Visual Basic позволяет работать с достаточно сложными эле-

ментами интерфейса пользователя, библиотеками кода (например, COM-серверами) и средствами доступа к данным при минимальных затратах времени и сил. Visual Basic в гораздо большей степени, чем MFC, прячет от пользователя вызовы Win32 API и предоставляет большой набор интегрированных средств быстрой разработки.

Однако у Visual Basic есть и недостатки. Главный из них – это гораздо меньшие возможности, которые он предоставляет, по сравнению с C++ (это утверждение справедливо, по крайней мере, для версий более ранних, чем Visual Basic.NET). Visual Basic – это язык «для работы с объектами», а не объектно-ориентированный язык в обычном понимании этого слова. В Visual Basic нет классического наследования, поддержки создания параметризованных классов, собственных средств создания многопоточных приложений – и этот список можно продолжать.

Java. Язык программирования Java – это полностью объектно-ориентированный язык, который в отношении синтаксиса многое унаследовал от C++. Конечно, преимущества Java далеко не исчерпываются межплатформенностью. Язык Java в синтаксическом отношении проще и логичнее, чем C++. Java как платформа предоставляет в распоряжение программистов большое количество библиотек (пакетов), в которых содержится большое количество описаний классов и интерфейсов для многих случаев. С их помощью можно создавать стопроцентные приложения Java с возможностью обращения к базам данных, поддержкой передачи почтовых сообщений, с клиентской частью, которой необходим только Web-браузер, или, наоборот, с клиентской частью, обладающей изощренным интерфейсом.

Java – это очень элегантный и красивый язык. Однако при его использовании проблем также избежать не удастся. Одна из серьезных проблем заключается в том, что при создании сложного приложения на Java придется использовать только этот язык для создания всех частей приложения. В Java предусмотрено не так много средств для межъязыкового взаимодействия (что понятно ввиду предназначения Java быть единым многоцелевым языком программирования). В реальном мире существуют миллионы строк готового кода, которые хотелось бы интегрировать с новыми приложениями на Java. Однако это сделать очень трудно.

Java – это далеко не идеальный язык во многих ситуациях. Простой пример: если попытаться создать только на Java приложение, активно работающее с 3D-графикой, скорее всего, обнаружится, что работать такое приложение будет не очень быстро. В этой связи можно прийти к выводу, что для работы с 3D-графикой лучше использовать код, написанный на языке с более развитыми низкоуровневыми возможностями (например, на C++). Однако интегрировать такой код с кодом на Java будет очень сложно.

Поскольку возможности для обращения к API компонентов, созданных на других языках, в Java очень ограничены, говорить о реальном межъязыковом взаимодействии на основе Java не приходится.

COM. Современное состояние дел таково, что если не создаются Java-приложения, то велика вероятность, что имеется дело с технологией Microsoft COM. Этот стандартный механизм, включающий интерфейсы, с помощью которых одни объекты предоставляют свои сервисы другим, является основой многих объектных технологий, в том числе OLE и ActiveX. COM-технология провозглашает: если создаются классы в точном соответствии с требованиями COM, то получится блок повторно используемого программного кода.

Достоинство двоичного COM-сервера заключается в том, что к нему можно обращаться из любого языка. Например, программисты, использующие C++, могут создавать классы, которые можно будет использовать из приложения на Visual Basic. Программисты, использующие Delphi, могут использовать классы, созданные на C, и т. д. Однако в межъязыковом взаимодействии COM есть свои ограничения. Например, невозможно произвести новый тип COM от существующего (т. е. нельзя использовать классическое наследование). Для повторного использования существующих типов COM придется использовать другие, гораздо менее надежные и эффективные средства.

Большое преимущество COM заключается в том, что программист может не заботиться о физическом местонахождении компонентов. Такие средства, как Application Identifiers (AppIDs, идентификаторы приложений), стабы (Stubs), прокси, среда выполнения COM, позволяют избегать при обращении к компонентам по сети необходимости помещать в приложение код для работы с сокетами, вызовами RPC (Remote Procedure Call, удаленный вызов процедуры) и прочими низкоуровневыми механизмами. Например, RPC служит средством передачи сообщений, которое позволяет распределенному приложению вызывать сервис различных компьютеров в сети, обеспечивает процедурно-ориентированный подход в работе с сетью и применяется в распределенных объектных технологиях, таких как DCOM, CORBA и Java RMI. Достаточно посмотреть на код на Visual Basic версии 6.0 для клиента COM, который предназначен для активации класса COM, созданного на любом языке. Класс COM может быть расположен в любом месте на локальном компьютере или в сети.

Объектная модель COM используется очень широко. Однако внутреннее устройство компонентов весьма сложно. Чтобы научиться разбираться в нем, придется потратить, по крайней мере, несколько месяцев. Написание приложений с использованием COM-компонентов можно упростить, используя стандартные библиотеки, например STL – библиоте-

ку стандартных шаблонов (шаблонных классов) и ATL – библиотеку активных шаблонов (шаблонных классов), использующуюся для создания элементов управления ActiveX, со своим набором готовых классов, шаблонов и макросов.

Некоторые языки (например, Visual Basic) также позволяют скрыть сложность инфраструктуры COM. Однако все равно всех сложностей избежать не удастся. Например, даже если работать с относительно простым и поддерживающим COM Visual Basic, придется решать не всегда простые вопросы, связанные с регистрацией компонентов на компьютере и развертыванием (установкой) приложений.

Windows DNA. Необходимо также принять во внимание сеть Internet. За несколько последних лет Microsoft добавила в свои операционные системы большое количество средств для работы с этой средой, в том числе и средства, призванные помочь в создании Internet-приложений. Однако построение законченного web-приложения с использованием технологии Windows DNA (Distributed iNternet Architecture – распределенная межсетевая архитектура) до сих пор остается весьма сложной задачей.

Значительная часть сложностей возникает, оттого что Windows DNA требует использования разнородных технологий и языков (ASP (Active Server Pages), HTML (Hypertext Markup Language), XML (Extensible Markup Language), JavaScript, VBScript, COM, COM+, ADO (ActiveX Data Objects) и т. д.). Одна из проблем заключается в том, что синтаксически все эти языки и технологии очень мало похожи друг на друга. Например, синтаксис JavaScript больше похож на синтаксис C, в то время как VBScript является подмножеством Visual Basic. COM-серверы, предназначенные для работы в среде выполнения COM+, созданы на основе совершенно иных подходов, нежели ASP-страницы, которые к ним обращаются. Конечным результатом является сложное смешение технологий. Помимо всего прочего в каждом языке, который входит в состав Windows DNA, предусмотрена своя система типов, что также не является удобством для программистов. Например, тип данных int в JavaScript – это не то же самое, что int в C, который, в свою очередь, отличен от integer в Visual Basic [1–4].

Основополагающие понятия и особенности платформы. NET. На этом можно считать обращение к новейшей истории программирования законченным. Главный вывод таков, что современные Windows-программисты стоят перед своего рода дилеммой – выбором и непосредственной работой с той или иной технологией разработки программных приложений. На этом фоне возможности, предлагаемые платформой .NET, позволяют радикально облегчить их положение. Один из главных принципов .NET заключается в том, что дозволено вносить изменения во все, что угодно и откуда угодно. .NET – это совершенно новая модель для создания приложений под

Windows (а в будущем, видимо, и под другими операционными системами). Основные возможности .NET:

- Полные возможности взаимодействия с существующим кодом. Существующие двоичные компоненты COM отлично работают вместе с двоичными файлами .NET.
- Полное и абсолютное межъязыковое взаимодействие. В отличие от классического COM, в .NET поддерживаются межъязыковые наследование, обработка исключений и отладка.
- Общая среда выполнения для любых приложений .NET, вне зависимости от того, на каких языках они были созданы. Один из важных моментов при этом – то, что для всех языков используется один и тот же набор встроенных типов данных.
- Библиотека базовых классов, которая обеспечивает сокрытие всех сложностей, связанных с непосредственным использованием вызовов API, и предлагает целостную объектную модель для всех языков программирования, поддерживающих .NET.
- Отсутствие присущей COM сложности. Например, IClassFactory, IUnknown, код IDL (Interface Definition Language, язык описания (определения) интерфейсов (используется для спецификации интерфейсов объектов COM) и VARIANT-совместимые типы данных (BSTR, SAFEARRAY и остальные) не существуют в коде программ .NET.
- Действительное упрощение процесса развертывания приложения. В .NET нет необходимости регистрировать двойные типы в системном реестре. Более того, .NET позволяет разным версиям одного и того же модуля DLL без проблем сосуществовать на одном компьютере.

Строительные блоки .NET (CLR, CTS и CLS). Технологии CLR, CTS и CLS очень важны для понимания смысла платформы .NET. С точки зрения программиста .NET вполне можно рассматривать просто как новую среду выполнения и новую библиотеку базовых классов. Среда выполнения .NET как раз и обеспечивается с помощью CLR – стандартной языковой среды выполнения. Главная роль CLR заключается в том, чтобы обнаруживать и загружать типы .NET и производить управление ими в соответствии с командами. CLR берет на себя всю низкоуровневую работу – например, автоматическое управление памятью, межъязыковым взаимодействием, развертывание (с отслеживанием версий) различных двоичных библиотек.

Еще один строительный блок платформы .NET – это CTS – стандартная система типов. CTS полностью описывает все типы данных, поддерживаемые средой выполнения, определяет, как одни типы данных могут взаимодействовать с другими и как они будут представлены в формате метаданных .NET.

Важно понимать, что не во всех языках программирования .NET обязательно должны поддерживаться все типы данных, которые определены в CTS. CLS – это набор правил, определяющих подмножество общих типов данных, в отношении которых гарантируется, что они безопасны при использовании во всех языках .NET. Если создаются типы .NET с использованием только тех возможностей, которые совместимы с CLS, тем самым создаются условия для их пригодности в контексте любых языков .NET.

Библиотека базовых классов .NET. Помимо спецификаций CLR и CTS / CLS платформа .NET предоставляет в распоряжение также библиотеку базовых классов, доступную из любого языка программирования .NET. Библиотека базовых классов не только прячет обычные низкоуровневые операции, такие как файловый ввод-вывод, обработка графики и взаимодействие с оборудованием компьютера, но и обеспечивает поддержку большого количества служб, используемых в современных приложениях.

В качестве примера можно привести встроенные типы для обращения к базам данных, работы с XML, обеспечения безопасности при работе приложения, создания приложений для работы в Web (конечно, обеспечивается и поддержка обычных консольных и оконных приложений). С концептуальной точки зрения существуют некоторые отношения между уровнем среды выполнения и библиотекой базовых классов .NET (рис. 1.1).

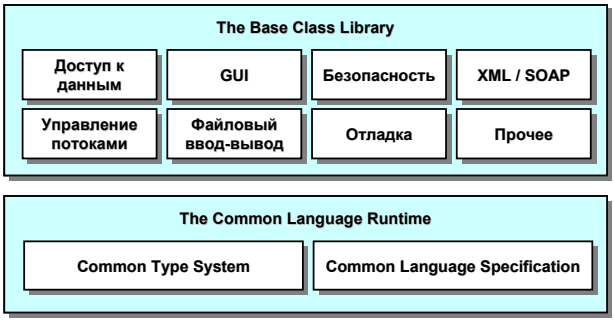


Рис. 1.1. Отношения между средой выполнения и библиотекой базовых классов .NET

Таким образом, реализации, предлагаемые платформой .NET, в некоторой степени решают те проблемы и ограничения, с которыми столкнулись существующие в настоящее время традиционные подходы к программированию. Важнейшими компонентами указанной платформы являются среда выполнения CLR и библиотека базовых классов [1–4].

1.2. Особенности языков программирования .NET. Назначение сборок. Роль и преимущества промежуточного языка IL. Сущность метаданных

Особенности языков программирования .NET. Специально для платформы .NET Microsoft был разработан новый язык программирования C#. C# – это язык программирования, синтаксис которого очень похож на синтаксис Java (но не идентичен ему). Например, в C# (как и в Java) определение класса состоит из одного файла (*.cs), в отличие от C++, где определение класса разбито на заголовок (*.h) и реализацию (*.cpp). Однако называть C# клоном Java было бы неверно. Как C#, так и Java основаны на синтаксических конструкциях C++. Если Java во многих отношениях можно назвать очищенной версией C++, то C# можно охарактеризовать как очищенную версию Java.

Синтаксические конструкции C# унаследованы не только от C++, но и от Visual Basic. Например, в C#, как и в Visual Basic, используются свойства классов. Как C++, C# позволяет производить перегрузку операторов для созданных типов (Java не поддерживает ни ту, ни другую возможность). C# – это фактически гибрид разных языков. При этом C# синтаксически не менее (если не более) чист, чем Java, так же прост, как Visual Basic, и обладает практически той же мощью и гибкостью, что и C++. Основные особенности C#:

- Нет необходимости в работе с указателями (однако если это потребуется, то возможности для этого сохранены).
- Управление памятью производится автоматически.
- Предусмотрены встроенные синтаксические конструкции для работы с перечислениями, структурами и свойствами классов.
- Осталась возможность перегружать операторы, унаследованные от C++. При этом значительная часть возникавших при этом сложностей ликвидирована.
- Предусмотрена полная поддержка использования программных интерфейсов. Однако в отличие от классического COM применение интерфейсов – это не единственный способ работы с типами, используя различные двоичные модули. .NET позволяет передавать объекты (как ссылки или как значения) через границы программных модулей.
- Также предусмотрена полная поддержка аспектно-ориентированных программных технологий (таких как атрибуты). Это позволяет присваивать типам характеристики (что во многом напоминает COM IDL) для описания в будущем поведения данной сущности.

Возможно, самое важное, что необходимо сообщить про язык C#, – то, что он генерирует код, предназначенный для выполнения только в среде выполнения .NET. Например, невозможно использовать C# для создания классического COM-сервера. Согласно терминологии Microsoft код, предназначенный для работы в среде выполнения .NET, – это управляемый код (Managed code). Двоичный файл, который содержит управляемый файл, называется сборкой (Assembly).

Во время анонса платформы .NET на конференции 2000 Professional Developers Conference (PDC) докладчики называли фирмы, работающие над созданием .NET-версий своих компиляторов. К 2001 году компиляторы для создания .NET-версий приложений разрабатывались более чем для 30 различных языков. Помимо языков, поставляемых с Microsoft Visual Studio (Visual Basic, Visual C#, Visual C++ и Visual J#), ожидаются .NET-версии Smalltalk, COBOL, Pascal, Python, Perl и множества остальных известных языков программирования.

Все компиляторы, ориентированные на .NET, генерируют IL-инструкции и метаданные (рис. 1.2).

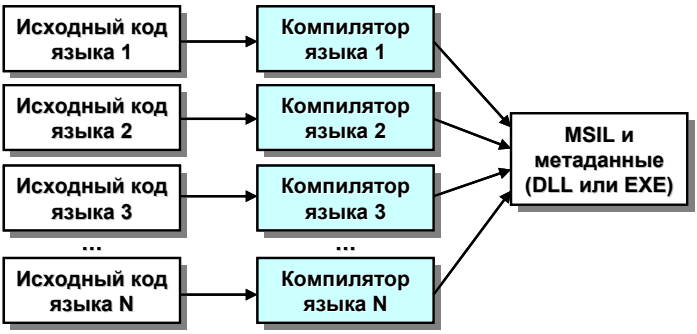


Рис. 1.2. Схема работы компиляторов, ориентированных на .NET

Двоичные файлы .NET, для которых используются стандартные расширения DLL и EXE, по своему внутреннему содержанию не имеют абсолютно ничего общего с обычными исполняемыми файлами. Например, файлы DLL не предоставляют свои методы в распоряжение приложений на компьютере. В отличие от компонентов COM двоичные файлы .NET не описываются с помощью кода IDL и регистрируются в системном реестре. Однако, пожалуй, самое важное отличие заключается в том, что двоичные файлы .NET не содержат зависящих от платформы команд. Содержимое двоичных файлов .NET – это платформенно-независимый «промежуточный язык», который офици-

ально называется Microsoft Intermediate Language (MSIL, промежуточный язык Microsoft), или просто IL [1–4].

Назначение сборок. Когда с помощью компилятора для платформы .NET создается модуль DLL или EXE, содержимое этого модуля – это так называемая сборка на языке IL. Базовые свойства этого нового формата исполняемых файлов позволяют лучше понять особенности среды выполнения .NET.

Как уже упоминалось, сборка содержит код на «промежуточном языке» – IL. Назначение IL концептуально аналогично байт-коду Java – он компилируется в платформенно-специфичные инструкции, только когда это абсолютно необходимо. «Абсолютная необходимость» возникает тогда, когда к блоку инструкций IL (например, реализации метода) обращается для использования среда выполнения .NET.

Помимо инструкций IL двоичные модули .NET содержат также метаданные, которые подробно описывают все типы, использованные в модуле. Например, если внутри сборки есть некоторый класс, то в метаданных этой сборки будет содержаться информация о базовом классе для него, какие интерфейсы предусмотрены для него (если они вообще предусмотрены), а также полное описание всех методов, свойств и событий этого класса.

Во многих отношениях метаданные .NET являются значительным усовершенствованием по сравнению с классической информацией о типах в COM. Классические двоичные файлы COM обычно описываются с помощью ассоциированной библиотеки типов (которая очень похожа на двоичную версию кода IDL). Проблема с информацией о типах в COM заключается в том, что никто не может гарантировать, что эта информация окажется полной. Код IDL не может создать полный каталог внешних серверов, которые могут быть необходимы для нормальной работы содержащихся в модуле COM-классов. Существование метаданных .NET, напротив, обеспечивается тем, что метаданные автоматически генерируются компилятором, создающим приложение .NET.

Метаданные описывают не только типы, используемые в сборке, но и саму сборку. Эта часть метаданных называется манифестом (manifest). В манифесте содержится информация о текущей версии сборки, об использованных ограничениях по безопасности, о поддерживаемом естественном языке (английском, русском и т. д.), а также список всех внешних сборок, которые потребуются для нормального выполнения. Существуют различные средства, которые можно использовать для анализа кода IL внутри сборки, метаданных для типов и манифеста.

Сборка может состоять как из одного, так и из нескольких двоичных файлов. В сборке из одного файла (Single file assembly) этот единственный файл содержит и манифест, и метаданные, и инструкции IL.

В сборке из нескольких двоичных файлов (multifile assembly) каждый двоичный файл называется модулем (module). При создании таких многофайловых сборок один из двоичных файлов должен содержать манифест сборки (в нем могут также находиться и другие данные, в том числе инструкции IL). Все остальные модули могут содержать только метаданные типов и инструкции IL.

Единственная причина для создания многофайловой сборки – большая гибкость при развертывании приложения. Например, если пользователь обращается к удаленной сборке, которая должна быть загружена на его локальный компьютер, среда выполнения загрузит лишь те модули сборки, которые действительно необходимы. Такое решение позволит избежать ненужного сетевого трафика и увеличить скорость работы программы [1–4].

Роль и преимущества промежуточного языка IL. Код IL не зависит от платформы, на которой будет производиться выполнение. При этом компилятор для платформы .NET сгенерирует код IL вне зависимости от того, какой язык программирования (Visual Basic, Visual C#, Eiffel и т. п.) использовался для создания программы.

Пример сложения чисел 5 и 10 в коде на C#:

```
namespace Calculator
{
	using System;

	public class Calc
	{
		public Calc() { }

		public int Add(int x, int y)
		{
			return x + y;
		}

		public static int Main(string[] args)
		{
			Calc c = new Calc();
			int ans = c.Add(5, 10);
			Console.WriteLine("5 + 10 is {0}.", ans);
			return 0;
		}
	}
}
```

В классе Calculator определен метод Add(), а также точка входа приложения – метод Main().

После того как этот исходный файл будет обработан компилятором C# (csc.exe), в распоряжении окажется исполняемый файл C# – сборка из одного файла. Внутри этого файла можно будет обнаружить манифест, инструкции IL и метаданные, описывающие класс Calc. Если заглянуть внутрь этой сборки при помощи специальных средств, то помимо всего прочего можно найти следующий блок инструкций IL, относящихся к методу Add():

```
.method public hidebysig instance int32 Add(int32 x, int32 y) il managed
{
        .maxstack 2
        .locals ([0] int32 V_0)
        IL_0000: ldarg.1
        IL_0001: ldarg.2
        IL_0002: add
        IL_0003: stloc.O
        IL_0004: br.s IL0006
        IL_0006: ldloc.0
        IL_0007: ret
}
```

Компилятор C# генерирует не платформенно-зависимый набор инструкций, а код IL. То же самое справедливо и для других компиляторов .NET, например для Visual Basic исходный текст программы:

```
Module Module1

        Class Calc

                Public Function Add(ByVal x As Integer, ByVal y As Integer) As
Integer
                        Return x + y
        End Function
        End Class

        Sub Main()
                Dim ans As Integer
                Dim c As New Calc()
                ans = c.Add(10, 84)
                Console.WriteLine("10 + 84 is: {0}.", ans)
        End Sub

End Module
```

Код внутри сборки, относящийся к методу Add():

```
.method public instance int32 Add(int32 x, int32 y) il managed
{
        .maxstack 2
        .locals init ([0] int32 Add)
        IL_0000: nop
        IL_0001: ldarg.1
        IL_0002: ldarg.2
        IL_0003: add.ovf
        IL_0004: stloc.0
        IL_0005: nop
        IL_0006: br.s IL_0008
        IL_0008: nop
        IL_0009: ldloc.0
        IL_000a: ret
}
```

Получившийся код IL практически идентичен предыдущему. Незначительные отличия возникают вследствие особенностей компиляторов C# и Visual Basic.

Преимущества IL:

- Возможность полного межъязыкового взаимодействия. Поскольку любой код на любом языке программирования .NET компилируется в стандартный набор инструкций IL, проблем во взаимодействии между блоками кода IL не будет. При этом взаимодействие будет производиться, как и положено, на двоичном уровне.
- Потенциальная независимость от компьютерной платформы. Существует большая вероятность, что среда выполнения .NET будет распространена на самые разные компьютерные платформы и операционные системы (отличные от Windows). В результате .NET может пойти по стопам Java – т. е. с помощью языков .NET можно будет создавать программы, которые будут работать под самыми разными операционными системами (и при этом в отличие от Java еще и пользоваться преимуществами языковой независимости). Таким образом, .NET потенциально позволяет создавать приложения на любом языке, которые будут работать на любой платформе и под любой операционной системой.

Однако в отношении межплатформенности пока ключевое слово – «потенциально». Поэтому пока можно считать, что приложения .NET работают только под Windows [1–4].

Сущность метаданных. Программистам, работающим с COM, хорошо знакома концепция IDL. IDL – это «метаязык», который позволяет, исклю-

чив любую двусмысленность, описать типы, используемые внутри сервера COM. IDL компилируется в двоичный формат (называемый библиотекой типов) с использованием компилятора midl.exe. Этот компилятор может использоваться любым языком, предназначенным для работы с COM.

IDL полностью описывает все типы данных, используемые в двоичном файле COM, но информация о самом этом двоичном файле в нем минимальна. Фактически она ограничивается номером версии (к примеру, 1.0, 2.0 или 2.4) и информацией о локализации (например, English, German, Russian и т. д.). Кроме того, наличие или отсутствие метаданных (и их полноту) должен вручную контролировать создающий сервер COM-программист – таким образом, необходимых метаданных в двоичном файле COM может вообще не оказаться.

В отличие от COM при использовании платформы .NET вообще не придется думать об IDL. Однако общий принцип описания типов в строго определенном двоичном формате остался.

Сборки .NET всегда содержат полные и точные метаданные, поскольку метаданные в них генерируются автоматически. Как и в IDL, в метаданных .NET содержится исчерпывающая информация об абсолютно всех типах, которые используются в сборке (классах, структурах, перечислениях и пр.), а также о каждом свойстве, методе или событии каждого типа.

Еще одно отличие метаданных .NET от информации IDL заключается в том, что метаданные .NET гораздо более подробны. В них перечислены все ссылки на внешние модули и сборки, которые потребуются для нормального выполнения сборки .NET. За счет этого можно считать сборки .NET фактически самодокументируемыми. В результате, к примеру, отпадает необходимость регистрировать двоичные файлы .NET в системном реестре.

Пример метаданных для метода Add() рассмотренного ранее приложения (метаданные для этого метода в Visual Basic будут точно такими же):

```
Method #2

MethodName      : Add (06000002)
Rags            : [Public] [HldeBySig] [ReuseSlot] (00000086)
RVA             : 0x00002058
ImplFlags       : [IL] [Managed] (00000000)
CalcCnvntn      : [DEFAULT]
hasThis
ReturnType      : 14
2 Arguments
        Argument #1: 14
        Argument #2: 14
```

2 Parameters
(1) ParamToken : (08000001) Name : x flags: [none] (00000000) default;
(2) ParamToken : (08000002) Name : y flags: [none] (00000000) default;

В коде ясно представлены название метода, тип возвращаемого значения и данные об ожидаемых аргументах. Все метаданные автоматически создаются компилятором C#.

К метаданным обращаются и сама среда выполнения .NET (очень часто), и самые разные средства разработки и отладки. Например, средство IntelliSense в Microsoft Visual Studio (которое пытается помочь закончить начатую строку) берет необходимую ему информацию именно из метаданных. Метаданные также активно используются утилитами просмотра, отладки и, конечно, самим компилятором C#.

Поскольку в сборках содержится платформенно-независимый код IL, а выполняются в конечном итоге именно платформенно-зависимые инструкции, что-то должно взять на себя работу по оперативной компиляции IL в такие инструкции. Это средство называется JIT – компилятор синхронного времени выполнения. Он реализует концепцию управления, предполагающую поставку ресурса как раз в тот момент, когда его нужно использовать. Этот компилятор для перевода IL в платформенно-зависимые инструкции входит в состав среды выполнения .NET. Используя код IL, разработчики могут не думать об особенностях архитектуры центрального процессора (CPU (Central Processing Unit) данного компьютера – эти особенности будут учтены JIT.

Откомпилированные из IL платформенно-зависимые инструкции JIT помещают в кэш-памяти, что очень ускоряет работу приложения. Например, если был вызван какой-либо метод определенного класса, при первом вызове этого метода JIT откомпилирует относящийся к этому методу код IL в платформенно-зависимые инструкции. При повторных вызовах этого метода JIT уже не будет заниматься компиляцией, а просто возьмет готовый откомпилированный код из кэша в оперативной памяти.

Таким образом, любые двоичные файлы .NET, называемые «сборками», работают только в среде выполнения CLR. Сборки содержат в себе инструкции промежуточного кода и метаданные самой сборки (манифест) и типов. Промежуточный код превращается в платформенно-зависимый код в момент выполнения приложения. Эту задачу выполняет компилятор синхронного времени выполнения JIT [1–4].

1.3. Типы и пространства имен .NET. Особенности стандартной среды выполнения языков CLR. Назначение стан-

дартной системы типов CTS и особенности ее классов, структур, интерфейсов, членов типов, перечислений, делегатов и встроенных типов данных. Назначение стандартной спецификации языков CLS

Типы и пространства имен .NET. Сборка (не важно, однофайловая или многофайловая) может содержать любое количество самых разных типов. В .NET тип – это общий термин, который может относиться к классам, структурам, интерфейсам, перечислениям и прочему. При создании приложения .NET (например, на языке C#) потребуется организовывать взаимодействие этих типов. Например, сборка может определять класс с несколькими интерфейсами; один интерфейс может принимать в качестве параметров только значения определенного перечисления.

Есть возможность использовать пространства имен при создании собственных типов. Пространство имен – это логическая структура для организации имен, используемых в приложении .NET. Основное назначение пространств имен – предупредить возможные конфликты между именами в разных сборках.

Например, при создании приложения, основанном на элементах управления Windows Forms, которое обращается к двум внешним сборкам. В каждой сборке есть тип с именем Type, при этом эти типы отличаются друг от друга. При написании кода можно точно указать, к какому именно типу и из какой сборки происходит обращение. Для этого достаточно к имени типа добавить имя соответствующего пространства имен: например, Namespace_1.Type или Namespace_2.Type [1 - 4].

Особенности стандартной среды выполнения языков CLR. Среду выполнения (runtime) можно рассматривать как набор служб, необходимых для работы блока программного кода. К таким службам можно отнести и требуемые библиотеки. Например, если создано приложение MFC, то в качестве компонента среды выполнения потребуется весьма объемистая библиотека времени выполнения Microsoft Foundation Classes – mfc42.dll. К примеру, программы на Visual Basic версии 6.0 привязаны к такому компоненту среды выполнения, как библиотека msvbvm60.dll, а программам на Java необходим большой набор файлов, входящих в состав виртуальной машины Java.

Своя среда выполнения требуется и приложениям .NET. Главное отличие этой среды выполнения от перечисленных выше заключается в том, что единая среда выполнения .NET используется приложениями, написанными на любых языках программирования .NET. Как уже упоминалось, среда выполнения .NET носит официальное название CLR.

Сама CLR состоит из двух главных компонентов. Первый компонент – это ядро среды выполнения, которое реализовано в виде библиотеки mscoree.dll. При обращении к приложению .NET эта библиотека автоматически загружается в память и, в свою очередь, управляет процессом загрузки в память сборки данного приложения. Ядро среды выполнения ответственно за множество задач. Оно занимается поиском физического местонахождения сборки, обнаружением внутри сборки запрошенного типа (класса, интерфейса, структуры и т. п.) на основе информации метаданных, компилирует IL в платформенно-зависимые инструкции, выполняет проверки, связанные с обеспечением безопасности, – и этот перечень далеко не полон.

Второй главный компонент CLR – это библиотека базовых классов. Сама библиотека разбита на множество отдельных сборок, однако главная сборка представлена файлом mscorlib.dll. В библиотеке базовых классов содержится огромное количество типов для решения распространенных задач при создании приложения. Приложение .NET будет обязательно использовать сборку mscorlib.dll и по мере необходимости – другие сборки (как встроенные, так и создаваемые).

На рис. 1.3 представлен путь, который проходит исходный код приложения, прежде чем воплотиться в выполнение каких-либо действий на компьютере [1–4].

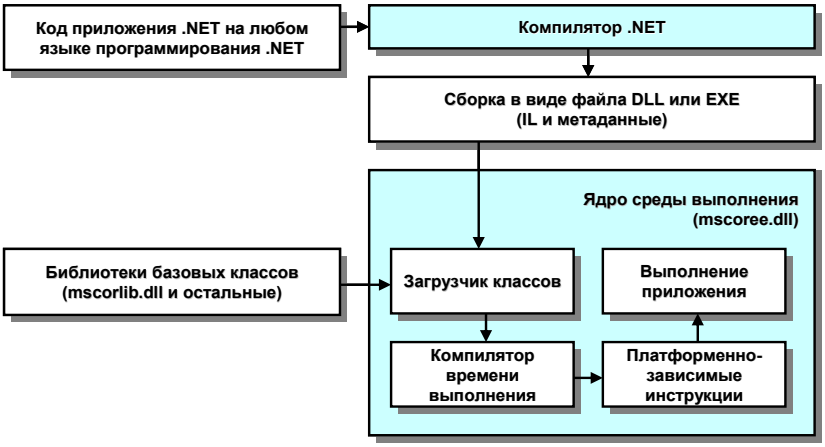


Рис. 1.3. Роль среды выполнения .NET

Назначение стандартной системы типов CTS и особенности ее классов, структур, интерфейсов, членов типов, перечислений, делегатов и встроенных типов данных. Стандартная система типов (Common Type System, CTS) – это формальная спецификация, которая определяет, как какой-либо тип (класс, структура, интерфейс, встроенный тип данных и тому подобное) должен быть определен для его правильного восприятия средой выполнения .NET. CTS определяет синтаксические конструкции (в качестве примера можно взять перегрузку операторов), которые могут поддерживаться, а могут и не поддерживаться конкретным языком программирования .NET. Если необходимо создать сборки, которые смогут использоваться всеми языками программирования .NET, придется при создании типов следовать правилам Common Language Specification – CLS. Спецификация CTS применяется ко многим типам, обладающим своими особенностями.

Концепция классов – краеугольный камень любого объектно-ориентированного программирования. Она поддерживается всеми языками программирования .NET. Класс (class) – это набор свойств, методов и событий, объединенных в единое целое. В CTS предусмотрены абстрактные члены классов, что обеспечивает возможность применения полиморфизма в производных классах. Множественное наследование в CTS запрещено. Самые важные характеристики классов представлены в табл. 1.1.

Таблица 1.1

Самые важные характеристики классов CTS

Характеристика	Смысл
Является ли класс «закрытым»?	«Закрытые» классы не могут становиться базовыми для других классов
Предусмотрены ли в классе какие-либо интерфейсы?	Интерфейс – это набор абстрактных членов, который обеспечивает связь между объектом и пользователем. В CTS в классе может быть любое количество интерфейсов
Является ли класс абстрактным?	Объекты абстрактных классов создать невозможно. Единственное назначение абстрактных классов – выполнять роль базовых для других классов. За счет механизма наследования абстрактные классы обеспечивают производные классы общими наборами членов
Какова область видимости для данного класса?	Для каждого класса должен быть определен атрибут области видимости (Visibility). Как правило, значение этого атрибута определяет, можно ли обращаться к этому классу из внешних сборок или только из той, которая его содержит

Помимо классов в CTS также предусмотрена концепция структур (Structures). Этот пользовательский тип данных сохранился и в .NET (правда, надо отметить, что он немного изменился). В принципе структуры можно грубо рассматривать как упрощенные разновидности классов. Структуры CTS могут иметь любое количество полей, конструкторов с параметрами (конструктор без параметров зарезервирован) и методов. С помощью конструкторов с параметрами можно установить значение любого поля объекта структуры в момент создания этого объекта. Например:

```
struct Baby
{

        public string name;

        public Baby(string name)
        {
                this.name = name;
        }

        public void Cry()
        {
                Console.WriteLine("A-a-a!!!");
        }

        public bool IsSpleeping()
        {
                return false;
        }

        public bool IsChanged()
        {
                return false;
        }
}
```

Структура в действии выглядит как

```
Baby barnaBaby = new Baby ("Федор");
Console.WriteLine ("Изменен?: {0}", barnaBaby.IsChanged().ToString());
Console.WriteLine ("Спит?: {0}", barnaBaby.IsSleeping().ToString());

for (int i=0; i<10000; i++)
barnaBaby.Cry();
```

Все CTS-совместимые структуры произведены от единого базового класса System.ValueType. Этот базовый класс определяет структуру как тип данных для работы только со значениями, но не со ссылками. В структуре может быть любое количество интерфейсов. Однако структуры не могут быть унаследованы от остальных типов данных и они всегда являются «закрытыми», – т. е. они не могут выступать в качестве базовых для целей наследования.

Интерфейсы (interfaces) – это просто наборы абстрактных методов, свойств и определений событий. В отличие от классической технологии COM, интерфейсы .NET не являются производными от единого общего интерфейса, каким в мире COM был интерфейс IUnknown. В интерфейсах самих по себе смысла не очень много. Однако если известно, что какой-либо класс реализует известный интерфейс, можно требовать от этого класса определенной функциональности. При создании собственного интерфейса на .NET-совместимом языке программирования можно произвести этот интерфейс сразу от нескольких базовых интерфейсов.

Как указано ранее, в классах и структурах может быть любое количество членов. Член (member) – это либо метод, либо свойство, либо поле, либо событие. Для любого члена в .NET существует набор характеристик.

Например, любой член в .NET характеризуется своей областью видимости (Public, Private, Protected и т. д.). Член можно объявить как абстрактный, чтобы воспользоваться возможностями полиморфизма при работе с производными классами. Члены могут быть статическими (Static), которые можно совместно использовать всеми объектами данного класса, и обычными – принадлежащими только одному объекту данного класса.

Перечисление (Enumeration) – это удобная программная конструкция, которая позволяет объединять пары «имя-значение» под указанным именем перечисления. Предположим, необходимо создать программу, в которой можно выбирать тип пользователя с соответствующим уровнем привилегированности. В этой ситуации очень удобно будет воспользоваться перечислением:

```
enum User-Type
{
        Type_1=100,
        Type_2=200,
        Type_3=300
}
```

В CTS все перечисления являются производными от единственного базового класса System.Enum. Этот базовый класс содержит множество по-

лезных членов, которые помогут в извлечении (и выполнении прочих операций) с парами «имя-значение».

Делегаты (Delegates) – в .NET это безопасный для типов эквивалент указателя на функцию в C. Однако между ними есть и существенное отличие. Делегат .NET – это уже не просто адрес в оперативной памяти, а класс, производный от базового класса MulticastDelegate. Делегаты очень полезны в тех ситуациях, когда нужно, чтобы одна сущность передала вызов другой сущности. Делегаты – это краеугольный камень в технологии обработки событий .NET.

В .NET предусмотрен богатый набор встроенных типов данных. Помимо всего прочего, этот набор еще и един для всех языков программирования .NET. Названия типов данных в языках .NET могут выглядеть поразному, но эти названия – всего лишь псевдонимы для встроенных системных типов данных .NET, определенных в библиотеке базовых типов. Перечень встроенных типов данных .NET представлен в табл. 1.2.

Таблица 1.2

Встроенные типы данных

Встроенный тип данных .NET	**Название в Visual Basic**	**Название в Visual C#**	**Название в Visual C++**
System.Byte	Byte	byte	char
System.SByte	Не поддерживается	sbyte	signed char
System.Int16	Short	short	short
System.Int32	Integer	int	int или long
System.Int64	Long	long	_int64
System.UInt16	Не поддерживается	ushort	unsigned short
System.UInt32	Не поддерживается	uint	unsigned int или unsigned long
System.UInt64	Не поддерживается	ulong	unsigned _int64
System.Single	Single	float	float
System.Double	Double	double	double
System.Object	Object	object	Object*
System.Char	Char	char	_wchar_t
System.String	String	string	String*
System.Decimal	Decimal	decimal	Decimal
System.Boolean	Boolean	bool	bool

Как видно из таблицы, не все языки .NET могут работать с некоторыми встроенными типами данных CTS. Поэтому очень важно определить

такой набор типов (и программных конструкций), с которым гарантированно смогут работать любые .NET-совместимые языки программирования. Такой набор есть, и он называется CLS [1–4].

Назначение стандартной спецификации языков CLS. Одни и те же программные конструкции в разных языках выглядят абсолютно поразному. Например, в C# объединение строк (конкатенация) производится с помощью оператора плюс (+), в то время как в Visual Basic для этой же цели используется амперсанд (&). Аналогично на Visual Basic функция, не возвращающая значений, выглядит как

```
Public Sub Foo()
End Sub
```

на C#:

```
public void Foo()
{
}
```

Для среды выполнения .NET такая разница в синтаксисе абсолютно безразлична: все равно соответствующие компиляторы (в данном случае vbc.exe и csc.exe) создадут одинаковый код IL. Однако языки программирования отличаются не только синтаксисом, но и возможностями. Например, в одних языках программирования разрешена перегрузка операторов, а в других нет. Одни языки могут использовать беззнаковые (unsigned) типы данных, а в других такие типы данных не предусмотрены. Вывод очевиден: нужны некие единые правила для всех языков .NET. Если им следовать, то гарантируется, что программные модули, написанные на разных языках, будут нормально взаимодействовать друг с другом. Такой набор правил определен в спецификации CLS (Common Language Specification).

Набор правил, определяемый CLS, не только гарантирует нормальное взаимодействие блоков кода, созданных на разных языках. Такой набор правил еще и определяет минимальные требования, которые предъявляются к любому .NET-совместимому компилятору. Необходимо помнить, что в CLS – это лишь часть тех возможностей, которые определены в CTS.

Правилам CLS должны удовлетворять и инструментальные средства среды разработки – если необходимо обеспечить межъязыковое взаимодействие, они должны генерировать только такой код, который соответствует требованиям CLS. У каждого правила CLS есть простое название (например, CLS Rule 6 – правило CLS номер 6). Например, самое важное правило – правило номер 1 – гласит: «Правила CLS относятся только к тем частям типа, которые предназначены для взаимодействия за пределами сборки, в которой они определены». Из этого правила явствует, что во внутренней логике при реализации какого-либо типа можно сколько угод-

но нарушать правила CLS – это ни на что не повлияет. Например, создается приложение .NET, которое взаимодействует с другими программными модулями с помощью трех классов, а в каждом из этих классов есть только одна функция. Правилам CLS в этом случае должны удовлетворять только три этих функции-члена (в отношении области видимости, соглашений об именовании, типов принимаемых параметров и так далее). Во внутренней реализации этих функций, классов или приложения в целом может быть сколько угодно отступлений от правил CLS – за пределами программного модуля это не видно.

Конечно, в CLS существует не только правило 1, но и множество других правил. В CLS, к примеру, строгие требования предъявлены к представлению символьных значений, к определению перечислений, к использованию статических членов и так далее. Однако заучивать эти правила совершенно не обязательно (конечно, если не создавать .NET-совместимый компилятор для оригинального языка программирования). Дополнительная информация по правилам CLS содержится в справочной системе Microsoft Developer Network Library for Visual Studio (MSDN Library for Visual Studio). Система также содержит собрание документов компании Microsoft со сведениями обо всех ее разработках.

Таким образом, в Microsoft Visual Studio в единую систему сведены все допустимые типы данных. Для этого используются концепции CTS и CLS [1–4].

1.4. Работа с пространствами имен .NET. Использование пространств имен в коде приложения. Обзор и основные возможности интегрированной среды разработки Microsoft Visual Studio

Работа с пространствами имен. Следует уделить внимание некоторым особенностям библиотеки базовых классов .NET. Важность библиотек кода очевидна. Например, библиотека MFC определяет набор классов C++ для создания диалоговых окон, меню и панелей управления. В результате программисты могут не заниматься изобретением того, что давным-давно уже сделано до них, а сосредоточиться на уникальных аспектах создаваемого ими приложения. Аналогичные средства существуют и в Visual Basic, и в Java, и во всех остальных языках программирования.

Однако в отличие от MFC, Visual Basic и Java в C# не существует библиотеки базовых классов только для этого языка. Можно сказать, что библиотека базовых классов C# вообще не существует. Вместо этого разработчики на C# используют библиотеку базовых типов для всей

среды .NET. А для лучшей организации типов внутри этой библиотеки используется концепция пространств имен.

Главное отличие от библиотек, привязанных к конкретному языку (типа MFC), заключается в том, что в любом .NET-совместимом языке используются те же самые типы и те же самые пространства имен, что и в C#. Пример приложений на трех разных .NET-совместимых языках:

Visual C#:

```
using System;
public class MyApp
(
        public static void Main()
        {
                Console.WriteLine ("Привет от Visual C#!");
        }
}
```

Visual Basic:

```
Imports System
Public Module MyApp
        Sub Main()
                Console.WriteLine ("Привет от Visual Basic!");
        End Sub
End Module
```

Visual C++:

```
using namespace System:
void main()
{
        Console: :WriteLine("Привет от Visual C++!");
}
```

В последнем примере среда выполнения .NET сама помещает глобальную функцию main внутрь определения класса. Если нужен класс Console, то в любом языке .NET используется одно и то же пространство имен System. Если не обращать внимания на синтаксические различия, то код приложений на разных языках .NET практически идентичен. Платформе .NET свойственно изящество единого стиля программирования.

Эффективность работы программиста, использующего .NET, напрямую зависит от того, насколько он знаком с тем массивом типов, который

определен в пространствах имен библиотеки базовых классов. Самое важное пространство имен в Visual C# – это System. В нем определены классы, которые обеспечивают самые важные функции Visual C#. Не удастся создать ни одно работоспособное приложение Visual C# без использования этого пространства имен.

Пространство имен – это просто способ организации типов (классов, перечислений, интерфейсов, делегатов и структур) в единую группу. Конечно, обычно в одном пространстве имен объединяются взаимосвязанные типы. Например, тип System.Drawing содержит набор типов, которые призваны помочь в организации вывода изображений на графическое устройство. В .NET предусмотрены пространства имен для организации типов для работы с базами данными, Web, многопоточностью, защитой данных и множества других задач. В табл. 1.3 приведены некоторые пространства имен .NET [1–4].

Таблица 1.3

Пример пространства имен .NET

Пространство имен .NET	**Назначение**
System	Внутри – множество низкоуровневых классов для работы с простыми типами, выполнения математических операций, сборки мусора и т. п.
System.Collections	Для работы с контейнерными объектами, такими как ArrayList, Queue, SortedList
System.Data System.Data.Common System.Data.OleDb System.Data.SqlClient	Для обращений к базам данных
System.Diagnostics	В этом пространстве имен содержатся многочисленные типы, используемые .NET-совместимыми языками для трассировки и отладки программного кода
System.Drawing System.Drawing.Drawing2D System.Drawing.Printing	Типы для примитивов GDI+ – растровых изображений, шрифтов, значков, поддержки печати. Предусмотрены также специальные классы для вывода более сложных изображений

Продолжение таблицы 1.3

Пространство имен .NET	Назначение
System.IO	Как следует из названия, в этом пространстве имен объединены типы, отвечающие за операции ввода-вывода – в файл, буфер т. п.
System.Net	Это пространство имен (как и все остальные, связанные с ним) содержит типы, относящиеся к передаче данных по сети (запрос-ответ, создание сокетов и т. п.)
System.Reflection System.Reflection.Emit	Классы, предназначенные для обнаружения, создания и вызова во время выполнения пользовательских типов
System.Runtime.InteropServices System.Runtime.Remoting	Средства для взаимодействия с «традиционным» кодом (Win32 DLL, COM-серверы) и типы, используемые для удаленного доступа (например, по коммутируемым соединениям)
System.Security	В мире .NET средства обеспечения безопасности интегрированы как со средой выполнения, так и с библиотекой базовых типов. В этом пространстве имен находятся классы для работы с разрешениями, криптографией и т. п.
System.Threading	Это пространство имен для типов, которые используются при работе с потоками (например, Mutex, Thread или Timeout)
System.Web	Классы, которые предназначены для использования в web-приложениях, включая ASP.NET
System.Windows.Forms	Классы для работы с элементами интерфейса Windows – окнами, элементами управления и пр.
System.XML	Множество классов для работы с данными в формате XML

Использование пространств имен в коде приложения. Пространство имен – это средство для логической группировки типов. С человеческой точки зрения выражение System.Console означает тип Console в простран-

стве имен System. Однако с точки зрения среды выполнения .NET System.Console – это единая сущность, к которой можно обратиться разными способами.

Слово using нужно для простоты обращения к типам в конкретном пространстве имен. Если по каким-либо причинам использовать слово using не-желательно, вполне можно обойтись и без него. Например, необходимо создать обычное оконное приложение Windows, которое должно представлять на круговой диаграмме информацию, извлекаемую из базы данных. Кроме того, еще нужно поместить на главную форму приложения растровый рисунок с логотипом компании. Можно определить, что в приложении потребуются классы из следующих пространств имен:

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Data;
using System.OleDb;
```

После того как определится использование конкретного пространства имен (с использованием ключевого слова using), можно обращаться к типам, содержащимся в этом пространстве. Например, если потребовалось создать экземпляр класса Bitmap (определенном в пространстве имен System.Drawing), код может быть таким:

```
using System.Drawing;

class MyClass
{
        public void DoIt()
        {
                Bitmap bm = new Bitmap (20, 20);
        }
}
```

Поскольку явно указано использование пространства имен System.Drawing с помощью ключевого слова using, компилятор сможет понять, что класс Bitmap – это член данного пространства имен. Если в примере, приведенном выше, опустить строку со словом using, получится сообщение компилятора об ошибке. Однако можно обойтись и без using, если использовать полное имя класса:

```
class MyClass
{
```

```
        public void DoIt()
        {
        System.Drawing.Bitmap bm = new.System.Drawing.Bitmap (20, 20);
        }
}
```

Главная идея: если явно указывать используемое пространство имен, строки при обращении к классам этого пространства имен получаются гораздо меньшего размера.

Помимо того, что можно явно указать используемое пространство имен с помощью ключевого слова using, иногда может потребоваться еще и явно указать физическое местонахождение сборки (библиотеки базовых классов .NET) с необходимым кодом IL. Многие важнейшие пространства имен .NET физически связаны с файлом mscorlib.dll. Типы пространства имен System.Drawing физически находятся внутри файла System.Drawing.dll. По умолчанию встроенные сборки .NET находятся в подкаталоге:

<имя_устройства>:\<каталог_операционной_системы>\Microsoft.NET\Framework\<номер_версии>

В зависимости от того, какие средства разработки используются для создания приложений .NET, существует много способов сообщить компилятору, какие именно сборки необходимо задействовать во время процесса компиляции.

Дополнительную информацию о сборках (пространствах имен и типах) можно найти, используя специальные инструменты:

- документация .NET SDK (в MSDN Library for Visual Studio);
- утилита Microsoft .NET Framework IL Disassembler;
- Web-приложение ClassViewer;
- графическое приложение Windows Forms Class Viewer;
- ObjectBrowser, входящий в комплект Microsoft Visual Studio [1–4].

Обзор и основные возможности интегрированной среды разработки Microsoft Visual Studio. Первое, что необходимо отметить относительно интегрированной среды разработки (Integrated Development Environment, IDE) Microsoft Visual Studio, – то, что эта среда теперь едина для всех языков программирования .NET. Таким образом, какой бы тип проекта ни создавался (ATL, MFC, Visual C#, Visual Basic, FoxPro, стандартный C++ и т. п.), программист все равно будет работать в одной и той же среде.

Microsoft Visual Studio – это новая версия Visual Studio и оболочки (каркаса) .NET Framework, превосходящая предыдущие версии по очень многим параметрам: она поддерживает новые и улучшенные объекты, включает среду разработки с обновленным интерфейсом и отличается интегрированной поддержкой системы управления базами данных Microsoft SQL Server 2005, позволяя создавать и развертывать проекты с применением сервера баз данных. Главное изменение в новой версии Visual Studio – расширение ее функциональных возможностей, выходящих за рамки среды разработки приложений. Из инструмента программиста, пишущего и отлаживающего код, в некоторых версиях Microsoft Visual Studio превратилась в полноценное инструментальное средство, позволяющее автоматизировать деятельность всех членов команды, работающих над проектом.

На первый взгляд, Microsoft Visual Studio не так уж сильно отличается от предыдущих версий (так, интерфейс Microsoft Visual Studio традиционно выполнен в одном стиле с последней версией Microsoft Office). Однако отличия есть. В частности, появились новые инструменты. Например, инструмент для контроля текущего процесса разработки Task List. В список задач помещают информацию об ошибках и о необходимых доработках. Каждому пункту можно назначить приоритет, а после выполнения установить флажок, сообщающий о завершении указанной задачи. Task List поддерживает сортировку записей по тексту, по приоритету и статусу. Свойства проекта в Microsoft Visual Studio можно редактировать с помощью встроенного инструмента, который позволяет изменять настройки и подписи сборки, ссылки на внешние модули и набор прав, необходимых для функционирования сборки.

И все это производится из одного окна, реализованного в виде набора вкладок. Кроме того, разработчик легко может сохранить настройки своего пользовательского IDE в файле настроек и применять его в случае перехода на другой компьютер.

Сам процесс написания и отладки программного кода упростился. В сравнении с предыдущей версией форматирование стало более ясным и интеллектуальным. Среда Microsoft Visual Studio еще в большей степени, чем в предыдущих версиях, ориентирована на быструю разработку качественных, надежных и производительных приложений с наименьшими затратами времени и сил.

Основная трудность, с которой встречаются программисты, заключается в ранее сделанных ошибках, которые либо снижают производительность, либо ведут к неверным результатам выполнения, поэтому важной задачей является отслеживание и исправление таких ошибок в момент написания кода. Механизм предупреждений разработчика о

недочетах программного кода в новой версии Microsoft Visual Studio получил дальнейшее развитие.

Одной из важнейших составляющих приложения является пользовательский интерфейс (та часть приложения, с которой имеет дело конечный пользователь). Эта часть приложения очень важна, так как на основе дизайна пользовательского интерфейса заказчик зачастую делает вывод о работе всего приложения. В новой версии упростилась процедура прорисовки дизайна пользовательского интерфейса за счет улучшенного механизма работы дизайнера форм и добавления в Framework новых компонентов Windows Forms.

Еще одна особенность Microsoft Visual Studio – его «сближение» с сервером баз данных. Одновременный выпуск этих двух продуктов не случаен, а вполне закономерен. Теперь Microsoft SQL Server полностью интегрирован с CLR. Это означает, что программист получает в распоряжение все возможности, предоставляемые .NET Framework. Разработчик баз данных может использовать объектно-ориентированные языки программирования, такие как Visual C# и Visual Basic, опираясь на широчайший спектр встроенных возможностей классов и методов .NET Framework. Кроме того, программист может воспользоваться компонентами, написанными сторонними компаниями. Разработчикам баз данных доступны два новых типа объектов – агрегаты и пользовательские типы. Объекты баз данных, написанные с применением .NET Framework, становятся более защищенными, так как управляемый код выполняется в среде CLR. Общая среда разработки всех компонентов приложения дает возможность создавать управляемые объекты Microsoft SQL Server, не покидая привычного окружения. Это позволяет использовать для создания и отладки объектов баз данных те же инструменты, что и для написания других компонентов и служб .NET Framework.

С появлением новой версии Microsoft SQL Server был усовершенствован механизм доступа к данным. В новой версии ADO.NET появился ряд дополнительных возможностей. Используя механизм асинхронного запуска команд, программист может посылать команды на выполнение с помощью методов с приставкой Begin и End. Эти методы реализуют асинхронный запуск командных объектов на основе стандартных механизмов .NET. Технология Multiple Active Result Sets (MARS) позволяет использовать соединение даже в том случае, когда открыт объект DataReader. Установив специальный параметр MultipleActiveResultSets в состояние true, программист получает возможность выполнять несколько запусков DataReader через одно соединение. В ADO.NET появился новый класс SqlBulkCopy, позволяющий организовать копирование данных в таблицу, используя DataSet или DataReader. Используя параметр Update-BathSize при обновле-

нии DataSet в источник данных, разработчик может регулировать количество записей для обновления в одном блоке. Увеличение этого числа может повлиять на производительность обновления. Наконец, ADO.NET поддерживает новые типы данных, появившиеся в новой версии Microsoft SQL Server. Таким образом, ADO.NET – наиболее подходящий инструмент для разработки приложений, взаимодействующих с Microsoft SQL Server.

Однако новая версия Microsoft Visual Studio – это не только удобный инструмент разработки и отладки широкого спектра приложений, но и упрощенный доступ к хранилищам данных, возможность выполнения тестирования и улучшение управляемости программного проекта в целом. Различные версии (редакции) Microsoft Visual Studio, предлагаемые клиентам, обладают разными новыми возможностями.

Компания Microsoft позаботилась о начинающих программистах и студентах. Именно для этой категории пользователей компания выпустила серию продуктов под редакцией Microsoft Visual Studio Express Edition. Эта версия отличается простотой и доступностью. В серию Express входят:

- инструмент для разработки Web-сайтов и Web-служб Visual Web Developer;
- инструменты программирования на разных языках (Visual Basic, Visual C#, Visual C++, Visual J#);
- Microsoft SQL Server Express Edition.

Microsoft SQL Server Express Edition – это система управления базами данных начального уровня. С помощью этих компактных, а главное, простых в изучении и использовании инструментов, начинающие программисты и любители могут создавать динамичные Windows-приложения и Web-сайты.

Редакция Microsoft Visual Studio Standard Edition представляет собой профессиональный инструмент начального уровня. В нем сочетается простота версий Express и мощные средства разработки, необходимые для создания клиентских приложений, работающих с данными, многоуровневых клиент-серверных приложений с использованием Web-служб и разнообразных Web-приложений. Среди улучшений в языке и среде разработки можно отметить пространство имен My, функции IntelliSense, технологии Code Snippets и функцию Edit and Continue. В сочетании с возможностями инструмента по созданию бизнес-приложений он представляет собой привлекательное предложение для миграции пользователей Visual Basic. Microsoft Visual Studio Standard Edition обеспечивает разработку бизнес-приложений, работающих с данными, хорошо приспособлен для разработки Web-приложений и при этом обладает упрощенным интерфейсом. Для повышения удобства и эффективности Web-разработки в Microsoft Visual Studio Standard

Edition имеются прекомпиляция Web-сайтов и базовая поддержка локализации сайтов. Microsoft Visual Studio Standard Edition позволяет:

- создавать приложения с использованием языков Visual Basic, Visual C#, Visual C++, Visual J#;
- создавать Windows- и Web-приложения для портативных устройств;
- создавать клиент-серверные приложения;
- использовать генератор отчетов SQL Reporting Services;
- создавать корпоративные Web-приложения масштаба предприятия.

Редакция Microsoft Visual Studio Professional Edition предназначена для создания надежных многоуровневых приложений для Microsoft Windows (Smart clients), Internet, мобильных устройств и для приложений Microsoft Office.

Наконец, профессионалы могут выбрать одну из трех ролевых версий (вариантов) Microsoft Visual Studio Team System (Microsoft Visual Studio Team Edition for Software Architects для архитекторов приложений, Microsoft Visual Studio Team Edition for Software Developers для разработчиков и Microsoft Visual Studio Team Edition for Software Testers для тестеров) и Microsoft Visual Studio Team Foundation Server (рис. 1.4 и 1.5).

Рис. 1.4. Microsoft Visual Studio Team System

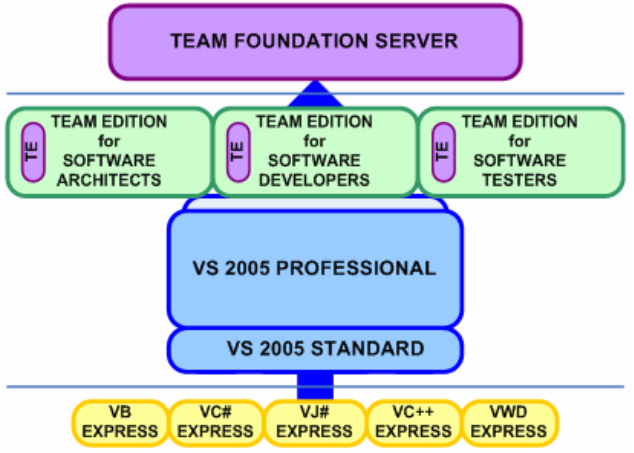


Рис. 1.5. Microsoft Visual Studio Team Foundation Server

Эти редакции ориентированы на архитекторов решений, аналитиков, разработчиков уровня предприятия, инженеров по тестированию, специалистов по развертыванию приложений, которые будут совместно работать в команде над созданием крупномасштабных распределенных приложений.

Версия для разработчиков приложений расширена за счет совершенно новых инстументов, которых Microsoft никогда ранее не производила. Это и анализ кода на C и C++, и средства для написания тестов модулей (unit-test), и встроенные в среду Microsoft Visual Studio инструменты для анализа управляемого кода и профилирования кода.

Дизайнеры распределенных систем (Distributed system designers), которые входят в версию для архитекторов, являются частью Dynamic Systems Initiative (DSI) – новой всеобьемлющей инициативы компании Microsoft. Эти дизайнеры и сопровождающие их инструменты представляют собой первую волну средств программирования, направленных на то, чтобы упростить и ускорить процесс разработки и внедрения сервис-ориентированных приложений.

Версия для тестеров четко обозначает роль и обязанности тестера-профессионала и включает средства для создания и управления различного вида тестов (например, стресс-тест) и их агентов, и виртуализацию машин с помощью виртуального сервера (Virtual server). Для того чтобы увеличить емкость нагрузочного тестирования, компания может приобрести агент для дополнительных нагрузочных тестов Microsoft Visual Studio Team Test Load Agent.

Взаимодействие, сотрудничество и обмен информацией является одной из самых серьезных проблем в командах разработчиков программного обеспечения. Решение этой проблемы компанией Microsoft – Microsoft Team Foundation Server, новый продукт в номенклатуре Microsoft Visual Studio. Microsoft Team Foundation Server – это ядро и связующее звено в цепи процесса групповой разработки программного продукта. Он предоставляет функции управления версиями, наблюдения за элементами потока операций (Work item tracking) и автоматизацию сборки программ (Build automation). За счет интеграции с другими сервисами и серверными процессами (как собственно Microsoft Visual Studio, так и других продуктов компании) Microsoft Team Foundation Server предоставляет практически неограниченные возможности для управления проектами. Например, при использовании Windows SharePoint Services создается портал для каждого проекта, а при помощи SQL Server Reporting Services можно публиковать различные отчеты о ходе проекта.

Лицензирование Microsoft Team Foundation Server работает по тому же принципу, что и лицензирование Microsoft SQL Server. Microsoft Team Foundation Server требует лицензию доступа клиента CAL (Client Access License) для каждого клиента, желающего присоединиться к серверу. Все продукты из серии Team System включают одну или несколько CAL и Team Explorer (обозреватель для доступа к общекомандным функциям). Для тех, кто хочет иметь продукты из различных изданий на одной машине, Microsoft выпускает Microsoft Visual Studio Team Suite, которая объединяет все три продукта из Team Edition в один (рис. 1.6).

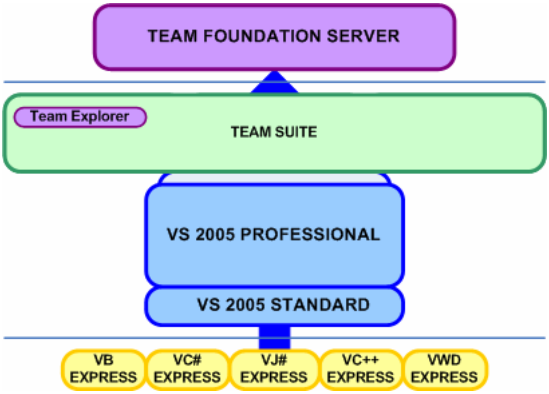


Рис. 1.6. Microsoft Visual Studio Team Suite

Кроме того, можно купить CAL лицензии для использования Team Explorer с Microsoft Visual Studio Professional Edition или Microsoft Visual Studio Standard Edition и таким образом обеспечить интеграцию изданий продукта с сервером Microsoft Team Foundation Server (рис. 1.7 и 1.8).

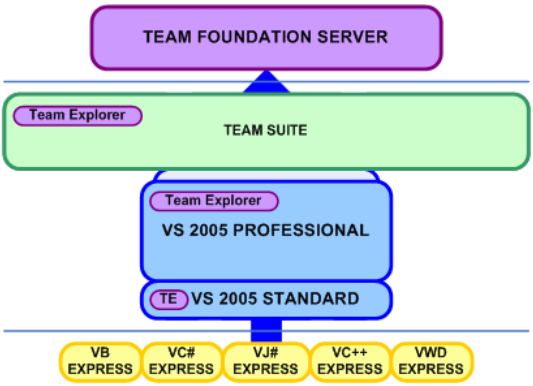


Рис. 1.7. Microsoft Visual Studio Team Explorer

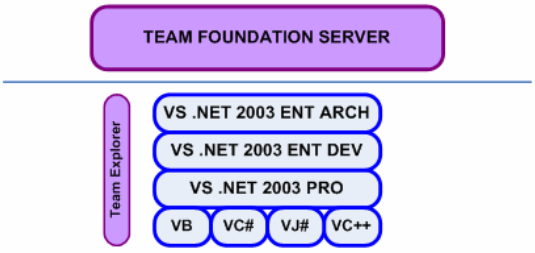


Рис. 1.8. Microsoft Visual Studio Team Explorer и Microsoft Visual Studio .NET

Если необходимо работать над проектом, который нельзя по каким-то причинам перевести или конвертировать на Microsoft Visual Studio, но желательно использовать возможности Microsoft Team Foundation Server в проекте, то можно воспользоваться Team Explorer для соединения с Microsoft Team Foundation Server из предыдущих версий Microsoft Visual Studio (при условии покупки лицензии CAL). Получается доступ к системе управления версиями, наблюдения за элементами потока операций и автоматизации сборки программ, но, к сожалению, в этом случае невозможно воспользоваться интегрированной версией и оценить все достоинства

и преимущества интеграции Microsoft Team Foundation Server и Microsoft Visual Studio.

Таким образом, Microsoft Visual Studio – это новый виток эволюции для систем программирования. С выходом продукта Microsoft Visual Studio Team System корпорация Microsoft выходит на рынок инструментов обеспечения полного жизненного цикла разработки приложений.

Начало работы со средой Microsoft Visual Studio. Прежде всего, конечно, надо запустить Microsoft Visual Studio и в меню File выбрать New, а затем Project. Как можно убедиться, типы проектов сгруппированы (по большей части) по используемым языкам программирования. Для наглядности в среде разработки Microsoft Visual Studio создана консольная версия программы, которая называется Test (рис. 1.9).

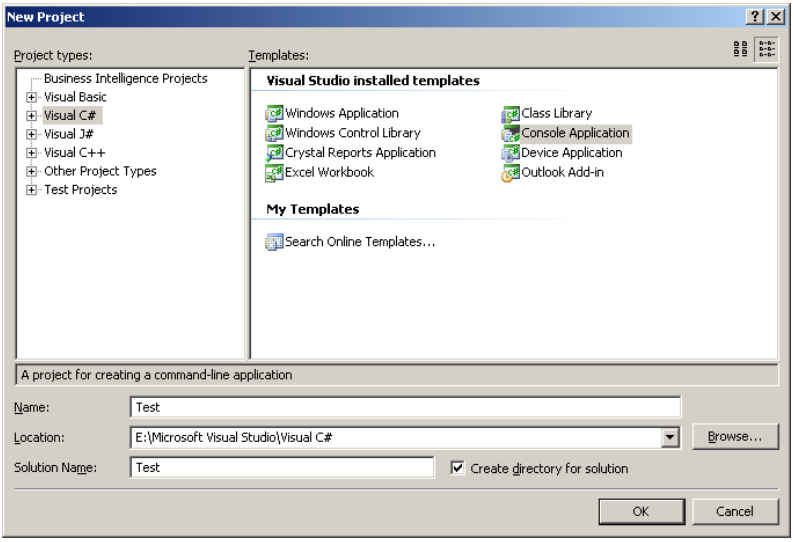


Рис. 1.9. Создание нового консольного проекта Visual C#

Окно Solution Explorer. В интегрированной среде разработки Microsoft Visual Studio проекты логически организуются в решения (Solutions). Каждое решение состоит из одного или нескольких проектов. В свою очередь, каждый проект может состоять из любого количества исходных файлов, ссылок на внешние сборки и прочих ресурсов, которые и образуют приложение. Можно открыть любой из данных ресурсов с помощью окна Solution Explorer – проводника решений (рис. 1.10).

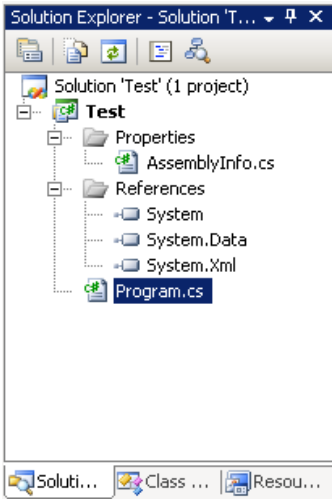


Рис. 1.10. Окно Solution Explorer

Следует обратить внимание, что по умолчанию первому классу в приложении присваивается имя Program.cs.

Окно Class View. Помимо обычной вкладки Solution Explorer в этом окне также предусмотрена вкладка Class View – для «объектно-ориентированного» отображения иерархии приложения (рис. 1.11).

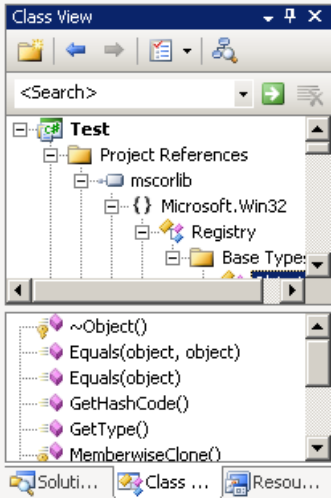


Рис. 1.11. Вкладка Class View

При щелчке правой кнопкой мыши на любом элементе в окне Class View в распоряжении будет контекстное меню, с помощью которого можно воспользоваться любым из множества CASE-средств, например для добавления в тип новых членов – методов, свойств, полей и т. д. (рис. 1.12).

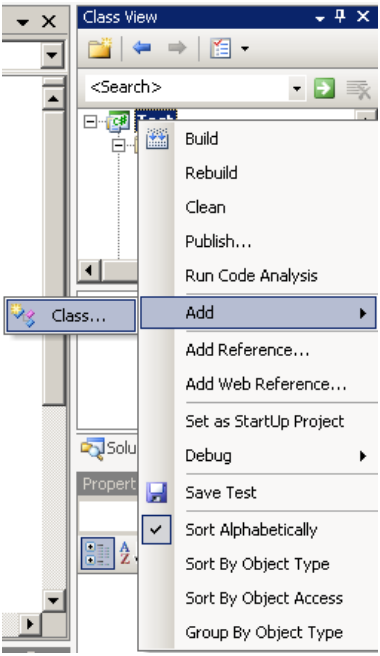


Рис. 1.12. Контекстное меню в окне Class View

Для лучшего понимания языка весь код лучше писать вручную, однако можно также воспользоваться интегрированными мастерами, что позволяет сэкономить время.

Окно Properties. Другой важный элемент интегрированной среды разработки – это окно Properties (Свойства). В этом окне отображаются важнейшие характеристики выделенного в настоящий момент элемента. Этим элементом может быть и исходный файл, и элемент управления графического интерфейса пользователя, и проект в целом. Например, чтобы изменить имя исходного файла, надо выбрать его в окне Solution Explorer и изменить значение свойства FileName в окне Properties (рис. 1.13).

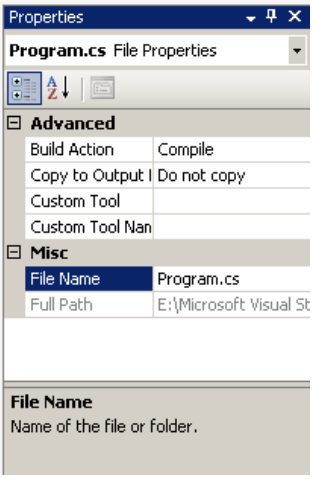


Рис. 1.13. Изменение имени файла с помощью окна Properties

Изменение имени класса производится в окне Class View – надо выбрать нужный класс и через контекстное меню переименовать его (рис. 1.14).

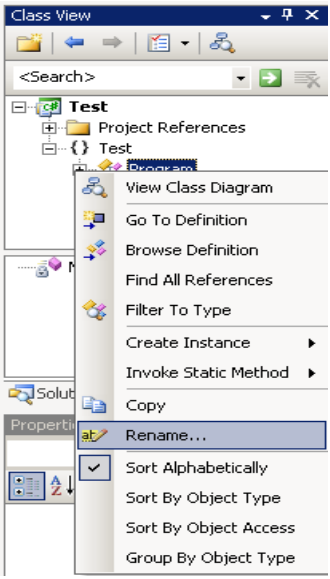


Рис. 1.14. Изменение имени класса с помощью окна Class View

Следует обратить внимание, что изменения автоматически будут произведены во всем коде.

Экономное размещение кода на экране. Еще одна полезная особенность среды разработки Microsoft Visual Studio – это возможность при отображении сжимать и разжимать участки программного кода, как при работе с обычной иерархией из дерева и узлов (рис. 1.15).

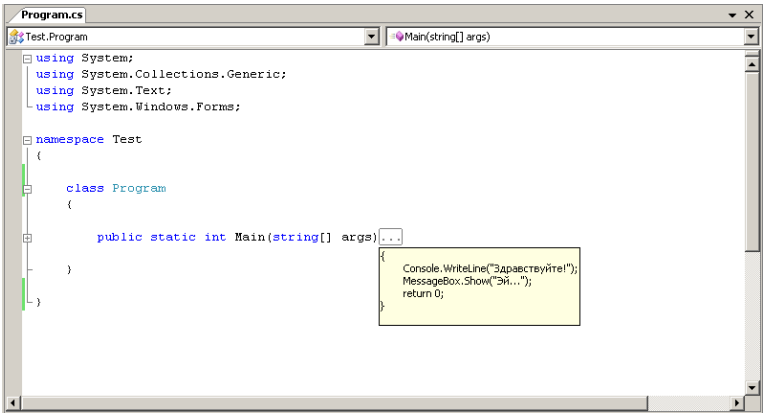


Рис. 1.15. Скрытие части кода

Нажатие на значок «+» приводит к открытию участка программного кода для выбранного элемента, нажатие на значок «–» позволяет скрыть данный отрезок кода, освободив место на рабочем столе. При наведении курсора мыши на многоточие, означающее скрытый для удобства код, данный код будет показан во всплывающем окне.

Конечно же, в среде разработки Microsoft Visual Studio предусмотрена полная поддержка технологии IntelliSense, «подсказывающей» в то время, когда набирается код, и предлагающей закончить за программиста начатую строку (рис. 1.16).

Ссылки на внешние сборки. Если потребовалось добавить ссылку на внешнюю сборку (в примере это была библиотека System.Windows.Forms.dll) в текущий проект, можно сделать это двумя способами: либо воспользоваться меню Project / Add Reference, либо сделать то же самое через контекстное меню для узла References (Ссылки) в окне Solution Explorer или Project References (Ссылки проекта) в окне Class View. В любом случае должно открыться диалоговое окно, представленное на рис. 1.17.

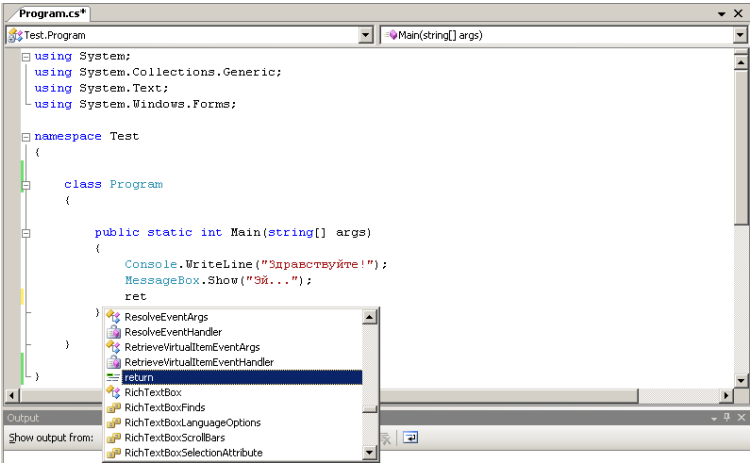


Рис. 1.16. Работа технологии IntelliSense

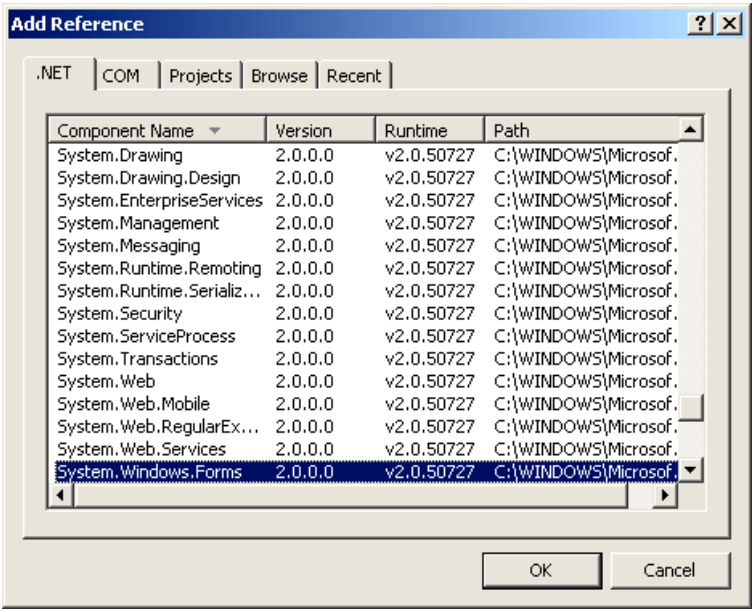


Рис. 1.17. Добавление ссылки на внешнюю сборку

Из рисунка видно, что в проект можно добавлять ссылки не только на внешние сборки .NET, но и на двоичные файлы COM, и на другие проекты. Для данного приложения в этом окне нужно добавить ссылку на System.Windows.Forms.dll.

Аналогично в окне Solution Explorer, выбрав необходимую ссылку на сборку, ее можно удалить (рис. 1.18).

Отладка в Microsoft Visual Studio. Как и в предыдущих версиях, в Microsoft Visual Studio, конечно же, предусмотрен интегрированный отладчик. Добавить контрольную точку можно, щелкнув мышью на самом левом сером столбце в окне кода – появится брейкпойнт (рис. 1.19).

При работе в режиме отладки выполнение приложения будет прерываться на каждом брейкпойнте. С помощью панели инструментов Debug (отладка) можно производить пошаговое выполнение (и также возвращаться назад). В интегрированном отладчике предусмотрено множество специальных окон (Breakpoints, Autos, Locals, Call Stack, Modules и так далее). Чтобы скрывать ненужные окна и, наоборот, открывать нужные, необходимо использовать меню Debug / Windows.

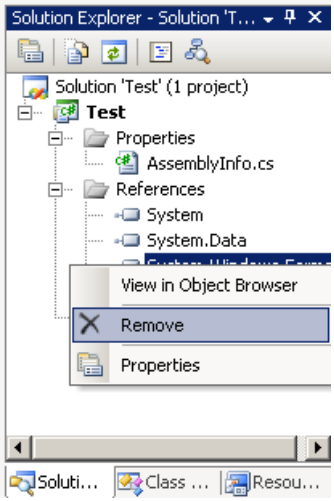


Рис. 1.18. Удаление ссылки на внешнюю сборку

Работа с окном Server Explorer. Еще одно очень важное средство Microsoft Visual Studio называется Server Explorer. Окно Server Explorer можно открыть с помощью меню View (рис. 1.20).

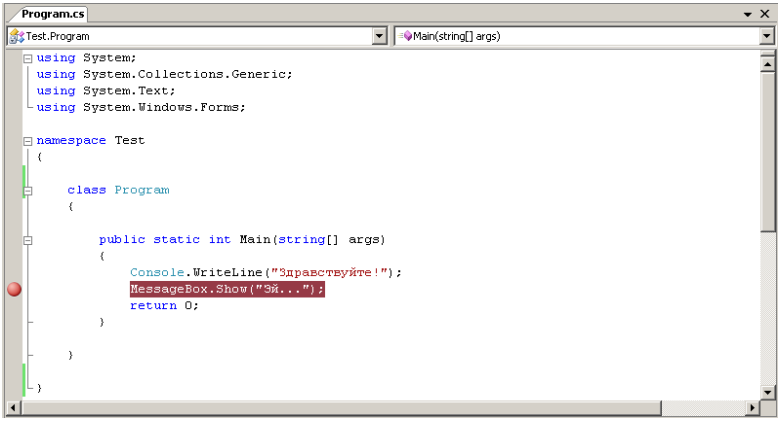


Рис. 1.19. Установка брейкпойнтов

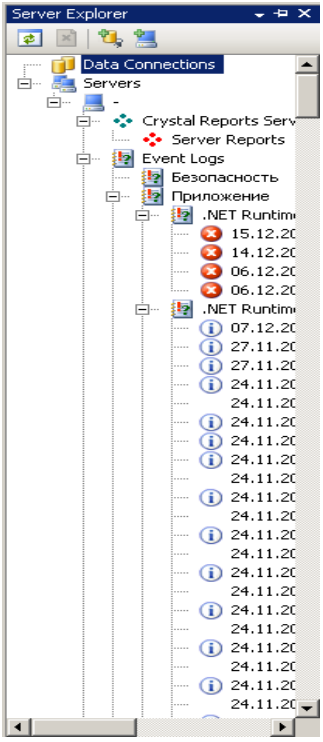


Рис. 1.20. Окно Server Explorer

Server Explorer можно рассматривать как командный центр при создании распределенных приложений. Используя Server Explorer, можно подключать локальные и удаленные базы данных и выполнять в них различные операции, организовать работу с очередями сообщений, работать с журналом событий, получать информацию о работающих службах и производить множество других действий.

Средства для работы с XML. В Microsoft Visual Studio предусмотрены встроенные средства для работы с XML (как и HTML) – расширяемой спецификацией языка, предназначенного для создания Web-страниц. Многие из этих средств были унаследованы от прежнего Visual InterDev. После подключения (или создания) файла XML к приложению можно производить редактирование его кода при помощи множества графических средств. В качестве примера на рис. 1.21 показано содержимое файла XML, который может быть создан с использованием возможностей технологии ADO.NET.

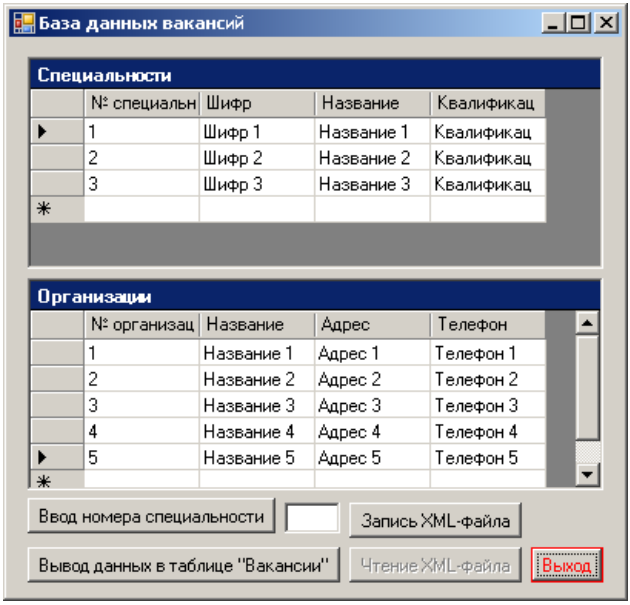


Рис. 1.21. Редактор XML

Поддержка диаграмм классов и UML. Microsoft Visual Studio предоставляет инструменты для создания диаграмм классов посредством использования контекстного меню в окнах Solution Explorer и Class View (рис. 1.22).

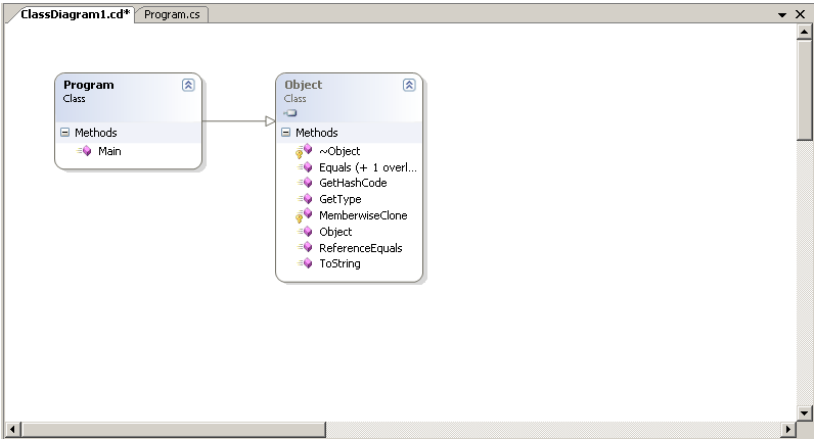


Рис. 1.22. Диаграмма классов

Также при помощи средств Microsoft Visio можно создавать диаграммы UML (Unified Modeling Language) типов приложения.

Утилита Object Browser. Помимо трех отдельных утилит для просмотра информации о типах можно воспользоваться встроенным просмотрщиком объектов Microsoft Visual Studio. Окно Object Browser открывается через меню View / Object Browser. Пример окна Object Browser представлен на рис. 1.23.

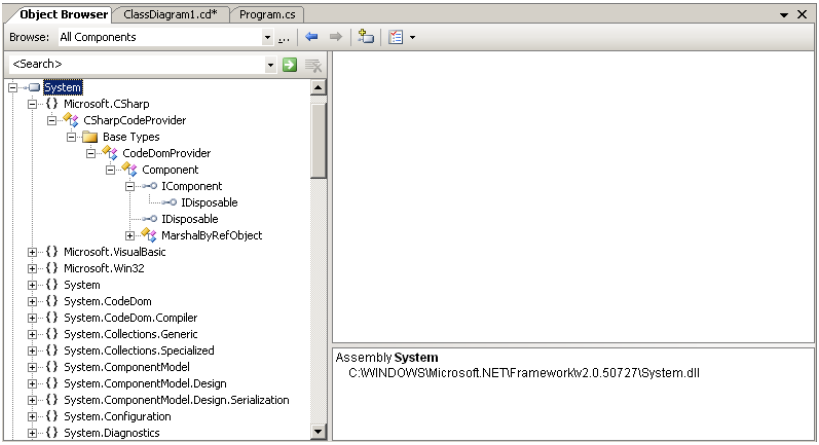


Рис. 1.23. Встроенная утилита Object Browser

Средства для работы с базами данных. В Microsoft Visual Studio предусмотрены встроенные средства для организации соединений с базами данных. После того как настроено подключение к базе данных в окне Server Explorer, можно производить с этой базой данных и ее объектами любые операции. На рис. 1.24 представлен пример структуры базы данных, созданной в Microsoft SQL Server.

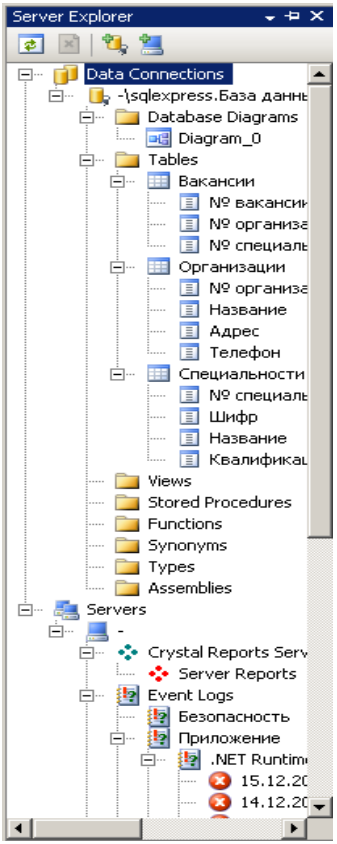


Рис. 1.24. Интегрированные средства для работы с базами данных

Встроенная справка. Последний аспект работы с Microsoft Visual Studio, о котором нельзя не упомянуть, – это встроенная справка. Вместо, того чтобы постоянно переключаться с помощью клавиш Alt + Tab между средой разработки и MSDN Library for Visual Studio, в Microsoft Visual Studio можно воспользоваться предусмотренным для этих целей окном

Dynamic Help. Содержимое этого окна динамически меняется в зависимости от того, какой именно элемент (окно, меню, ключевое слово в коде и т. п.) выделен в настоящий момент. Например, если поместить курсор на объявление метода Main(), то окно Dynamic Help примет вид, представленный на рис. 1.25.

Рис. 1.25. Окно Dynamic Help

Конечно же, по щелчку на гиперссылке в окне Dynamic Help откроется связанная с ней информация (рис. 1.26) [1–4].

Задание. Разработать приложение, по функциональности аналогичное приложению, исходный текст которого приведен на рис. 1.19. Отличие должно состоять в создаваемом решении, структура которого приведена на рис. 1.27.

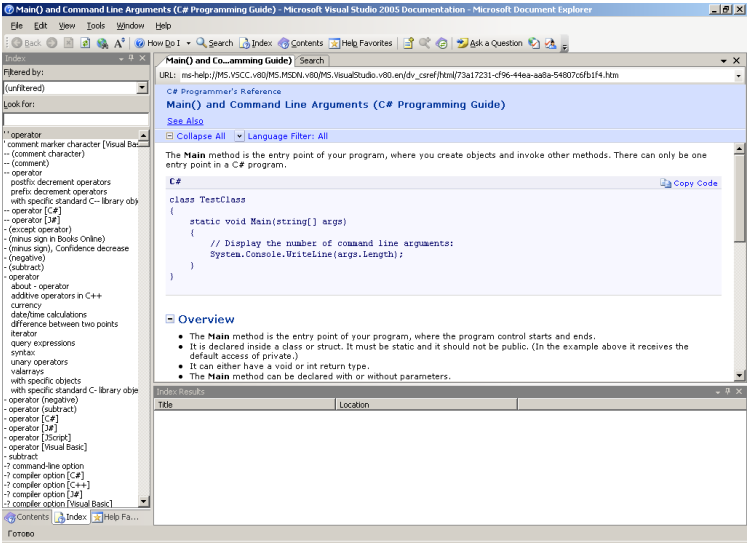


Рис. 1.26. Одна из страниц встроенной справки Microsoft Visual Studio

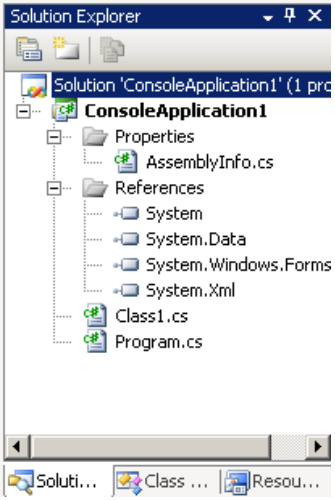


Рис. 1.27. Структура решения

В одном классе должен содержаться метод для вывода сообщения в графическое окно, а во втором – вывод сообщения в консоль и экземпляр класса, обладающего указанным методом (рис. 1.28).

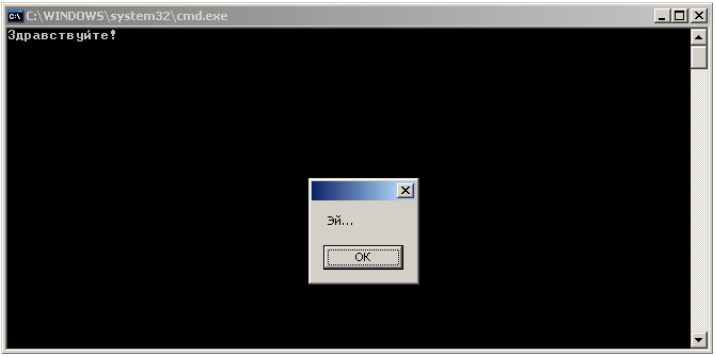


Рис. 1.28. Результат работы приложения

2. ОСОБЕННОСТИ ПРОЕКТИРОВАНИЯ И ПРОГРАММНОЙ РЕАЛИЗАЦИИ ИНФОРМАЦИОННЫХ СИСТЕМ С ИСПОЛЬЗОВАНИЕМ ДОСТУПА К ДАННЫМ ПРИ ПОМОЩИ ТЕХНОЛОГИИ ADO.NET

2.1. Обзор и причины создания технологии ADO.NET. Пространства имен. Типы пространства имен System.Data. Тип DataColumn. Создание объекта DataColumn и его добавление в DataTable. Создание столбца в качестве первичного ключа таблицы. Настройка автоматического увеличения значений для столбцов и представления столбца в формате XML. Тип DataRow. Работа со свойствами RowState и ItemArray

Обзор и причины создания технологии ADO.NET. В настоящее время базы данных занимают одну из самых популярных позиций у пользователей информационных систем. В платформе .NET определено множество типов (организованных в соответствующие пространства имен), которые обеспечивают взаимодействие с локальными и удаленными хранилищами данных. Общее название пространств имен с этими типами – ADO.NET. Это не просто перенос классической модели ADO, предполагающей работу с объектами данных ActiveX, на платформу .NET, но и ее коренная переработка.

Например, наиболее важные типы (классы) из пространства имен System.Data, такие как DataColumn, DataRow и DataTable, обеспечивают возможность взаимодействия с наборами данных, помещенных в память локального компьютера. Главный тип ADO.NET – DataSet. DataSet – это помещаемое в память компьютера представление о наборе взаимосвязанных таблиц. Можно программно моделировать отношения между таблицами, создавать пользовательские представления, производить запросы и создавать объекты DataSet, заполненные данными из распространенных СУБД, таких как Microsoft SQL Server, Oracle и Microsoft Access. Важно также иметь представление о концепции управляемых поставщиков (Managed providers) .NET и типах OleDbDataAdapter и SqlDataAdapter.

ADO.NET – это не просто улучшенная и расширенная версия классической ADO. У традиционной ADO и ADO.NET существуют как схожие черты (концепция объектов «соединения » и «командных» объектов), так

и существенные различия (к примеру, в ADO.NET уже нет такого важного типа, как Recordset). Самые важные типы ADO.NET, в свою очередь, не имеют эквивалентов в мире классического ADO (это справедливо, например, для DataSet).

ADO.NET – это новая технология доступа к базам данных, специально оптимизированная для нужд построения рассоединенных (Disconnected) систем на платформе .NET. Разработчики ADO.NET ориентировались на приложения N-tier – архитектуру многоуровневых приложений, которая в настоящее время стала фактически стандартом для создания распределенных систем.

Основные отличия ADO.NET от классической ADO:

- «старое» ADO было ориентировано прежде всего на создание клиент-серверных приложений, когда клиент и сервер должны постоянно взаимодействовать друг с другом;
- ADO.NET расширяет концепцию объектов-наборов записей в базе данных новым типом DataSet, который представляет локальную копию сразу множества взаимосвязанных таблиц. При помощи объекта DataSet пользователь может локально производить различные операции с содержимым базы данных, будучи физически рассоединен с СУБД, и после завершения этих операций передавать внесенные изменения в базу данных при помощи соответствующего «адаптера данных» (Data adapter);
- в ADO.NET реализована полная поддержка представления данных в XML-совместимых форматах. В ADO.NET закачанные для локальной обработки наборы данных представлены именно как XML (в этом же формате они и передаются с сервера баз данных). Поскольку данные в форматах XML очень удобно передавать при помощи обычного HTTP, сразу же решаются многие проблемы с установлением соединений через брандмауэры;
- в классическом ADO для перемещения данных между уровнями использовался протокол *маршалинга* (процесса упаковки запроса, включая параметры, в стандартный формат, пригодный для передачи по сети), использующего стандартные механизмы COM. Этот протокол имеет множество неоспоримых преимуществ, но у него есть и серьезные ограничения. Например, большинство брандмауэров не будут пропускать пакеты RPC при установлении соединений по этому протоколу, в результате чего развертывание многоуровневого приложения в реальных условиях предприятия может оказаться очень непростой задачей;
- ADO.NET – это библиотека управляемого кода, и взаимодействие с ней производится, как с обычной сборкой .NET. Типы ADO.NET используют возможности управления памятью CLR и могут использоваться во многих .NET-совместимых языках. При этом обращение к типам ADO.NET

(и их членам) производится практически одинаково вне зависимости от того, какой язык используется.

- Все типы ADO.NET предназначены для выполнения единого набора задач:
- установить соединение с хранилищем данных;
- создать и заполнить данными объект DataSet;
- отключиться от хранилища данных и вернуть изменения, внесенные в объект DataSet, обратно в хранилище данных.

Объект DataSet – это очень важный и интересный тип данных, представляющий локальный набор таблиц и информацию об отношениях между ними. В некоторых отношениях DataSet очень напоминает рассоединенный Recordset из «старой» ADO. Главное различие между рассоединенным Recordset и DataSet заключается в том, что Recordset представляет собой единственную таблицу, а DataSet – набор связанных таблиц. На практике ничто не мешает создать на клиенте объект DataSet, который будет представлять полную копию удаленной базы данных.

После создания объекта DataSet и его заполнения данными можно программными средствами производить запросы к нему, перемещаться по таблицам, выполнять все операции, как при работе с обычными базами данных (добавлять в таблицы новые записи, удалять и изменять существующие, применять к ним фильтры и т. п.). После того как клиент завершит внесение изменений, информация о них будет отправлена в хранилище данных для обработки.

DataSet создается при помощи *управляемого провайдера* (Managed provider). Управляемый провайдер – это набор классов, реализующих интерфейсы, определенные в пространстве имен System.Data. Речь идет об интерфейсах IDbCommand, IDbDataAdapter, IDbConnection и IDataReader (рис. 2.1).

В состав ADO.NET включены два управляемых провайдера: провайдер SQL и провайдер OleDb. Провайдер SQL специально оптимизирован под взаимодействие с Microsoft SQL Server. Для других источников данных предлагается использовать провайдер OleDb, который можно использовать для обращения к любым хранилищам данных, поддерживающим протокол OLE DB. Однако следует учесть, что провайдер OleDb работает при помощи «родного» OLE DB и требует возможности взаимодействия при помощи COM.

Несомненно, что скоро появятся управляемые провайдеры от производителей СУБД, которые будут эффективно взаимодействовать со «своими» источниками данных. Однако до этого времени в большинстве случаев придется использовать OleDb-провайдер [1, 5–7].

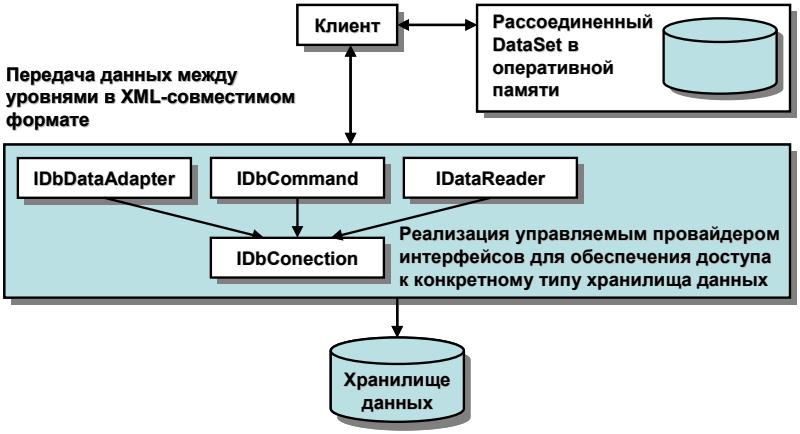


Рис. 2.1. Взаимодействие клиента с управляемыми провайдерами

Пространства имен. Все возможности ADO.NET заключены в типах, определенных в соответствующих пространствах имен. Краткий обзор главных пространств имен ADO.NET представлен в табл. 2.1.

Таблица 2.1

Пространства имен ADO.NET

Пространство имен	Описание
System.Data	Главное пространство имен ADO.NET. Типы, представляющие таблицы, столбцы, записи, ограничения и, конечно же, самый важный тип – DataSet. Здесь нет типов для подключения к источнику данных – только типы, представляющие сами данные
System.Data.Common	Типы, общие для всех управляемых провайдеров. Многие из них выступают в качестве базовых классов для классов из пространств имен для провайдеров SQL и OleDb
System.Data.OleDb	Типы для установления соединений с OLE DB-совместимыми источниками данных, выполнения к ним SQL-запросов и заполнения данными объектов DataSet. Здесь типы очень похожи на типы классического ADO

Продолжение таблицы 2.1

Пространство имен	Описание
System.Data.SqlClient	Типы, которые составляют управляемый провайдер SQL. С их помощью можно выполнять очень эффективное взаимодействие с Microsoft SQL Server напрямую, минуя промежуточные преобразования (как в случае использования OleDb)
System.Data.SqlTypes	«Родные» типы данных Microsoft SQL Server. Конечно же, можно использовать и стандартные типы данных CLR, но использование этих типов позволяет добиться наивысшей производительности при работе с Microsoft SQL Server

Все пространства имен ADO.NET расположены в одной сборке – System.Data.dll (рис. 2.2).

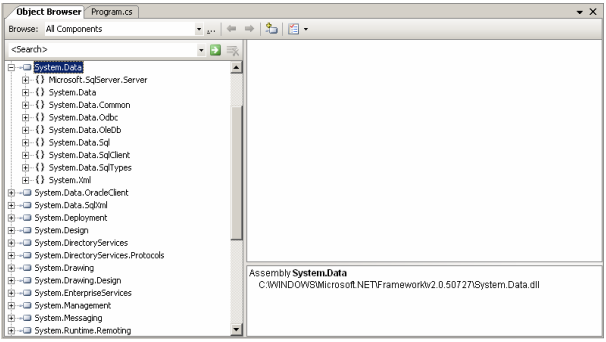


Рис. 2.2. Сборка System.Data.dll

Это означает, что в любом проекте, использующем ADO.NET, необходимо добавить ссылку на эту сборку.

Таким образом, в любом приложении ADO.NET необходимо использовать по крайней мере одно пространство имен – System.Data. Кроме того, практически во всех ситуациях потребуется использовать еще либо пространство имен System.Data.OleDb или System.Data.SqlClient – для установления соединения с источником данных. Для создания приложения ADO.NET необходимы определенные условия и знание типов основных пространств имен [1, 5–7].

Типы пространства имен System.Data. Эти типы предназначены для представления данных, полученных из источника (но не для установления соединения непосредственно с источником). В основном эти типы представляют собой объектные представления примитивов для работы с базами данных – таблицами, строками, столбцами, ограничениями и т. п. Наиболее часто используемые типы System.Data представлены в табл. 2.2.

Таблица 2.2

Типы пространства имен System.Data

Тип	Назначение
DataColumnCollection DataColumn	DataColumn представляет один столбец в объекте DataTable, DataColumnCollection – все столбцы
ConstraintCollection Constraint	Constraint – объектно-ориентированная оболочка вокруг ограничения (например, внешнего ключа или уникальности), наложенного на один или несколько DataColumn, ConstraintCollection – все ограничения в объекте DataTable
DataRowCollection DataRow	DataRow представляет единственную строку в DataTable, DataRowCollection – все строки в DataTable
DataRowView DataView	DataRowView позволяет создавать настроенное представление единственной строки, DataView – созданное программным образом представление объекта DataTable, которое может быть использовано для сортировки, фильтрации, поиска, редактирования и перемещения
DataSet	Объект, создаваемый в оперативной памяти на клиентском компьютере. DataSet состоит из множества объектов DataTable и информации об отношениях между ними
ForeignKeyConstraint UniqueConstraint	ForeignKeyConstraint представляет ограничение, налагаемое на набор столбцов в таблицах, связанных отношениями «первичный-внешний ключ». UniqueConstraint – ограничение, при помощи которого гарантируется, что в столбце не будет повторяющихся записей
DataRelationCollection DataRelation	Тип DataRelationCollection представляет набор всех отношений (т. е. объектов DataRelation) между таблицами в DataSet
DataTableCollection DataTable	Тип DataTableCollection представляет набор всех таблиц (объектов DataTable) в DataSet

Кроме того, в этом пространстве имен определены важные исключения, которые могут быть сгенерированы при работе с базами данных (NoNullAllowedException, RowNotInTableException, MissingPrimaryKeyException и т. п.).

Работать с типами из System.Data можно как вручную (создавать населенный объект DataSet), так и при помощи управляемого провайдера [1, 5–7].

Тип DataColumn. Тип DataColumn представляет отдельный столбец в таблице (которая, в свою очередь, должна быть представлена объектом DataTable). Собственно, столбцы с необходимыми атрибутами и составляют таблицу. Например, есть таблица «Сотрудники» с тремя столбцами: «Номер сотрудника», «Фамилия» и «Имя». Для представления данных этой таблицы потребуются три объекта DataColumn – по числу столбцов. Как вскоре станет очевидным, объект DataTable работает с внутренней коллекцией типов DataColumn, доступ к которой производится через свойство Column.

Известно, что на столбцы в таблице могут быть наложены ограничения. Например, столбец может быть определен в качестве первичного ключа, ему может быть назначено значение по умолчанию, его можно сделать доступным только для чтения и т. п. Кроме того, каждому столбцу в таблице должен соответствовать определенный тип данных (например, int, varchar и т. п.). В таблице «Сотрудники», например, можно использовать для «Номер сотрудника» целочисленный тип данных int, а для FirstName и LastName – символьные массивы varchar. Для того чтобы реализовать возможности, связанные с ограничениями, в классе DataColumn предусмотрено большое число свойств. Наиболее важные из них представлены в табл. 2.3 [1, 5–7].

Таблица 2.3

Свойства класса DataColumn

Свойство	**Описание**
AllowDbNull	Используется для определения того, может ли столбец содержать значения типа NULL (пустые значения). По умолчанию – может (значение этого свойства равно True)
AutoIncrement AutoIncrementSeed AutoIncrementStep	Эти свойства используются для настройки автоматического приращения значений в столбце. Это может быть полезно, если необходимо обеспечить уникальность значений в столбце (например, для первичного ключа). По умолчанию автоматическое приращение значений в столбцах отключено

Продолжение таблицы 2.3

Свойство	Описание
Caption	Определяет заголовок столбца для отображения в пользовательском приложении (например, этот заголовок может быть использован в DataGrid)
ColumnMapping	Определяет, как будет представлен столбец (объект DataColumn) при сохранении DataSet в формате XML (при использовании метода DataSetWriteXml())
ColumnName	Позволяет получить или установить имя столбца в коллекции Columns (внутренняя коллекция для столбцов в DataTable). Если имя столбца не определено явно, будут использованы значения по умолчанию: Column1, Column2, Column3 и т. д.
DataType	Определяет тип данных (Boolean, String, Float и т. п.), используемый для значений в столбце
DefaultValue	Позволяет установить или получить значение по умолчанию для столбца. Это значение будет автоматически использовано, если при вставке новой строки не укажется явно другое значение
Expression	Позволяет получить или установить выражение, используемое для фильтрации новых строк, вычисления значения в столбце или создания столбцов с агрегатными значениями
Ordinal	Позволяет установить порядковый номер столбца в коллекции Columns в DataTable
Readonly	Определяет, будет ли столбец только для чтения (если да, то значения в этом столбце после добавления строки изменять уже будет невозможно). По умолчанию имеет значение False
Table	Возвращает DataTable, которому принадлежит данный объект DataColumn
Unique	Позволяет определить, будут ли в столбце допускаться повторяющиеся значения. Если столбец является первичным ключом, то это свойство должно иметь значение True

Создание объекта DataColumn и его добавление в DataTable. Чтобы показать возможности работы с DataColumn, можно смоделировать при помощи этого типа столбец «Имя» (символьного типа). Также по условиям задачи необходимо, например, чтобы этот столбец был доступен только для чтения. Код для создания соответствующего объекта DataColumn может быть таким:

```
private DataColumn colFName = new DataColumn();

private void button1_Click(object sender, System.EventArgs e)
{

        colFName.DataType = Type.GetType("System.String");
        colFName.ColumnName = "Имя";
        colFName.Caption = "Имя";
        colFName.ReadOnly = true;
        string temp =
                "Тип данных: " + colFName.DataType + "\n" +
                "Заголовок: " + colFName.Caption + "\n" +
                "Имя: " + colFName.ColumnName + "\n" +
                "Только для чтения: " + colFName.ReadOnly + "\n" +
                "Пустые значения: " + colFName.AllowDBNull;
        MessageBox.Show(temp, "Вывод свойств столбца 'Имя'");
}
```

Результат выполнения этого кода, иллюстрирующий возможности некоторых свойств типа DataColumn представлен на рис. 2.3.

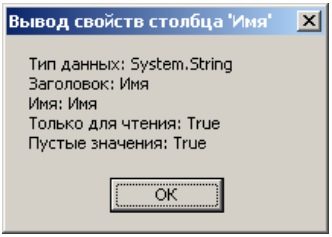


Рис. 2.3. Некоторые свойства типа DataColumn

Для типа DataColumn предусмотрено несколько перегруженных конструкторов, при помощи которых можно определять многие свойства объекта в момент его создания:

```
DataColumn colFName =new DataColunn ("Имя",
Type.GetType("System.String"));
```

Помимо рассмотренных свойств в типе DataColumn также имеется набор методов, информацию о которых можно получить при помощи электронной документации к C# в MSDN Library for Visual Studio.

Как правило, столбцы существуют не сами по себе, а внутри таблицы. Чтобы поместить столбец (объект DataColumn) в таблицу, потребуется

объект таблицы (DataTable). Создание объекта DataTable будет рассмотрено позднее, а пока предполагается, что он уже создан. Объект DataColumn добавляется во внутреннюю коллекцию DataTable.DataColumnCollection, а производится эта операция при помощи свойства Columns:

```
private DataTable Table = new DataTable("Сотрудники");

private DataColumn colFName = new DataColumn();

Table.Columns.Add(colFName);
```

Таким образом, свойство Columns возвращает объект типа DataColumnCollection, а для добавления столбца в таблицу используется метод Add() [1, 5–7].

Создание столбца в качестве первичного ключа таблицы. Один из основных принципов проектирования баз данных гласит: в каждой таблице обязательно должен быть первичный ключ (Primary key). Ограничение первичного ключа используется для того, чтобы однозначно идентифицировать любую строку таблицы. В примере с таблицей «Сотрудники» первичным ключом будет столбец «Номер сотрудника». Значения в столбце, определенном как первичный ключ, должны быть уникальными, кроме того, в нем не допускаются значения типа NULL (пустые значения), поэтому придется настроить соответствующие значения свойств AllowDBNull и Unique:

```
DataColumn colEmpID = new DataColumn();

colEmpID.DataType = Type.GetType("System.Int32");
colEmpID.ColumnName = "Номер сотрудника";
colEmpID.Caption = "Номер сотрудника";
colEmpID.AllowDBNull = false;
colEmpID.Unique = true;
```

Однако это пока не все: чтобы столбец стал первичным ключом таблицы, еще потребуется воспользоваться свойством PrimaryKey, которое будет рассмотрено далее [1, 5–7].

Настройка автоматического увеличения значений для столбцов и представления столбца в формате XML. В реальных таблицах очень часто используется свойство автоинкремента – автоматического увеличения значения столбца при добавлении в таблицу новых строк. Как только добавляется новая строка в таблицу, в соответствующее поле этой строки будет автоматически добавлено новое значение, основанное на значении

предыдущей строки и на шаге приращения. Обычно такие столбцы называются столбцами счетчика (identity columns). Такое решение позволяет создавать уникальные (в большинстве случаев) значения для строк автоматически.

При определении столбца счетчика потребуется настроить значения свойств AutoIncrement (определяет сам тип столбца и указывает, что значения для него нужно будет автоматически увеличивать), AutoIncrementSeed (начальное значение для столбца) и AutoIncrementStep (шаг приращения). Соответствующий код может быть таким:

```
DataColumn colEmpID = new DataColumn();

colEmpID.DataType = Type.GetType("System.Int32");
colEmpID.ColumnName = "Номер сотрудника";
colEmpID.Caption = "Номер сотрудника";
colEmpID.AutoIncrement = true;
colEmpID.AutoIncrementSeed = 1;
colEmpID.AutoIncrementStep = 1;
colEmpID.AllowDBNull = false;
colEmpID.Unique = true;
Table.Columns.Add(colEmpID);
```

Таким образом, определены исходное значение для столбца «Номер сотрудника», равное 1, и шаг приращения – 1. Первому значению в этом столбце будет присвоено значение 1, а последующим – соответственно 1, 2, 3 и т. д.

Для проверки сказанного, можно вставить столбец «Номер сотрудника» в таблицу, затем добавить в нее несколько строк и посмотреть, что будет со значениями указанного столбца для этих строк:

```
private void button1_Click(object sender, System.EventArgs e)
{

        DataColumn colEmpID = new DataColumn();

        colEmpID.DataType = Type.GetType("System.Int32");
        colEmpID.ColumnName = "Номер сотрудника";
        colEmpID.Caption = "Номер сотрудника";
        colEmpID.AutoIncrement = true;
        colEmpID.AutoIncrementSeed = 1;
        colEmpID.AutoIncrementStep = 1;
        colEmpID.AllowDBNull = false;
```

```
        colEmpID.Unique = true;
        Table.Columns.Add(colEmpID);
        DataRow dr1;

        for (int i = 0; i < 10; i++)
        {
                dr1 = Table.NewRow();
                Table.Rows.Add(dr1);
        }

}

private void button3_Click(object sender, EventArgs e)
{
        string temp = "";
        DataRowCollection rows = Table.Rows;

        for (int i = 0; i < Table.Rows.Count; i++)
        {
                DataRow currRow = rows[i];
                temp += currRow["Номер сотрудника"] + " ";
        }

        MessageBox.Show(temp, "Вывод автоматически увеличенных
значений");
}
```

После запуска приложения и нажатия на соответствующую кнопку появится окно, представленное на рис. 2.4.

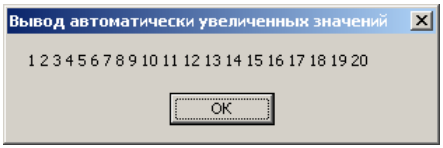


Рис. 2.4. Столбец счетчика – автоматическое увеличение значений

Подавляющее большинство свойств типа DataColumn, которые остались нерассмотренными, являются вполне очевидными (особенно если программист владеет соответствующей терминологией для работы с базами данных). Однако на одном свойстве хотелось бы остановиться подробнее. Это свойство – ColumnMapping, и оно определяет, как данный столбец

будет представлен в формате XML при извлечении содержимого столбца при помощи метода WriteXmL(). Для свойства ColumnMapping используются значения из перечисления MappingType (табл. 2.4).

Таблица 2.4

Значения перечисления MappingType

Значение перечисления MappingType	**Описание**
Attribute	Столбцу соответствует атрибут XML
Element	Столбцу соответствует элемент XML (это значение используется по умолчанию)
Hidden	Столбцу соответствует внутренняя структура
TableElement	Столбцу соответствует значение таблицы
Text	Столбцу соответствует значение текста

По умолчанию для ColumnMapping используется значение MappingType.Element. Это значит, что, например, во время записи содержимого DataSet в текстовый файл в формате XML при использовании значения Element столбцу «Номер сотрудника» в текстовом файле будут соответствовать значения вида:

```
<Сотрудники>
        <Номер сотрудника>1</Номер сотрудника >
</Сотрудники >
```

Если же установить для свойства ColumnMapping значение MappingType.Attribute, то столбцу «Номер сотрудника» в текстовом файле XML будут соответствовать уже следующие строки:

```
<Сотрудники Номер сотрудника = "1"/>
```

Подробнее возможности интеграции ADO.NET и XML будут приведены при рассмотрении типа DataSet [1, 5–7].

Тип DataRow. Из вышерассмотренного следует, что структура таблицы определяется как коллекция объектов DataColumn. Для хранения этой коллекции в объекте DataTable используется внутренний объект DataColumnCollection. Помимо этого, для DataTable важна еще одна коллекция: коллекция объектов DataRow, которая и определяет собственно данные, хранящиеся в таблице. Если в таблице «Сотрудники» предусмотрено 20 записей для сотрудников, каждую запись будет представлять от-

дельный объект DataRow. При помощи членов класса DataRow можно производить операции вставки, изменения и удаления строк из таблицы, а также сравнивать значения, которые содержатся в строках.

Работа с DataRow несколько отличается от работы с DataColumn, поскольку объект DataRow не нужно создавать напрямую. Вместо этого получается на него ссылка при помощи объекта DataTable. Например, потребовалось вставить новую строку в таблицу «Сотрудники». Для этого потребуется метод NewRow() (предполагается, что объект row уже создан):

```
row = Table.NewRow();
row["Номер сотрудника"] = 20;
row["Имя"] = "Иван";
row["Фамилия"] = "Иванов";
Table.Rows.Add(row);
```

Как видно из примера, для хранения данных о строках в объекте DataTable используется еще одна внутренняя коллекция – DataRowCollection.

Наиболее важные свойства типа DataRow представлены в табл. 2.5.

Таблица 2.5

Члены класса DataRow

Член	**Назначение**
AcceptChanges() RejectChanges()	Для записи в строку (или отказа от них) всех изменений, произведенных начиная с момента, когда последний раз был вызван метод AcceptChanges()
BeginEdit() EndEdit() CancelEdlt()	Начать, закончить и прекратить операции редактирования для объекта DataRow
Delete()	Помечает строку для удаления при следующем вызове метода AcceptChanges()

Продолжение таблицы 2.5

Член	Назначение
HasErrors GetColumnsInErrors() GetColumnError() ClearErrors() RowError	Свойство HasErrors возвращает логическое значение, определяющее, присутствуют ли ошибки в значениях столбцов для данной строки. Если такие ошибки есть, то для получения значений, которые нарушают установленные правила, можно использовать метод GetColumnsInError(). Для получения описания ошибки можно использовать GetColumnError(), а ClearErrors() просто удаляет все ошибочные значения из строки. Свойство RowError позволяет настроить текстовое описание для ошибки в столбце
IsNull()	Возвращает информацию о том, содержит ли строка в указанном поле пустое значение (значение типа NULL)
ItemArray	Позволяет получить или установить значения всех полей строки при помощи массива объектов
RowState	Позволяет получать информацию о текущем состоянии объекта DataRow. Используются значения из перечисления RowState
Table	Это свойство используется для получения указателя на таблицу, содержащую текущий объект DataRow

Кроме того, класс DataRow определяет индексатор, при помощи которого можно получить значение из поля строки по порядковому номеру. Конечно же, то же самое значение можно будет получить и по имени столбца [1, 5–7].

Работа со свойствами RowState и ItemArray. Большинство методов и свойств класса DataRow используется только в контексте размещения объекта DataRow в таблице (DataTable). Перед ознакомлением с операциями вставки, удаления и изменения строк следует рассмотреть свойство RowState класса DataRow. Главное назначение этого свойства – определять (в процессе выполнения программы), в каком состоянии находятся выбранные строки в таблице: были ли они изменены, только что вставлены и т. п. Для свойства RowState используются значения из перечисления DataRowState (табл. 2.6).

Таблица 2.6

Значения перечисления DataRowState

Значение	Описание
Added	Строка была добавлена при помощи метода Add()
Deleted	Строка была изменена при помощи метода Delete()
Detached	Строка была создана, но она еще не является частью DataRowCollection. Обычно строка находится в таком состоянии непосредственно после вставки или после принудительного вывода из коллекции DataRowCollection
Modified	Строка была изменена, но метод AcceptChanges() еще не вызывался
New	Строка была добавлена в коллекцию DataRowCollection, но метод AcceptChanges() еще не был вызван
Unchanged	Строка не была изменена с момента последнего вызова метода AcceptChanges()

Проиллюстрировать текущее состояние DataRow можно при помощи следующего кода:

```
private void button4_Click(object sender, EventArgs e)
{
        string temp = "Состояние: " + row.RowState;
        MessageBox.Show(temp, "Вывод текущего состояния введенной
строки");
}
```

Результат работы представлен на рис. 2.5.

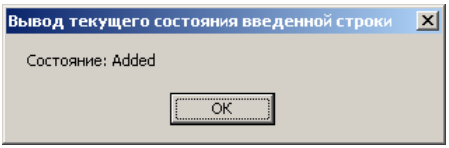


Рис. 2.5. Состояние строки

Таким образом, ADO.NET позволяет отслеживать различные состояния строк. Очень важно, что при помощи свойства RowState можно выяснить, какие строки были изменены, а какие – нет. Основываясь на этой информации, ADO.NET передает по сети для записи в базу данных только те строки, которые были реально изменены пользователем, что во многих случаях экономит значительное количество сетевого трафика. Знание воз-

можностей работы со свойством RowState позволит оптимизировать передачу данных от одного уровня приложения другому.

Еще один очень важный член класса DataRow – свойство ItemArray. Это свойство позволяет получить полный «снимок» текущей строки в виде массива объектов типа System.Object. Кроме того, при помощи этого свойства можно вставить в таблицу новую строку, не указывая явно значения для каждого столбца. Например, предполагается, что в таблице есть два столбца – «Номер сотрудника» и «Имя». Можно добавить новые строки, передавая массив объектов через свойство ItemArray:

```
object[] Vals = new object[3];
DataRow dr2;

for (int i = 11; i <= 19; i++)
{
        Vals[0] = i;
        Vals[1] = "Имя: " + i;
        dr2 = Table.NewRow();
        dr2.ItemArray = Vals;
        Table.Rows.Add(dr2);
}

foreach (DataRow r in Table.Rows)
{

        foreach (DataColumn c in Table.Columns)
        {
                textBox1.AppendText(r[c].ToString() + "\n");
        }

}
```

Результат работы программы представлен на рис. 2.6.

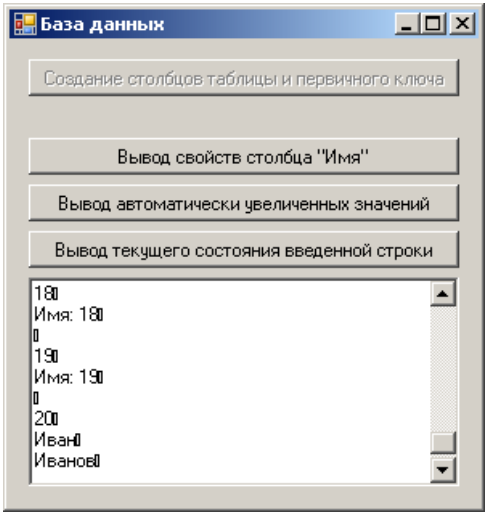


Рис. 2.6. Применение свойства ItemArray

Таким образом, ADO.NET – это новая технология доступа к данным, специально разработанная для применения в многоуровневых приложениях, в которых обеспечить постоянное соединение с источником данных не представляется возможным. Большинство типов, которые необходимы для обеспечения взаимодействия со строками, столбцами, таблицами и представлениями, находятся в пространстве имен System.Data. В пространствах имен System.Data.SqlClient и System.DataOleDb определены типы, которые позволяют устанавливать соединение с источниками данных Microsoft SQL Server и OLE DB [1, 5–7].

2.2. Тип DataTable. Создание объекта DataTable. Удаление строк из таблицы. Применение фильтров и порядка сортировки. Внесение изменений в строки. Тип DataView. Возможности и члены класса DataSet. Создание объекта DataSet. Моделирование отношения между таблицами при помощи класса DataRelation. Переход между таблицами, участвующими в отношении. Чтение и запись объектов DataSet в формате XML

Тип DataTable. Класс DataTable используется для создания в оперативной памяти моделей табличных наборов данных. Можно создавать объекты DataTable программным образом, однако чаще в приложениях объект DataTable создается автоматически с помощью возможностей DataSet и типов, определенных в пространствах имен System.Data.OleDb и System.Data.SqlClient. Наиболее важные свойства DataTable представлены в табл. 2.7.

Таблица 2.7

Свойства класса DataTable

Свойство	Описание
CaseSensitive	Определяет, будет ли при сравнении символьных данных в таблице учитываться регистр символов. По умолчанию – false (не будет)
ChildRelations	Возвращает коллекцию подчиненных отношений (DataRelationCollection) для объекта DataTable (если такие отношения есть)
Columns	Возвращает набор столбцов для таблицы
Constraints	Позволяет получить коллекцию ограничений, определенных в таблице (ConstraintCollection)
DataSet	Позволяет получить ссылку на объект DataSet, к которому принадлежит данная таблица (если такой объект есть)
DefauftView	Позволяет получить настроенное представление для таблицы, которое может включать в себя, например, только некоторые выбранные пользователем столбцы или данные о положении курсора
MinimumCapacity	Позволяет получить или установить исходное количество строк в таблице (по умолчанию используется значение 25)
ParentRelations	Позволяет получить коллекцию родительских отношений для данного объекта DataTable

Продолжение таблицы 2.7

Свойство	Описание
PrimaryKey	Позволяет получить или установить массив столбцов, которые являются первичным ключом в таблице
Rows	Возвращает набор строк, относящихся к таблице
TableName	Позволяет получить имя таблицы или определить его. Значение для этого свойства может быть установлено через конструктор для таблицы

Графическая схема наиболее важных компонентов DataTable представлена на рис. 2.7.

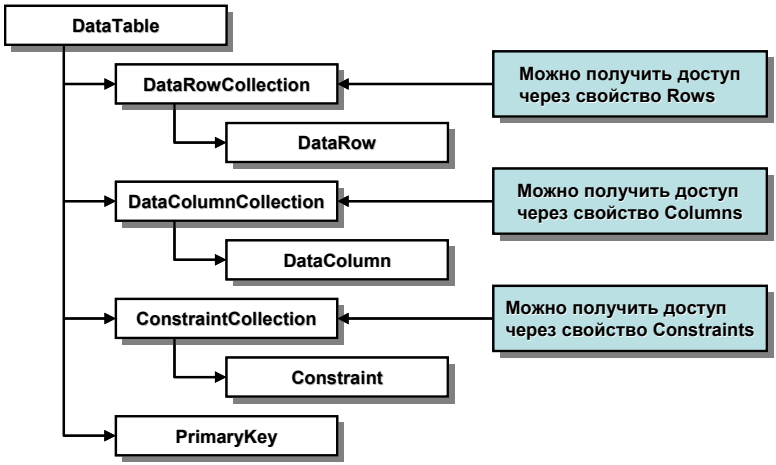


Рис. 2.7. Отношения компонентов DataTable

Следует обратить внимание, что схема не имеет никакого отношения к иерархии классов (к примеру, класс DataRow не является классом, производным от DataRowCollection). Эта схема представляет логические отношения «иметь» («has-a») между наиболее важными компонентами DataTable (например, объекты DataRow принадлежат к объекту DataRowCollection) [1, 5–7].

Создание объекта DataTable. Теперь, когда произведено знакомство с классом DataTable, можно решить следующую задачу – создать полный объект DataTable и продемонстрировать возможности работы с этим объектом. Например, предполагается, что цель – создать объект DataTable, представляющий информационный список в базе данных сотрудников. В таблице «База данных сотрудников» будет шесть столб-

цов: «N» (идентификатор сотрудника), «Фамилия» (фамилия), «Имя» (имя), «Отчество» (отчество), «Стаж» (стаж) и «Телефон» (телефон). Столбец «N» будет столбцом счетчика (т. е. значения в нем будут увеличиваться автоматически) – и, кроме того, он не будет допускать значения типа NULL и будет первичным ключом таблицы. То, как таблица может выглядеть, представлено на рис. 2.8.

N	Фамилия	Имя	Отчество	Стаж	Телефон
1	Фамилия 1	Имя 1	Отчество 1	1	Телефон 1
2	Фамилия 2	Имя 2	Отчество 2	2	Телефон 2
3	Фамилия 3	Имя 3	Отчество 3	3	Телефон 3
4	Фамилия 4	Имя 4	Отчество 4	4	Телефон 4
5	Фамилия 5	Имя 5	Отчество 5	5	Телефон 5

Рис. 2.8. Таблица «База данных сотрудников»

Первое, что нужно сделать, – создать объект DataTable. Очень удобно указать дружественное имя таблицы в качестве параметра конструктора. Это дружественное имя затем можно будет использовать, например, при обращении к нашей таблице из DataSet. Код для создания объекта DataTable в данном случае будет выглядеть так:

```
private DataTable Baza_Dannikh = new DataTable("База данных сотрудников");
```

Следующая задача – программным образом произвести добавление необходимых столбцов в коллекцию DataColumnCollecton при помощи метода Add(). Нужно будет добавить шесть столбца: «N», «Фамилия», «Имя», «Отчество», «Стаж» и «Телефон» (тип данных для каждого столбца определяется при помощи свойства DataType):

```
DataColumn myDataColumn;

myDataColumn = new DataColumn();

myDataColumn.DataType = Type.GetType("System.Int32");
myDataColumn.ColumnName = "N";
myDataColumn.ReadOnly = true;
myDataColumn.AllowDBNull = false;
myDataColumn.Unique = true;
myDataColumn.AutoIncrement = true;
myDataColumn.AutoIncrementSeed = 1;
myDataColumn.AutoIncrementStep = 1;
```

```
Baza_Dannikh.Columns.Add(myDataColumn);

myDataColumn = new DataColumn();

myDataColumn.DataType = Type.GetType("System.String");
myDataColumn.ColumnName = "Фамилия";
Baza_Dannikh.Columns.Add(myDataColumn);

myDataColumn = new DataColumn();

myDataColumn.DataType = Type.GetType("System.String");
myDataColumn.ColumnName = "Имя";
Baza_Dannikh.Columns.Add(myDataColumn);

myDataColumn = new DataColumn();

myDataColumn.DataType = Type.GetType("System.String");
myDataColumn.ColumnName = "Отчество";
Baza_Dannikh.Columns.Add(myDataColumn);

myDataColumn = new DataColumn();

myDataColumn.DataType = Type.GetType("System.Int32");
myDataColumn.ColumnName = "Стаж";
Baza_Dannikh.Columns.Add(myDataColumn);

myDataColumn = new DataColumn();

myDataColumn.DataType = Type.GetType("System.String");
myDataColumn.ColumnName = "Телефон";
Baza_Dannikh.Columns.Add(myDataColumn);
```

Перед тем как приступить к добавлению строк, еще потребуется указать первичный ключ для таблицы. Для этого положено использовать свойство PrimaryKey. Первичный ключ может включать в себя несколько столбцов (композитный первичный ключ), поэтому это свойство в качестве параметра принимает массив объектов DataColumn. В данной таблице роль первичного ключа будет выполнять единственный столбец «N», поэтому соответствующий код может выглядеть так:

```
DataColumn[] PK = new DataColumn[1];

PK[0] = Baza_Dannikh.Columns["N"];
Baza_Dannikh.PrimaryKey = PK;
```

Теперь в созданную таблицу можно помещать данные. Предполагается, что уже есть массив Sotrudniki (типа ArrayList) объектов Sotrudnik. Перенос из него данных о сотрудниках в таблицу может выглядеть так:

```
foreach (Sotrudnik s in Sotrudniki)
{
        DataRow newRow;
        newRow = Baza_Dannikh.NewRow();
        newRow["Фамилия"] = s.familiya;
        newRow["Имя"] = s.imya;
        newRow["Отчество"] = s.otchestvo;
        newRow["Стаж"] = s.stazh;
        newRow["Телефон"] = s.telefon;
        Baza_Dannikh.Rows.Add(newRow);
}
```

Теперь задача – убедиться, что все прошло нормально. Предполагается, что есть приложение на элементах управления Windows Forms, на главной форме которого размещен элемент управления DataGridView (или DataGrid). Для того чтобы привязать DataTable к DataGridView, используется свойство DataSource. По выполнении всех необходимых действий должен получиться интерфейс, представленный на рис. 2.9.

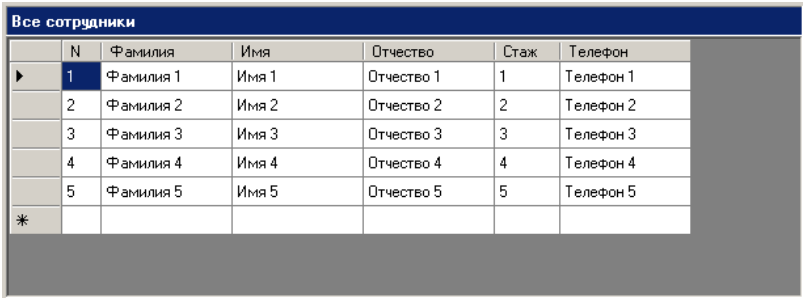


Рис. 2.9. Отображение содержимого DataTable при помощи элемента управления DataGridView

В данном примере добавляются строки с явным указанием имени каждого столбца. Однако то же самое можно сделать и по-другому, указывая вместо имени порядковый номер столбца в таблице. Код для создания строк таблицы при этом может выглядеть так (результаты работы будут совершенно одинаковы) [1, 5–7]:

```
foreach (Sotrudnik s in Sotrudniki)
{
        DataRow newRow;
        newRow = Baza_Dannikh.NewRow();
        newRow[1] = s.familiya;
        newRow[2] = s.imya;
        newRow[3] = s.otchestvo;
        newRow[4] = s.stazh;
        newRow[5] = s.telefon;
        Baza_Dannikh.Rows.Add(newRow);
}
```

Удаление строк из таблицы. Если потребуется удалить из таблицы одну или несколько строк, можно сделать это несколькими способами. Первый способ – воспользоваться методом Delete(). При этом придется указать порядковый номер удаляемой строки.

Предполагается, что в графическом интерфейсе приложения появились следующие компоненты: кнопка «Удаление записи №» (Button) и компонент для ввода текстовых значений (TextBox) (рис. 2.10).

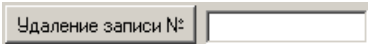


Рис. 2.10. Графические элементы для возможности удаления строк

При нажатии на размещенную на форме кнопку будут удаляться строки с указываемыми номерами. Для выполнения этой операции достаточно поместить в обработчик события Click для кнопки следующий код:

```
protected void button2_Click(object sender, EventArgs e)
{
        try
        {
                Baza_Dannikh.Rows[(int.Parse(textBox1.Text)-1)].Delete();
                Baza_Dannikh.AcceptChanges();
        }
        catch
        {
                MessageBox.Show("Номер не введен или отсутствует",
"Ошибка");
        }
}
```

Метод Delete() можно было бы, наверное, назвать MarkedAsDeletable(), поскольку при его вызове реального удаления строки не происходит. Вместо этого столбец лишь помечается как подлежащий удалению, а само удаление происходит при вызове метода AcceptChanges(). Пока AcceptChanges() не вызван, можно даже восстановить удаленную строку [1, 5–7]:

```
protected void button2_Click(object sender, EventArgs e)
{
        try
        {
                Baza_Dannikh.Rows[(int.Parse(textBox1.Text)-1)].Delete();
                Baza_Dannikh. RejectChanges ();
        }
}
```

Применение фильтров и порядка сортировки. Очень часто возникает необходимость получить подборку данных таблицы, основываясь на каком-либо критерии отбора (фильтре). Например, предполагается, что из таблицы потребовалось извлечь только строки, относящиеся к номерам телефонов, соответствующим определенной фамилии (фильтр по значению столбца «Фамилия»). В этой ситуации можно использовать метод Select(). Еще раз следует обновить графический интерфейс приложения, чтобы получить возможность выводить данные только о фамилиях и соответствующих номерах телефонов сотрудников (рис. 2.11).

Рис. 2.11. Графические элементы для возможности использования фильтра для отбора строк

Для метода Select() предусмотрено множество перегруженных вариантов. Чаще всего этому методу передается параметр, который представляет условие для отбора строк. Например, в данном случае для обработчика события Click кнопки отбора записей можно использовать следующий код:

```
private void button3_Click(object sender, EventArgs e)
{
        string filterStr = "Фамилия = '" + textBox2.Text + "'";
        string strFamiliya = null;
        DataRow[] familiya = Baza_Dannikh.Select(filterStr);
        if (familiya.Length == 0)
```

```
        {
            MessageBox.Show("Фамилия отсутствует", "Ошибка");
        }
        else
        {

            for (int i = 0; i < familiya.Length; i++)
            {
                DataRow temp = familiya[i];
                strFamiliya += temp["Телефон"].ToString() + "\n";
            }

            MessageBox.Show(strFamiliya, textBox2.Text );
        }
}
```

Вначале создана текстовая строка, содержащая критерии отбора. Если бы пользователь ввел в текстовом поле «Фамилия 1», то условие поиска выглядело бы как «Фамилия = 'Фамилия 1'». Затем это условие поиска передано методу Select() и получен в ответ массив объектов DataRow. При этом каждому из этих объектов соответствовала строка, удовлетворяющая условию поиска (рис. 2.12).

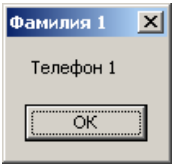


Рис. 2.12. Отобранные при помощи фильтра данные

Условие поиска может содержать в себе любое количество операторов. Например, если бы потребовалось найти все строки со значением «N» большим, чем три, то можно было бы использовать следующий код:

```
DataRow[] nomer;
string newFilterStr = "N > '3'";
string strNomer = null;
nomer = Baza_Dannikh.Select(newFilterStr);

for (int i = 0; i < nomer.Length; i++)
{
```

```
        DataRow temp = nomer[i];
        strNomer += temp["Фамилия"].ToString() + " имеет номер " +
temp["N"] + "\n";
}

MessageBox.Show(strNomer, "Сотрудники с номерами более 3");
```

То, что должно получиться в результате выполнения программы, представлено на рис. 2.13.

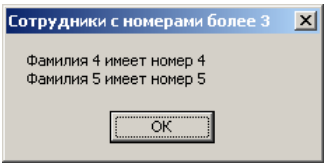


Рис. 2.13. Отбор диапазона данных

Условия поиска формулируются при помощи стандартного синтаксиса SQL. Например, предполагается, что нужно упорядочить возвращаемые методом Select() данные по номерам телефонов в порядке возрастания (или в алфавитном порядке для букв). В терминах SQL это значит, что необходимо провести сортировку по столбцу «Телефон». Ничего особо сложного делать не придется, поскольку для метода Select() предусмотрен перегруженный вариант и на такой случай:

```
DataRow[] familiya = Baza_Dannikh.Select(filterStr, "Телефон");
```

Результаты запроса будут возвращены в упорядоченном виде.

Если желательно, чтобы значения были упорядочены по убыванию, запрос можно переписать следующим образом:

```
DataRow[] familiya = Baza_Dannikh.Select(filterStr, "Телефон DESC");
```

Формально говоря, условия сортировки задаются путем указания имен столбцов, после которых может следовать ключевое слово ASC (от ascending – сортировка по возрастанию; она используется по умолчанию) или DESC (от descending – сортировка по убыванию). Если есть необходимость провести сортировку сразу по нескольким столбцам, названия этих столбцов можно перечислить через запятую [1, 5–7].

Внесение изменений в строки. Последний очень важный вопрос, который необходимо прояснить, – внесение изменений в уже существующие

строки в объекте DataTable. Наиболее простой способ – вначале воспользоваться тем же методом Select(), чтобы отобрать интересующие строки, а затем внести изменения в те объекты DataRow, которые вернет этот метод. Например, предполагается, что в приложении появилась еще одна кнопка, при нажатии на которую происходит увеличение значений стажа на единицу. Код обработчика события Click для этой кнопки может выглядеть следующим образом:

```
private void button4_Click(object sender, EventArgs e)
{
        string filterStr = "Стаж >= '0'";
        DataRow[] stazh = Baza_Dannikh.Select(filterStr);

        for (int i = 0; i < stazh.Length; i++)
        {
                DataRow temp = stazh[i];
                temp["Стаж"] = ((int)(temp["Стаж"])) + 1;
                stazh[i] = temp;
        }

}
```

Кроме того, в классе DataRow также предусмотрены методы BeginEdit(), EndEdit() и CancelEdit(), которые позволяют редактировать содержимое строки, отключив на время все правила проверки целостности данных, связанные с соответствующей строкой. Как только вызывается метод BeginEdit() для объекта DataRow, он переводится в режим редактирования. В этом режиме вносятся необходимые изменения, а затем вызывается либо EndEdit() для сохранения внесенных изменений, либо CancelEdit() для их отмены. Например:

```
rowToUpdate.BeginEdit();
if(ChangeValuesForThisRow(rowToUpdate))
{
        rowToUpdate.EndEdit();
}
else
{
        rowToUpdate.CancelEdit();
}
```

Методы BeginEdit(), EndEdit() и CancelEdit() можно вызывать вручную, но чаще они вызываются автоматически в процессе редактирования строк через элемент управления DataGridView (или DataGrid), связанный с DataTable. Например, если выбрана строка для внесения изменений в DataGridView, то эта строка автоматически переводится в режим редактирования. При перемещении фокуса на новую строку автоматически вызывается метод EndEdit(). Проверить это можно очень просто, увеличив значение стажа на единицу для всех строк через элемент управления DataGridView приложения, а затем точно так же, как в примере с номером, запросив все строки со значением стажа более определенного значения. Должны вернуться все строки, в которые внесены через DataGridView соответствующие изменения [1, 5–7].

Тип DataView. В технологии баз данных представление (view) – это специальным образом настроенное отображение данных из таблицы (или нескольких таблиц). Например, можно создать представление на основе таблицы «База данных сотрудников», которое будет отображать только информацию о сотрудниках, имеющих стаж работы более пяти лет. В ADO.NET представления (объекты DataView) позволяют программным образом получать выборку данных на основе базовой таблицы (объекта DataTable).

Для одной и той же таблицы можно создать неограниченное число представлений. Создание нескольких представлений для одной таблицы может потребоваться, например, если планируется использовать эти представления в разных элементах управления (например, если на форме размещено несколько элементов управления DataGridView или DataGrid). Например, первый DataGridView может быть привязан к объекту DataView, который позволяет получить строки для всех сотрудников в таблице, второй DataGridView работает через DataView, выводящий информацию только о сотрудниках с определенным номером телефона, и т. п. В классе DataTable предусмотрено свойство DefaultView, при помощи которого можно получить представление по умолчанию для определенной таблицы.

Например, на главной форме приложения теперь будет размещен не один элемент управления DataGridView, а два. Первый, как и раньше, будет выводить информацию о всех сотрудниках в DataTable, второй – только о сотрудниках, имеющих стаж работы более пяти лет (значения в столбце «Стаж» более пяти). Выглядеть все это может так, как показано на рис. 2.14.

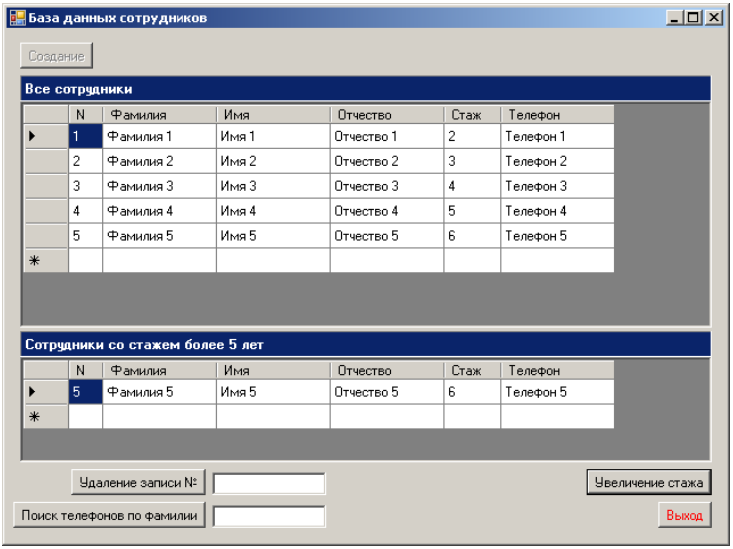


Рис. 2.14. Несколько элементов DataGridView,
каждый из которых основан на своем DataView

Чтобы все это реализовать на практике, первое, что необходимо сделать, – объявить необходимые переменные класса DataView:

```
DataView Stazh;
```

Предполагается, что в распоряжении есть реализованная вспомогательная функция Vid(), которая вызывается непосредственно после создания объекта DataTable и служит для создания представления:

```
private void Vid()
{
        Stazh = new DataView(Baza_Dannikh);
        Stazh.RowFilter = "Стаж > '5'";
        dataGridView2.DataSource = Stazh;
        dataGridView2.Columns["N"].Width = 30;
        dataGridView2.Columns["Стаж"].Width = 50;
}
```

Следует обратить внимание, что конструктору DataView передается в качестве параметра объект DataTable, на основе которого создается представление. В классе DataView предусмотрено свойство RowFilter, для кото-

рого используется текстовая строка с критериями отбора объектов DataRow. После того как объекты DataView полностью сконфигурированы, они просто указываются в качестве источника данных для элементов управления DataGridView (или DataGrid) через свойство DataSource. Эти элементы управления умеют отслеживать изменения, которые вносятся в источник данных для них. Поэтому если заменить в DataTable содержимое какой-либо ячейки, то значения во всех DataGridView автоматически изменятся, отобразив эти переменные.

Некоторые наиболее важные члены класса DataView представлены в табл. 2.8 [1, 5–7].

Таблица 2.8

Члены класса DataView

Член	Назначение
AddNew()	Добавляет новую строку через DataView
AllowDelete AllowEdit AtlowNew	Позволяют настроить возможность проведения через DataView операций по удалению, редактированию и добавлению новых строк в таблицу
Delete()	Позволяет удалить существующую строку через DataView (необходимо указать ее порядковый номер)
RowFilter	Позволяет получить или установить выражение, используемое для фильтрации строк, отображаемых через DataView
Sort	Позволяет настроить порядок сортировки строк в DataView
Table	Позволяет получить или указать базовую таблицу для DataTable

Возможности и члены класса DataSet. К этому моменту должно сформироваться представление о возможности создания таблиц с данными (объектов DataTable) в оперативной памяти и выполнения с ними всех необходимых операций. В принципе, это может быть вполне достаточно для многих приложений. Однако гораздо чаще таблицы в базах данных используются не сами по себе, а во взаимодействии с другими таблицами. В ADO.NET возможности работы с наборами таблиц, взаимосвязанных друг с другом, предоставляет класс DataSet. В действительности при использовании объектов для доступа к источникам данных, поставляемых с ADO.NET, чаще всего не удастся получить объект DataTable с данными напрямую – все интерфейсы для доступа к данным в ADO.NET возвращают именно объект DataSet (как набор взаимосвязанных объектов DataTable).

Объект DataSet – это создаваемый в оперативной памяти набор таблиц (объектов DataTable), связанных между собой отношениями и снабженными средствами проверки целостности данных (для них в DataSet предусмотрены свои объекты). Разобраться в иерархии классов, входящих в DataSet, поможет схема на рис. 2.15.

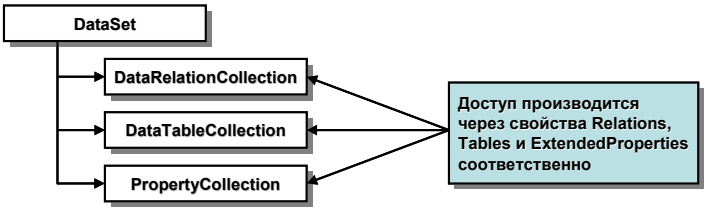


Рис. 2.15. Внутренние коллекции DataSet

Через свойство Tables объекта DataSet можно получить доступ к отдельным объектам DataTable, которые хранятся в коллекции DataTableCollection. Через другое свойство – Relations – можно получить доступ к объектам DataRelation, которые хранятся в коллекции DataRelationCollection. Поскольку DataSet – это фактически представление базы данных, которое помещается в оперативную память клиента (и с которой можно работать при разорванном соединении), то в этой модели объекты DataRelation представляют отношения между таблицами базы данных.

Например, предполагается, что в одной таблице существует внешний ключ, ссылающийся на первичный ключ в другой таблице. Это отношение представлено объектом DataRelation (и его можно добавить в DataSet при помощи свойства Relations). После этого можно использовать это отношение при выполнении запросов к таблицам.

Свойство ExtendedProperties обеспечивает доступ к объектам, хранящимся в коллекции PropertyCollection. Основное назначение всей этой конструкции со свойством ExtendedProperties и коллекцией PropertyCollection – обеспечить возможность ассоциировать дополнительную информацию (метаданные) в виде пар «имя-значение» с объектом DataSet. В принципе этой дополнительной информацией может быть все, что угодно. Например, можно самостоятельно определить, что у объекта DataSet будет свойство University, а в качестве значения этого свойства указать имя университета. Выглядеть это будет так:

```
DataSet ds = new DataSet ("MyDataSet");
ds.ExtendedProperties.Add("University","ISEU");
Console.WriteLine(ds.ExtendedProperties["University"].ToString());
```

Можно использовать расширенные свойства для хранения самой разной информации – например, о внутреннем пароле для доступа к данным, об интервале синхронизации данных и т. п. Кроме того, расширенные свойства (т. е. свойство ExtendedProperties) предусмотрены не только для DataSet, но и DataTable.

Перед тем как начать вникать в особенности применения DataSet в приложении, следует рассмотреть его наиболее важные свойства. Эти свойства обеспечивают доступ к внутренним коллекциям DataSet, позволяют представлять данные из DataSet в формате XML и обеспечивают возможность получения информации об ошибках. Некоторые наиболее важные свойства DataSet представлены в табл. 2.9.

Таблица 2.9

Свойства DataSet

Свойство	**Описание**
CaseSensitive	Определяет, будет ли во время операций по сравнению текстовых строк в объектах DataTable учитываться регистр букв
DataSetName	Позволяет получить или задать имя для данного объекта DataSet. Обычно значение этого свойства задается как параметр, передаваемый конструктору
DefaultViewManager	Позволяет определить представление по умолчанию для отображения данных в DataSet
EnforceConstraints	Позволяет отключить (или включить снова) проверку соответствия ограничениям при выполнении операций обновления данных в DataSet
HasErrors	Позволяет получить значение, определяющее наличие ошибок в DataSet (т. е. ошибок в любой строке любой таблицы DataSet)
Relations	Позволяет обратиться к коллекции отношений между таблицами DataSet
Tables	Позволяет получить доступ к коллекции таблиц DataSet

Многие методы DataSet дублируют возможности, которые обеспечиваются свойствами. Помимо взаимодействия с потоками данных в формате XML, методы DataSet позволяют копировать содержимое DataSet, устанавливать начало и конец пакетных изменений данных в DataSet и т. п. Самые важные методы DataSet представлены в табл. 2.10 [1, 5–7].

Таблица 2.10

Методы DataSet

Метод	Описание
AcceptChanges()	Позволяет сохранить в DataSet все изменения, произведенные с момента последнего вызова этого метода
Clear()	Полная очистка DataSet – удаляются все строки из всех таблиц
Clone()	Клонирует структуру DataSet, включая структуру таблиц, отношения между таблицами и ограничения
Copy()	Копирует DataSet (структуру вместе с данными)
GetChanges()	Возвращает копию DataSet, которая содержит все изменения, внесенные в оригинальный DataSet с момента последнего вызова для него метода AcceptChanges()
GetChildRelations()	Возвращает коллекцию подчиненных отношений для указанной таблицы
GetParentRelations()	Возвращает коллекцию родительских отношений для указанной таблицы
HasChanges()	Этот перегруженный метод позволяет получить информацию об изменениях, внесенных в DataSet (отдельно по вставленным, удаленным и измененным строкам)
Merge()	Этот перегруженный метод позволяет производить слияние разных объектов DataSet
ReadXml() ReadXmlSchema()	Позволяют считывать данные в формате XML в DataSet из потока (файла, оперативной памяти и сетевого ресурса)
RejectChanges()	Отменяет все изменения, внесенные в DataSet с момента его создания или последнего вызова AcceptChanges()
WriteXml() WriteXmlSchema()	Позволяют записать данные в формате XML из DataSet в поток

Создание объекта DataSet. Чтобы проиллюстрировать применение DataSet на практике, можно создать специальное приложение на Windows Forms, содержащее реляционную базу данных «База данных вакансий». В этом приложении будет использоваться объект DataSet с тремя внутренними таблицами (объектами DataTable) – «Организации», «Специальности» и «Вакансии». В каждой таблице будет свой первичный ключ, при этом система первичных и внешних ключей таблиц позволит использовать объекты DataRelation для моделирования отношений между таблицами.

Общая структура базы данных, которую необходимо реализовать при помощи объекта DataSet, представлена на рис. 2.16.

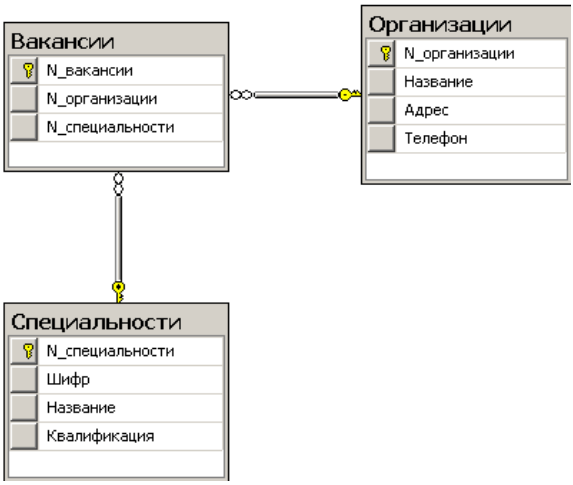


Рис. 2.16. Структура базы данных «База данных вакансий»

Таблица «Организации» является родительской таблицей для таблицы «Вакансии», и для первичного ключа (столбца «N_организации») в таблице «Организации» предусмотрен внешний ключ в таблице «Вакансии». Таблица «Специальности» также является родительской для таблицы «Вакансии», и эти таблицы связаны отношением «первичный / внешний ключ» по столбцу «N_специальности». Как станет очевидным позднее, добавление в DataSet объектов DataRelation, моделирующих эти отношения, позволит получать взаимосвязанные данные для таблиц.

Создание объекта DataSet начинается с объявления переменных, представляющих таблицы и сам объект DataSet. Выглядеть это будет так:

```
private DataTable Organizatsii = new DataTable("Организации");

private DataTable Spetsialnosti = new DataTable("Специальности");

private DataTable Vakansii = new DataTable("Вакансии");

private DataSet Baza_Dannikh = new DataSet("База данных вакансий");
```

Теперь для наглядности целесообразно создать два очень простых класса – Organizatsiya и Spetsialnost, для хранения информации о которых,

в сущности, и будет использован объект DataSet. Следует обратить внимание, что в классе Spetsialnost предусмотрена специальная переменная для хранения информации об организации, которая предлагает вакансию:

```
public class Organizatsiya
{

        public string nazvaniye, adres, telefon;

        public Organizatsiya(string nazvaniye, string adres, string telefon)
        {
                this.nazvaniye = nazvaniye;
                this.adres = adres;
                this.telefon = telefon;
        }

}

public class Spetsialnost
{

        public int nomer;

        public string shifr, nazvaniye, kvalifikatsiya;

        public Spetsialnost(string shifr, string nazvaniye, string kvalifikatsiya, int
nomer)
        {
                this.shifr = shifr;
                this.nazvaniye = nazvaniye;
                this.kvalifikatsiya = kvalifikatsiya;
                this.nomer = nomer;
        }

}
```

На главной форме также будут использованы два массива (объекты ArrayList) для хранения информации об организациях и специальностях. Считается, что эти два массива будут заполняться данными в конструкторе для формы. Кроме того, в конструкторе будет производиться вызов нескольких вспомогательных функций для создания таблиц и отношений между ними. И еще одна операция будет производиться в конструкторе: привязка объектов DataTable для таблиц «Организации» и «Специальности» к соответствующим элементам управления DataGrid. Подобная привязка

через объект DataSet производится при помощи метода SetDataBinding() с параметрами:

```
private ArrayList AL_Organizatsii, AL_Spetsialnosti;

public Form1()
{
        InitializeComponent();

        AL_Organizatsii = new ArrayList();
        AL_Organizatsii.Add(new Organizatsiya("Название 1", "Адрес 1",
"Телефон 1"));
        AL_Organizatsii.Add(new Organizatsiya("Название 2", "Адрес 2",
"Телефон 2"));
        AL_Organizatsii.Add(new Organizatsiya("Название 3", "Адрес 3",
"Телефон 3"));
        AL_Organizatsii.Add(new Organizatsiya("Название 4", "Адрес 4",
"Телефон 4"));
        AL_Organizatsii.Add(new Organizatsiya("Название 5", "Адрес 5",
"Телефон 5"));

        AL_Spetsialnosti = new ArrayList();
        AL_Spetsialnosti.Add(new Spetsialnost("Шифр 1", "Название 1",
                "Квалификация 1", 1));
        AL_Spetsialnosti.Add(new Spetsialnost("Шифр 2", "Название 2",
                "Квалификация 2", 2));
        AL_Spetsialnosti.Add(new Spetsialnost("Шифр 3", "Название 3",
                "Квалификация 3", 3));

        T_Organizatsii();
        T_Spetsialnosti();
        T_Vakansii();

        Otnoshenie();

        dataGrid1.SetDataBinding(Baza_Dannikh, "Организации");
        dataGrid2.SetDataBinding(Baza_Dannikh, "Специальности");
}
```

Все объекты DataTable будут создаваться точно так же, как это производилось ранее. Чтобы проще было сосредоточиться на логике DataSet, основная часть кода для создания таблиц опускается. Однако следует отметить, что в каждой таблице предусмотрен столбец счетчика, который

является первичным ключом таблицы. Наиболее важная часть кода по созданию объектов DataTable для таблицы «Вакансии» представлена ниже:

```
Baza_Dannikh.Tables.Add(Vakansii);

for (int i = 1; i < AL_Spetsialnosti.Count; i++)
{
        DataRow newRow;
        newRow = Vakansii.NewRow();
        Spetsialnost s = (Spetsialnost)AL_Spetsialnosti[i];
        newRow["N_специальности"] = i;
        newRow["N_организации"] = s.nomer-1;
        Baza_Dannikh.Tables["Вакансии"].Rows.Add(newRow);
}
```

Вспомогательные функции T_Organizatsii() и T_Spetsialnosti() для формирования остальных таблиц в данном случае будут создаваться точно по таким же принципам [1, 5–7].

Моделирование отношения между таблицами при помощи класса DataRelation. Самая интересная из всего набора вспомогательных функций в конструкторе формы – это, безусловно, функция Otnoshenie(), которая ответственна за создание отношений между таблицами DataSet. После того как в DataSet появилось несколько объектов таблиц, можно программным образом определить отношения между этими таблицами. Конечно, создание таких отношений не является обязательным, но во многих случаях это предоставляет важные дополнительные возможности, характерные для реляционных баз данных. Например, станет возможным переходить между строками разных таблиц «на лету» и осуществлять запросы сразу к нескольким таблицам.

Объектно-ориентированную оболочку вокруг отношений между таблицами представляет класс System.Data.DataRelation. При создании объекта этого класса необходимо будет указать дружественное имя этого объекта, а также родительскую таблицу (например, «Организации») и подчиненную таблицу («Вакансии»). Чтобы отношение было успешно установлено, в каждой из таблиц должен быть столбец с одинаковым названием («N_организации») и типом данных (в этом случае Int32). Основное определение функции Otnoshenie() без реализации проверки ошибок приведено ниже:

```
private void Otnoshenie()
{

        DataRelation dr = new DataRelation("Организации-вакансии",
```

```
            Baza_Dannikh.Tables["Организации"].Columns["N_организации"],
                    Baza_Dannikh.Tables["Вакансии"].Columns["N_организации"]);

            Baza_Dannikh.Relations.Add(dr);

            dr = new DataRelation("Специальности-вакансии",

            Baza_Dannikh.Tables["Специальности"].Columns["N_специальности"],

            Baza_Dannikh.Tables["Вакансии"].Columns["N_специальности"]);

            Baza_Dannikh.Relations.Add(dr);
}
```

Объекты DataRelation хранятся в коллекции DataRelationCollection, поддерживаемой DataSet. В типе DataRelation предусмотрены свойства, которые позволяют получать ссылки на родительскую и подчиненную таблицу, участвующую в отношении, определять имя отношения т. п. Наиболее часто используемые свойства представлены в табл. 2.11 [1, 5–7].

Таблица 2.11

Свойства типа DataRelation

Свойство	Описание
ChildColumns ChildKeyConstraint ChildTable	Позволяют получить информацию о подчиненной таблице, участвующей в отношении, а также ссылку на саму эту таблицу
DataSet	Позволяет получить ссылку на объект DataSet, которому принадлежит данное отношение
ParentColumns ParentKeyConstraint ParentTable	Позволяют получить информацию о родительской таблице, участвующей в отношении, а также ссылку на саму эту таблицу
RelationName	Позволяет получить или задать имя для данного отношения

Переход между таблицами, участвующими в отношении. Объект DataRelation позволяет осуществлять программным образом переход между двумя таблицами, участвующими в отношении. Пусть на главной форме приложения появятся еще два элемента управления: кнопка и текстовое поле. В текстовое поле пользователь сможет вводить номер специальности, а при нажатии на кнопку будет выводиться информация о вакансии, предлагаемой соответствующей организацией. Вывод ин-

формации для простоты будет производиться через обычный MessageBox так, как показано на рис. 2.17.

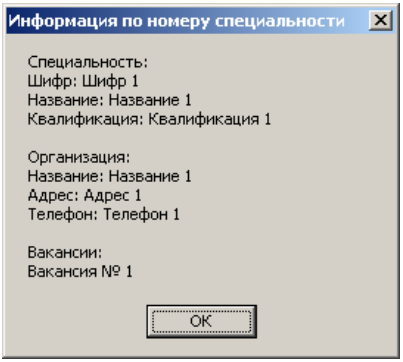


Рис. 2.17. Переход между таблицами с использованием объектов DataRelation

Код обработчика события Click для кнопки может быть таким:

```
private void button1_Click(object sender, EventArgs e)
{
        string strInfo = "";

        DataRow dr_Spetsialnost = null;

        DataRow[] drs_Vakansiya = null;

        int Nomer = int.Parse(this.textBox1.Text)-1;
        dr_Spetsialnost = Baza_Dannikh.Tables["Специальности"].Rows[Nomer];
        strInfo += "Специальность: " + "\n";
        strInfo += "Шифр: " + dr_Spetsialnost["Шифр"].ToString() + "\n";
        strInfo += "Название: " + dr_Spetsialnost["Название"].ToString() + "\n";
        strInfo += "Квалификация: " + dr_Spetsialnost["Квалификация"].ToString()
+
                "\n\n";
        drs_Vakansiya = dr_Spetsialnost.GetChildRows(Baza_Dannikh.Relations
                ["Специальности-вакансии"]);
        if (drs_Vakansiya != null)
        {
                DataRow[] drs_Organizatsiya =
                        drs_Vakansiya[0].GetParentRows(Baza_Dannikh.
                        Relations["Организации-вакансии"]);
```

```
                if (drs_Organizatsiya.Length > 0)
                {

                        foreach (DataRow r in drs_Organizatsiya)
                        {
                                strInfo += "Организация:" + "\n";
                                strInfo += "Название: " + r["Название"] + "\n";
                                strInfo += "Адрес: " + r["Адрес"] + "\n";
                                strInfo += "Телефон: " + r["Телефон"] + "\n\n";
                        }

                }

        }

        if (drs_Vakansiya != null)
        {
                strInfo += "Вакансии: " + "\n";

                foreach (DataRow r in drs_Vakansiya)
                {
                        strInfo += "Вакансия № " + r["N_вакансии"] + "\n";
                }

                MessageBox.Show(strInfo, "Информация по номеру
специальности");
        }

}
```

В целом алгоритм вышеприведенной функции заключается в следующем:

- получается «N_специальности» из текстового поля;
- основываясь на этом значении, получается вся строка из таблицы «Специальности»;
- получается информация о специальности;
- производится переход от таблицы «Специальности» к таблице «Вакансии»;
- производится переход от таблицы «Вакансии» к таблице «Организации»;
- получается информация об организации;
- получается «N_вакансии».

Как можно убедиться, перемещение между таблицами производится при помощи методов, определенных в типе DataRow. Целесообразно проанализировать приведенный код шаг за шагом. Первое, что сделано, – получено из текстового поля значение, означающее номер специальности, в информации о которой заинтересован пользователь. Далее, используя полученное значение «N_специальности», извлекается из таблицы «Специальности» (при помощи свойства Rows) вся строка, в которой присутствует указанное значение «N_специальности», целиком и, соответственно, информация, хранящаяся в ней. Следующее действие – переход от таблицы «Вакансии» к таблице «Организации», для чего используется отношение (объект DataRelation) «Специальности-вакансии». Следует обратить внимание, что метод GetChildRows() позволяет считывать строки из подчиненной таблицы. Получив эти строки, можно извлечь из них нужную информацию. Аналогично осуществляется переход от таблицы «Вакансии» к таблице «Организации» с использованием метода GetParentRows(). Далее получаются информация об организации и номер вакансии.

Код еще одного примера, посвященного переходам между таблицами программным образом, приведенный ниже, позволяет выводить значения из таблицы «Вакансии» при помощи отношения «Организации-вакансии»:

```
private void button2_Click(object sender, EventArgs e)
{
        DataRelationCollection relCol;

        DataRow[] arrRows;

        string info = "";
        relCol = Baza_Dannikh.Tables["Организации"].ChildRelations;
        info += "Связь: " + relCol[0].RelationName + "\n\n";
        info += "№ вакансии" + "\t";
        info += "№ организации" + "\t";
        info += "№ специальности" + "\n";

        foreach (DataRelation drel in relCol)
        {

                foreach (DataRow dr in Organizatsii.Rows)
                {
                        arrRows = dr.GetChildRows(drel);
                        for (int i = 0; i < arrRows.Length; i++)
                        {
```

```
                    foreach (DataColumn dc in
arrRows[i].Table.Columns)
                    {
                        info += arrRows[i][dc] + "\t\t";
                    }

                    info += "\n";
                }

            }

            MessageBox.Show(info, "Данные в таблице 'Вакансии'");
        }

}
```

Результат работы программы представлен на рис. 2.18.

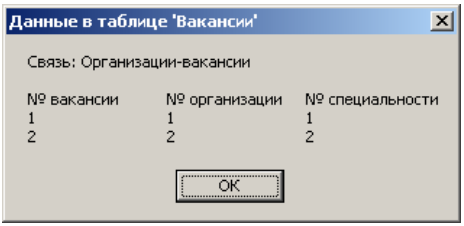


Рис. 2.18. Перемещение между таблицами при помощи отношений

Таким образом, DataSet предоставляет возможность работы с представлением базы данных в оперативной памяти точно так же, как и с обычной базой данных; можно производить запросы к таблицам, выполнять вставку, изменение и удаление данных, перемещаться между таблицами при помощи отношений между ними т. п. Однако до сих пор не рассмотренным и более важным является то, как получается объект DataSet из реальной базы данных, хранящейся где-нибудь на сервере баз данных. Перед тем как перейти к рассмотрению этого вопроса, следует познакомиться с еще одним исключительно важным моментом: представлением данных из DataSet в формате XML [1, 5–7].

Чтение и запись объектов DataSet в формате XML. Одна из главных целей, которая преследовалась при разработке ADO.NET, – это упрощение работы с данными в формате XML. При помощи типа DataSet можно представлять таблицы (со всеми данными), отношения между ними, ограничения и все остальное в формате XML и записывать их в поток (на-

пример, текстовый файл). Если уже создан объект DataSet, то для записи его содержимого в формате XML достаточно просто вызывать метод WriteXml():

```
private void button3_Click(object sender, EventArgs e)
{
        Baza_Dannikh.WriteXml("База данных вакансий.xml");
        MessageBox.Show("Запись XML-файла произведена");
}
```

Открыв этот файл в Microsoft Visual Studio (рис. 2.19), можно убедиться, что весь объект DataSet (т. е. вся база данных) записан в формате XML.

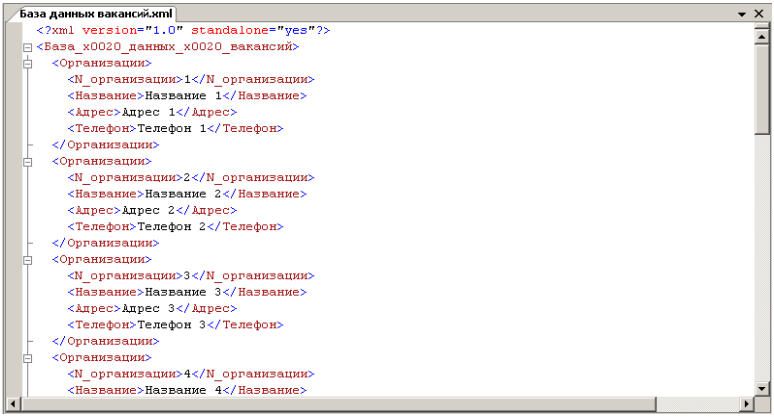


Рис. 2.19. Представление объекта DataSet в формате XML

Считывание текстовых данных из текстового файла в формате XML в объект DataSet производится при помощи метода ReadXml(). Чтобы убедиться в том, что это действительно так, можно создать в приложении специальную кнопку, при нажатии на которую DataSet вначале будет полностью очищаться, а затем восстанавливаться на основе информации из только что созданного файла в формате XML. Код для этой кнопки может выглядеть так:

```
private void button4_Click(object sender, EventArgs e)
{
        Baza_Dannikh.Clear();
        Baza_Dannikh.Dispose();
        MessageBox.Show("Очистка базы данных произведена");
```

```
            Baza_Dannikh = new DataSet("База данных вакансий");
            MessageBox.Show("Создание базы данных произведено");
            Baza_Dannikh.ReadXml("База данных вакансий.xml");
            MessageBox.Show("Чтение XML-файла произведено");
            button4.Enabled = false;
            dataGrid1.SetDataBinding(Baza_Dannikh, "Организации");
            dataGrid2.SetDataBinding(Baza_Dannikh, "Специальности");
}
```

Если поглубже исследовать то, что делают WriteXml() и ReadXml(), то выяснится, что они работают с типами, определенными в сборке System.Xml.dll (конкретно с классами XmlWriter и XmlReader). Поэтому в ранних версиях Microsoft Visual Studio потребуется явное указание, во-первых, ссылки на эту сборку, а во-вторых, на использование соответствующего пространства имен.

Окончательный вид описанного приложения представлен на рис. 2.20.

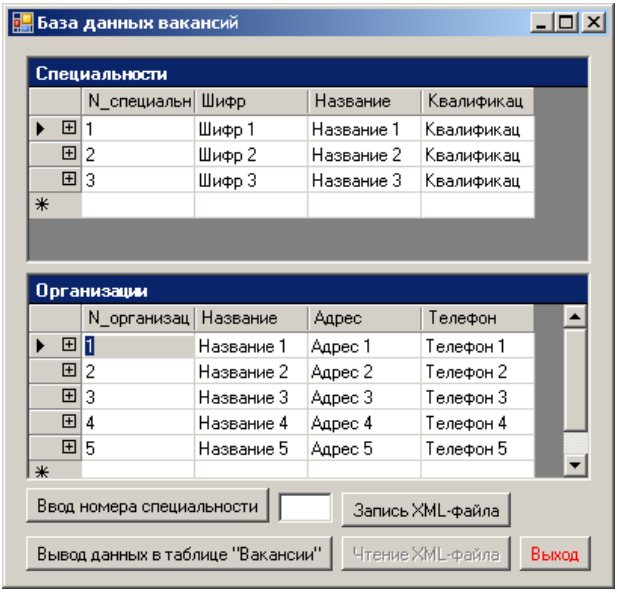


Рис. 2.20. Приложение «База данных вакансий» (окончательный вид)

Таким образом, главный тип в ADO.NET – это класс DataSet. DataSet предназначен для представления в оперативной памяти любого количества таблиц, отношений между ними, ограничений и выражений. То, что при помощи DataSet со стороны клиента представлены не только таблицы, но

и отношения между ними, позволяет производить переходы между таблицами без необходимости всякий раз устанавливать соединение с удаленным источником данных [1, 5–7].

2.3. Управляемые провайдеры ADO.NET. Управляемый провайдер OLE DB. Установление соединения при помощи типа OleDbConnection. Построение команды SQL. Работа с OleDbDataReader. Подключение к базе данных Microsoft Access. Выполнение хранимых процедур. Тип OleDbDataAdapter. Заполнение данными объекта DataSet при помощи OleDbDataAdapter

Управляемые провайдеры ADO.NET. Можно сказать, что управляемый провайдер (managed provider) в ADO.NET – это аналог провайдера OLE DB в классическом ADO. Другими словами, управляемый провайдер – это шлюз к хранилищу данных (например, на сервере баз данных), при помощи которого можно произвести загрузку данных из этого внешнего хранилища в объект DataSet.

Как уже упоминалось, вместе с ADO.NET поставляются два управляемых провайдера.

Первый из них – это провайдер OLE DB, который реализуется при помощи типов, определенных в пространстве имен System.Data.OleDb. Провайдер OLE DB позволяет обращаться к данным, расположенным в любом хранилище, к которому можно подключиться по протоколу OLE DB. Таким образом, аналогично классическому ADO можно использовать в ADO.NET управляемый провайдер OLE DB для доступа, например, к базам данных Microsoft SQL Server, Microsoft Access и Oracle. Однако поскольку при этом типы из пространства имен System.Data.OleDb должны взаимодействовать с обычным (не .NET) кодом драйверов OLE DB, то при таком обращении будет производиться множество преобразований вызовов .NET в вызовы COM, что в некоторых ситуациях может привести к падению производительности.

Другой управляемый провайдер – провайдер SQL – предлагает уже прямой доступ к хранилищам данных, при котором производительность будет максимальной. Однако с его помощью можно обращаться только к базам данных на Microsoft SQL Server, начиная с версии 7.0. Типы, используемые провайдером SQL, содержатся в пространстве имен System.Data.SqlClient. Функциональные возможности обоих управляемых провайдеров практически одинаковы (даже названия типов в System.Data.OleDb и System.Data.SqlClient во многом совпадают).

Главное различие между провайдерами заключается в том, что провайдер SQL не использует классические протоколы OLE DB или ADO и за счет этого обеспечивает значительный выигрыш в производительности.

В пространстве имен System.Data.Common определено множество абстрактных типов, которые обеспечивают общий интерфейс для всех управляемых провайдеров. Оба управляемых провайдера реализуют интерфейс IDbConnection, который используется для конфигурирования и открытия сеанса подключения к источнику данных. Типы, которые реализуют другой интерфейс – IDbCommand, – используются для выполнения SQL-запросов к базам данных. IDataReader обеспечивает считывание данных при помощи однонаправленного курсора только для чтения. Типы, которые реализуют IDbDataAdapter, ответственны за заполнение объекта DataSet данными из базы данных.

В большинстве случаев не потребуется взаимодействовать с типами из пространства имен System.Data.Common напрямую. Однако для применения любого из управляемых провайдеров придется указать использование соответствующих пространств имен [1, 5–7]:

```
using System.Data;
using System.Data.OleDb;
```

или

```
using System.Data;
using System.Data.SqlClient;
```

Управляемый провайдер OLE DB. Работа со всеми управляемыми провайдерами очень похожа, и если разобраться с одним управляемым провайдером, работа с другими не представит больших трудностей. Можно начать рассмотрение возможностей подключения к другим базам данных в ADO.NET с управляемого провайдера OLE DB. Этот провайдер потребуется во всех случаях, когда необходимо подключаться к источнику данных, отличному от Microsoft SQL Server. Главные типы, используемые этим провайдером (они определены в пространстве имен System.Data.OleDb), представлены в табл. 2.12 [1, 5–7].

Таблица 2.12

Наиболее важные типы пространства имен System.Data.OleDb

Тип	Описание
OleDbCommand	Представляет запрос SQL, производимый к источнику данных
OleDbConnection	Представляет открытое соединение с источником данных
OleDbDataAdapter	Представляет соединение с базой данных и набор команд, используемых для заполнения объекта DataSet, а также обновления исходной базы данных после внесения изменений в DataSet
OleDbDataReader	Обеспечивает метод считывания потока данных из источника в одном направлении (вперед)
OleDbErrorCollection OleDbError OleDbException	OleDbErrorCollection представляет набор ошибок и предупреждений, возвращаемых источником данных. Сами эти ошибки и предупреждения представлены объектами OleDbError. При возникновении ошибки может быть сгенерировано исключение, представленное объектом OleDbException
OleDbParameterCollection OleDbParameter	Используются для передачи параметров процедуре, хранимой на источнике данных. Параметры представлены объектами OleDbParameter, весь набор – объектом OleDbParameterCollection

Установление соединения при помощи типа OleDbConnection. При работе с управляемым провайдером OLE DB первое, что необходимо сделать, – установить соединение с источником данных при помощи типа OleDbConnection. Работа с OleDbConnection во многом напоминает работу с объектом Connection в классическом ADO. Для OleDbConnection предусмотрено использование строки подключения (connection string), состоящей из пар «имя-значение». С ее помощью можно задать имя компьютера, к которому производится подключение, параметры безопасности подключения, имя базы данных, к которой производится подключение, а также, конечно, имя самого провайдера OLE DB. Полное описание всех возможных вариантов пар «имя-значение» можно найти в электронной документации к Microsoft Visual Studio (или MSDN Library for Visual Studio), тем не менее, следует ограничиться некоторыми наиболее важными моментами.

Создать строку подключения можно при помощи свойства ConnectionString (или передав ее в качестве параметра конструктора). Предполагается, что необходимо подключиться к базе данных «База данных вакансий» на компьютере с именем «-» (при работе локально можно использовать имя «(local)») через управляемый провайдер OLE DB. Код для этого может быть таким:

```
OleDbConnection connection = new OleDbConnection();

connection.ConnectionString =
        "Provider=SQLOLEDB.1;" +
        @"Data Source=-\SQLEXPRESS;" +
        "Initial Catalog=База данных вакансий;" +
        "Integrated Security=SSPI;" +
        "Persist Security Info=False;";
```

Provider – это имя провайдера OLE DB, который будет использован для обращения к источнику данных (возможные значения для этого имени приведены в табл. 2.13).

Таблица 2.13

Наиболее часто используемые провайдеры OLE DB

Значение	Описание
Microsoft.JET.OLED8.4.0	Этот провайдер OLE DB используется для подключения к базам данных JET 9, т. е. Microsoft Access
MSDAORA	Для подключения к базам данных Oracle
SQLOLEDB.1	Для подключения к базам данных Microsoft SQL Server

Data Source – это, конечно, имена компьютера и сервера баз данных, с которыми устанавливается соединение. Initial Catalog – имя базы данных, к которой происходит подключение (в данном случае «База данных вакансий»).

После настройки строки подключения следующее, что необходимо сделать, – открыть сеанс соединения с источником данных:

```
connection.Open();
```

После этого выполняются нужные действия и разрывается соединение:

```
connection.Close();
```

Конечно, ConnectionString, Open() и Close() – не единственные члены класса OleDbConnection. Кроме них в этом классе предусмотрено множество членов, которые позволяют настраивать самые разные параметры подключения. Их краткий перечень представлен в табл. 2.14 [1, 5–7].

Таблица 2.14

Члены класса OleDbConnettion

Член	Описание
BeginTransaction() CommitTransaction() RollbackTransaction()	Используются для того, чтобы программным образом начать транзакцию, закончить ее или отменить
Close()	Закрывает соединение с источником данных (наиболее рекомендуемый способ)
ConnectionString	Позволяет настроить строку подключения при установлении соединения или получить ее содержание
ConnectionTimeout	Позволяет получить или установить время тайм-аута при установке соединения
Database	Позволяет получить или установить название текущей базы данных во время подключения
DataSource	Позволяет получить или установить имя
Open()	Открывает соединение с базой данных, используя текущие настройки свойств соединения
Provider	Позволяет получить или установить имя провайдера
State	Позволяет получить информацию о текущем состоянии соединения

Построение команды SQL. Объектно-ориентированным представлением запроса на языке SQL в ADO.NET является класс OleDbCommand. Сам текст команды определяется через свойство CommandText. Множество типов ADO.NET принимают объект OleDbCommand в качестве параметра для того, чтобы передать запрос к источнику данных. Помимо свойства CommandText, которое позволяет определить сам текст запроса, в классе OleDbCommand предусмотрено также множество других членов, которые позволяют определить характеристики запроса (табл. 2.15).

Таблица 2.15

Члены класса OleDbCommand

Член	Описание
Cancel()	Прекращает выполнение команды
CommandText	Позволяет получить или задать текст запроса на языке SQL (возможно, с особенностями диалекта, определяемыми типом источника данных), который будет передан источнику данных
CommandTimeout	Позволяет получить время тайм-аута при выполнении команды. По умолчанию это время равно 30 сек.
CommandType	Позволяет получить или установить значение, определяющее, как именно будет интерпретирован текст запроса, заданный через свойство. Позволяет получить ссылку на объект OleDbConnection, для которого используется данный объект CommandText
Connection	Позволяет получить ссылку на объект OleDbConnection, для которого используется данный объект OleDbCommand
ExecuteReader()	Возвращает объект OleDbDataReader
Parameters	Возвращает коллекцию параметров OleDbParameterCollection
Prepare()	Готовит команду к выполнению (например, она будет откомпилирована) на источнике данных

Работа с типом OleDbCommand производится очень просто, при этом, как и в случае с OleDbConnection, в распоряжении есть несколько способов добиться того же результата. В качестве примера можно привести два варианта выполнения одной и той же команды SQL к объекту OleDbConnection. Для каждого из них считает, что уже открыто соединение, представленное объектом OleDbConnection с именем connection.
Первый вариант:

```
string strSQL = "SELECT Название FROM Организации WHERE
Адрес='Адрес 1'";

OleDbCommand OleDbC_1 = new OleDbCommand(strSQL, connection);
```

Второй вариант [1, 5–7]:

```
string strSQL = "SELECT Название FROM Организации WHERE
Адрес='Адрес 1'";
```

```
OleDbCommand OleDbC_1 = new OleDbCommand();

OleDbC_1.Connection = connection;
OleDbC_1.CommandText = strSQL;
```

Работа с OleDbDataReader. После того как открыто соединение с источником данных и создан объект – команда SQL, следующая задача – передать эту команду (запрос) источнику данных. Это можно сделать несколькими способами, но использование OleDbDataReader – наиболее простой и быстрый способ получения информации от источника данных, хотя и наименее гибкий. Этот класс представляет однонаправленный (только вперед), доступный только для чтения поток данных, который за один раз возвращает одну строку в ответ на запрос SQL.

Класс OleDbDataReader очень полезен, когда необходимо последовательно обработать большое количество данных и в то же время не нужно выполнять с этими данными какие-либо операции в памяти. Например, если запрос возвращает 20 000 записей из таблицы для помещения их в текстовый файл, организовывать для них промежуточное хранение в оперативной памяти через объект DataSet было бы непрактичным (и с точки зрения требований к оперативной памяти, и с точки зрения производительности). Гораздо лучше поток данных, возвращаемый из источника данных, перенаправить напрямую в другой поток, производящий запись в текстовый файл. В этой ситуации и потребуется объект OleDbDataReader. Следует обратить внимание, что в отличие от DataSet при использовании OleDbDataReader соединение с базой данных не закрывается автоматически, а сохраняется активным до тех пор, пока не закроется явным образом.

Следует проиллюстрировать все вышеприведенное на примере. Предполагается, что в распоряжении есть класс, который производит простой запрос к базе данных «База данных вакансий» при помощи метода ExecuteReader(). Этот метод возвратит объект класса OleDbDataReader, после чего можно воспользоваться методом Read() для записи возвращаемых из базы данных записей в стандартный поток ввода-вывода:

```
class OleDb
{

        static void Main(string[] args)
        {
                OleDbConnection connection = new OleDbConnection();

                connection.ConnectionString =
```

```
                    "Provider=SQLOLEDB.1;" +
                    @"Data Source=-\SQLEXPRESS;" +
                    "Initial Catalog=База данных вакансий;" +
                    "Integrated Security=SSPI;" +
                    "Persist Security Info=False;";
                connection.Open();
                string strSQL = "SELECT Название FROM Организации
WHERE
                Адрес='Адрес 1'";

                OleDbCommand OleDbC_1 = new OleDbCommand(strSQL,
connection);

                OleDbDataReader myDataReader;
                myDataReader = OleDbC_1.ExecuteReader();

                while (myDataReader.Read())
                {
                        Console.WriteLine("Организация по адресу 1: {0}",
myDataReader
                         ["Название"].ToString());
                }

                myDataReader.Close();
                connection.Close();
        }

}
```

В результате будет выведен список всех организаций с адресом «Адрес 1» в базе данных «База данных вакансий» (рис. 2.21).

Поскольку объекты DataReader (вскоре произойдет знакомство и с другими их разновидностями) обеспечивают лишь однонаправленный поток данных только для чтения, перемещаться по этим данным в приложении невозможно. Все, что можно сделать, – считать каждую запись, возвращаемую запросом, и использовать ее в приложении, как это видно из примера.

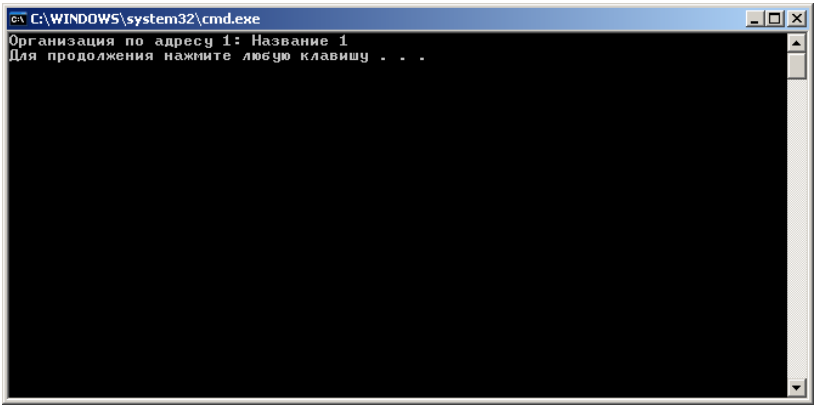


Рис. 2.21. Класс OleDbDataReader в действии

После того как DataReader станет больше не нужен, следует не забыть явным образом закрыть соединение с базой данных. Это делается при помощи метода Close(). Конечно же, помимо методов Read() и Close(), в классе OleDbDataReader предусмотрены и другие методы, например для получения значения из указанного столбца в нужном формате (GetBoolean(), GetByte() и т. д.). Кроме того, можно отметить свойство FieldCount, которое возвращает количество столбцов в текущей записи [1, 5–7].

Подключение к базе данных Microsoft Access. Наряду с обращением к базам данных Microsoft SQL Server аналогичным образом можно устанавливать соединение и с базами данных других видов. Например, можно подключаться к распространенному виду баз данных – базам данных Microsoft Access. Для примера можно просто изменить уже созданное приложение таким образом, чтобы считывать данные из базы данных Microsoft Access (файла MDB).

Как в классическом ADO, в ADO.NET, чтобы подключиться к другому источнику данных из уже готового кода, обычно достаточно лишь поменять содержимое строки подключения. Всего потребуется внести два изменения: в строке Provider поместить ссылку на Microsoft.JET.OLED8.4.0 вместо SQLOLEDB.1 и в имени источника данных указать путь к файлу MDB:

```
OleDbConnection connection = new OleDbConnection();

connection.ConnectionString =
        "Provider=Microsoft.JET.OLEDB.4.0;" +
```

```
@"Data Source=E:\Базы данных\Microsoft Access\База данных
вакансий.mdb";
connection.Open();
```

После того как соединение с базой данных установлено, можно выполнять все необходимые операции точно так же, как и в предыдущем варианте приложения. Единственное, о чем необходимо помнить: при подключении к базе данных Microsoft Access можно использовать только типы из пространства имен System.Data.OleDb. System.Data.SqlClient – только для баз данных Microsoft SQL Server [1, 5–7].

Выполнение хранимых процедур. Одно из важнейших решений, которые придется принять при создании распределенного приложения, – где будет реализована бизнес-логика. Один из возможных подходов – реализовать библиотеки кода с возможностью повторного использования (COM-компонент), которые будут управляться каким-либо вспомогательным процессом вроде Microsoft Windows 2000 Component Services Manager. Другой подход – поместить бизнес-логику непосредственно на источник данных, реализовав ее в виде хранимых процедур (еще один распространенный подход – использовать одновременно и то и другое).

Хранимая процедура – это набор команд SQL, который в виде объекта со своим именем хранится в базе данных. Хранимая процедура может возвращать набор строк (или просто значений указанных типов данных), а также принимать любое количество обязательных и необязательных параметров. По своему назначению хранимая процедура очень похожа на обычную функцию, например в C#, с теми очевидными отличиями, которые вытекают из сущности хранимой процедуры как объекта базы данных. Например, можно добавить в базу данных «База данных вакансий» простую хранимую процедуру «Адрес», которая будет принимать единственный параметр типа int. В качестве входящего параметра, конечно же, будет использован номер организации, а возвращаться будет адрес – значения столбца «Адрес» (типа nchar). Синтаксис (скрипт) для создания хранимой процедуры может быть таким:

```
USE [База данных вакансий]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE Адрес
```

```
        @p1 int,
        @p2 nchar(50) output
AS
BEGIN
        SET NOCOUNT ON;
        SELECT @p2 = Адрес from Организации where N_организации = @p1
END
```

Теперь, когда указанный скрипт выполнен и хранимая процедура создана, необходимо каким-то образом ее вызвать. Первое, что, конечно, для этого необходимо, – создать объект OleDbConnection, настроить строку подключения и открыть соединение с базой данных. После этого потребуется создать объект OleDbCommand, указав имя хранимой процедуры и установив нужное значение свойства CommandType:

```
OleDbConnection connection = new OleDbConnection();

connection.ConnectionString =
        "Provider=SQLOLEDB.1;" +
        @"Data Source=-\SQLEXPRESS;" +
        "Initial Catalog=База данных вакансий;" +
        "Integrated Security=SSPI;" +
        "Persist Security Info=False;";
connection.Open();

OleDbCommand OleDbC_2 = new OleDbCommand("Адрес", connection);

OleDbC_2.CommandType = CommandType.StoredProcedure;
```

Для свойства CommandType используются значения из одноименного перечисления. Значения этого перечисления приведены в табл. 2.16.

Таблица 2.16

Значения перечисления CommandType

Значение	**Описание**
StoredProcedure	Используется при настройке объекта OleDbCommand, который обеспечивает запуск на сервере баз данных хранимой процедуры
TableDirect	Для OleDbCommand достаточно будет указать имя таблицы – в результате будут возвращены все данные из этой таблицы
Text	Объект OleDbCommand будет представлять стандартную команду на языке SQL. Это значение используется по умолчанию

При выполнении обычных команд SQL о свойстве CommandType задумываться не надо, поскольку значение по умолчанию (Text) в этом случае вполне подходит. Однако если команда представляет собой хранимую процедуру, то необходимо будет указать для свойства CommandType значение StoredProcedure.

Обычно при запуске на выполнение хранимой процедуры необходимо определить передаваемые ей параметры. Для этой цели используется класс OleDbParameter – объектная оболочка для параметров, которые передаются хранимой процедуре или возвращаются как результат ее работы. Для класса OleDbParameter предусмотрено большое количество свойств, которые позволяют определить имя, размер и тип данных для параметра, а также передается ли он хранимой процедуре или принимается от нее. Наиболее важные свойства типа OleDbParameter представлены в табл. 2.17.

Таблица 2.17

Свойства класса OleDbParameter

Свойство	**Описание**
DataType	Определяет тип параметра в терминах .NET
DbType	Позволяет получить или задать тип данных, используемый на источнике данных. Для этого свойства используются значения из перечисления OleDbType
Direction	Определяет, будет ли этот параметр передаваться на источник данных, возвращаться с источника данных или он сможет использоваться для передачи и в том и в другом направлении
IsNullable	Позволяет определить, будет ли данный параметр допускать пустые значения (значения типа NULL)
ParameterName	Позволяет получить или установить имя параметра
Precision	Позволяет получить или установить максимальное количество цифр, используемых для значения параметра (определяемое через свойство Value)
Scale	Позволяет получить или установить количество десятичных разрядов, используемых для значения параметра
Size	Позволяет получить или установить максимальный размер данных для используемого параметра
Value	Позволяет получить или установить значение параметра

В данной ситуации хранимая процедура принимает один параметр и один параметр возвращает. Соответствующий код приведен ниже. Следует обратить внимание, что параметры (т. е. объекты OleDbParameter) добав-

ляются в коллекцию ParametersCollection объекта OleDbCommand при помощи свойства Parameters.

```
OleDbParameter OleDbP = new OleDbParameter();

OleDbP.ParameterName = "@p1";
OleDbP.OleDbType = OleDbType.Integer;
OleDbP.Direction = ParameterDirection.Input;
OleDbP.Value = 1;
OleDbC_2.Parameters.Add(OleDbP);

OleDbP = new OleDbParameter();

OleDbP.ParameterName = "@p2";
OleDbP.OleDbType = OleDbType.Char;
OleDbP.Size = 50;
OleDbP.Direction = ParameterDirection.Output;
OleDbC_2.Parameters.Add(OleDbP);
```

Последнее, что осталось сделать, – запустить программу на выполнение при помощи OleDbCommand.ExecuteNonQuery(). Следует обратить внимание, что для получения возвращаемых хранимой процедурой значений (в данном случае – адреса организации) используется свойство Value объекта OleDbParameter:

```
OleDbC_2.ExecuteNonQuery();
Console.WriteLine("Информация хранимой процедуры:");
Console.WriteLine("№ организации: " + OleDbC_2.Parameters["@p1"].Value);
Console.WriteLine("Адрес: " + OleDbC_2.Parameters["@p2"].Value);
```

Результат работы программы представлен на рис. 2.22 [1, 5–7].

Тип OleDbDataAdapter. К этому моменту уже известно, как подключиться к источнику данных при помощи класса OleDbConnection, выполнить команду SQL, используя типы OleDbCommand и OleDbParameter, и получить однонаправленный поток данных при помощи OleDbDataReader. Всего этого вполне достаточно для выполнения хранимой процедуры или перенаправления потока данных из базы данных в поток вывода. Однако очень часто требуется заполнить полученными с сервера данными объект DataSet и выполнить с ними определенные операции. Наиболее гибкий способ, который позволяет это сделать, – использование класса OleDbDataAdapter.

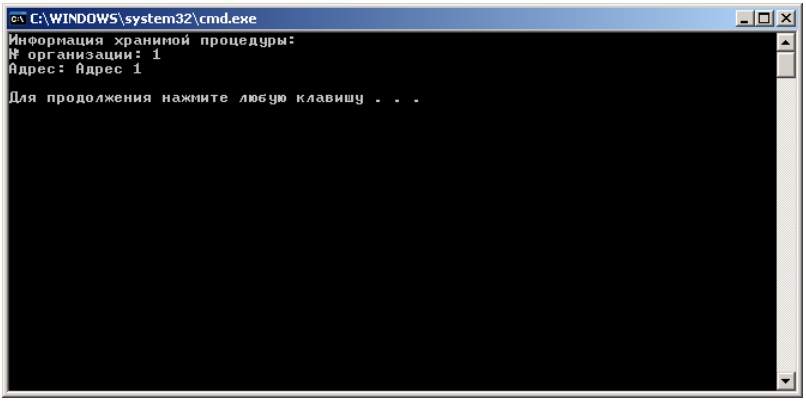


Рис. 2.22. Запуск хранимой процедуры

Основное назначение этого класса – извлечь информацию из источника данных и заполнить ею объект DataTable в DataSet при помощи метода Fill(). Метод Fill() многократно перегружен. Его два наиболее часто используемых варианта (возвращаемое значение int позволяет получить информацию о количестве записей, полученных из источника данных):

```
public Int Fill(DataSet DS, string tableName};

public Int Fill(DataSet DS, string tableName, int startRecord, int maxRecord);
```

При использовании первого варианта заполняется объект DataSet данными, полученными из таблицы на источнике данных с указанный именем, второго – которые находятся в указанных границах.

Конечно, перед тем как вызывать этот метод, потребуется уже созданный объект OleDbDataAdapter. Конструктор OleDbDataAdapter также многократно перегружен, но обычно необходимо указать информацию о параметрах подключения к базе данных и команду SELECT на языке SQL, которая будет использована для заполнения DataTable.

OleDbDataAdapter позволяет не только заполнять объект DataTable внутри DataSet данными, полученными из источника, но и помещать измененные данные обратно в источник данных при помощи стандартных команд SQL. В табл. 2.18 представлены члены класса OleDbDataAdapter, которые позволяют это сделать, а также некоторые другие важнейшие члены этого класса.

Таблица 2.18

Наиболее важные члены OleDbDataAdapter

Член	Описание
DeleteCommand InsertCotnmand SelectCommand UpdateCommand	Используются для определения того, какая именно команда SQL будет передана на источник данных при вызове метода Update(). Каждое из этих свойств определяется при помощи объектов OleDbCommand
Fill()	Заполняет указанную таблицу в DataSet определенным пользователем количеством записей
GetFillParameters()	Возвращает все параметры, использованные при выполнении запроса SELECT к источнику данных
Update()	Вызывает соответствующие команды INSERT, UPDATE, DELETE к источнику данных для каждой вставленной, измененной или удаленной строки в таблице объекта DataSet

При использовании свойств DeleteCommand, InsertCommand, UpdateCommand и SelectComand объект OleDbDataAdapter автоматически переводит внесенные изменения в таблицу данных в DataSet в соответствующие команды на языке SQL, сохраняя, таким образом, внесенные в DataSet изменения в источнике данных. Эти свойства позволяют определить соответствующие команды SQL в деталях. Однако прежде чем приступить к их рассмотрению, следует познакомиться с тем, как можно использовать OleDbDataAdapter в коде программы для заполнения данными объекта DataSet [1, 5–7].

Заполнение данными объекта DataSet при помощи OleDbDataAdapter. Заполнить данными объект DataSet (в котором имеется только одна таблица) при помощи OleDbDataAdapter можно, если использовать следующий код:

```
class Program
{

        static void Main(string[] args)
        {

                OleDbConnection connection = new OleDbConnection();

                connection.ConnectionString =
                        "Provider=SQLOLEDB.1;" +
                        @"Data Source=-\SQLEXPRESS;" +
                        "Initial Catalog=База данных вакансий;" +
```

```
                    "Integrated Security=SSPI;" +
                    "Persist Security Info=False;";
                connection.Open();
                string selectCmd = "SELECT * FROM Организации";

                OleDbDataAdapter dAdapt = new OleDbDataAdapter(selectCmd,
                    connection);

                DataSet Baza_Dannikh = new DataSet("База данных вакансий");

                try
                {
                    dAdapt.Fill(Baza_Dannikh, "Организации");
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                }
                finally
                {
                    connection.Close();
                }

                PrintTable(Baza_Dannikh);
        }

}
```

Следует обратить внимание, что создание таблицы «Организации» (объекта DataTable) было совсем не похоже на то, как делалось это ранее. Вместо того чтобы создавать DataTable из отдельных элементов, а затем добавлять ее в DataSet, просто указано имя создаваемой таблицы («Организации») в качестве второго параметра для метода Fill(). Все остальное этот метод сделал автоматически: создал объект DataTable, присвоил ему указанное имя, создал внутри DataTable объекты DataColumn и заполнил таблицу строками из источника данных, полученных в результате выполнения запроса SELECT. Команда SELECT была передана источнику данных как параметр конструктора OleDbDataAdapter.

С точки зрения объектно-ориентированного программирования, конечно, привычнее было бы создать для команды SELECT отдельный объект (вместо переменной string) и использовать именно его. Можно сделать и так. Понятно, что команду SELECT в этом случае будет представлять объект OleDbComrnand. Чтобы связать этот объект с объектом OleDbDataAdapter, используется свойство SelectCommand:

```
OleDbCommand selectCmd = new OleDbCommand("SELECT * FROM
Организации",
        connection);

OleDbDataAdapter dAdapt = new OleDbDataAdapter();

dAdapt.SelectCommand = selectCmd;
```

В любом случае результат будет одним и тем же. Однако, чтобы программа действительно заработала, осталось создать ее последнюю часть – метод PrintTable(). Он не слишком сложен:

```
public static void PrintTable(DataSet ds)
{
        Console.WriteLine("Список организаций:\n");

        DataTable T_Organizatsii = ds.Tables["Организации"];

        for (int curCol = 0; curCol < T_Organizatsii.Columns.Count; curCol++)
        {

        Console.Write(T_Organizatsii.Columns[curCol].ColumnName.Trim() + "\t");
        }

        Console.WriteLine();
        Console.WriteLine();

        for (int curRow = 0; curRow < T_Organizatsii.Rows.Count; curRow++)
        {

                for (int curCol = 0; curCol < T_Organizatsii.Columns.Count;
curCol++)
                {

        Console.Write(T_Organizatsii.Rows[curRow][curCol].ToString().
                                Trim() + "\t");
                }

                Console.WriteLine();
        }

        Console.WriteLine();
}
```

Результат работы программы представлен на рис. 2.23.

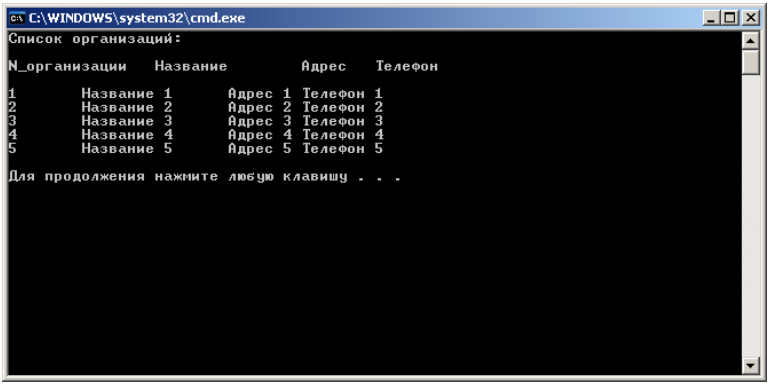


Рис. 2.23. Объект OleDbDataAdapter в действии

Таким образом, тип OleDbDataAdapter позволяет (при помощи свойств SelectCommand, InsertCommand, UpdateCommand и DeleteCommand) вносить изменения в исходную базу данных на источнике данных [1, 5–7].

2.4. Управляемый провайдер SQL. Пространство имен System.Data.SqlTypes. Вставка новых записей при помощи SqlDataAdapter. Изменение записей в таблице при помощи SqlDataAdapter. Автоматическое создание команд SQL. Заполнение объекта DataSet с несколькими таблицами и добавление объектов DataRelations. Настройка системы управления базами данных Microsoft SQL Server в SQL Server Configuration Manager. Работа в Microsoft SQL Server Management Studio

Управляемый провайдер SQL. Перед тем как приступить к вставке, изменению и удалению записей в источнике данных при помощи объектов DataAdapter, следует рассмотреть еще один вопрос – особенности работы с управляемым провайдером SQL. Как уже отмечалось, по своим функциональным возможностям этот провайдер схож с управляемым провайдером OLE DB, но он предназначен только для работы с базами данных на сервере Microsoft SQL Server и хорошо оптимизирован именно для такой работы.

Типы, которые составляют управляемый провайдер SQL, определены в пространстве имен System.Data.SqlClient. В нем можно найти много об-

щего с уже знакомыми типами из пространства имен System.Data.OleDb. Список основных типов приведен в табл. 2.19.

Таблица 2.19

Наиболее важные типы пространства имен System.Data.SqlClient

Тип	Описание
SqlCommand	Представляет запрос SQL, производимый к источнику данных – Microsoft SQL Server
SqlConnection	Представляет открытое соединение с источником данных
SqlDataAdapter	Представляет соединение с базой данных и набор команд, используемых для заполнения объекта DataSet, а также обновления исходной базы данных после внесения изменений в DataSet
SqlDataReader	Обеспечивает метод считывания потока данных из источника в одном направлении (вперед)
SqlErrors SqlError SqlException	SqlErrors представляет набор ошибок и предупреждений, возвращаемых источником данных. Сами эти ошибки и предупреждения представлены объектами SqlError. При возникновении ошибки может быть сгенерировано исключение, представленное объектом SqlException
SqlParameterCollection SqlParameter	Используются для передачи параметров хранимой процедуре на источнике данных. Параметры представлены объектами SqlParameter

Работа с этими типами данных практически идентична работе с аналогичными типами данных из пространства имен System.Data.OleDb. Однако, чтобы сделать работу с типами данных управляемого провайдера SQL более привычной, в оставшейся части будут приведены примеры для работы именно с ними [1, 5–7].

Пространство имен System.Data.SqlTypes. При использовании управляемого провайдера SQL очень удобно использовать классы, которые предназначены для представления «родных» типов данных Microsoft SQL Server. Эти классы определены в пространстве имен System.Data.SqlTypes. В качестве дополнительной информации их перечень приведен в табл. 2.20 [1, 5–7].

Таблица 2.20

Типы пространства имен System.Data.SqlTypes

Тип	Тип данных на Microsoft SQL Server
SqlBinary	binary, varbinary, timestamp, image
SqlInt64	bigint
SqlBit	bit
SqlDateTime	datetime, smalldatetime
SqlNumeric	decimal
SqlDouble	float
SqlInt32	int
SqlMoney	money, smallmoney
SqlString	nchar, ntext, nvarchar, sysname, text, varchar, char
SqlNumeric	numeric
SqlSingle	real
SqlInt16	smallint
System.Object	sql_variant
SqlByte	tinyint
SqlGuid	uniqueidentifier

Вставка новых записей при помощи SqlDataAdapter. Сделав переход от типов управляемого провайдера OLE DB к типам управляемого провайдера SQL, необходимо по-прежнему выяснить, как обеспечить запись измененных данных в базу данных на источнике при помощи объектов DataAdapter. Первое, с чем следует познакомиться, – со вставкой новых записей при помощи SqlDataAdapter (еще раз надо отметить, что с точки зрения использования в коде программы разницы между SqlDataAdapter и OleDbDataAdapter почти никакой нет). Как обычно, первое, что необходимо сделать, – открыть соединение с базой данных:

```
SqlConnection connection = new SqlConnection(@"Server=-\SQLEXPRESS;
Uid=sa;
Pwd=; Database=База данных вакансий; Integrated Security=SSPI");

SqlDataAdapter dAdapt = new SqlDataAdapter("SELECT * FROM
Организации",
        connection);

connection.Open();

SqlCommand killCmd = new SqlCommand("DELETE FROM Организации
WHERE " +
        "N_организации = '6'", connection);
```

```
killCmd.ExecuteNonQuery();
connection.Close();
```

Первое, что обращает на себя внимание, – изменение содержания строки подключения. При работе с управляемым провайдером SQL не указывается значение для Provider (поскольку всегда происходит подключение к Microsoft SQL Server). Кроме того, в целом используется несколько другой набор пар «имя-значение». После того как соединение с использованием строки подключения установлено, можно уже действовать привычным путем: создается объект SqlDataAdapter и определяется для него текст запроса SQL через конструктор.

Шаг второй – это устранение возможных проблем, которые могут быть связаны с тем, что в таблице в базе данных уже есть запись с совпадающим значением «N_организации». Иначе в этой ситуации просто невозможно будет произвести вставку, так как на этот столбец наложено ограничение первичного ключа. Поэтому на всякий случай эту проблему следует решить наиболее простым способом. После этого следующая задача – создать новый объект для представления команды SQL INSERT. Конечно же, для представления этой команды будет использован объект SqlCommand, а для параметров команды INSERT – объект SqlParameter:

```
dAdapt.InsertCommand = new SqlCommand("INSERT INTO Организации " +
        "(N_организации, Название, Адрес, Телефон) VALUES
(@N_организации, " +
        "@Название, @Адрес, @Телефон)", connection);

SqlParameter workParam = null;

workParam = dAdapt.InsertCommand.Parameters.Add(new SqlParameter
         ("@N_организации", SqlDbType.Int));
workParam.SourceColumn = "N_организации";
workParam.SourceVersion = DataRowVersion.Current;
workParam = dAdapt.InsertCommand.Parameters.Add(new
SqlParameter("@Название",
        SqlDbType.NChar));
workParam.SourceColumn = "Название";
workParam.SourceVersion = DataRowVersion.Current;
workParam = dAdapt.InsertCommand.Parameters.Add(new
SqlParameter("@Адрес",
        SqlDbType.NChar));
workParam.SourceColumn = "Адрес";
workParam.SourceVersion = DataRowVersion.Current;
```

```
workParam = dAdapt.InsertCommand.Parameters.Add(new
SqlParameter("@Телефон",
        SqlDbType.NChar));
workParam.SourceColumn = "Телефон";
workParam.SourceVersion = DataRowVersion.Current;
```

Теперь произведена подготовка к тому, чтобы вставить новую строку в DataSet (предварительно его следует создать и заполнить данными из источника) и записать произведенные изменения из DataSet обратно в источник данных. Кроме того, для наглядности можно также вывести то, что получилось, на консоль при помощи ранее созданной функции PrintTable():

```
DataSet Baza_Dannikh = new DataSet();

dAdapt.Fill(Baza_Dannikh, "Организации");
PrintTable(Baza_Dannikh);

DataRow newRow = Baza_Dannikh.Tables["Организации"].NewRow();

newRow["N_организации"] = 6;
newRow["Название"] = "Название 6";
newRow["Адрес"] = "Адрес 6";
newRow["Телефон"] = "Телефон 6";
Baza_Dannikh.Tables["Организации"].Rows.Add(newRow);
try
{
        dAdapt.Update(Baza_Dannikh, "Организации");
        Baza_Dannikh.Dispose();

        Baza_Dannikh = new DataSet();

        dAdapt.Fill(Baza_Dannikh, "Организации");
        PrintTable(Baza_Dannikh);
}
catch (Exception e)
{
        Console.Write(e.ToString());
}
```

Результат работы приложения представлен на рис. 2.24 [1, 5–7].

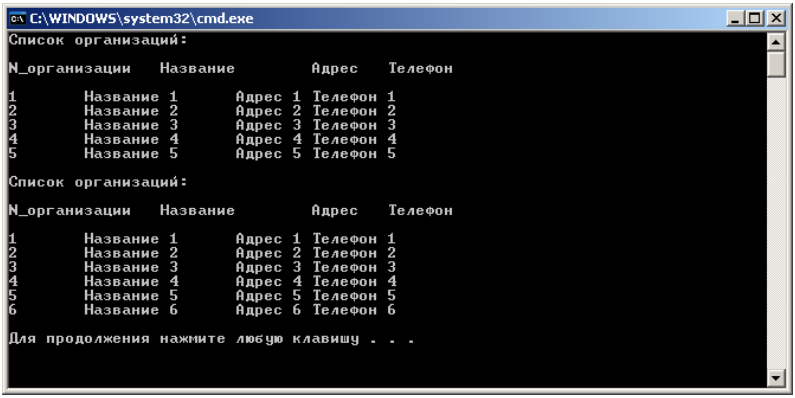


Рис. 2.24. Применение свойства InsertCommand

Изменение записей в таблице при помощи SqlDataAdapter. Процесс изменения существующих строк в таблице на источнике данных очень похож на процесс вставки новых строк. Как обычно, все начинается с создания объекта SqlDataAdapter и открытия соединения с базой данных. Далее задача – воспользоваться свойством UpdateCommand:

```
dAdapt.UpdateCommand = new SqlCommand("UPDATE Организации SET
Название " +
        "= @Название, Адрес = @Адрес, Телефон = @Телефон WHERE "+
        "N_организации = @N_организации", connection);

SqlParameter workParam = null;

workParam = dAdapt.UpdateCommand.Parameters.Add(new SqlParameter
        ("@N_организации", SqlDbType.Int));
workParam.SourceColumn = "N_организации";
workParam.SourceVersion = DataRowVersion.Current;
workParam = dAdapt.UpdateCommand.Parameters.Add(new
SqlParameter("@Название",
        SqlDbType.NChar));
workParam.SourceColumn = "Название";
workParam.SourceVersion = DataRowVersion.Current;
workParam = dAdapt.UpdateCommand.Parameters.Add(new
SqlParameter("@Адрес",
        SqlDbType.NChar));
workParam.SourceColumn = "Адрес";
workParam.SourceVersion = DataRowVersion.Current;
```

```
workParam = dAdapt.UpdateCommand.Parameters.Add(new
SqlParameter("@Телефон",
        SqlDbType.NChar));
workParam.SourceColumn = "Телефон";
workParam.SourceVersion = DataRowVersion.Current;

DataSet Baza_Dannikh = new DataSet();

dAdapt.Fill(Baza_Dannikh, "Организации");
PrintTable(Baza_Dannikh);

DataRow changeRow = Baza_Dannikh.Tables["Организации"].Rows[5];

changeRow["Название"] = "Название X";
changeRow["Адрес"] = "Адрес X";
changeRow["Телефон"] = "Телефон X";
try
{
        dAdapt.Update(Baza_Dannikh, "Организации");
        Baza_Dannikh.Dispose();

        Baza_Dannikh = new DataSet();

        dAdapt.Fill(Baza_Dannikh, "Организации");
        PrintTable(Baza_Dannikh);
}
catch (Exception e)
{
        Console.Write(e.ToString());
}
```

Результат работы представлен на рис. 2.25 [1, 5–7].

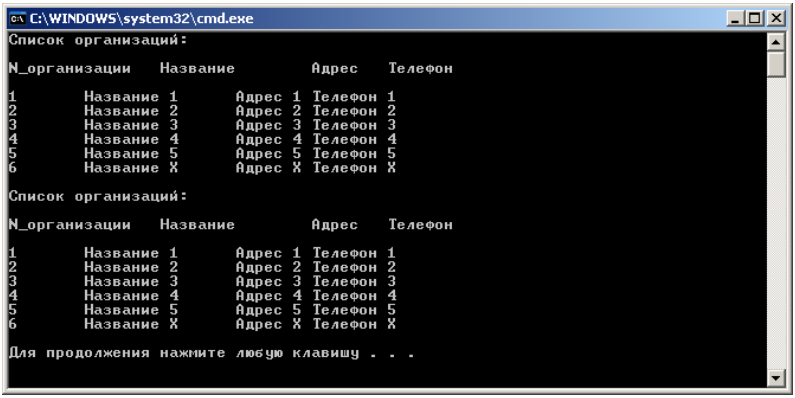


Рис. 2.25. Внесение изменений в существующие записи в базе данных

Автоматическое создание команд SQL. Работа по вставке, удалению и изменению данных при помощи объектов OleDbDataAdapter и SqlDataAdapter не является особенно сложной, но создание отдельного параметра для каждого из столбцов таблицы и передача их через свойства InsertCommand, UpdateCommand и DeleteCommand указанных типов может показаться несколько утомительной. Существует более удобный способ.

Этот способ заключается в применении класса SqlCommandBuilder. Если производится работа с объектом DataTable, состоящим из единственной таблицы (а не результатом объединения нескольких таблиц), то SqlCommandBuilder автоматически настроит свойства InsertCommand, UpdateCommand и DeleteCommand в соответствии с тем, что изначально использовалось для свойства SelectComrnand. Однако применяться SqlCommandBuilder может не всегда. Помимо ограничений, связанных с объединениями, о которых уже упоминалось, должно обязательно выполняться еще одно условие: для таблицы должен быть определен первичный ключ, и этот ключ должен использоваться в исходном запросе – команде SELECT. Главное же преимущество применения SqlCommandBuilder заключается в том, что нет необходимости создавать объекты SqlParameter вручную.

Следует рассмотреть применение SqlCommandBuilder на примере. Предполагается, что в распоряжении имеется форма Windows, на которой расположен элемент управления DataGrid. Пользователь может редактировать записи прямо в DataGrid, а по завершении он должен нажать специальную кнопку, чтобы сохранить изменения в базе данных. Конструктор для формы будет выглядеть так:

```
private SqlConnection connection = new SqlConnection(@"Server=-
\SQLEXPRESS;
Uid=sa; Pwd=; Database=База данных вакансий; Integrated Security=SSPI");

private SqlDataAdapter dAdapt;

private SqlCommandBuilder org_Builder;

private DataSet Baza_Dannikh = new DataSet();

public Form1()
{
        InitializeComponent();
        CenterToScreen();

        dAdapt = new SqlDataAdapter("SELECT * FROM Организации",
connection);

        org_Builder = new SqlCommandBuilder(dAdapt);

        dAdapt.Fill(Baza_Dannikh, "Организации");
        dataGrid1.DataSource =
Baza_Dannikh.Tables["Организации"].DefaultView;
}
```

Теперь у SqlDataAdapter есть вся необходимая информация для записи данных из DataGrid обратно в источник данных. Код для события Click кнопки будет таким:

```
private void button1_Click(object sender, EventArgs e)
{
        try
        {
                dataGrid1.Refresh();
                dAdapt.Update(Baza_Dannikh, "Организации");
        }
        catch (Exception ex)
        {
                MessageBox.Show(ex.ToString());
        }
}
```

Как обычно, вызван метод Update() и указаны объект DataSet и таблица в нем, в которую следует записать изменения. Приложение будет выглядеть примерно так, как представлено на рис. 2.26 (перед нажатием на кнопку следует убедиться, что произведен выход из режима редактирования).

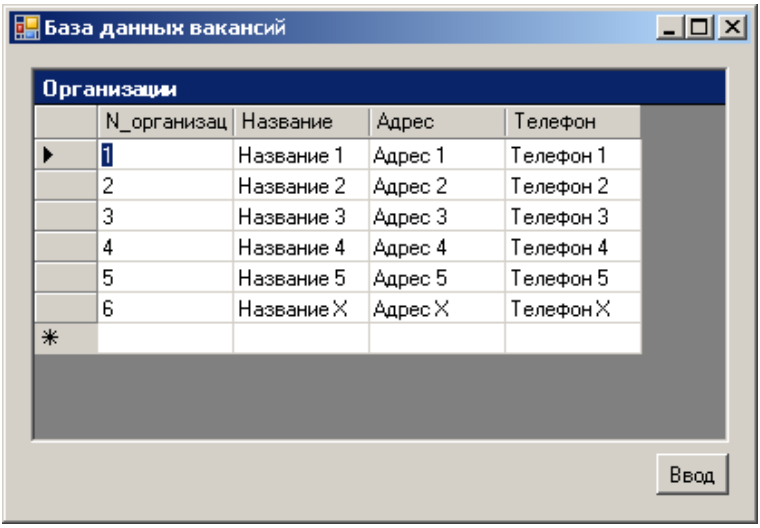


Рис. 2.26. Приложение, использующее SqlCommandBuilder

Очевидно, что применение автоматически генерируемых команд SQL гораздо проще, чем создание таких команд вручную. Однако, конечно же, как правило, все имеет свои недостатки. К примеру, если объект DataTable создан на основе объединения двух таблиц в источнике данных, использовать эту технологию не удастся. Кроме того, создание команд вручную предоставляет в распоряжение гораздо больше возможностей в различных специфических ситуациях [1, 5–7].

Заполнение объекта DataSet с несколькими таблицами и добавление объектов DataRelations. Следует завершить эту тему созданием приложения, очень похожего на то, что было создано в самом начале этой главы. Его интерфейс уже знаком: на форме три элемента управления DataGrid, в качестве источников данных для которых служат записи, полученные из таблиц «Организации», «Специальности» и «Вакансии» базы данных «База данных вакансий». Единственная кнопка формы предназначена для передачи измененных данных из DataGrid обратно в базу данных (рис. 2.27).

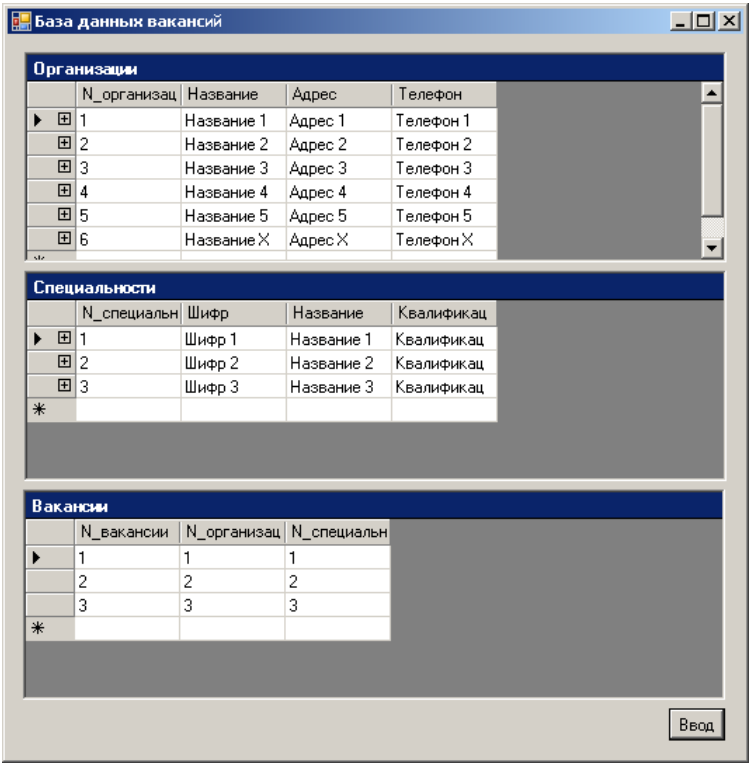


Рис. 2.27. Информация из многотабличного объекта DataSet выведена на форму

Чтобы сделать все максимально простым, нужно использовать для каждого из трех объектов SqlDataAdapter (по одному на каждую таблицу) автоматически генерируемые команды SQL. Прежде всего следует настроить соединение с базой данных и подготовить все объекты, которые понадобятся при использовании формы:

```
private SqlConnection connection = new SqlConnection(@"Server=-
\SQLEXPRESS;  "Uid=sa; Pwd=; Database=База данных вакансий; Integrated
Security=SSPI");

private SqlDataAdapter org_TableAdapter;

private SqlDataAdapter spets_TableAdapter;

private SqlDataAdapter vak_TableAdapter;
```

```
private SqlCommandBuilder org_Builder = new SqlCommandBuilder();

private SqlCommandBuilder spets_Builder = new SqlCommandBuilder();

private SqlCommandBuilder vak_Builder = new SqlCommandBuilder();

DataSet Baza_Dannikh = new DataSet();
```

Вся работа по созданию переменных для работы с данными и заполнению объекта DataSet производится в конструкторе формы. В нем также будет производиться вызов к вспомогательной функции Otnoshenie() для создания отношения между таблицами. Код для конструктора будет выглядеть так:

```
public Form1()
{
        InitializeComponent();
        CenterToScreen();

        org_TableAdapter = new SqlDataAdapter("SELECT * FROM
Организации",
                connection);

        spets_TableAdapter = new SqlDataAdapter("SELECT * FROM
Специальности",
                connection);

        vak_TableAdapter = new SqlDataAdapter("SELECT * FROM
Вакансии",
                connection);

        org_Builder = new SqlCommandBuilder(org_TableAdapter);

        spets_Builder = new SqlCommandBuilder(vak_TableAdapter);

        vak_Builder = new SqlCommandBuilder(spets_TableAdapter);

        org_TableAdapter.Fill(Baza_Dannikh, "Организации");
        spets_TableAdapter.Fill(Baza_Dannikh, "Специальности");
        vak_TableAdapter.Fill(Baza_Dannikh, "Вакансии");
        Otnoshenie();
}
```

Код для вспомогательной функции Otnoshenie() будет выглядеть в полном соответствии с тем, который приводился ранее:

```
private void Otnoshenie()
{
        DataRelation dr = new DataRelation("Организации-вакансии",

        Baza_Dannikh.Tables["Организации"].Columns["N_организации"],
                Baza_Dannikh.Tables["Вакансии"].Columns["N_ организации
"]);

        Baza_Dannikh.Relations.Add(dr);

        dr = new DataRelation("Специальности-вакансии",

        Baza_Dannikh.Tables["Специальности"].Columns["N_специальности"]
,
                Baza_Dannikh.Tables["Вакансии"].Columns["N_
специальности "]);

        Baza_Dannikh.Relations.Add(dr);
        dataGrid1.SetDataBinding(Baza_Dannikh, "Организации");
        dataGrid2.SetDataBinding(Baza_Dannikh, "Специальности");
        dataGrid3.SetDataBinding(Baza_Dannikh, "Вакансии");
}
```

Теперь, когда объект DataSet заполнен, можно работать с ним (производить все необходимые операции с данными прямо в элементах управления DataGrid) без какого-либо соединения с базой данных – полностью локально. После того как все необходимые изменения внесены, можно нажать кнопку формы «Ввод». Код для события Click этой кнопки будет таким [1, 5–7]:

```
private void button1_Click(object sender, EventArgs e)
{
        try
        {
                org_TableAdapter.Update(Baza_Dannikh, "Организации");
                spets_TableAdapter.Update(Baza_Dannikh, "Специальности");
                vak_TableAdapter.Update(Baza_Dannikh, "Вакансии");
        }
        catch (Exception ex)
        {
```

```
                MessageBox.Show(ex.Message);
            }
}
```

Настройка системы управления базами данных Microsoft SQL Server в SQL Server Configuration Manager. Система управления базами данных Microsoft SQL Server позволяет работать как с локальными, так и удаленными базами данных как со стороны сервера, так и со стороны клиента. Перед ее использованием целесообразно настроить некоторые ее параметры. Предполагается, что эта настройка производится сразу после установки Microsoft SQL Server с неизмененными параметрами.

Со стороны сервера для работы с удаленными базами данных необходимо в SQL Server Configuration Manager запустить сервис SQL Server Browser, в свойствах которого предварительно установить стартовый режим на автоматический (рис. 2.28).

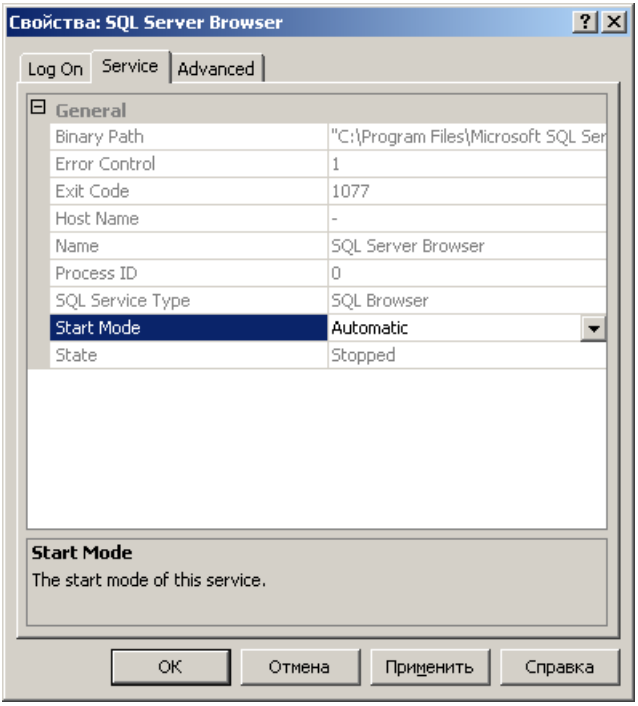


Рис. 2.28. Настройка режима запуска SQL Server Browser

Далее – включить поддержку протокола TCP / IP для соответствующего сервера (рис. 2.29).

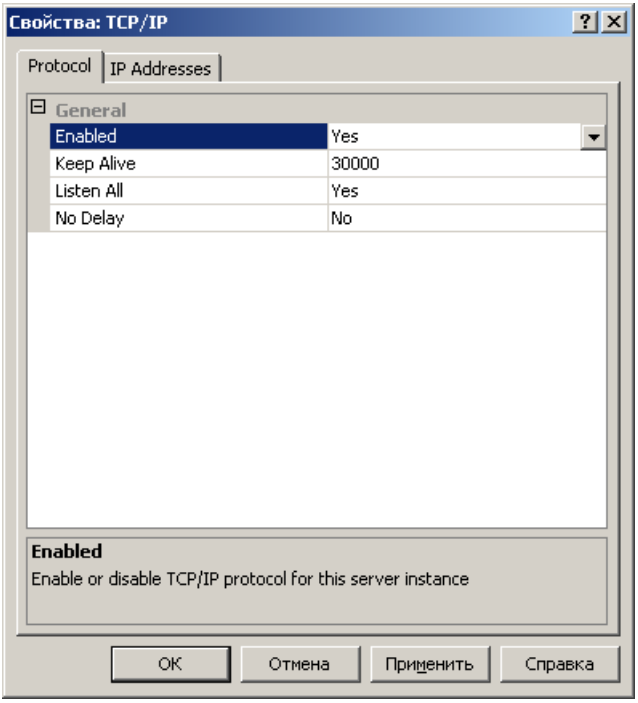


Рис. 2.29. Настройка включения протокола TCP / IP

Для работы со стороны клиента также целесообразно проверить настройку вышеуказанного протокола.

К тому же для возможности создания баз данных при локальной или удаленной работе с пользовательскими правами как со стороны сервера, так и клиента необходимо добавить группы, связанные с доступом к различным службам Microsoft SQL Server (рис. 2.30) [1, 5–7].

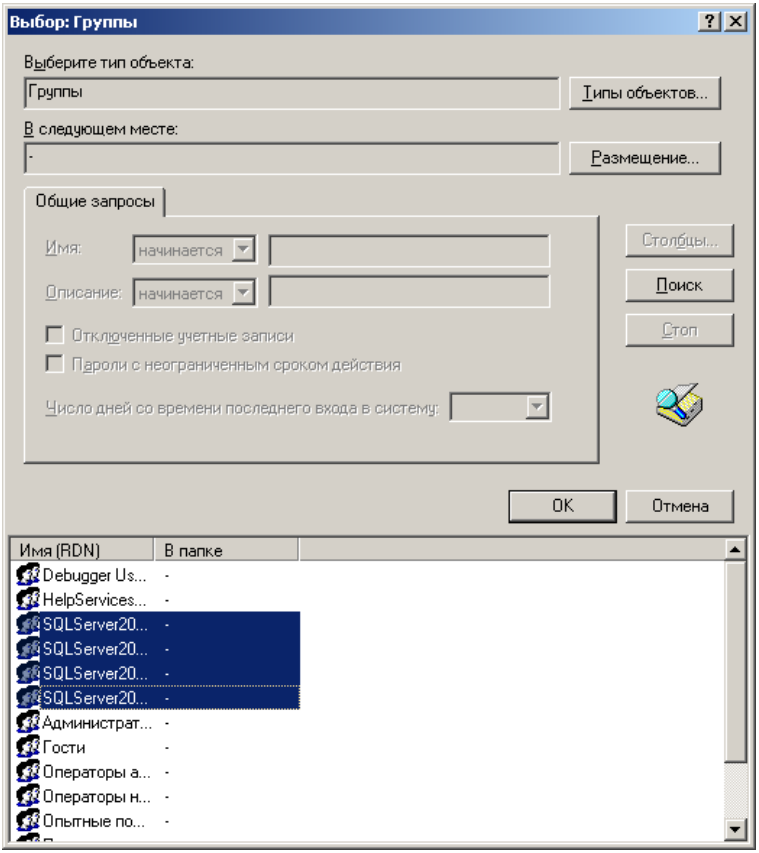


Рис. 2.30. Добавление групп для работы с Microsoft SQL Server

Работа в Microsoft SQL Server Management Studio. При подключении к серверу баз данных после загрузки Microsoft SQL Server Management Studio можно использовать два типа аутентификации: Microsoft Windows и Microsoft SQL Server. Для работы локально или удаленно достаточно использовать первый тип, однако при удаленном доступе пользователь должен обладать паролем и быть в числе локальных пользователей удаленной системы. В результате подключения станут доступны различные элементы списка в Object Explorer.

Для создания новой базы данных можно воспользоваться контекстным меню. После ее создания станут доступными новые элементы (контейнеры) для хранения диаграмм, таблиц, видов и т. д., которые также формируются посредством контекстного меню (рис. 2.31).

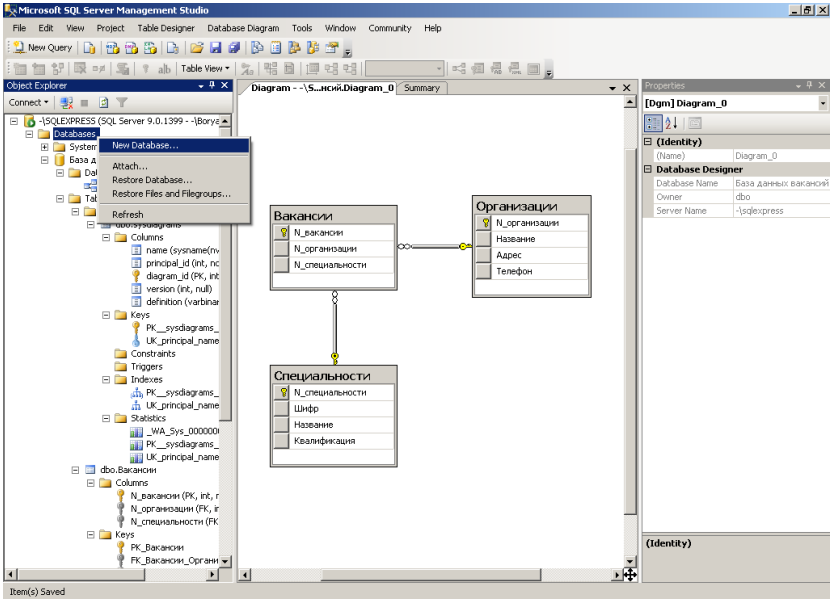


Рис. 2.31. Общий вид Microsoft SQL Server

Задание. Создать, заполнить и сохранить в виде скриптов, оформленных как проект (решение), базу данных «База данных вакансий», структура которой представлена на рис. 2.16, и добавить в нее следующую хранимую процедуру:

```
set ANSI_NULLS ON
set QUOTED_IDENTIFIER ON
go

CREATE PROCEDURE [dbo].[Адрес]
        @p1 int,
        @p2 nchar(50) output
AS
BEGIN
        SET NOCOUNT ON;
        SELECT @p2 = Адрес from Организации where N_организации =
@p1
END
```

После освоения объекта DataSet целесообразно научиться реализовывать доступ к данным, например на языке программирования C#, т. е. создавать объект DataSet при помощи соединений с реальными базами данных на примере созданной выше.

При проверке подключения к базе данных можно воспользоваться свойствами ServerVersion (версия сервера, к которому происходит подключение клиента) и State (состояние подключения) объекта OleDbConnection.

Задание. Разработать клиентское Windows-приложение для доступа к данным, хранящимся в созданной ранее базе данных, с проверкой подключения. Информацию о подключении вывести в новое окно. В приложении должны быть реализованы: 1) выполнение автоматически генерируемых команд SQL; 2) выполнение хранимой процедуры с выводом результата в новое окно; 2) возможность сохранения базы данных в файл формата XML и последующего его чтения (рис. 2.32).

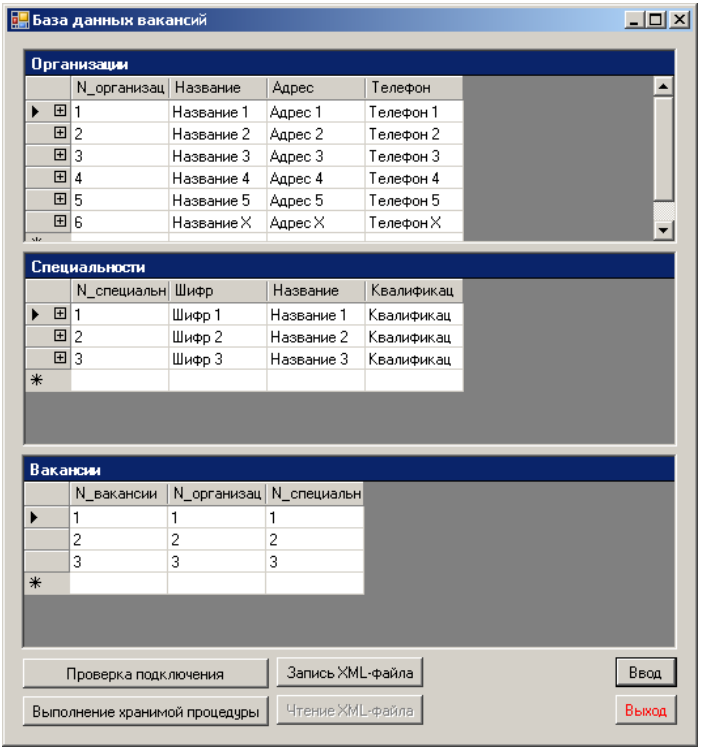


Рис. 2.32. Приложение для доступа к данным

К этому времени должны быть приобретены некоторые навыки работы с управляемыми провайдерами OLE DB и SQL, объектом DataSet и связанными с ним типами. Однако, конечно, в ADO.NET остается еще немало интересных вещей, таких как работа с транзакциями, вопросы безопасности при подключении к базе данных и т. п., что может быть рекомендовано в качестве материала для самостоятельного освоения.

Единственный момент, о котором хотелось бы еще упомянуть, – это то, что в Microsoft Visual Studio помещены специальные мастера для работы с созданием баз данных и подключением к источникам данных, которые во многих ситуациях позволяют сэкономить некоторое время без загрузки дополнительных средств, таких как Microsoft SQL Server Management Studio. Например, если воспользоваться закладкой Server Explorer и соответствующими мастерами, то можно создать новую базу данных Microsoft SQL Server или настроить соединение с существующей с получением строки подключения для типов SqlConnection и OleDbConnection. При перетаскивании элемента управления DataSet с панели Toolbox на форму получаются возможности для быстрого создания таблиц и их связей. Также при помощи мастеров можно создавать команды SELECT, INSERT, DELETE и UPDATE и многие другие вещи. После пройденного материала разобраться с тем, что создают эти мастера, не составит особого труда.

Таким образом, тип SqlDataAdapter, аналогично типу OleDbDataAdapter, позволяет вносить изменения в базу данных на источнике данных под управлением Microsoft SQL Server. В ADO.NET предусмотрено еще множество замечательных возможностей, но и с теми, которые были рассмотрены, уже можно создавать вполне работоспособные приложения, обращающиеся к базам данных [1, 5–7].

3. ПРИНЦИПЫ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ С ПРИМЕНЕНИЕМ ТЕХНОЛОГИИ ASP.NET

3.1. Понятия о технологии ASP.NET, Web-приложениях и Web-серверах. Назначение виртуальных каталогов. Структура документа HTML. Форматирование текста. Заголовки. Разработка форм и создание пользовательского интерфейса

Понятия о технологии ASP.NET, Web-приложениях и Web-серверах. ASP.NET – это технология создания веб-приложений и веб-сервисов от компании Microsoft. Она является составной частью платформы Microsoft.NET и развитием более старой технологии ASP.

ASP.NET берет свое название от более старой технологии ASP и внешне во многом сохраняет схожесть с ней, что позволяет разработчикам относительно легко перейти на ASP.NET. В то же время внутреннее устройство и принципы ASP.NET существенно отличаются от ASP, поскольку она основана на платформе .NET и, следовательно, использует все новые возможности, предоставляемые этой платформой.

Microsoft полностью перестроила ASP.NET, основываясь на Common Language Runtime (CLR), которая является основой всех приложений .NET. Программисты при создании приложений на ASP.NET могут писать код, используя различные «настоящие» языки программирования, поддерживаемые в .NET Framework, как коммерческие (Visual Basic, Visual C#, Visual C++, Visual J# и др.), так и «открытые» (Python, Perl и др.), а не только интерпретируемые языки скриптов.

Как станет очевидным, ASP.NET предлагает гораздо более надежную модель создания Web-приложений, нежели классическая ASP. Например, можно разделить логику представления на HTML и бизнес-логику при помощи техники, называемой Codebehind (фоновый код).

До настоящего момента все рассмотренные приложения были консольными или Windows-приложениями (приложениями Windows Forms). Сначала следует познакомиться также с новым типом приложений – Web-приложениями, для доступа к которым клиентам нужен лишь браузер. Целесообразно рассмотреть главные структурные элементы, без которых не обходится ни одно Web-приложение – HTML, запросы HTTP (POST и GET), скрипты, выполняемые в браузере клиента, на примере JavaScript, а также страницы классической технологии ASP (активные серверные страницы).

В этой связи не очень строгие определения следующие: Web-приложение – это набор взаимосвязанных файлов (HTM (HTML), ASP, ASPX, файлов изображений и т. п.), а также связанных с ними компонентов (двоичных файлов .NET или классической COM), которые размещены на Web-сервере. Web-сервер – это программный продукт, на котором размещаются Web-приложения и который обычно обеспечивает набор связанных с Web-приложениями служб, таких как интегрированные средства обеспечения безопасности, поддержка протокола FTP, поддержка средств передачи электронной почты т. п. Web-сервер (Web-сервисы) от Microsoft называется Internet Information Services (IIS).

При создании Web-приложений с использованием классических ASP или ASP.NET обязательно придется – прямо или опосредованно – работать с IIS. Обычно по умолчанию в операционных системах IIS не установлен, поэтому его следует установить стандартными средствами Microsoft Windows.

После того как установка IIS будет завершена, проще всего управлять им из консоли управления MMC. Оригинальные виртуальные Web-узлы использовать не обязательно и можно ограничиться лишь Web-узлом по умолчанию (рис. 3.1).

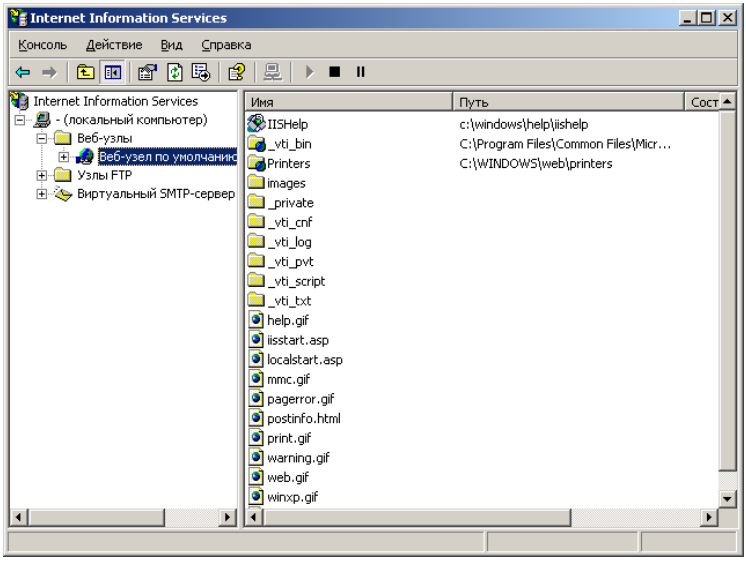


Рис. 3.1. Администрирование IIS

Затем следует рассмотреть вопросы, связанные непосредственно с применением технологии ASP.NET. Целесообразно ознакомление с архитектурой Web-приложения, с важнейшим типом Page и со свойствами, пришедшими из классической ASP, такими как Request, Response, Session и Application.

В конце следует рассмотреть серверные элементы управления WebForm и события сервера. Одна из главных задач в этой связи – подготовиться к созданию Web-служб ASP.NET, которые будут рассмотрены далее.

Таким обоазом, перед тем как освоить среду ASP.NET, необходимо рассмотреть основы архитектуры Web-приложений и некоторые базовые Web-технологии [1, 7–10].

Назначение виртуальных каталогов. На одном IIS может находиться множество Web-приложений. Каждое из этих Web-приложений должно размещаться в своем виртуальном каталоге (Virtual directory). Виртуальному каталогу на Web-сервере соответствует физический каталог на диске. Предполагается, что создано Web-приложение Web-prilozhenie. Например, если зарегистрировано доменное имя в системе DNS с именем домена COM, то удаленно к этому приложению можно будет обратиться по адресу URL http://www.Web-prilozhenie.com, а на компьютере этому приложению будет соответствовать физический каталог, например C:\Web-sayt. И именно в этом физическом каталоге будут находиться файлы, из которых состоит указанное Web-приложение.

При создании Web-приложения первое, что потребуется сделать, – создать на компьютере новый каталог, в котором будут храниться файлы Web-приложения. Следующее, что надо будет сделать, – создать на Web-сервере новый виртуальный каталог, которому будет соответствовать этот физический каталог. Сделать это можно разными способами, но самый простой – в древовидном списке в окне IIS выбрать элемент (контейнер) «Веб-узел по умолчанию» и через контекстное меню создать виртуальный каталог (рис. 3.2).

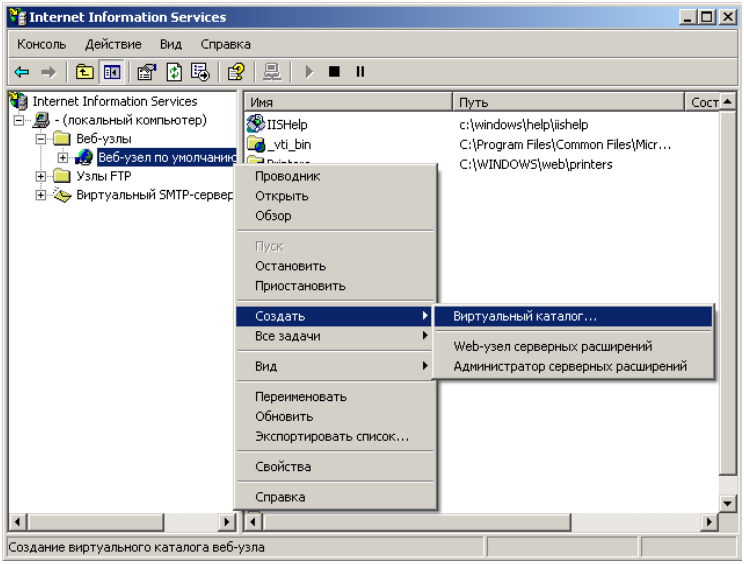


Рис. 3.2. Создание виртуального каталога

В результате запустится мастер создания виртуальных каталогов, который запросит псевдоним (краткое имя для ссылок), предназначенный для получения доступа к виртуальному каталогу. При этом рекомендуется использовать те же соглашения об именах, что и для физических каталогов. Далее для этого виртуального каталога следует ввести физический путь в операционной системе к каталогу с содержимым, публикуемым на Web-узле. Далее мастер предложит настроить параметры прав доступа (разрешений) к создаваемому виртуальному каталогу – возможность чтения, запуска сценариев (например, ASP, скриптов и т. д.), выполнения (например, приложений ISAPI и CGI и других исполняемых файлов), записи и обзора списка файлов из Web-браузера. В большинстве случаев вполне подойдут значения, предлагаемые мастером по умолчанию (если потребуется что-либо изменить, это несложно будет сделать через свойства виртуального каталога). После того как все эти действия будут завершены, можно увидеть созданный виртуальный каталог в списке каталогов элемента «Веб-узел по умолчанию» на сервере IIS.

Следует отметить, что в Microsoft Visual Studio при разработке Web-приложения ASP.NET можно обойтись без виртуального каталога, созданного IIS (работа не по протоколу HTTP). Во время компиляции приложения на локальном компьютере автоматически назначается порт и создается временный виртуальный каталог с названием, совпадающим с названием

каталога нижнего уровня, в котором находится приложение. Эти задачи в среде разработки решает ASP.NET Development Server.

Также следует обратить внимание, что если Microsoft .NET Framework установлена ранее, чем IIS, то необходимо переустановить текущую версию ASP.NET, обновив сопоставления (соответствия) обработчиков сценариев (скриптов) в метабазе IIS. Для этого при наличии версии ASP.NET, не более ранней, чем 2.0, можно воспользоваться утилитой (служебной программой) aspnet_regiis.exe, позволяющей администрировать ASP.NET на локальном компьютере и находящейся в том же подкаталоге, в котором размещены встроенные сборки .NET.

Запуск этой утилиты с опцией регистрации ASP.NET «-i» позволяет установить текущую версию ASP.NET и обновить сопоставления обработчиков сценариев в корне метабазы IIS и вложенных разделах (уровнях). При этом существующие сопоставления обработчиков сценариев более ранних версий обновляются до этой версии (рис. 3.3) [1, 7–10].

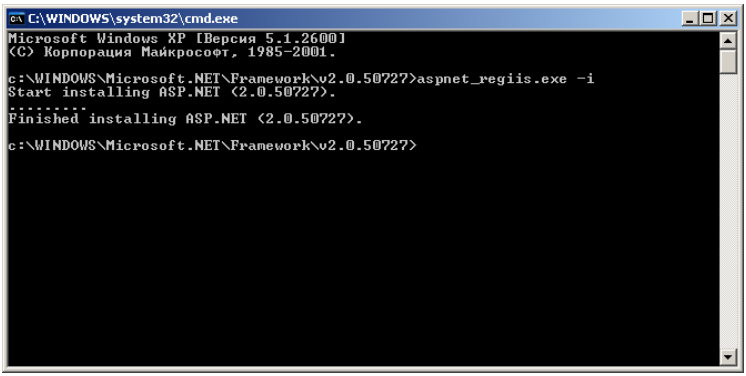


Рис. 3.3. Установка текущей версии ASP.NET

Структура документа HTML. Теперь, когда создан виртуальный каталог, можно приступать к созданию самого Web-приложения. При создании Web-приложений не обойтись без страниц на языке HTML. HTML – это стандартный язык гипертекстовой разметки, используемый для описания того, как текст, изображения, гиперссылки и стандартные элементы графического интерфейса будут отображаться в Web-браузере. Большинство современных сред разработки Web-приложений (в том числе Microsoft Visual Studio) позволяют создавать Web-страницы, почти не обращаясь непосредственно к самому коду HTML посредством специальных интегрированных средств разработки интерфейса, однако, тем не менее, разработчик Web-приложений должен, безусловно, знать этот язык.

Документ HTML обычно начинается с набора тегов, в которых содержится общая информация о документе (заголовок, метаданные файла и т. п.), за которой следует само тело документа (т. е. набор текста, изображений, таблиц, гиперссылок и т. д.). Теги HTML не чувствительны к регистру.

Разработку документа HTML можно начать с загрузки интегрированной среды разработки Microsoft Visual Studio и создания пустого шаблона Empty Web Site (рис. 3.4).

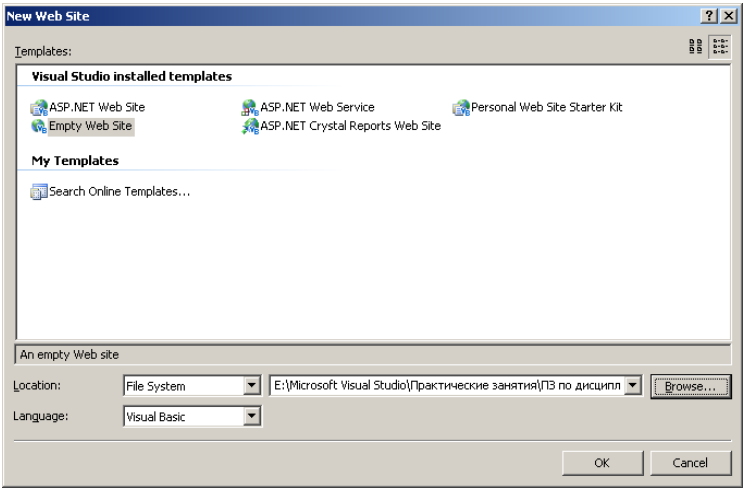


Рис. 3.4. Формирование пустого шаблона Empty Web Site

После этого необходимо добавить в созданный проект шаблон HTML Page, содержащий страницу HTML, которая будет включать код клиентской стороны (рис. 3.5).

Альтернативный способ указанных предварительных операций по разработке документа HTML в составе проекта и решения заключается в формировании страницы HTML вне проекта (рис. 3.6).

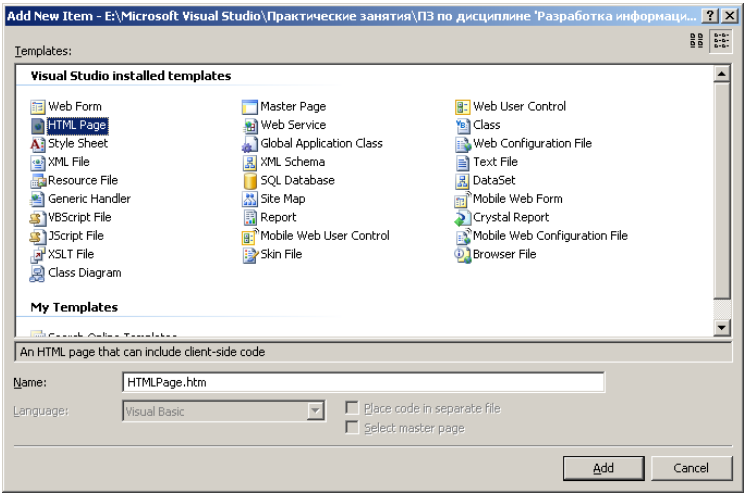


Рис. 3.5. Добавление шаблона HTML Page

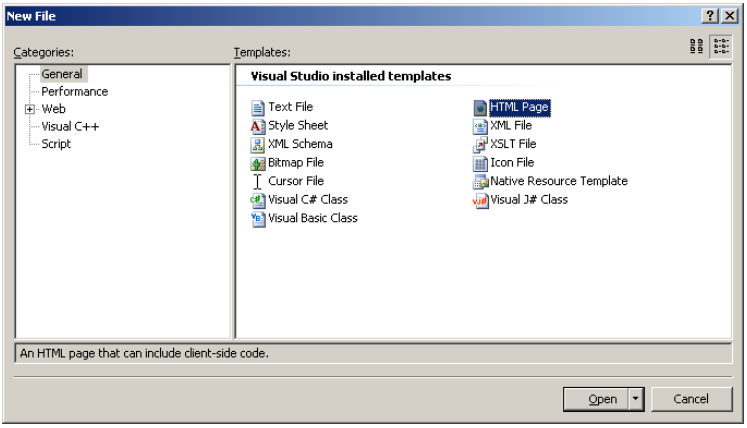


Рис. 3.6. Формирование шаблона HTML Page

Затем эту страницу можно сохранить в физическом каталоге проекта или добавить в проект. Как до создания страницы, так и после ей может быть присвоено любое имя в соответствии с правилами для символов имен файлов.

После указанных выше действий Microsoft Visual Studio автоматически добавит в созданный файл HTML следующие теги шаблона:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
        <title>Untitled Page</title>
</head>
<body>

</body>
</html>
```

Если название тега обозначить «X», то открывающий тег HTML выглядит как <X>, а закрывающий – как </X>, хотя существует множество тегов, которые закрывать не надо. Теги <html> и </html> помечают начало и конец документа HTML соответственно. Теги <head> и </head> выделяют метаданные для всего документа. Например, внутрь блока этих тегов можно поместить теги, которые описывают программу, использованную для создания документа, и содержимое файла. Здесь также может быть сформировано название страницы посредством тегов <title> и </title>:

```
<title>
        МГЭУ им. А. Д. Сахарова
</title>
```

Это название выводится как заголовок окна браузера, в котором открыт документ (рис. 3.7).

Само содержание документа HTML помещается между тегами <body> и </body>. Как правило, между этими тегами помещается множество дополнительных тегов, которые используются для представления и форматирования текстовой и графической информации. Все теги HTML, конечно, рассматривать не стоит, но самым необходимым и наиболее часто встречающимся и употребляемым следует уделить некоторое внимание [1, 7–10].

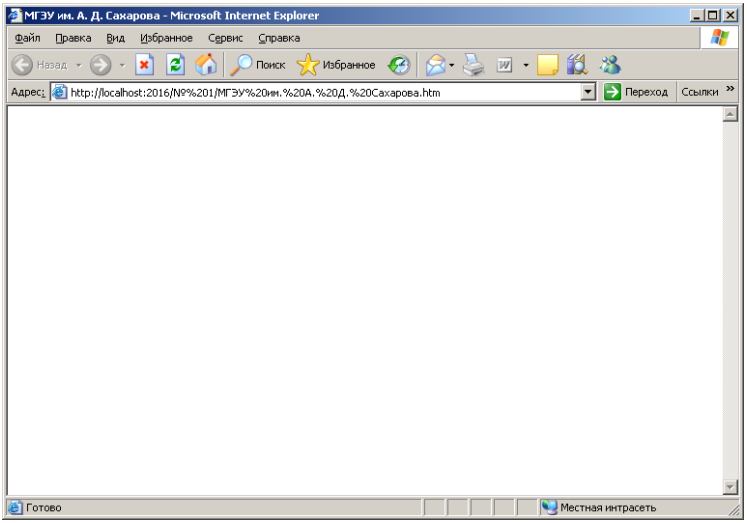


Рис. 3.7. Применение тега <title>

Форматирование текста. Исходное назначение HTML заключалось в представлении текстовой информации. Как уже упоминалось, текст документа в HTML обычно помещается между тегами <body> и </body>. Например, предполагается, что создается страница аутентификации пользователей. Текст HTML на ней может выглядеть следующим образом (надо обратить также внимание на синтаксис комментариев HTML):

```
<body>
        <!-- Приглашение пользователю к аутентификации -->
        МГЭУ им. А. Д. Сахарова
</body>
```

В этом примере к тексту не были применены какие-либо теги. Встречаясь с таким текстом, Web-браузер выводит его в своем окне так, как он был записан. Если изменить текст документа следующим образом:

```
<body>
        <!-- Приглашение пользователю к аутентификации -->
        МГЭУ им. А. Д. Сахарова
        Пожалуйста, введите свои имя и пароль!
</body>
```

то браузер не добавит ожидаемый переход на новую строку, т. е. для текстовой информации без тегов символы начала новых строк учитываться не будут (рис. 3.8).

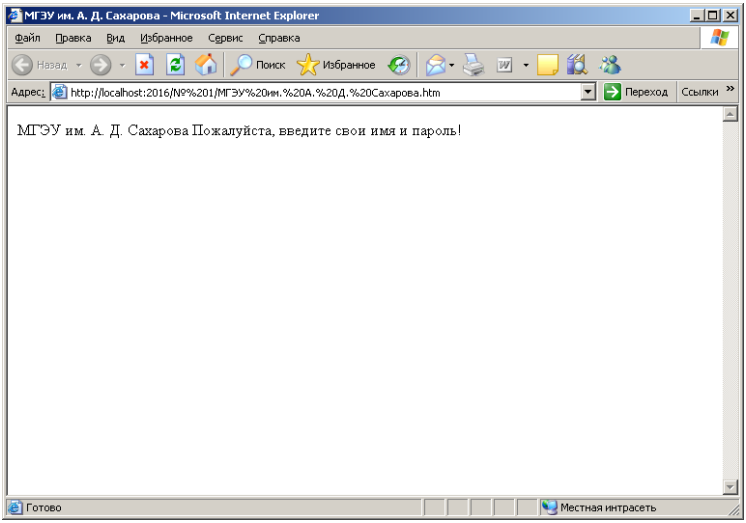


Рис. 3.8. Отсутствие применения тегов для перехода на новые строки

Чтобы начать новый абзац, необходимо выделить текст в этом абзаце при помощи тегов и , например, так:

```
<body>
        <!-- Приглашение пользователю к аутентификации -->
        МГЭУ им. А. Д. Сахарова
        <p>
                Пожалуйста, введите свои имя и пароль!
        </p>
</body>
```

Теперь в окне браузера все выглядит по-другому (рис. 3.9).

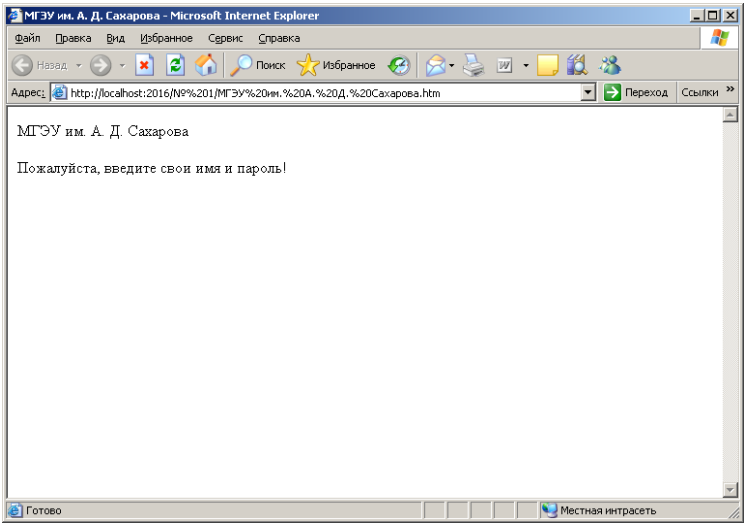


Рис. 3.9. Применение тега <p>

Можно и не начинать новый абзац, а просто добавить тег начала новой строки
:

```
<body>
        <!-- Приглашение пользователю к аутентификации -->
        МГЭУ им. А. Д. Сахарова
        <br />
        Пожалуйста, введите свои имя и пароль!
</body>
```

В этом случае браузер отобразит текст несколько иначе (рис. 3.10).

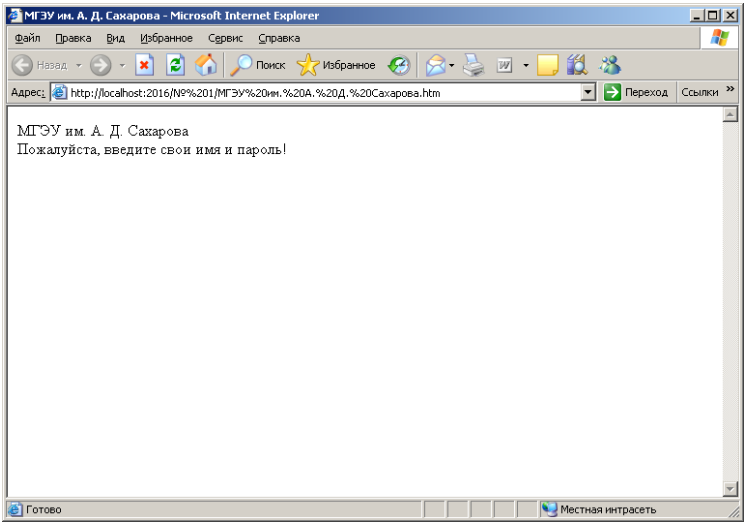


Рис. 3.10. Применение тега

В HTML предусмотрены средства для выделения участков текста полужирным и курсивным начертаниями. Для этого предусмотрены теги и и и соответственно:

```
<body>
        <!-- Приглашение пользователю к аутентификации -->
        <b>
                МГЭУ им. А. Д. Сахарова
        </b>
        <br />
        <i>
                Пожалуйста, введите свои имя и пароль!
        </i>
</body>
```

Результат представлен на рис. 3.11 [1, 7–10].

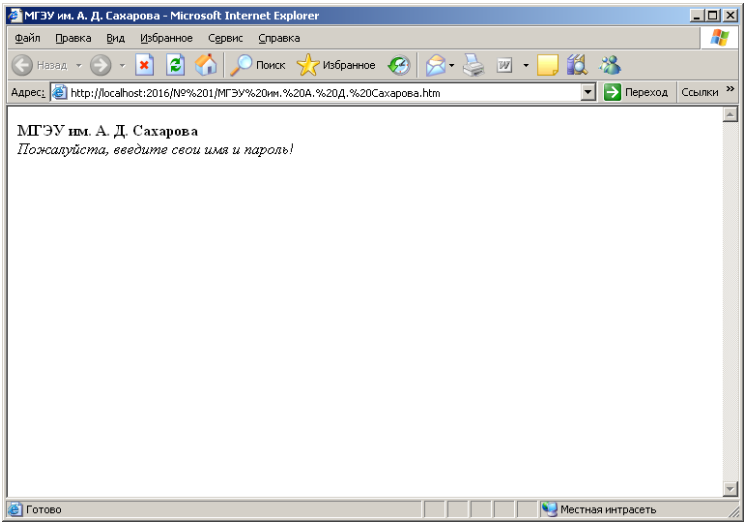


Рис. 3.11. Применение тегов <b> и <i>

Заголовки. Последний вид тегов для форматирования текста, который следует рассмотреть, – это теги заголовков HTML. Они выглядят как <h1>, <h2>, <h3>, <h4>, <h5> и <h6> (закрывающие теги в дальнейшем будут опускаться) и применяются для изменения размера выделенного ими текста. Тег
 при этом не обязателен, так как закрывающие теги заголовков переводят текст на новую строку. Наибольший относительный размер текста обеспечивает тег <h1> (заголовок первого уровня), например:

```
<body>
        <!-- Приглашение пользователю к аутентификации -->
        <h1>
                МГЭУ им. А. Д. Сахарова
        </h1>
        <br />
        <i>
                Пожалуйста, введите свои имя и пароль!
        </i>
</body>
```

Для того чтобы блок текста был выровнен посередине страницы, можно использовать тег :

```
<body>
        <!-- Приглашение пользователю к аутентификации -->
        <center>
                <h1>
                        МГЭУ им. А. Д. Сахарова
                </h1>
                <i>
                        Пожалуйста, введите свои имя и пароль!
                </i>
        </center>
</body>
```

То, как теперь выглядит страница, показано на рис. 3.12.

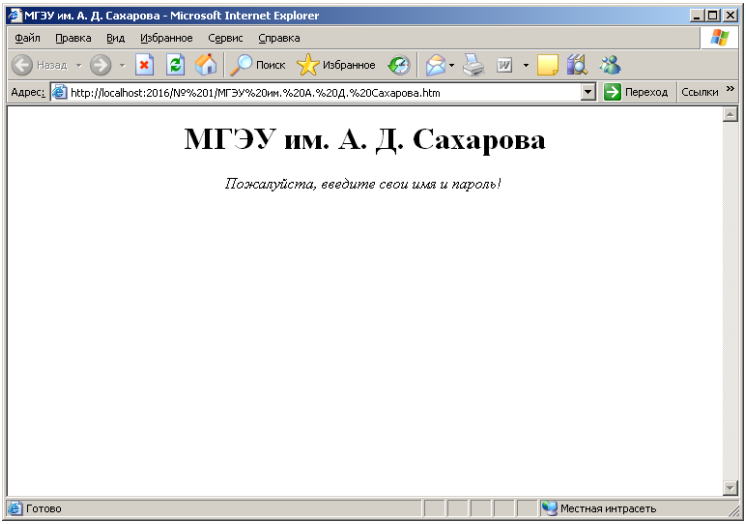


Рис. 3.12. Применение тегов <h1> и

В Microsoft Visual Studio при разработке Web-страниц предусмотрены специальные встроенные средства для редактирования HTML-страниц. С их помощью можно добавлять дополнительные теги HTML при форматировании и настройке свойств различных элементов.

Графические средства, которые применяются для управления отображением всего документа, находятся в свойствах объекта Document (рис. 3.13).

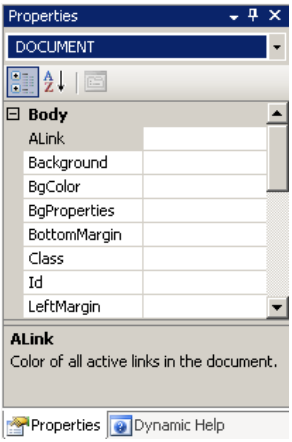


Рис. 3.13. Редактирование документа HTML при помощи графических средств Microsoft Visual Studio

Например, если при помощи указанных средств изменить значение свойства BgColor (Background color), определяющего цвет фона, то в коде документа HTML автоматически появится новый тег:

```
<body bgcolor="#006600">
        <!-- Приглашение пользователю к аутентификации -->
        <center>
                <h1>
                        МГЭУ им. А. Д. Сахарова
                </h1>
                <i>
                        Пожалуйста, введите свои имя и пароль!
                </i>
        </center>
</body>
```

В Microsoft Visual Studio также предусмотрена панель форматирования HTML. С ее помощью можно управлять представлением блоков текста, выбирая для них уровень заголовка, разметку списков, шрифт и его атрибуты и т. п. (рис. 3.14).

Рис. 3.14. Панель форматирования HTML

Таким образом, при помощи графических средств Microsoft Visual Studio можно оформлять гипертекстовые страницы – весь необходимый для этого код HTML будет сгенерирован автоматически. Эти средства позволяют сэкономить много времени, однако Web-разработчику часто приходится создавать весь код для страницы HTML вручную [1, 7–10].

Разработка форм и создание пользовательского интерфейса. После внешнего оформления страницы целесообразно наделить ее новыми свойствами – возможностью принимать ввод пользователя. Для этого придется прибегнуть к помощи элементов управления HTML. Как станет очевидным далее, в среде ASP.NET предусмотрен набор элементов управления WebForm, при применении которых все необходимые теги для элементов управления HTML будут генерироваться автоматически. Однако знакомство с тегами для создания элементов управления HTML также будет нелишним. Еще раз следует подчеркнуть, что элементы управления WebForm в ASP.NET и элементы управления HTML – это разные вещи, и в процессе выполнения Web-приложения первые преобразуются во вторые.

Форма HTML – это именованная группа элементов пользовательского интерфейса HTML, используемых для ввода пользователем данных. Затем эти данные передаются на Web-сервер по протоколу HTTP (подробнее об этом – немного позже). Теги для элементов пользовательского интерфейса на форме HTML помещаются между тегами <form> и </form>:

```
<form name="MainForm">
</form>
```

В этом коде создана форма и присвоено ей дружественное имя. Также можно указать идентификатор формы через свойство Id с передачей в него строки, например id="MainForm", так как не все браузеры поддерживают свойство Name. С технической точки зрения использовать имя и (или) идентификатор в принципе не обязательно, однако во многих ситуациях это очень удобно.

Как правило, в открывающий тег <form> помещается атрибут для действия, выполняемого этой формой. В нем содержится информация об адресе URL, на который будут передаваться данные, введенные пользователем, а также сведения о методе передачи данных (POST или GET). Вскоре эти моменты будут рассмотрены достаточно подробно, а до этого следует рассмотреть те элементы, которые могут быть помещены внутрь формы HTML. В Microsoft Visual Studio предусмотрена специальная панель Toolbox, в которой можно выбрать эти элементы (рис. 3.15).

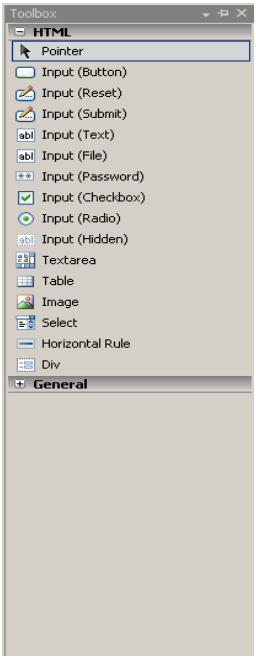


Рис. 3.15. Набор элементов управления HTML

Краткий перечень наиболее часто используемых элементов представлен в табл. 3.1.

Таблица 3.1

Элементы управления HTML

Элемент управления	**Описание**
Input (Button)	Эта разновидность кнопки обычно используется для того, чтобы выполнить отрезок кода клиентского скрипта
Input (Reset) Input (Submit)	Специальные кнопки на форме, при нажатии на которые все значения в форме принимают свой исходный вид (состояние) или производится отправка данных формы на Web-сервер соответственно
Input (Text) Textarea	Эти элементы управления предназначены для ввода пользователем одной строки текста или нескольких строк
Input (Password)	Специальное текстовое поле, предназначенное для ввода пользователем пароля. Все введенные данные в это поле отображаются одинаковыми символами

Продолжение таблицы 3.1

Элемент управления	Описание
Input (Checkbox) Input (Radio)	То же самое, что и аналогичные элементы управления Windows Forms, предназначенные для выбора или переключения соответственно
Table	Элемент управления для формирования таблицы
Image	Позволяет указать изображение, которое будет выведено на форме

В библиотеке базовых классов .NET предусмотрен набор типов .NET, которые соответствуют элементам управления HTML. Они определены в пространстве имен System.Web.UI.HtmlControls.

При создании пользовательского интерфейса первое, что нужно сделать, чтобы страница могла воспринимать ввод пользователя, – создать на ней форму HTML. Для этого следует поместить на страницу следующий код:

```
<body bgcolor="#006600">
        <!-- Приглашение пользователю к аутентификации -->
        <center>
                <h1>
                        МГЭУ им. А. Д. Сахарова
                </h1>
                <i>
                        Пожалуйста, введите свои имя и пароль!
                </i>

                <form name="MainForm">
                </form>

        </center>
</body>
```

Форма создана, теперь можно приступать к добавлению в нее элементов управления. Это можно сделать при помощи графических средств Microsoft Visual Studio, а можно создать все необходимые теги вручную. Каждый элемент управления описывается атрибутом имени (имя используется при выполнении программы, чтобы определить, например, в какой элемент управления были введены данные) и атрибутом типа (этот атрибут и определяет разновидность элемента управления). Для разных элементов управления существуют разные наборы дополнительных атрибутов, которые мо-

гут быть использованы для определения различных их параметров. Конечно же, эти дополнительные параметры можно также настроить при помощи окна свойств для соответствующего элемента управления в Microsoft Visual Studio.

Предполагается, что форма будет содержать два текстовых поля (одно – для ввода имени пользователя, другое, специальное, – для ввода пароля) и две кнопки – для передачи информации на сервер и для восстановления формы в исходном состоянии, если пользователь решает отменить свой ввод. Код HTML для формы может выглядеть следующим образом:

```
<form name="MainForm">
        <p>
                Имя:
                <input name="txtUserName" type="text" />
        </p>
        <p>
                Пароль:
                <input name="txtPassword" type="password" />
        </p>
        <input name="btnSubmit" type="submit" value="Подтверждение" />
        <input name="btnReset" type="reset" value="Сброс" />
</form>
```

Для каждого элемента управления определены уникальное имя (TxtUserName, TxtPassword, BtnSubmit и BtnReset) и тип (Text, Password, Submit и Reset) соответственно. Кроме того, для каждой кнопки определено очень важное свойство Value (значение): value="Сброс" означает, что все элементы управления на форме вернутся в исходное состояние, а value="Подтверждение" – что данные, введенные пользователем, отправятся получателю. Также эта информация необходима для вывода на экран названий кнопок.

Свойство Value можно применять не только для кнопок. Например, можно определить его для текстового поля TxtUserName. Например, если определить для TxtUserName свойство value="Студент", то это значит, что слово «Студент» станет значением по умолчанию для этого текстового поля, и оно будет помещаться в поле всякий раз при загрузке формы (рис. 3.16).

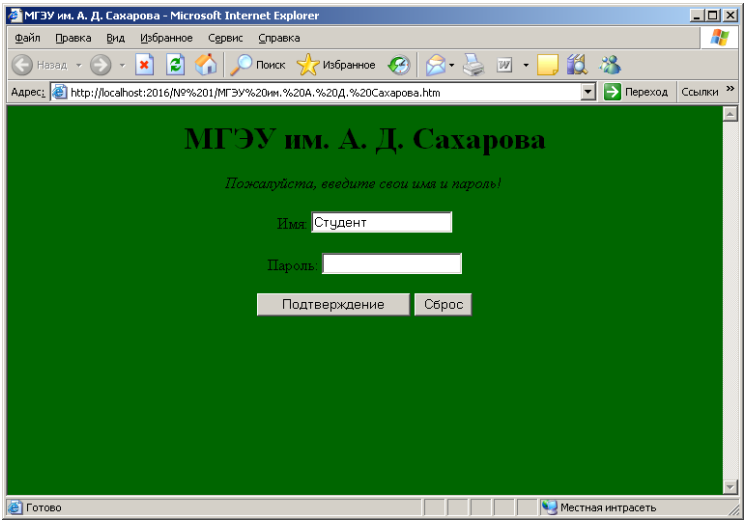


Рис. 3.16. Применение значения по умолчанию для текстового поля

Еще один вопрос, посвященный возможностям HTML, который следует рассмотреть, – это добавление на Web-страницу изображений. Изображения добавляются при помощи тега <img />:

```
<img alt="Корпус № 1" src="Корпус № 1.jpg" border="4" />
```

Свойство Alt (Alternative) используется для определения текстового альтернативного эквивалента изображения. Этот текст «всплывет», если поместить указатель мыши над изображением, или будет выведен вместо самого изображения, если браузер не поддерживает вывод графики или файл изображения не найден. В свойстве Src (Source) можно использовать как полный путь к графическому файлу (источнику), так и относительный, при котором подразумевается, что файл изображения будет находиться в одном каталоге с файлом HTM (HTML). Необязательное свойство Border определяет толщину рамки вокруг изображения. Доработанный код страницы может выглядеть следующим образом:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

        <head>
```

```
		<title>
			МГЭУ им. А. Д. Сахарова
		</title>
	</head>

	<body bgcolor="#006600">
		<center>
			<h1>
				<span style="color: #ccff66">
					МГЭУ им. А. Д. Сахарова
				</span>
			</h1>
			<img alt="Корпус № 1" src="Корпус № 1.jpg"
border="4"
			width="380" />
			<br />
			<h3>
				<span style="color: #ff3300">
					Пожалуйста, введите свои имя и
пароль!
				</span>
			</h3>

			<form name="MainForm">
				<p>
					<span style="color: #99ffff">
						<strong>
							   
 Имя:
						</strong>
					</span>
					<input id="txtUserName" type=text
style="width:
					139px" />
				</p>
				<p>
					<span style="color: #99ffff">
						<strong>
							Пароль:
						</strong>
					</span>
					<input name="txtPassword"
type="password" />
				</p>
```

```
                                        <input id="btnSubmit"
name="btnSubmit"
                                        type="submit"
value="Подтверждение" />
                                          
                                        <input name="btnReset" type="reset"
value="Сброс"
                                        />
                        </form>

                </center>
        </body>

</html>
```

Здесь тег определяет контейнер (объект) для назначения различных атрибутов и свойств линейного текста, в частности стиля шрифта Style, который в свою очередь определяется строкой, содержащей информацию о его цвете Color в шестнадцатеричном формате данных. Свойство Width задает ширину изображения в пикселях (в данном случае пропорционально высоте, так как отсутствует свойство Height). Также существует альтернативное указание ширины элементов управления через свойство Style в виде строки, как это реализовано для текстового поля. « » определяет вставку символа неразрывного пробела (пустого пространства). Страница после обновления должна выглядеть так, как показано на рис. 3.17 [1, 7–10].

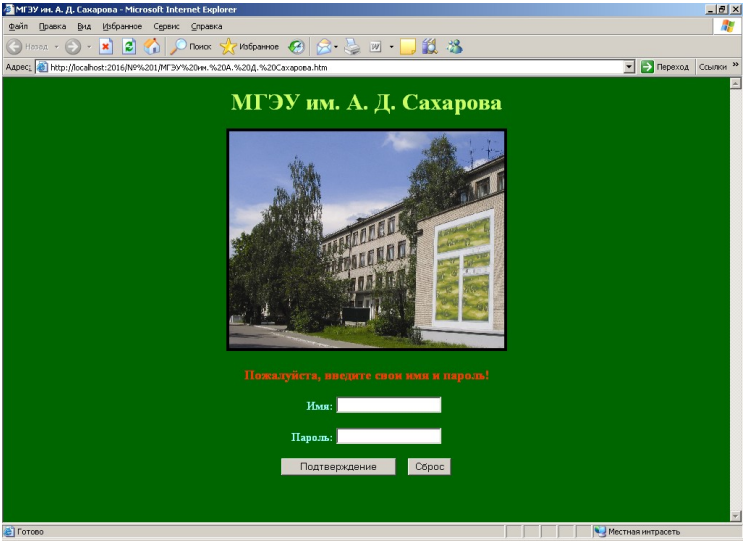


Рис. 3.17. Web-страница после добавления изображения

3.2. Клиентские скрипты. Реализация проверки введенных пользователем данных. Методы GET и POST для передачи данных формы. Синтаксис строки запроса HTTP. Проблемы, преимущества и отличия классической ASP от ASP.NET. Пространства имен ASP.NET. Некоторые типы пространства имен System.Web. Понятия о Web-приложении и сеансе подключения пользователя

Клиентские скрипты. Одной из больших проблем для множества Web-приложений является необходимость вновь обращаться с повторными запросами на Web-сервер для внесения изменений в то, что показывается пользователю в окне браузера. Конечно, во многих случаях такие обращения неизбежны, но если есть возможность сократить их количество, то этой возможностью надо пользоваться. Один из способов сократить количество запросов на Web-сервер заключается в применении клиентских (браузерных) скриптов, например для проверки введенных пользователем данных перед передачей этих данных на сервер.

Например, в рассмотренной выше ситуации пользователю необходимо ввести его имя и пароль для аутентификации. Если какое-либо из полей

останется незаполненным, аутентификация все равно не произойдет. Поэтому вполне можно сделать так, чтобы пользователь не мог отправить данные на сервер, не заполнив оба поля. Конечно, в этой ситуации только кодом HTML не обойтись: HTML – это язык разметки, а не программирования. Для того чтобы реализовать проверку введенных пользователем данных, придется использовать какой-либо из языков для работы со скриптами.

Существует множество языков для работы со скриптами, но для скриптов, выполняющихся в браузерах, подойдут далеко не все. Например, Microsoft Internet Explorer поддерживает два языка: VBScript (диалект Visual Basic для работы со скриптами) и JavaScript, а Netscape Navigator – только JavaScript. Если можно гарантировать, что клиенты приложения будут использовать только Microsoft Internet Explorer, можно использовать любой из языков для работы со скриптами. Но на обычных Web-сайтах, открытых для доступа самых разных клиентов, как правило, используется только один язык для браузерных скриптов – JavaScript.

JavaScript – очень популярный язык скриптов, который де-факто является стандартом для создания браузерных скриптов. Сразу следует заметить, что JavaScript никоим образом не является частью языка Java. JavaScript – это совершенно отдельный язык программирования, который предназначен для решения специальных задач и в котором предусмотрено гораздо меньше возможностей, чем в Java. JScript – это название реализации JavaScript от Microsoft.

Как правило, клиентские скрипты выполняются в ответ на события графических элементов управления HTML. Перехват таких событий происходит в соответствии с определенным принципом. Проще всего показать это на примере клиентского скрипта.

Предполагается, что работа производится с созданной ранее простой Web-страницей, представленной на рис. 3.17. Чтобы настроить перехват события, возникающего при нажатии, например, кнопки «Подтверждение», следует перейти в режим просмотра исходного текста и выбрать эту кнопку в левом ниспадающем списке. Затем в правом списке выбрать для этой кнопки событие onclick (рис. 3.18).

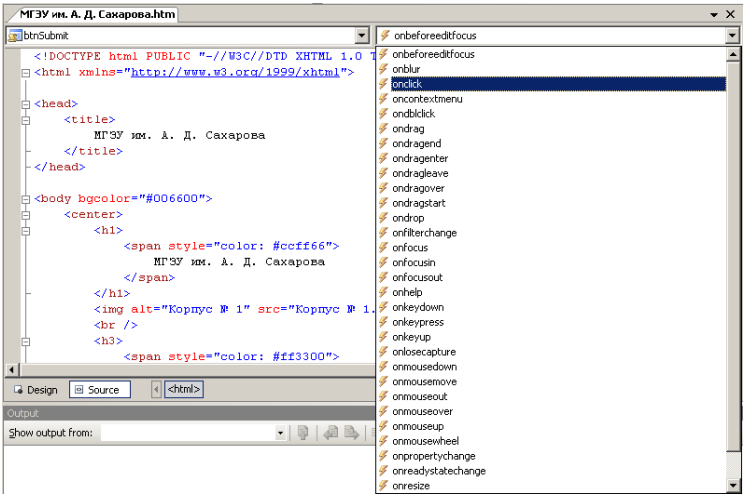


Рис. 3.18. Перехват событий элементов управления HTML
в режиме просмотра исходного текста

Возможен также другой вариант инициализации функции перехвата события – двойной щелчок на кнопке в режиме просмотра формы (рис. 3.19).

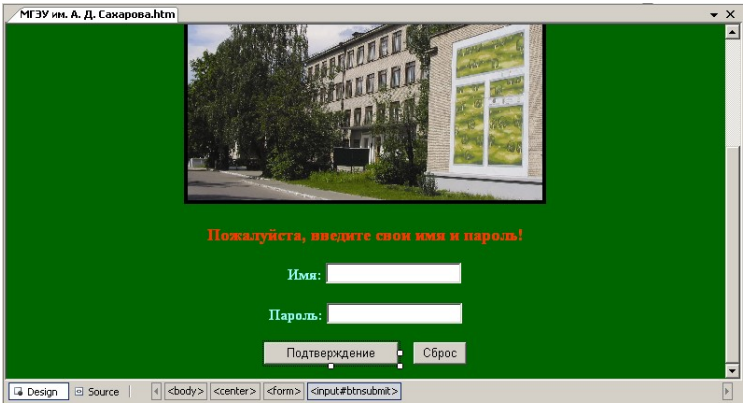


Рис. 3.19. Перехват событий элементов управления HTML в режиме просмотра формы

Выполнив эти действия, можно обнаружить в коде HTML следующие изменения:

```
<script language="javascript" type="text/javascript">
<!--

function btnSubmit_onclick() {

}

// -->
</script>

<input id="btnSubmit" name="btnSubmit" type="submit"
value="Подтверждение"
language="javascript" onclick="return btnSubmit_onclick()" />
```

Как видно, на странице в разделе <head> появился блок <script>, для которого в качестве используемого языка указан JavaScript. Следует обратить внимание, что сам код скрипта помещен в блок комментария HTML. Причина состоит в том, что если страница будет открыта в браузере, который не поддерживает JavaScript, то этот код будет воспринят как комментарий и проигнорирован.

Следует также обратить внимание, что в теге для кнопки появились новые свойства: Language и Onclick, который ссылается на метод (функцию) JavaScript. В результате при нажатии на эту кнопку будет вызван этот метод [1, 7–10].

Реализация проверки введенных пользователем данных. Часто необходимо проверять корректность ввода пользователя на страницах для исключения передачи неполноценных данных на сервер. Например, при нажатии на кнопку «Подтверждение» должен вызываться метод JavaScript, который будет проверять, не оставлено ли какое-либо из текстовых полей пустым. Если так оно и есть, пользователю будет выдаваться сообщение Microsoft Internet Explorer с информацией о допущенной им ошибке. Прежде всего потребуется определить для кнопки «Подтверждение» событие onclick. При возникновении этого события должен вызываться метод JavaScript btnSubmit_onclick(). Этот метод будет проводить проверку на отсутствие данных в текстовых полях. При этом необходимо использовать полные имена текстовых полей в формате «имя_формы.имя_поля». Содержание (код) метода для текущих целей может быть таким:

```
function btnSubmit_onclick()
{
        if((MainForm.txtUserName.value=="")||(MainForm.txtPassword.value==""))
        {
```

```
                alert("Вы должны ввести имя и пароль!");
                return false;
        }
        return true;
}
```

При нажатии на кнопку получится окно сообщения, представленное на рис. 3.20.

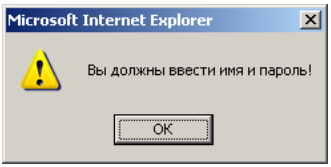


Рис. 3.20. Окно оповещения (Alert) Microsoft Internet Explorer

Таким образом, программно можно осуществлять проверку данных в форме. JavaScript обладает множеством различных возможностей. Для небольшого представления о других возможностях этого языка и браузерных скриптов в целом можно продемонстрировать возможность работы еще одной функции, которая будет вызываться при загрузке (обновлении) страницы и выводить информацию о дате и времени входа пользователя. Для этой функции (она будет называться Data_vremya()) потребуется еще один тег <script>. Следует обратить внимание на применение метода Write() объекта Document, представляющего текущий документ, загруженный в Microsoft Internet Explorer. Этот метод выступает как часть объектной модели Microsoft Internet Explorer:

```
function Data_vremya()
{
        return Date();
}

<span style="color: #ccff66">
        Сегодня:

        <script language="javascript" type="text/javascript">
                document.write(Data_vremya());
        </script>

</span>
```

После этой операции страница примет окончательный вид, представленный на рис. 3.21 [1, 7–10].

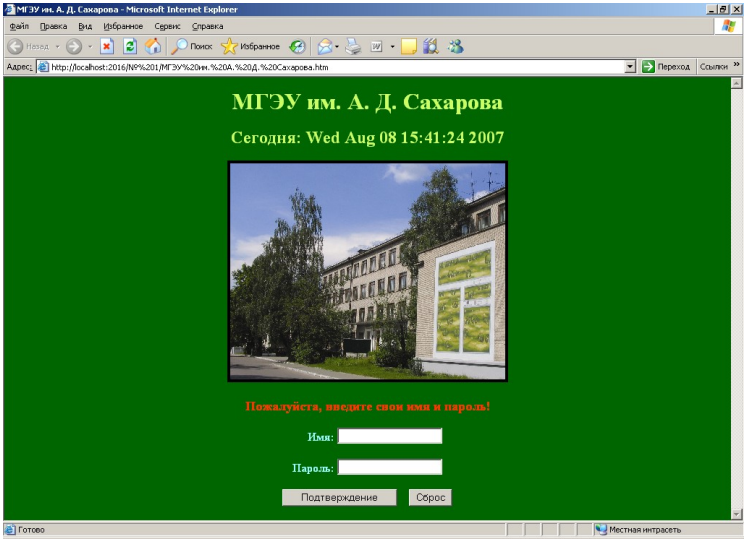


Рис. 3.21. Вывод информации о дате и времени входа пользователя

Методы GET и POST для передачи данных формы. Введенные и проверенные данные обычно необходимо отправить получателю – например, другому файлу HTML на Web-сервере, почтовому серверу, странице ASP т. п. Получатель данных формы указывается при помощи свойства Action, добавленного в ее тег:

```
<form name="MainForm" action="http://localhost/Web-prilozhenie/ASP.asp"
method="GET">
```

Такое значение атрибута Action означает, что при нажатии кнопки «Подтверждение» данные формы будут переданы странице ASP.asp, расположенной в определенном физическом каталоге, которому соответствует виртуальный каталог, на локальном компьютере. В качестве метода для передачи данных определен метод GET. Это значит, что данные формы будут добавлены к запросу в виде пар «имя-значение». Вместо метода GET можно использовать метод POST, в этом случае значение соответствующего атрибута должно выглядеть следующим образом:

```
<form name="MainForm" action="http://localhost/Web-prilozhenie/ASP.asp"
method="POST">
```

При использовании метода POST данные формы не будут добавляться к строке запроса. Вместо этого для них будет выделена отдельная строка в поле заголовка HTTP. При этом передаваемые данные не будут сразу же видны при отслеживании трафика, и поэтому метод POST можно считать немного более безопасным. Однако еще важнее, что при помощи метода POST можно передавать данные гораздо большего размера, чем при помощи GET. Тем не менее в примере будет использоваться более традиционный и простой метод GET [1, 7–10].

Синтаксис строки запроса HTTP. Файл ASP, которому передаются данные из формы, должен суметь извлечь эти данные из строки запроса. Сама строка запроса с данными формы выглядит как обычный адрес в адресной строке браузера с добавлением нескольких пар «имя-значение»:

```
http://localhost/Web-prilozhenie/ASP.asp?
txtUserName=%C8%EC%FF&
txtPassword=%CF%E0%F0%EE%EB%FC&
btnSubmit=%CF%EE%E4%F2%E2%E5%F0%E6%E4%E5%ED%E8%E5
```

Следует обратить внимание, что вся строка запроса разбивается на две части символом «?». Слева от этого знака находится адрес получателя данных, а справа – сами пары «имя-значение». Как можно убедиться, каждая пара «имя-значение» отделена от другой пары символом «&». Однако, к примеру, если требуется поместить внутрь какого-либо передаваемого значения пробел, то в строке запроса вместо пробелов подставляется символ «+».

В приведенном примере для совместимости символы пробелов в адресе и кириллицы в значениях представлены в специальном формате, при котором каждый символ начинается с символа «%», за которым следует шестнадцатеричное значение соответствующего символа ASCII. Это же касается различных служебных символов [1, 7–10].

Проблемы, преимущества и отличия классической ASP от ASP.NET. Форма уже в состоянии передавать данные, однако принимать их пока нечему. Этот недостаток Web-приложения ликвидируется созданием страницы-приемника данных. Вначале следует рассмотреть ее создание не на ASP.NET, а с помощью классической ASP.

Первое, с чего нужно начать, – добавить новую страницу ASP, присвоив ей то имя, которое указано в свойстве Action для формы

(ASP.asp), и сохранив ее в физическом каталоге, которому соответствует виртуальный каталог.

Страницу ASP можно воспринимать как набор из кода HTML и скриптов, предназначенных для выполнения на сервере. Можно отметить, что основной смысл ASP заключается в генерации кода HTML «на лету» при помощи серверных скриптов. Например, можно создать страницу ASP, которая будет считывать данные из источника данных (при помощи ADO) и представлять возвращаемые строки в виде кода HTML.

В примере страница ASP будет использовать встроенный объект Request для считывания значений из входящей строки запроса и выводить полученные от клиента данные в виде кода HTML. Клиенту будет возвращаться как бы эхо его запроса. Код соответствующего скрипта (следует обратить внимание на блоки <% … %>, в которые помещен скрипт) следующий:

```
<%@ Language=VBScript %>

<html>

        <head>
                <meta name="GENERATOR" content="Microsoft Visual
Studio">
        </head>

        <body>
                <center>
                                <h1>
                                        Вы ввели:
                                </h1>
                                <b>
                                        Имя:
                                </b>
                                <%

        =Request.QueryString("txtUserName")
                                %>
                                <br>
                                <b>
                                        Пароль:
                                </b>
                                <%
                                        =Request.QueryString("txtPassword")
```

```
                                %>
                </center>
        </body>

</html>
```

Первое, что необходимо отметить: на странице ASP используются те же теги <html>, <head> и <body>, что и на обычной Web-странице. Используется объект Request, который, как и положено объектам, поддерживает некоторое количество свойств, методов и событий. Для того чтобы извлечь данные в виде запроса от клиента, используется свойство QueryString().

Код HTML, который возвращается клиенту, генерируется посредством записей <% … %>, содержимое которых позволяет формировать информацию для HTTP-ответа. Кроме того, можно получить полный контроль над тем, что возвращается пользователю, при помощи объекта Response:

```
<%
        dim UserName
        UserName =Request.QueryString("txtUserName")
        Response.Write(UserName)
%>

<%
        dim Password
        Password =Request.QueryString("txtPassword")
        Response.Write(Password)
 %>
```

В данном случае с использованием метода Write() значения записываются в HTTP-ответ, и поэтому результат вывода будет такой же, как в предыдущем примере. При этом явное объявление переменных посредством оператора dim, выделяющего для переменной место в памяти и дающего возможность компилятору определить тип данных, который он может ожидать встретить в дальнейшем, можно опустить, используя неявное объявление. Однако в этом случае возможны проблемы с неверно написанными именами переменных, которые будут являться в коде источником ошибок и будут сложны в обнаружении в дальнейшем.

В типах Request и Response, конечно же, предусмотрено множество других очень полезных членов; кроме того, в распоряжении Web-разработчика классических ASP также находится набор дополнительных объектов, таких как Session, Server, Application и ObjectContext. Если по-

ставить цель разобрать классическую ASP во всех подробностях и рассмотреть эти объекты, то можно получить информацию о них в электронной документации по Microsoft Visual Studio. Однако следует отметить, что возможности этих типов реализованы в ASP.NET при помощи типа Page.

Для того чтобы запустить страницу ASP, надо просто открыть файл со страницей, ввести в текстовые поля значения для имени и пароля пользователя и нажать кнопку «Подтверждение». Сработает серверный скрипт ASP, и в окне браузера откроется сгенерированная на основе введенных данных страница (рис. 3.22).

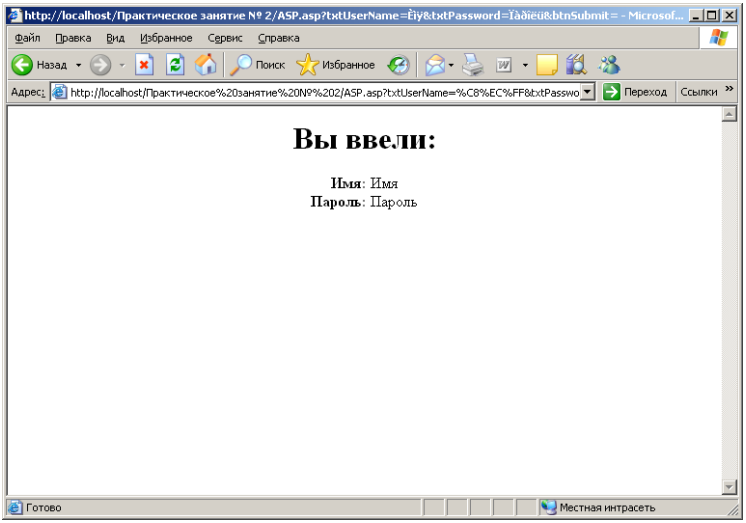


Рис. 3.22. Динамически созданная страница HTML с использованием метода GET

Приведенный пример хорошо иллюстрирует основной принцип работы ASP (и ASP.NET): данные передаются через форму HTML, обрабатываются серверным скриптом, и результат возвращается пользователю в виде сгенерированного кода HTML.

Следует также рассмотреть прием данных, переданных методом POST. В приведенном примере для передачи данных формы использовался метод GET, при котором пары «имя-значение» для элементов управления формы добавлялись к концу строки запроса. Затем значения принимались при помощи свойства QueryString(). Сразу следует отметить, что это свойство может использоваться только для приема значений, передаваемых методом GET. Если изменить значение соответствующего тега формы на

POST и снова запустить наше приложение, ничего положительного не произойдет: вернутся пустые значения.

Конечно же, в типе Request предусмотрены члены, которые позволяют принимать данные, отправленные и методом POST. Для этой цели используется свойство (коллекция) Form(). Выглядит это так:

```
<%
        dim UserName
        UserName =Request.Form("txtUserName")
        Response.Write(UserName)
%>

<%
        dim Password
        Password =Request.Form("txtPassword")
        Response.Write(Password)
 %>
```

Изменив код страницы ASP в соответствии с вышеприведенным примером и запустив Web-приложение, заново можно убедиться, что все работает практически аналогично. Результат может быть таким, как показано на рис. 3.23.

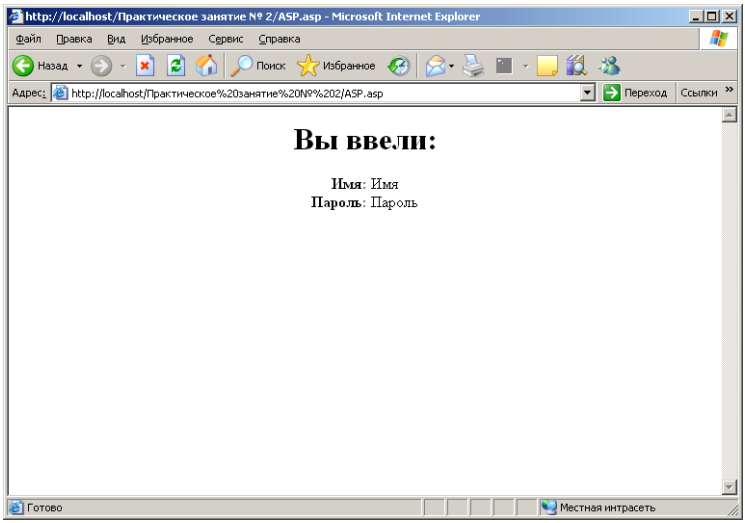


Рис. 3.23. Динамически созданная страница HTML с использованием метода POST

Следует также обратить внимание, что теперь в адресной строке браузера переданные значения не отображаются.

Страницы ASP размещаются на Web-сервере. Общая схема работы приложения ASP при использовании различных методов передачи данных представлена на рис. 3.24.

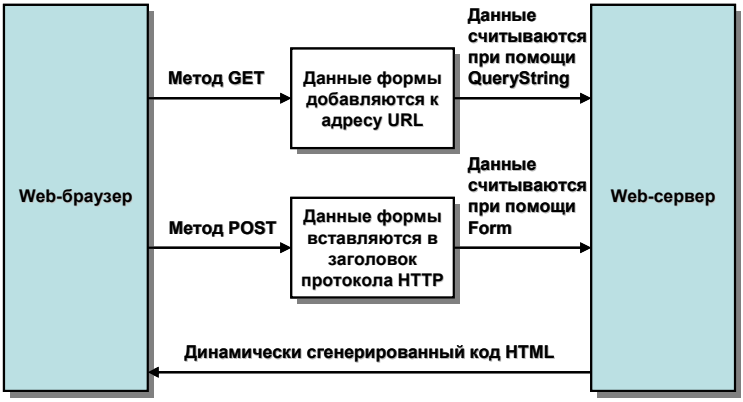


Рис. 3.24. Передача данных на страницу ASP при помощи методов GET и POST

Перед созданием приложений ASP.NET в Microsoft Visual Studio необходимо добавить в проект новую страницу ASP.NET, аналогично созданию страницы ASP, присвоив ей имя, указанное в свойстве Action для формы (ASP.NET.aspx) (рис. 3.25).

Так как проект создан для работы с физическим каталогом файловой системы, а не с виртуальным по протоколу HTTP, то необходимо в отличие от страницы ASP для страницы ASP.NET не указывать URL адрес – все операции по созданию временного виртуального каталога возьмет на себя ASP.NET Development Server. Достаточно в свойстве Action указать только имя файла ASPX:

```
<form name="MainForm" action="ASP.NET.aspx" method="GET">
```

или

```
<form name="MainForm" action="ASP.NET.aspx" method="POST">
```

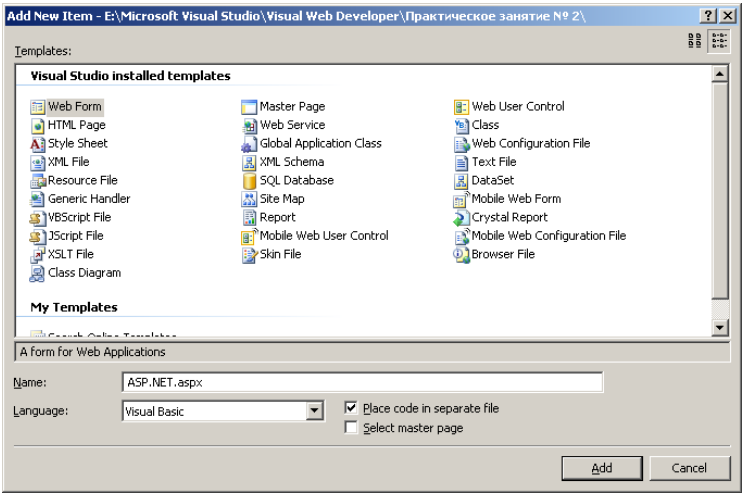


Рис. 3.25. Добавление файла ASP.NET

Все должно работать так же, как и раньше: то, что использовалось в файле классической ASP, используется и в ASP.NET. Однако, конечно же, в ASP.NET достаточно много отличий от классической технологии ASP, которые (как и общие возможности ASP и ASP.NET) будут рассмотрены далее.

Классическая ASP – это очень популярная архитектура создания Web-приложений, однако она не лишена некоторых проблем и недостатков. Главный ее недостаток заключается в том, что в ней используются языки скриптов. Несмотря на различные специальные приемы (например, кэширование откомпилированных скриптов для более быстрого повторного выполнения), языки скриптов – это большой проигрыш как в производительности (поскольку они являются интерпретируемыми), так и в возможностях (так как в них не поддерживаются многие технологии объектно-ориентированного программирования).

Еще одно неудобство классической ASP связано с тем, что в них код HTML смешан с кодом скриптов. В принципе классическая ASP позволяет размещать код HTML отдельно от кода скриптов, но суть дела от этого не меняется: логика представления (код HTML) не отделена от бизнес-логики (т. е. от собственно исполняемого кода).

Еще один момент, знакомый любому Web-разработчику, использующему ASP, заключается в том, что из проекта в проект приходится переносить одни и те же повторяющиеся блоки кода скриптов. В большинстве Web-приложений требуется выполнять одни и те же действия: проверять

данные, вводимые пользователем, обеспечивать форматирование HTML т. п. Гораздо удобнее было бы использовать уже готовые решения, а не копировать код скриптов из одного проекта в другой.

ASP.NET обладает некоторыми преимуществами по сравнению с классической ASP. Также в ASP.NET устранены многие недостатки классических ASP. Например, в файлах ASP.NET (ASPX) языки скриптов не используются. Вместо этого можно применять полнофункциональные языки программирования, такие как Visual C#, Visual Basic, Visual J# и др. В результате любые богатейшие возможности объектно-ориентированного программирования можно применять непосредственно в Web-приложениях. Поэтому на страницах ASP.NET можно использовать как типы из библиотеки базовых классов .NET, так и типы из пользовательских сборок. К тому же ASP.NET имеет преимущество в скорости по сравнению с другими технологиями, основанными на скриптах, потому что код на стороне Web-сервера обычно компилируется в одну или несколько динамических библиотек DLL.

Кроме того, в приложениях ASP.NET количество кода, которое приходится писать вручную, резко сокращается. Например, при помощи серверных объектов Web Controls можно автоматически генерировать код элементов управления HTML, который будет передаваться браузеру пользователя. Другие объекты Web Controls могут быть использованы для реализации проверки вводимых пользователем данных (в результате не нужно будет создавать браузерные клиентские скрипты вручную).

Кроме этого, применение ASP.NET предоставляет в распоряжение разработчика множество мелких, но очень существенных удобств. Например, создание приложений ASP.NET производится в стандартной среде разработки Microsoft Visual Studio, возможности которой (в том числе в отношении удобства отладки) далеко превосходят то, что было реализовано в предыдущих версиях.

Таким образом, ASP.NET имеет ряд преимуществ перед ASP. В частности, это:

- более быстрая разработка приложений за счет использования расширяемого набора элементов управления и библиотек классов;
- более быстрое выполнение компилируемого кода (большинство ошибок фиксируется еще на стадии разработки);
- опора на многоязыковые возможности .NET, что позволяет писать код страниц на различных объектно-ориентированных языках программирования;
- использование метафор уже применяющихся в Windows-приложениях, например, таких как элементы управления и события;

- значительно улучшенная обработка ошибок времени выполнения с использованием блоков «try … catch»;
- выделение часто используемых шаблонов, таких как меню сайта, пользовательскими элементами управления WebForm;
- возможность кэширования всей страницы или ее части для увеличения производительности;
- возможность разделения визуальной части и бизнес-логики в разные файлы [1, 7–10].

Пространства имен ASP.NET. В библиотеке базовых классов .NET предусмотрено множество пространств имен, которые имеют отношение к созданию Web-приложений. Эти пространства имен можно отнести к трем основным группам:

- основные элементы Web-приложений (например, типы для работы с протоколом HTTP, типы системы безопасности и т. п.);
- элементы графического интерфейса (элементы управления Web-Form Controls);
- Web-службы.

Следует ознакомиться с главными и наиболее важными типами этих пространств имен, используемых для создания приложений ASP.NET (табл. 3.2) [1, 7–10].

Таблица 3.2

Пространства имен ASP.NET

Пространство имен	Описание
System.Web	В этом пространстве имен определены наиболее важные типы для организации взаимодействия между браузером и Web-сервером (запрос и ответ, работа с cookie – небольшими персональными фрагментами данных о предыстории обращений пользователя к WWW-серверу, автоматически создаваемыми сервером на компьютере пользователя; передача файлов и т. п.)
System.Web.Caching	Здесь представлены типы для поддержки кэширования при работе Web-приложений
System.Web.Configuration	Типы этого пространства имен позволяют настроить Web-приложение в соответствии с файлами конфигурации проекта
System.Web.Security	Реализация системы безопасности Web-приложений

Продолжение таблицы 3.2

Пространство имен	Описание
System.Web.Services System.Web.Services.Description System.Web.Services.Discovery System.Web.Services.Protocols	В этих пространствах имен представлены типы для построения Web-служб
System.Web.UI System.Web.UI.WebControls System.Web.HtmlControls	Типы этих пространств имен нужны для построения графического интерфейса пользователя Web-приложений

Некоторые типы пространства имен System.Web. В пространстве имен System.Web определен минимальный набор типов, которые позволяют организовать взаимодействие между браузером клиента и Web-сервером. Перечень этих типов представлен в табл. 3.3 [1, 7–10].

Таблица 3.3

Наиболее важные типы пространства имен System.Web

Тип	Описание
HttpApplication	Этот класс определяет члены, которые являются общими для всех приложений ASP.NET. Как станет очевидно в дальнейшем, класс, производный от HttpApplication, определен в файле Global.asax
HttpApplicationState	Класс позволяет определять общую информацию приложения ASP.NET для множества запросов, сеансов и каналов передачи данных
HttpBrowserCapabilities	Позволяет собирать информацию о возможностях браузера, при помощи которого клиент обращается к Web-серверу
HttpCookie	Обеспечивает безопасный способ для работы с множеством объектов HTTP cookie
HttpRequest	Объекты этого класса обеспечивают прием информации, передаваемой клиентом
HttpResponse	Объекты класса используются для создания ответа HTTP, передаваемого Web-сервером клиенту (например, страниц HTML)

Понятия о Web-приложении и сеансе подключения пользователя. Web-приложение – это набор взаимосвязанных файлов, которые расположены в физическом каталоге, соответствующем виртуальному каталогу. В ASP.NET предусмотрен тип HttpApplication, который представляет методы, свойства и события, общие для всех Web-приложений. В файле Glob-

al.asax определен тип Global, который является производным от HttpApplication.

С классом HttpApplication тесно связан класс HttpApplicationState. Этот класс позволяет предоставлять общую информацию приложения в совместное использование множеству сеансов подключения. Сеанс подключения (Session) можно рассматривать как взаимодействие одного пользователя в данный конкретный момент с Web-приложением. Например, если с некоторым приложением в определенный момент работает 20 000 пользователей, то это значит, что к нему открыто 20 000 сеансов.

В ASP.NET для каждого открытого сеанса хранится своя уникальная информация, представленная при помощи типа HttpSessionState. В этой связи каждому пользователю выделена область оперативной памяти, в которой хранятся промежуточные результаты его взаимодействия с Web-приложением. Например, разные пользователи, подключившиеся к Web-приложению, могут интересоваться различной информацией. Отношения между Web-приложением и сеансами подключения к нему представлены на рис. 3.26.

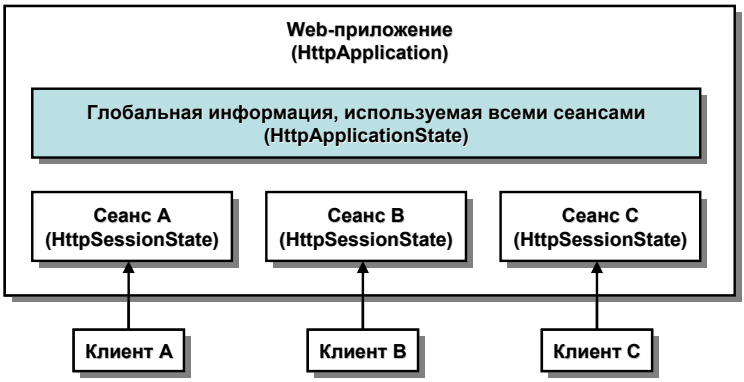


Рис. 3.26. Приложения и сеансы подключения

Таким образом, в классической ASP понятия приложения и сеанса представлены двумя отдельными типами (Application и Session); в ASP.NET они представлены вложенными типами HttpApplicationState и HttpSessionState, доступ к которым производится через свойства Application и Session типов, производных от Page, рассмотрение которых предполагается в дальнейшем [1, 7–10].

3.3. Процесс создания Web-приложения на языках программирования .NET. Исходный файл ASPX. Особенности конфигурационного и глобального файлов. Архитектура Web-приложения. Тип System.Web.UI.Page. Связка *.aspx / CodeFile. Свойства Request, Response и Application. Отладка и трассировка приложений

Процесс создания Web-приложения на языках программирования .NET. Начать рассмотрение разработки Web-приложений и основных принципов архитектуры ASP.NET можно на примере создания простого пробного Web-сайта ASP.NET с фоновым кодом на Visual C#.

Первое, что нужно сделать при создании нового Web-сайта, – сформировать пустой шаблон ASP.NET Web Site, выбрав для него соответствующие местонахождение и язык программирования (рис. 3.27).

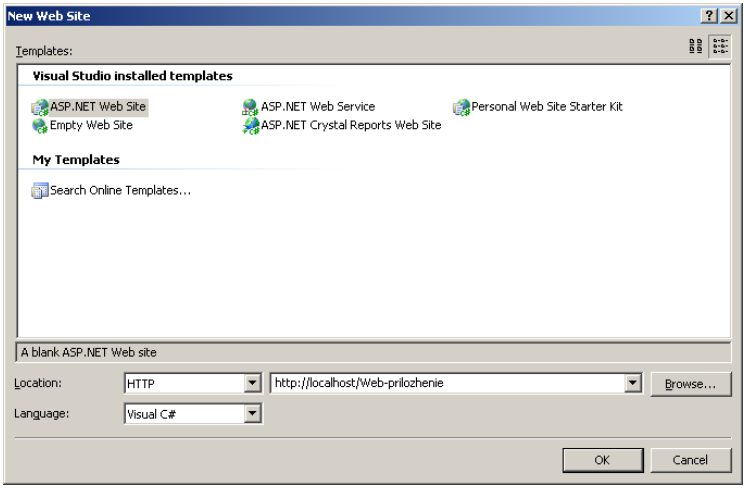


Рис. 3.27. Формирование пустого шаблона ASP.NET Web Site

Следует обратить внимание на то, что в поле Location необходимо выбрать протокол HTTP и указать путь не к физическому каталогу на жестком диске, в котором должен быть расположен Web-сайт, как обычно это делается для проектов, а к соответствующему виртуальному каталогу посредством адреса URL компьютера. В результате должны быть созданы файлы решения Microsoft Visual Studio (SLN и SUO) и открыться создан-

ный шаблон, включающий окна исходного текста и дизайна (проектирования) формы (рис. 3.28).

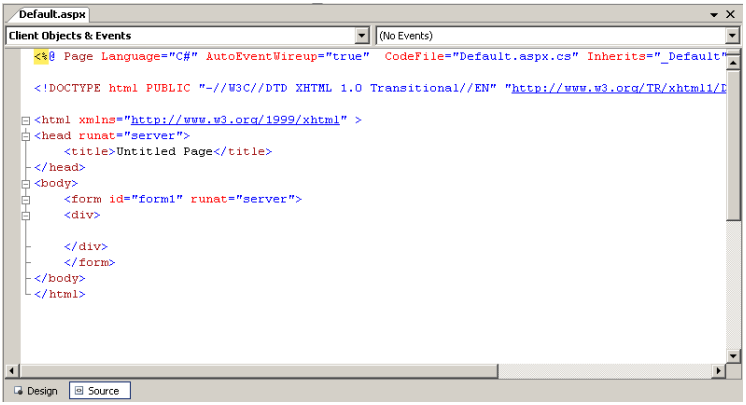


Рис. 3.28. Шаблон ASP.NET Web-сайта

В режиме дизайна этот шаблон действует, как обычный шаблон Windows Forms, представляя графический интерфейс создаваемой страницы ASP.NET. Главное отличие заключается в том, что при этом используются элементы управления не Windows Forms, а WebForm, основанные на соответствующем коде HTML. Следует также обратить внимание, что сформированному файлу страницы присвоено имя «Default.aspx», учитывая то, что в IIS в качестве документа, используемого по умолчанию (исходного), для сайтов ASP.NET, как правило, определено именно это имя. Таким образом, при обращении к этой странице удаленно в строке браузера можно будет не указывать этот файл, например:

http://<имя_компьютера>/Web-prilozhenie/

При работе локально вместо имени компьютера можно указывать «localhost».

В сравнении с обычным Web-приложением в окне Solution Explorer можно заметить, что по умолчанию в проект добавились каталог App_Data для хранения различных прикладных файлов данных, в частности локальной базы данных с информацией о правах доступа и файл Default.aspx.cs для содержания фонового кода.

Если на сервере IIS соответствующий виртуальный каталог не создан заранее, то Microsoft Visual Studio автоматически его сформирует, и если открыть IIS, то можно убедиться в этом. При этом каждый файл, который добавляется в проект, будет помещен в этот виртуальный каталог. Физиче-

ски этому виртуальному каталогу будет соответствовать каталог с расположенными файлами проекта, например [1, 7–10]:

<имя_ устройства>:\Inetpub\wwwroot\Web-prilozhenie

Исходный файл ASPX. Если открыть автоматически сгенерированный файл ASPX, то можно найти в нем минимальный набор тегов с единственной формой:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
        <title>Untitled Page</title>
</head>
<body>
        <form id="form1" runat="server">
        <div>

        </div>
        </form>
</body>
</html>
```

В этом коде достойны внимания несколько деталей. Во-первых, следует обратить внимание на атрибут Runat в открывающем теге <form>. Этот атрибут – один из важнейших в ASP.NET. Он означает, что данный элемент должен быть обработан средой выполнения ASP.NET, которая вернет результат браузеру клиента.

Во-вторых, кроме того, в коде предусмотрено сразу несколько моментов, относящихся ко всей странице в целом. В самом начале используется атрибут Language. Его значение определяет, что для создания кода HTML, возвращаемого браузеру клиента, будет использован Visual C#. Атрибут CodeFile определяет имя файла Visual C#, в котором содержится код этого языка и который будет использован для всех фоновых вычислений (обработок) на стороне сервера. Атрибут Inherits определяет имя класса, представляющего класс, определенный в CodeFile [1, 7–10].

Особенности конфигурационного и глобального файлов. Файл web.config – это текстовый файл в формате XML, который используется для определения множества параметров Web-приложения. Обычно этот файл расположен в корне виртуального каталога, соответствующего физическому, и используется для каждого подкаталога. По умолчанию в него помещается информация о компиляции, ошибках, безопасности, отладке и глобализации. Кроме того, в него могут быть помещены и другие необходимые данные. Таким образом, файл web.config позволяет настраивать основные параметры Web-приложения.

Как и в классических ASP, в ASP.NET используется глобальный файл (Global.asax), который позволяет взаимодействовать с событиями как уровня всего приложения, так и уровня сеанса подключения. Кроме того, этот файл делает возможным совместное использование различных общих данных. Это реализуется при помощи информации, представленной классом Global, являющимся производным от базового класса HttpApplicatlon.

В некоторых отношениях класс Global действует в качестве промежуточного звена между внешним клиентом и элементами управления WebForm, как в классических ASP. В общем, эти события позволяют реагировать на запуск и прекращение работы как Web-приложения в целом, так и отдельных сеансов подключения.

Для более глубокого понимания механизма работы Web-приложений ASP.NET следует рассмотреть пример кода на Visual C#, внедренного в страницу ASPX. Если после создания шаблона обратиться по адресу Web-приложения, то среда выполнения ASP.NET вернет пустую страницу. Можно исправить эту ситуацию, например, изменив содержание файла Default.aspx таким образом, чтобы возвращалась информация о произведенном запросе HTTP (свойство Response класса Page будет рассмотрено более подробно впоследствии):

```
<body>
        <span style="font-size: 14pt">
                <strong>
                        Источник:
                </strong>
        </span>
        <%=Request.ServerVariables["HTTP_USER_AGENT"] %>
        <br />
        <br />
        <span style="font-size: 14pt">
                <strong>
                        Приемник:
                </strong>
```

```
        </span>
        <%=this.ToString() %>
        <form id="form1" runat="server" method="post">
        </form>
</body>
```

Откомпилировав проект и запустив его на выполнение, вернется страница HTML с информацией о браузере, из которого был отправлен запрос, а также о сущности, которая этот запрос приняла (т. е. просто имя страницы ASP.NET) (рис. 3.29).

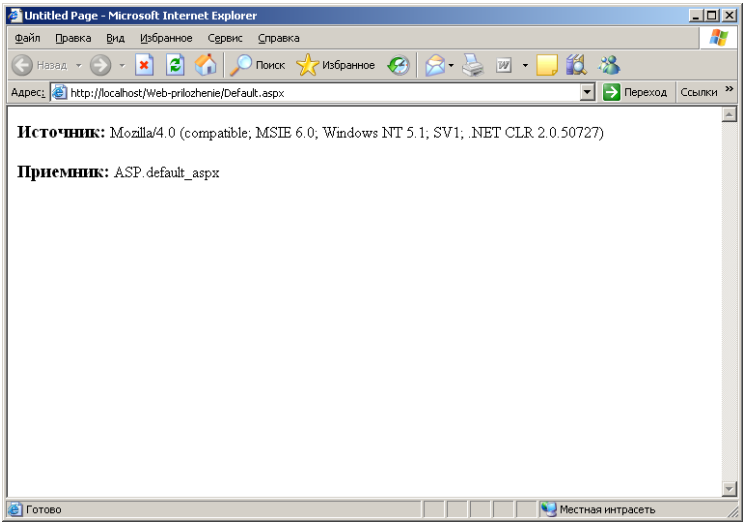


Рис. 3.29. Результат работы приложения ASP.NET

Все выглядит очень похоже на работу с классическими страницами ASP. Однако есть и существенные отличия. Например, в этом случае Request – это свойство объекта, производного от Page. Кроме того, в тегах <% … %> теперь находится не код языка скриптов, а полноценный код Visual C# [1, 7–10].

Архитектура Web-приложения. После создания простого приложения ASP.NET целесообразно разобраться с основными особенностями архитектуры ASP.NET в целом. Из предыдущего примера видно, что среда выполнения поместила в исходный скриптовый блок на странице следующий блок с атрибутом CodeFile:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs"
Inherits="_Default" %>
```

Как отмечалось ранее, главное различие между классическими страницами ASP и ASP.NET заключается в том, что страница ASPX, которую запрашивает внешний клиент, представлена уникальным классом (в данном случае _Default на Visual C#), на который и указывает атрибут CodeFile. Когда клиент запрашивает страницу, среда выполнения ASP.NET создает объект этого класса. Изначально содержимое файла с этим классом выглядит следующим образом:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
        protected void Page_Load(object sender, EventArgs e)
        {

        }
}
```

Как видно, конструктор класса устанавливает обработчик события Load, реализация которого вызывает метод Page_Load() по загрузке страницы [1, 7–10].

Тип System.Web.UI.Page. Очевидно, что любая страница ASP.NET представлена классом, производным от System.Web.UI.Page. Этот класс определяет свойства, методы и события, общие для всех страниц, предназначенных для выполнения в среде ASP.NET. Некоторые наиболее важные свойства этого класса представлены в табл. 3.4.

Таблица 3.4

Свойства класса Page

Свойство	Описание
Application	Возвращает объект HttpApplicationState
Cache	Возвращает объект Cache, в котором хранятся данные приложения, частью которого является данная страница
IsPostBack	Возвращает значение, определяющее, была ли данная страница загружена клиентом в первый раз или она загружена повторно в ответ на переданные клиентом данные
Request	Возвращает объект HttpRequest, используемый для получения информации о входящем запросе HTTP
Response	Возвращает объект HttpResponse, при помощи которого можно скомпоновать данные, возвращаемые браузеру клиента
Server	Возвращает объект HttpServerUtility
Session	Возвращает объект System.Web.SessionState.HttpSessionState, при помощи которого можно получить информацию о текущем сеансе подключения

Как видно, свойства класса Page обеспечивают те же возможности, которые были в распоряжении в классических страницах ASP. В классе Page также имеется несколько унаследованных методов (они используются редко) и несколько очень важных событий, представленных в табл. 3.5.

Таблица 3.5

Наиболее важные события класса Page

Событие	Описание
Init	Это событие происходит, когда страница инициализируется. Это – первое событие жизненного цикла страницы
Load	Событие происходит после события Init. Обработчик этого события можно использовать для настройки любых элементов управления WebForm
Unload	Происходит при выгрузке объекта из памяти. Можно использовать, к примеру, для освобождения ресурсов

Обработчик события Load – лучшее средство установления соединения с источником данных (например, для заполнения на форме элемента управления DataGrid) или выполнения других подготовительных действий. Обработчик события Unload можно использовать, например, для закрытия этого соединения и выполнения других аналогичных действий.

*Связка *.aspx / CodeFile.* Помимо готовых членов, унаследованных от Page, можно определить в созданном классе C# собственные члены, которые могут быть вызваны (не напрямую) при помощи блоков <% … %> в файле с расширением ASPX. В классических страницах ASP все дополнительные возможности приходилось определять непосредственно в коде файла с расширением ASP. В результате содержимое этого файла было сложно воспринимать (читать), а использовать его код повторно – еще сложнее вследствие множества различных тегов HTML и кода VBScript (или JavaScript).

В ASP.NET эта проблема решена за счет того, что код представления (т. е. код для генерации кода HTML) помещен в файл ASPX, а прочая программная логика – обычным образом в файл, например C# с расширением ASPX.CS.

Обращение к пользовательским членам класса, производного от Page, происходит следующим образом. Предполагается, что определена в таком классе простая функция, возвращающая текущее значение даты и времени:

```
public string GetDateTime()
{
        return DateTime.Now.ToString();
}
```

Обратиться к этой функции из файла ASPX можно так:

```
<%Response.Write(GetDateTime()); %>
```

Конечно, можно использовать нужные унаследованные от Page члены непосредственно внутри класса C#. Например, можно определить указанную функцию следующим образом [1, 7–10]:

```
public void GetDateTime()
{
        Response.Write(DateTime.Now.ToString());
}
```

А затем просто вызвать ее:

```
<%GetDateTime(); %>
```

Свойства Request, Response и Application. Множество Web-приложений построено по одному и тому же принципу: например, клиент заходит на Web-сайт, заполняет форму HTML своей пользовательской информацией и нажимает на кнопку Submit этой формы, чтобы передать информа-

цию на сервер. В большинстве случаев в теге форм используются атрибуты Action и Method. Первый определяет адрес получателя информации на сервере, а второй – метод ее передачи, например:

```
<form name="MainForm" action="http://localhost/Web-prilozhenie/Default.aspx"
method="POST">
```

В ASP.NET свойство Request позволяет получить доступ к данным, отправленным пользователем в виде запроса HTTP. Если разобраться, что делает это свойство, то выяснится, что оно взаимодействует с объектом класса HttpRequest. Некоторые наиболее важные свойства этого класса, также похожие на используемые в классических страницах ASP, представлены в табл. 3.6.

Таблица 3.6

Свойства типа HttpRequest

Свойство	Описание
ApplicationPath	Возвращает виртуальный путь к приложению, выполняющемуся на сервере
Browser	Позволяет получить информацию о возможностях браузера клиента
ContentType	Определяет тип содержимого MIME (Multipurpose Internet Mail Extensions – многоцелевых расширений электронной почты в сети Internet, набора стандартов для передачи мультимедийной информации посредством электронной почты) для входящего запроса. Это свойство доступно только для чтения
Cookies	Возвращает набор клиентских cookie
FilePath	Возвращает виртуальный путь к запрашиваемому файлу. Свойство доступно только для чтения
Files	Возвращает набор файлов, загруженных клиентов (формат MIME для файлов из нескольких частей)
Filter	Позволяет получить или установить фильтр, используемый для чтения потока входящих данных
Form	Позволяет получить набор переменных Form
Headers	Позволяет получить набор заголовков HTTP
HttpMethod	Определяет метод передачи данных, используемый клиентом (GET или POST). Доступно только для чтения
IsSecureConnection	Позволяет получить информацию о том, является ли установленное соединение защищенным (с применением SSL). Это свойство доступно только для чтения
Params	Возвращает комбинированный набор QueryString + Form + ServerVariable + Cookie

Продолжение таблицы 3.6

Свойство	Описание
QueryString	Возвращает набор переменных QueryString
RawUrl	Возвращает текущий запрос клиента в виде адреса URL
RequestType	Определяет метод передачи данных, используемых клиентом (GET или POST)
ServerVariables	Возвращает набор переменных Web-сервера
UserHostAddress	Возвращает IP-адрес (Internet Protocol – используется для идентификации узла в сети и для определения информации маршрутизации) компьютера удаленного клиента
UserHostName	Возвращает имя DNS (Domain Name System – служба имен доменов, механизм, используемый в сети Internet и устанавливающий соответствие между числовыми IP-адресами и текстовыми именами компьютеров в доменных записях) компьютера удаленного клиента

При помощи этих свойств можно получить любую возможную информацию о запросе пользователя. Рассмотренный объект уже был использован ранее, например, когда получалась информация о браузере пользователя при помощи строки следующего вида:

```
<%=Request.ServerVariables["HTTP_USER_AGENT"] %>
```

Реально обращение происходило к свойству возвращаемого объекта HttpRequest:

```
<%
        HttpRequest r;

        r=this.Request;
        Response.Write(r.ServerVariables["HTTP_USER_AGENT"]);
%>
```

Получение информации о передаваемых пользователем данных привычными средствами C# уже рассмотрено. Однако нужно определенным образом на них отреагировать. В этом поможет свойство Response и соответствующий ему класс HttpResponse.

Свойство Response возвращает объект класса HttpResponse. В этом классе предусмотрено множество свойств, которые предназначены для одной цели – помочь скомпоновать ответ в виде кода HTML (т. е. Web-

страницу), который будет возвращен браузеру клиента. Некоторые наиболее важные свойства этого класса представлены в табл. 3.7.

Таблица 3.7

Свойства класса HttpResponse

Свойство	Описание
Cache	Возвращает информацию о кэшировании для Web-страницы (время устаревания и т. п.)
ContentEncoding	Позволяет определить кодировку для возвращаемых клиенту данных
ContentType	Позволяет определить тип MIME для возвращаемых клиенту данных
Cookies	Возвращает коллекцию HttpCookie, отправленных в текущем запросе
Filter	Определяет объект фильтра, который может быть использован для внесения изменений в данные HTTP перед отправкой их клиенту
IsClientConnected	Позволяет получить информацию о том, подключен ли клиент к серверу
Output	Используется для добавления пользовательских данных в возвращаемый клиенту ответ на запрос
OutputStream	То же самое, но для добавления двоичных данных
StatusCode	Позволяет определить код состояния HTTP для переданных клиенту данных
StatusDescription	Позволяет получить строку состояния HTTP для переданных клиенту данных
SupressContent	Позволяет получить или установить значение, определяющее, будут ли данные отправлены клиенту

Кроме того, в классе HttpResponse определены важные методы, представленные в табл. 3.8.

Таблица 3.8

Методы класса HttpResponse

Метод	Описание
AppendHeader()	Добавляет заголовок HTTP в возвращаемые клиенту данные
AppendToLog()	Добавляет пользовательскую информацию в файл журнала IIS
Clear()	Очищает все заголовки и все содержимое буфера для возвращаемых данных
Close()	Закрывает соединение с клиентом

Продолжение таблицы 3.8

End()	Отправляет все содержимое буфера для возвращаемых данных клиенту, а после этого закрывает соединение
Flush()	Отправляет все содержимое буфера для возвращаемых данных клиенту
Redirect()	Перенаправляет клиента по указанному адресу URL
Write()	Добавляет значение в данные, возвращаемые клиенту
WriteFile()	Метод многократно перегружен. Используется для направления файла напрямую браузеру клиента

Наиболее важный и часто используемый метод класса HttpResponse – это метод Write(), который позволяет добавлять значения в набор возвращаемых клиенту данных. Этот метод можно вызывать как явно:

```
<%
        HttpRequest r;

        r=this.Request;

        HttpResponse rs;

        rs=this.Response;
        rs.Write(r.ServerVariables["HTTP_USER_AGENT"]);
%>
```

так и косвенно, в стиле классических страниц ASP, как было рассмотрено ранее:

```
<%=Request.ServerVariables["HTTP_USER_AGENT"] %>
```

И в том и в другом случае результат будет совершенно одинаковым.

Свойство Application класса Page обеспечивает доступ к объекту класса HttpApplicationState. Как уже отмечалось, HttpApplicationState предоставляет разработчикам возможность управления информацией, общей для всех сеансов подключения к приложению ASP.NET. Некоторые наиболее важные свойства HttpApplicationState представлены в табл. 3.9.

Таблица 3.9

Свойства типа HttpApplicationState

Свойство	Описание
AllKeys	Возвращает набор всех объектов, относящихся к состоянию приложения
Count	Позволяет получить количество объектов в наборе, относящихся к состоянию приложения
Keys	Возвращает объект NameObjectCollectionBase.KeysCollection, используемый для хранения всех ключей состояния приложения объекта NameObjectCollectionBase
StaticObjects	Позволяет получить доступ ко всем объектам, объявленным в теге <X runat="server"></X> в файле приложения ASP.NET

При создании переменной, которая будет доступна из всех сеансов подключения, необходимо использовать пару «имя-значение» (например, Organizatsiya="Организация 1"), а затем добавить ее во внутреннюю коллекцию KeysCollection. Для этого можно использовать индексатор класса:

```
Application["Organizatsiya"] = "Организация 1";
```

Если затем потребуется обратиться к этому значению, необходимо извлечь его при помощи того же свойства [1, 7–10]:

```
string Peremennaya = "Организация:" + Application["Organizatsiya"];
```

Отладка и трассировка приложений. В отличие от предыдущих версий сред разработки программного обеспечения отлаживать приложения ASP.NET в средах последних версий стало гораздо удобнее. В этом отношении при создании проектов ASP.NET можно использовать те же самые средства отладки, что и для любых других проектов Microsoft Visual Studio. Например, можно устанавливать точки прерываний (брейкпойнты) в скриптовых блоках (или в файлах C#), запускать сеансы отладки, производить пошаговое выполнение и т. п. (рис. 3.30).

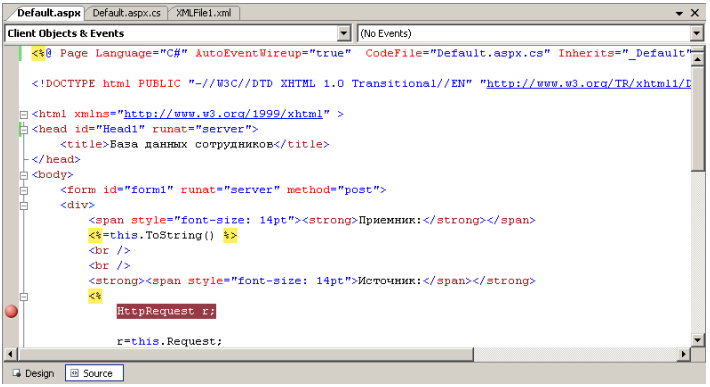


Рис. 3.30. Установка точек прерываний

Кроме того, можно осуществлять трассировку файлов ASPX просто путем добавления атрибута Trace в открывающий скриптовый блок:

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" Trace="true" %>
```

В результате в конец возвращаемого клиенту файла HTML будет добавлена информация трассировки (рис. 3.31).

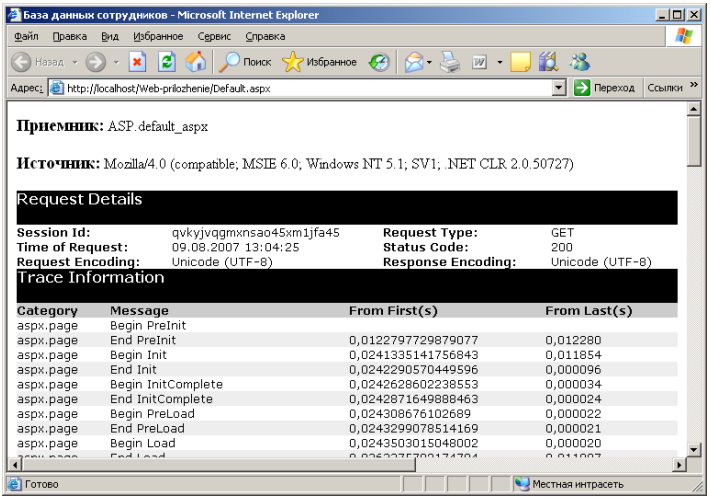


Рис. 3.31. Получение информации трассировки

Можно добавлять и собственные данные трассировки. Для этого используется класс Trace. Каждый раз, когда нужно добавить оригинальное сообщение трассировки, просто используется метод Write этого класса ():

```
Trace.Write("Категория", "Сообщение");
```

Результат добавления пользовательского сообщения трассировки представлен на рис. 3.32.

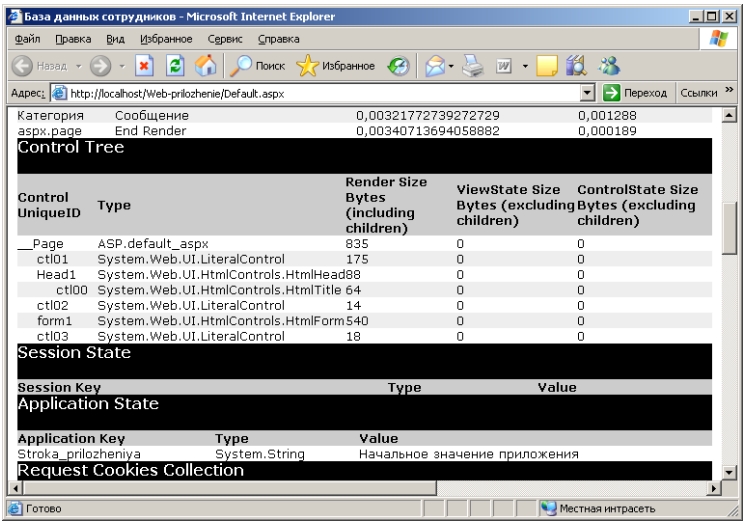


Рис. 3.32. Пользовательское сообщение трассировки

Таким образом, рассмотрены запросы HTTP и создание кода HTML в ответ на эти запросы. Однако еще стоит рассмотреть элементы графического интерфейса, которые можно использовать как для приема данных пользователя, так и для возврата информации в ответ на пользовательский запрос, хотя здесь ASP.NET предлагает еще большее количество новых и очень интересных возможностей [1, 7–10].

3.4. Создание и иерархия классов элементов управления WebForm. Виды и базовые элементы управления. Элементы управления с дополнительными возможностями. Настройка доступа к базам данных в Microsoft SQL Server Management Studio. Элементы управления для работы с источниками данных. Элементы управления для проверки вводимых пользователем данных. Обработка событий элементов управления

Создание и иерархия классов элементов управления WebForm. Одно из важнейших достоинств ASP.NET заключается в том, что при ее использовании резко упрощается создание элементов пользовательского интерфейса на Web-страницах. Элементы управления WebForm – их также называют серверными элементами управления (Server Controls) или элементами управления Web (Web Controls) – определены в пространстве имен System.Web.UI.WebControls. Их основное назначение – избавление от трудоемкой работы по созданию элементов управления HTML на Web-странице вручную.

Например, если создается классическая страница ASP и есть необходимость разместить на ней несколько текстовых полей, то придется писать теги для каждого текстового поля на странице вручную. В ASP.NET достаточно будет поместить на шаблон времени разработки графические элементы управления из панели Toolbox, а затем их настроить. Например, для текстового поля будет автоматически сгенерирован следующий код:

```
<asp:TextBox ID="TextBox1" runat="server">
</asp:TextBox>
```

Когда придет время отвечать на запрос пользователя, среда выполнения ASP.NET автоматически преобразует этот код в тег HTML следующего вида:

```
<input name="TextBox1" type="text" id="Text1" />
```

Можно написать указанный выше код для текстового поля и вручную – все будет казаться проще, и в итоге окажется меньше кода на странице ASP. Однако, во-первых, при использовании элементов управления WebForm может вообще не потребоваться писать вручную код HTML, а, во-вторых, не все элементы управления такие простые, как текстовые поля. Например, если нужно поместить на генерируемую Web-страницу боль-

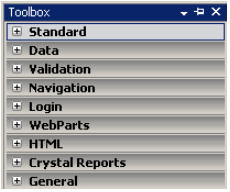
шую таблицу, баннерную рулетку, календарный элемент управления или элемент управления DataGrid, то применение элементов управления WebForm может сэкономить достаточно много времени.

Еще одно преимущество элементов управления WebForm заключается в том, что с ними очень удобно работать с программной точки зрения: каждому из этих элементов управления соответствует класс в библиотеке базовых классов .NET, и можно работать с ними, как с любыми другими классами, и в файле ASPX, и в производном от Page классе C#. Кроме того, в любом элементе управления WebForm также определен набор событий, которые будут обрабатываться на сервере (более подробно это рассмотрено ниже).

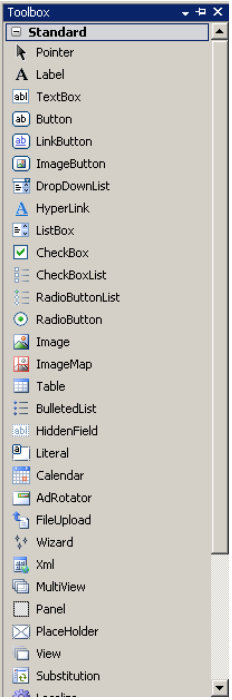
Также стоит упомянуть о том, что при использовании элементов управления WebForm в распоряжении появляется целый набор возможностей для проверки ввода данных пользователем. Таким образом, можно избавиться не только от необходимости написания тегов для элементов управления HTML вручную, но и от ручного создания клиентских скриптов JavaScript для проверки вводимых данных. Конечно, если все же нужно использовать код JavaScript на странице, это делать возможно.

При создании проекта, например, на основе шаблона ASP.NET Web Site в распоряжении всегда есть набор элементов управления WebForm. Для того чтобы ими воспользоваться, достаточно перейти на соответствующую вкладку в Toolbox (рис. 3.33).

Настройку параметров каждого из элементов управления WebForm можно производить при помощи окна свойств Properties в Microsoft Visual Studio. Эти элементы управления очень похожи на элементы управления Windows Forms, и поэтому проблем с настройкой подавляющего большинства свойств не должно быть. Например, для текстового поля в распоряжении набор свойств, представленный на рис. 3.34.



а



б

Рис. 3.33. Набор элементов управления WebForm: а – все разделы; б – раздел Standard

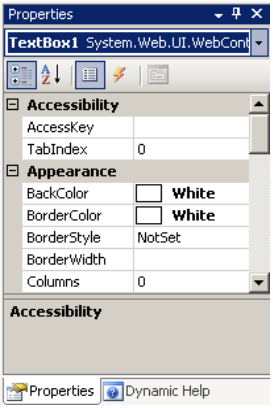


Рис. 3.34. Набор свойств для текстового поля

При внесении изменений через окно свойств для элемента управления все изменения сразу же записываются напрямую в файл ASPX. Например, если для текстового поля TextBox1 изменить (задать) значения свойств BackColor, BorderColor, BorderStyle, BorderWidth и ToolTip, в теге <asp:TextBox> появятся новые пары «имя-значение»:

```
<asp:TextBox ID="TextBox1" runat="server" BackColor="White"
BorderColor="White"
        BorderStyle="None" BorderWidth="1" ToolTip="Фамилия, имя и
отчество">        </asp:TextBox>
```

Итоговый результат в виде кода HTML, который вернется пользователю, выглядит следующим образом:

```
<input name="TextBox1" type="text" id="Text1" title="Фамилия, имя и
отчество"          style="background-color:White;border-color:White;border-
width:1px;border-style:None;"
/>
```

Следует разобраться с синтаксисом записей для элементов управления WebForm в файле ASPX. Для каждого элемента управления используется синтаксис, очень напоминающий формат XML. Открывающим тегом всегда будет <asp:тип_элемента_управления runat="server">, а закрывающим – </asp:тип_элемента_управления> (например, как в приведенной выше записи). Пара Runat="server" означает, что это – элемент управления WebForm, предназначенный для выполнения на сервере. Прежде чем его код отправится к клиенту, он будет преобразован средой выполнения ASP.NET в привычный код HTML.

В ранних версиях Microsoft Visual Studio в файле C#, который указан в атрибуте Codebehind (аналог CodeFile в поздних версиях), код также изменяется. В нем появляются новые объекты, представляющие элементы управления. Имена этих объектов совпадают с идентификаторами элементов в тегах файла ASPX. Конечно же, можно программным образом работать с этими объектами привычными способами C#.

Все элементы управления WebForm – это классы, производные от базового класса System.Web.UI.WebControls.WebControl. Для этого класса, в свою очередь, базовым является System.Web.UI.WebControls.Control, который происходит непосредственно от System.Object. Например, иерархия классов для элементов управления выглядит так, как показано на рис. 3.35.

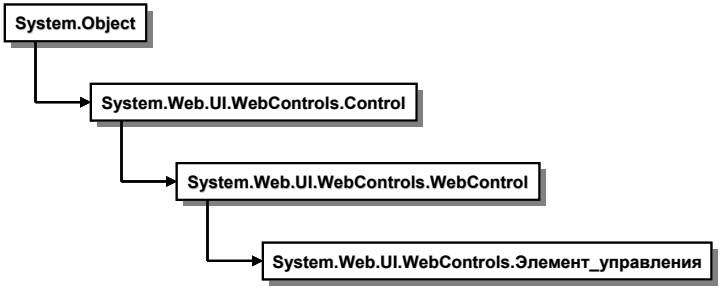


Рис. 3.35. Иерархия классов для элементов управления

И класс Control, и класс WebControl наделяют производные классы каждый своим набором очень важных членов. Свойства, определенные в классе Control, представлены в табл. 3.10.

Таблица 3.10

Свойства базового класса Control

Свойство	**Описание**
ID	Позволяет получить или установить идентификатор элемента управления
MaintainState	Позволяет работать с состоянием просмотра для элемента управления (более подробно это будет рассмотрено ниже)
Page	Возвращает объект Page, т. е. страницу, на которой находится данный элемент управления
Visible	Позволяет получить или определить видимость элемента управления (т. е. будет он выводиться на странице или нет)

Свойства, определенные в классе WebControl, в основном позволяют определять внешний вид элемента управления (табл. 3.11) [1, 7–10].

Таблица 3.11

Свойства базового класса WebControl

Свойство	**Описание**
BackColor	Позволяет получить или установить цвет фона элемента управления
BorderColor	Позволяет получить или установить цвет рамки вокруг элемента управления
BorderStyle	Позволяет получить или установить стиль рамки вокруг элемента управления
BorderWidth	Позволяет получить или установить толщину рамки вокруг элемента управления

Продолжение таблицы 3.11

Свойство	Описание
Enabled	Позволяет получить или установить значение, определяющее доступность элемента управления
Font	Позволяет получить информацию об используемом шрифте
ForeColor	Позволяет получить или установить цвет переднего плана (обычно цвет надписи) для элемента управления
Height Width	Определяют соответственно высоту и ширину элемента управления
TabIndex	Позволяет получить или установить значение TabIndex (очередности перехода по клавише табуляции Tab)
ToolTip	Позволяет получить или установить значение ToolTip – всплывающей подсказки, появляющейся при наведении на элемент управления указателя мыши

Виды и базовые элементы управления. Все множество элементов управления можно поделить на четыре основные разновидности:

- базовые элементы управления;
- элементы управления с дополнительными возможностями;
- элементы управления для работы с источниками данных;
- элементы управления для проверки вводимых пользователем данных.

Назначение указанных элементов управления схоже с назначением элементов управления Windows Forms. Единственное, что постоянно следует иметь в виду,– это само принципиальное различие между этими элементами управления. Элементы управления WebForm в итоге преобразуются в набор тегов HTML, а Windows Forms – в набор вызовов Win32 API.

К базовым элементам управления WebForm относятся те из них, которым есть прямые соответствия в HTML. Например, для отображения списка сотрудников можно использовать элемент управления ListBox с прилагающимися к нему элементами управления ListItem:

```
<asp:ListBox ID="ListBox1" runat="server" ToolTip="Список сотрудников">
        <asp:ListItem>
                Сотрудник 1
        </asp:ListItem>
        <asp:ListItem>
                Сотрудник 2
        </asp:ListItem>
        <asp:ListItem>
                Сотрудник 3
```

```
        </asp:ListItem>
</asp:ListBox>
```

Результат работы указанных элементов управления представлен на рис. 3.36.

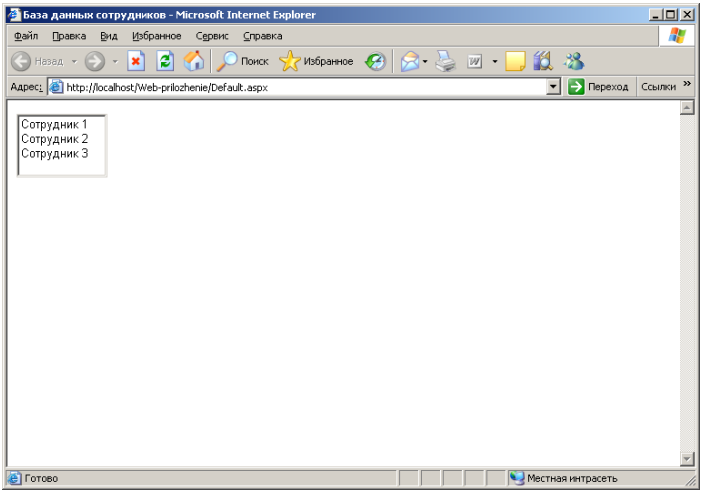


Рис. 3.36. Элемент управления ListBox

Перед тем как отправиться к клиенту, этот код будет преобразован средой выполнения ASP.NET в следующий набор тегов HTML:

```
<select size="4" name="ListBox1" id="Select1" title="Список сотрудников">
        <option value="Сотрудник 1">
                Сотрудник 1
        </option>
        <option value="Сотрудник 2">
                Сотрудник 2
        </option>
        <option value="Сотрудник 3">
                Сотрудник 3
        </option>
</select>
```

Некоторые базовые элементы управления WebForm представлены в табл. 3.12.

Таблица 3.12

Базовые элементы управления WebForm

Элемент управления	Описание
Button ImageButton	Разновидности кнопок
CheckBox CheckBoxList	Флажок (CheckBox) или окно списка с несколькими флажками (CheckBoxList)
DropDownList ListBox ListItem	Эти типы предназначены для создания различных разновидностей списков (ниспадающий, обычный и элемент списка соответственно)
Image Pand Label	Эти типы представляют контейнеры для статического текста и изображений (а также средство для их группировки)
RadioButton RadioButtonList	Стандартный переключатель (RadioButton) или окно списка с набором переключателей (RadioButtonList)
TextBox	Текстовое окно для ввода данных пользователем. Может быть настроено для приема одной или нескольких строк текста

Работа с базовыми элементами управления WebForm очень похожа на работу с их аналогами Windows Forms. Следует остановиться на нескольких стандартных ситуациях, связанных с использованием элементов управления WebForm.

Группа переключателей. Переключатели обычно объединяются в группы. В одной группе одновременно может быть выбран только один переключатель. Например, необходимо создать пользовательский интерфейс, представленный на рис. 3.37.

Код для него может быть таким:

```
<strong>
        <span>
                Средство связи с сотрудником:
        </span>
</strong>
<br /><br />
<asp:RadioButtonList ID="RadioButtonList1" runat="server"
BackColor="#E0E0E0"
        BorderColor="Silver" Width="494px" BorderStyle="Outset"
BorderWidth="3px">
        <asp:ListItem Selected="True">
                Адрес
```

```
        </asp:ListItem>
        <asp:ListItem>
                Телефон
        </asp:ListItem>
        <asp:ListItem>
                E-mail
        </asp:ListItem>
</asp:RadioButtonList>
```

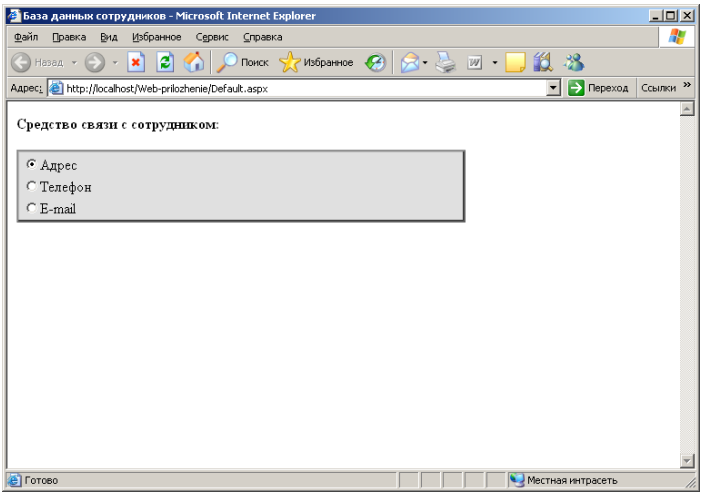


Рис. 3.37. Группа переключателей на Web-странице

Следует обратить внимание, что для каждого объекта RadioButton предусмотрен атрибут GroupName. Если значение этого атрибута у нескольких переключателей одно и то же (как в данном случае), одновременно может быть выбран только один из них.

Текстовое поле для ввода нескольких строк с полосой прокрутки. Еще одним часто используемым элементом управления является текстовое поле для ввода нескольких строк (рис. 3.38).

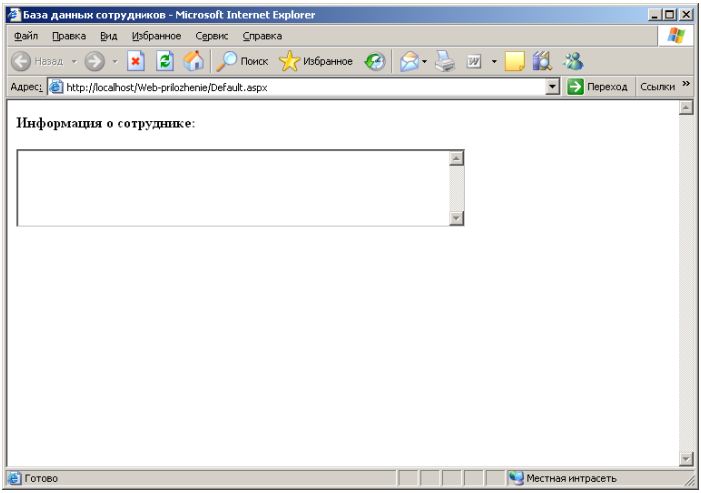


Рис. 3.38. Текстовое поле для ввода нескольких строк

Очевидно, чтобы в поле можно было вводить несколько строк, необходимо установить значение соответствующего атрибута. Выглядеть это может так:

```
<span>
        <strong>
                Информация о сотруднике:
        </strong>
</span>
<br /><br />
<asp:TextBox ID="TextBox2" runat="server" BorderColor="White" Rows="5"
        TextMode="MultiLine" Width="489px" ToolTip="Информация о
сотруднике">
</asp:TextBox>
```

При установке для атрибута TextMode значения MultiLine у текстового поля автоматически возникает (активизируется) полоса прокрутки, когда введенные в него данные уже не могут поместиться в отображаемой области [1, 7–10].

Элементы управления с дополнительными возможностями. К этой группе относятся элементы управления, для которых не предусмотрено прямых аналогов в HTML. Их всего два, и они представлены в табл. 3.13.

Таблица 3.13

Элементы управления WebForm с дополнительными возможностями

Элемент управления	Описание
AdRotator	Баннерная рулетка: набор из нескольких пар «изображение-альтернативный текст», которые сменяют друг друга. Для настройки используется специальный код в формате XML
Calendar	Этот элемент управления возвращает код HTML, представляющий календарь

Элемент управления AdRotator (баннерная рулетка). Элемент управления AdRotator (баннерная рулетка; обычно в компьютерной терминологии баннером называется графический объект, помещаемый на Web-страницу и имеющий гиперссылку на рекламируемую страницу) был и в классических страницах ASP, однако в ASP.NET он дополнен новыми возможностями. Задача этого элемента управления проста и понятна: менять изображения в окне браузера через заданные промежутки времени. Обычно, конечно, с его помощью делают рекламу. При размещении этого элемента управления на шаблоне страницы времени разработки на ней будет лишь помечено место, где будут находиться баннеры. Все остальное придется делать вручную. Если точнее, то придется указать для свойства AdvertisementFile путь и имя файла в формате XML, в котором будут храниться настройки баннерной рулетки, а затем написать этот самый файл.

Формат AdvertisementFile очень прост. Для каждого баннера создается отдельный тег <Ad> (от Advertisement). Как минимум в этом теге должны быть указаны:

- ImageUrl – путь к файлу изображения, т. е. баннера;
- NavigateUrl – адрес URL, на который клиент перейдет при щелчке на этом баннере;
- AlternateText – альтернативный текст, который будет периодически сменять изображение или появляться при наведении на него указателя мыши;
- Impressions – вес конкретного баннера в общем времени показа.

Например, можно создать файл в формате XML (пусть он называется XMLFile1.xml) со следующим содержимым:

```
<?xml version="1.0" encoding="utf-8"?>
<Advertisements>
        <Ad>
                <ImageUrl>Фото\Фирма 1.jpg</ImageUrl>
```

```
        <NavigateUrl>http://www.firma_1.com</NavigateUrl>
        <AlternateText>Фирма 1</AlternateText>
        <Impressions>80</Impressions>
    </Ad>
    <Ad>
        <ImageUrl>Фото\Фирма 2.jpg</ImageUrl>
        <NavigateUrl>http://www.firma_2.com</NavigateUrl>
        <AlternateText>Фирма 2</AlternateText>
        <Impressions>80</Impressions>
    </Ad>
</Advertisements>
```

После этого осталось убедиться, что файл XML и файлы изображений помещены в один виртуальный каталог со страницей ASPX, и настроить для элемента управления AdRotator значения атрибутов AdvertisementFile, Height и Width, например, так:

```
<asp:AdRotator ID="AdRotator1" runat="server"
AdvertisementFile="~/XMLFile1.xml"
Height="150px" Width="199px" />
```

Свойства Height и Width элемента управления AdRotator определяют высоту и ширину баннера соответственно. Если изображение не будет подходить под эти размеры, оно будет растянуто или сжато.

Разные «повороты» созданной баннерной рулетки представлены на рис. 3.39 и 3.40.

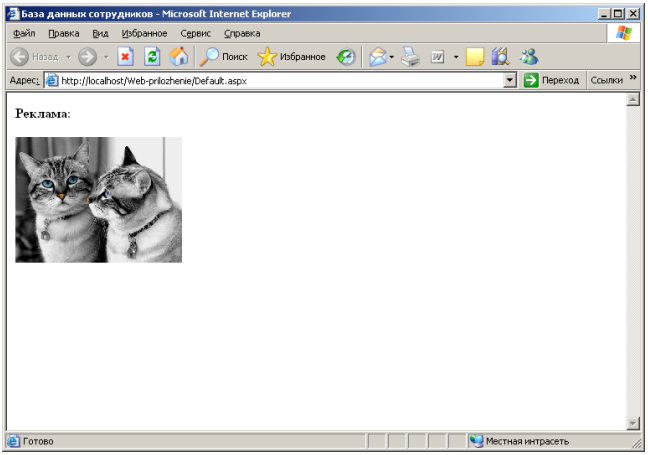


Рис. 3.39. Первый баннер

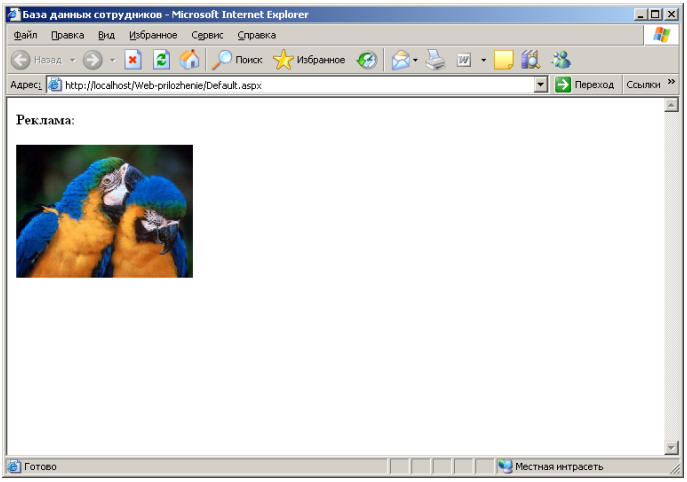


Рис. 3.40. Второй баннер

Элемент управления Calendar. Для элемента управления Calendar не существует прямого эквивалента в HTML. Однако необходимость в помещении на Web-страницу календаря возникает очень часто. Поэтому в ASP.NET был предусмотрен специальный элемент управления, который преобразуется средой выполнения ASP.NET в набор тегов HTML, представляющий календарь. Например, этот элемент управления может размещаться на Web-странице при помощи следующего кода:

```
<asp:Calendar runat="server"></asp:Calendar>
```

Можно увидеть, какое большое количество кода HTML сгенерировала среда выполнения ASP.NET, обнаружив на странице такую строку. Это делается путем помещения (перетаскивания) объекта Calendar из Toolbox на графический шаблон времени разработки (в режиме Design), сохранения файла ASPX и обращения к нему из Web-браузера. После этого можно, например, через контекстное меню просмотреть HTML-код для элемента управления Calendar (рис. 3.41).

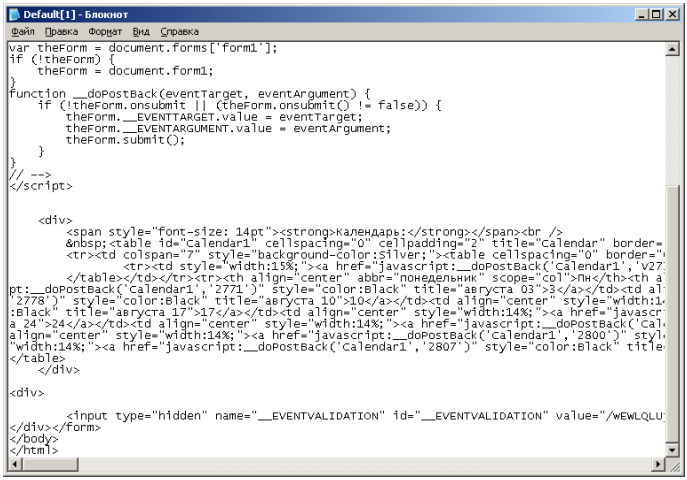


Рис. 3.41. Для элемента управления Calendar генерируется весьма объемный код HTML

В элементе управления Calendar предусмотрено множество возможностей для настройки. Одно из свойств, которое может представлять интерес, – это свойство SelectionMode. По умолчанию в календаре можно выбирать только один день (что соответствует значению по умолчанию SelectionMode – Day). Однако можно воспользоваться и другими допустимыми значениями этого свойства:

- None – вообще ничего нельзя выбирать, т. е. календарь будет предназначен исключительно для справочных целей;
- DayWeek – можно выбирать один день или целую неделю;
- DayWeekMonth – можно выбирать день, неделю или месяц.

Например, если установить значение DayWeekMonth, в возвращаемом коде HTML будет предусмотрена дополнительная пара «имя-значение», а на странице для элемента управления Calendar будет сформирован столбец одинарных флажков слева (для выбора недели целиком) и двойной флажок в верхнем левом углу (для выбора всего месяца). Код, использующий некоторые из возможных свойств указанного элемента управления, следующий (приведенный код можно сформировать автоматически, используя окно свойств в Microsoft Visual Studio):

```
<asp:Calendar ID="Calendar1" runat="server" BackColor="White"
BorderColor="White"
        BorderWidth="1px" Font-Names="Times New Roman Cyr" Font-
Size="9pt"
```

```
        ForeColor="Black" Height="190px" NextPrevFormat="FullMonth"
Width="350px"
        OnSelectionChanged="Calendar1_SelectionChanged">
        <SelectedDayStyle BackColor="#333399" ForeColor="White" />
        <TodayDayStyle BackColor="#CCCCCC" />
        <OtherMonthDayStyle ForeColor="#999999" />
        <NextPrevStyle Font-Bold="True" Font-Size="8pt"
ForeColor="#333333"
                VerticalAlign="Bottom" />
        <DayHeaderStyle Font-Bold="True" Font-Size="8pt" />
        <TitleStyle BackColor="White" BorderColor="Black"
BorderWidth="4px"
                Font-Bold="True" Font-Size="12pt" ForeColor="#333399" />
</asp:Calendar>
```

То, как может выглядеть календарь в окне браузера клиента, показано на рис. 3.42 [1, 7–10].

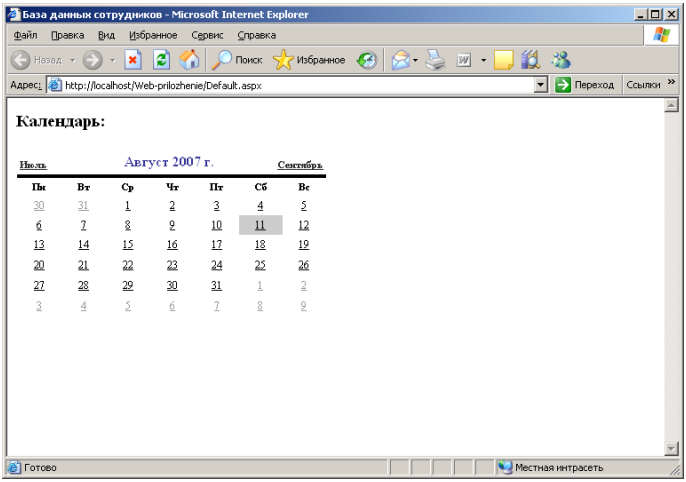


Рис. 3.42. Элемент управления Calendar в окне Microsoft Internet Explorer

Настройка доступа к базам данных в Microsoft SQL Server Management Studio. Прежде чем приступать к использованию элементов управления для работы с источниками данных, необходимо определенным образом произвести настройку доступа к базам данных в Microsoft SQL Server Management Studio (рис. 3.43 и 3.44).

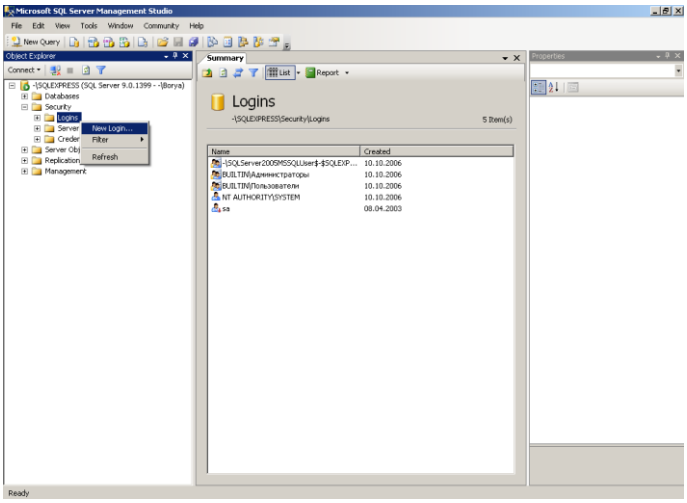
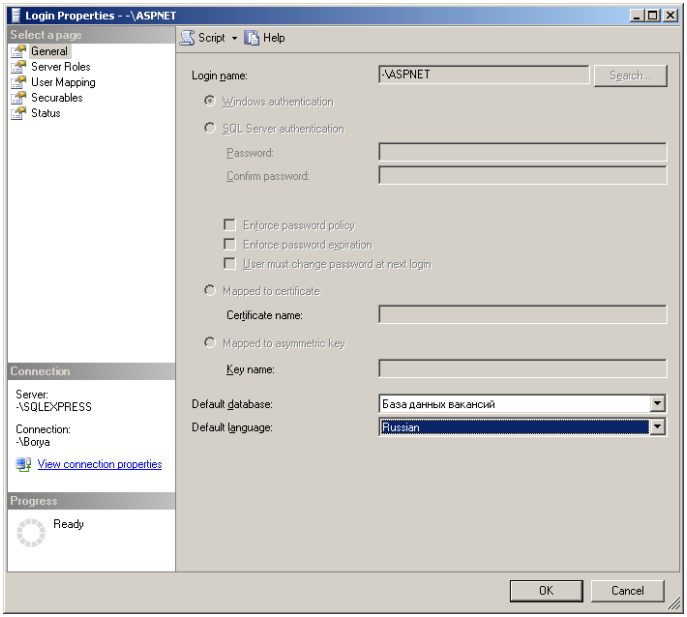
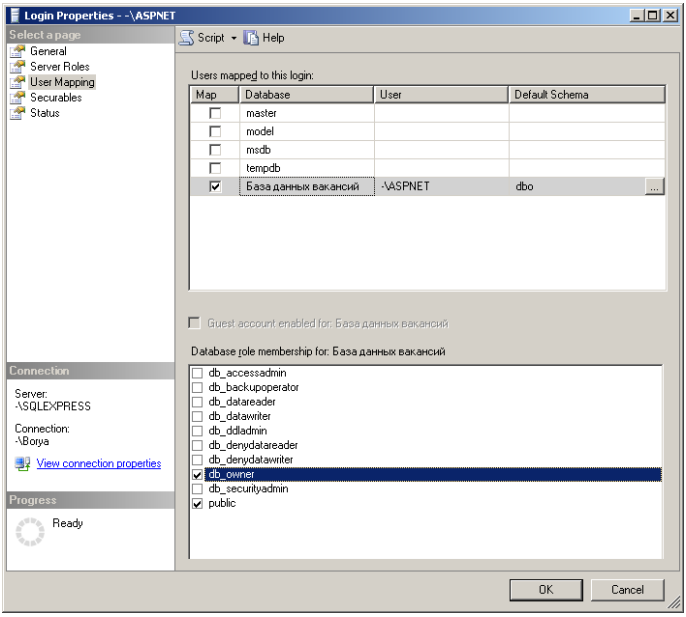


Рис. 3.43. Добавление имени новой учетной записи (сеанса)



а



б

Рис. 3.44. Настройка свойств учетной записи ASPNET:
а – указание исходных базы данных и языка; б – выбор ролевого членства для базы данных

Фиксированная роль db_owner предоставляет возможности всех ролей базы данных, в частности конфигурирования, сопровождения и выполнения хранимых процедур базы данных. Разрешения этой роли включают разрешения всех остальных фиксированных ролей базы данных. В последних версиях Microsoft SQL Server члены указанной роли могут удалять базу данных [1, 7–10].

Элементы управления для работы с источниками данных. В ASP.NET предусмотрено два элемента управления WebForm, предназначенных для отображения данных, полученных из источника (обычно в качестве источника в приложениях ASP.NET выступает объект ADO.NET DataSet, который, в свою очередь, может быть, например, заполнен данными с сервера баз данных). Эти элементы управления представлены в табл. 3.14.

Таблица 3.14

Элементы управления WebForm,
предназначенные для работы с источниками данных

Элемент управления	**Описание**
DataGrid	Элемент управления, который отображает содержимое объекта ADO.NET DataSet в виде таблицы
DataList	Элемент управления для выбора значений, заполняемый из источника данных

Кроме того, для работы с источниками данных можно настроить некоторые из базовых типов данных.

Элемент управления DataGrid. Одна из наиболее часто встречающихся задач Web-приложения – нахождение каких-то данных в источнике данных по запросу пользователя и возврат их в табличном формате. В классических ASP это делалось путем создания объекта ADO Recordset и таблицы HTML «на лету» с использованием данных из этого объекта. Тех же самых результатов гораздо проще можно достичь при помощи элемента управления WebForm – DataGrid.

Следует рассмотреть применение DataGrid на примере. Предполагается, что необходимо предоставить пользователю в ответ на его запрос данные из базы данных «База данных вакансий», созданной ранее. Первое, что нужно сделать, – создать обработчик для события Load страницы. В нем следует установить соединение с базой данных, создать и заполнить объект DataSet и указать его в качестве источника данных для элемента управления DataGrid. Соответствующий код на C# может выглядеть так:

```
protected void Page_Load(object sender, EventArgs e)
{
        if (!IsPostBack)
        {
                SqlConnection connection = new SqlConnection();

                connection.ConnectionString = @"Data Source=-
\SQLEXPRESS;
                Initial Catalog=База данных вакансий; Integrated
Security=sspi";

                SqlDataAdapter dsc = new SqlDataAdapter("Select * from
Организации",
                        connection);
```

```
                DataSet ds = new DataSet();

                dsc.Fill(ds, "Организации");
                DataGrid1.DataSource =
ds.Tables["Организации"].DefaultView;
                DataGrid1.DataBind();
        }
}
```

Возможный результат представлен на рис. 3.45.

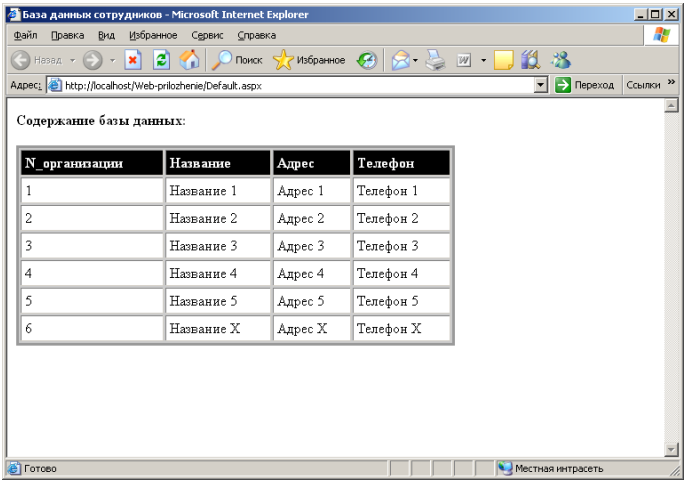


Рис. 3.45. Элемент управления DataGrid с данными, полученными из Microsoft SQL Server

Элемент управления ListBox. Следует привести еще некоторую информацию об источниках данных. Как можно было убедиться, выводить содержимое объекта DataTable при помощи элементов управления Web-Form (например, DataGrid) можно легко и просто. Однако достаточно часто возникает необходимость выводить на Web-странице данные, которые хранятся другими способами. И в элементах управления WebForm предусмотрена возможность делать это, т. е. выводить данные, которые находятся в каком угодно виде.

Например, предполагается, что возникла следующая ситуация: необходимо заполнить элемент управления ListBox данными, которые в настоящее время хранятся в обычном строковом массиве (такая потребность возникает очень часто). Заполнение ListBox данными из этого массива

производится точно так же просто, как DataGrid из объекта DataTable, приведенное выше:

```
protected void Page_Load(object sender, EventArgs e)
{
        if (!IsPostBack)
        {
                string[] Sotrudnik = { "Сотрудник 1", "Сотрудник 2",
"Сотрудник 3" };
                ListBox1.DataSource = Sotrudnik;
                ListBox1.DataBind();
        }
}
```

То, что может получиться, представлено на рис. 3.46.

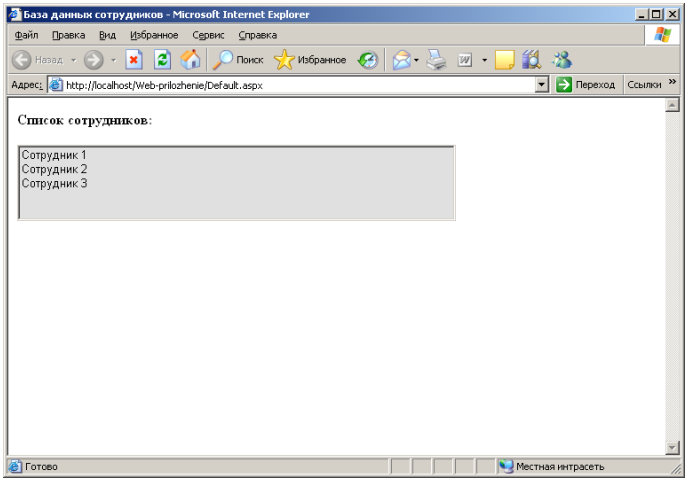


Рис. 3.46. Привязка данных к элементу управления ListBox

Все массивы .NET происходят от единого общего предка – класса System.Array, а в классе System.Array реализован интерфейс IEnumerable. Это упоминается к тому, что любой класс, в котором реализован этот интерфейс, может быть привязан к элементу управления WebForm (или Windows Forms) в качестве источника данных. Например, если данные находились в объекте ArrayList, все будет точно так же:

```
protected void Page_Load(object sender, EventArgs e)
{
        if (!IsPostBack)
        {
                ArrayList Sotrudnik = new ArrayList();
                Sotrudnik.Add("Сотрудник 1");
                Sotrudnik.Add("Сотрудник 2");
                Sotrudnik.Add("Сотрудник 3");
                ListBox1.DataSource = Sotrudnik;
                ListBox1.DataBind();
        }
}
```

Результат, естественно, остался тем же [1, 7–10].

Элементы управления для проверки вводимых пользователем данных. Последняя разновидность элементов управления WebForm – это элементы управления, которые применяются для проверки вводимых пользователем данных. Наиболее важные элементы управления этого типа представлены в табл. 3.15.

Таблица 3.15

Элементы управления для проверки данных

Элемент управления	Описание
CompareValidator	Сравнивает значение, введенное в один элемент управления, со значением, введенным во второй
CustomValidator	Позволяет определить пользовательский метод, при помощи которого будет производиться проверка
RangeValidator	Определяет, попадает ли введенное пользователем значение в определенный диапазон
RegularExpressionValidator	Проверяет введенное значение на соответствие подстановочному выражению
RequiredFieldValidator	Позволяет убедиться, что в соответствующий элемент управления действительно введено значение (оно не оставлено пустым)
ValidationSummary	Отображает все ошибки, обнаруженные при проверке ввода, в виде списка, маркированного списка или обычного абзаца. Ошибки могут отображаться на Web-странице или в специальном окне оповещения браузера

А теперь следует рассмотреть, как можно проверять правильность вводимых пользователем данных при помощи элементов управления WebForm. Например, нужно убедиться, что текстовое поле E-mail пользователь не оставил пустым. Проверка будет производиться при нажатии кнопки «Ввод», т. е. при попытке пользователя отправить данные на сервер (рис. 3.47).

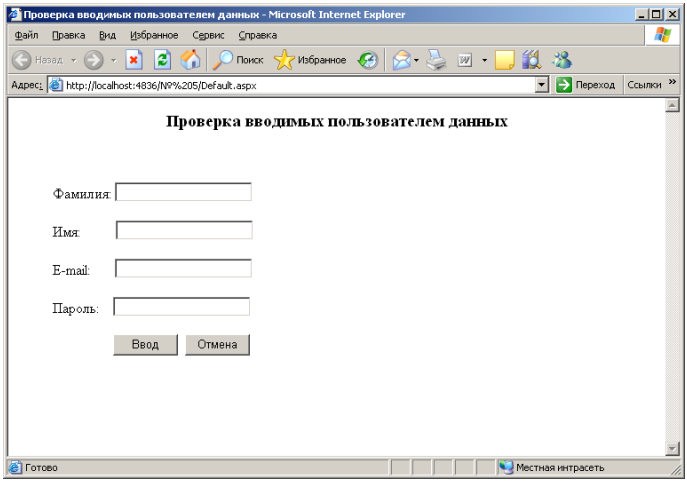


Рис. 3.47. Интерфейс для ввода данных

Самый простой способ реализовать такую проверку – воспользоваться элементом управления RequiredFieldValidator.

Следует добавить указанный объект на страницу при помощи Toolbox и открыть его свойства. Свойств, которые обязательно нужно настроить, два:

- ControlToValidate – имя текстового поля, обязательного для заполнения;
- ErrorMessage – сообщение об ошибке, которое будет выдаваться пользователю.

Код в файле ASPX, сгенерированный средой разработки Microsoft Visual Studio и измененный, может быть таким:

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
        ControlToValidate="TextBox1" ErrorMessage="Введите E-mail,
пожалуйста!">
</asp:RequiredFieldValidator>
```

При этом, как уже было упомянуто ранее, в последних версиях Microsoft Visual Studio в производном от Page классе C# для страницы ASP.NET (в том файле ASPX.CS, который указан в атрибуте CodeFile) новой переменной не появится.

После запуска проекта с виду ничто не изменится. Однако если попробовать нажать кнопку «Ввод», не заполнив текстовое поле E-mail, то на странице, например, рядом с этим полем появится сообщение об ошибке «Введите E-mail, пожалуйста!» (рис. 3.48).

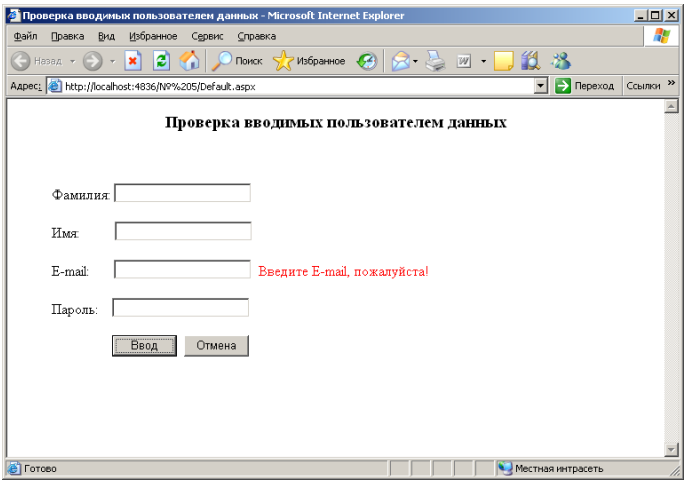


Рис. 3.48. Интерфейс для ввода данных с сообщением об ошибке

При вводе данных в указанное поле и повторном нажатии «Ввод» надпись исчезнет.

Если в окне браузера просмотреть код Web-страницы, то можно заметить в нем функции JavaScript, которые были созданы автоматически. Если же элемент управления WebForm обнаружит, что браузер не поддерживает работу со скриптами, то логика проверки вводимых данных будет реализована для выполнения на сервере [1, 7–10].

Обработка событий элементов управления. События, генерируемые элементами управления WebForm, можно перехватывать и обрабатывать двумя способами. Первый способ – делать все непосредственно в браузере клиента при помощи клиентских браузерных скриптов JavaScript. Это традиционный подход, который наиболее удобен в тех ситуациях, когда нужно выполнять форматирование на Web-странице, выводить оповещения в окне браузера или осуществлять прочие взаимодействия с объектной моделью,

реализованной в браузерах. Но элементы управления WebForm предлагают и другой способ – обрабатывать и перехватывать их события на сервере. Для этого достаточно добавить обработчик события при помощи окна Properties свойств элемента управления (пиктограмма Events). Обычно такой способ наиболее удобен для выполнения операций, не связанных с графическим интерфейсом – например, для производства каких-то вычислений, редактирования таблицы с данными т. п.

Следует рассмотреть применение обработки событий элементов управления WebForm на сервере. В качестве элемента управления будет использован Calendar, а реагировать можно на событие SelectionChanged. Выглядеть это может так:

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
        Label1.Text = (Calendar1.SelectedDate.Date.ToString());
}
```

Теперь при выборе пользователем даты в календаре сработает событие SelectionChanged и в окне браузера появится надпись, показанная на рис. 3.49.

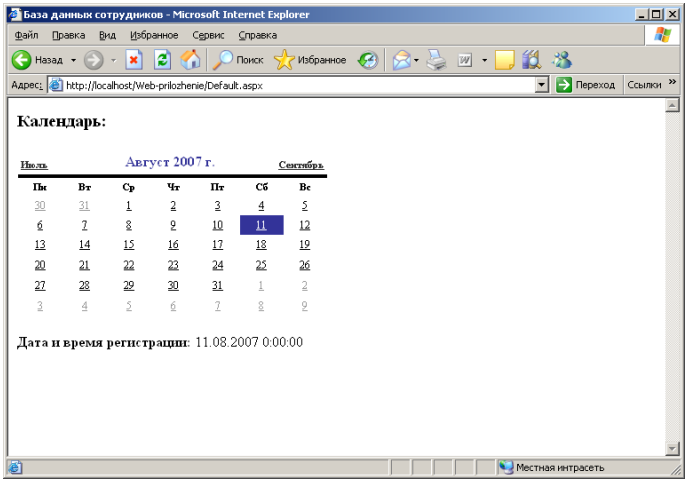


Рис. 3.49. Обработка событий элемента управления Calendar на сервере

Конечно, можно использовать для подобных целей и клиентские браузерные скрипты. Все зависит от ситуации – свои достоинства есть у каждо-

го способа. Клиентским скриптам не нужно обращаться на сервер, поэтому они будут работать гораздо быстрее. С другой стороны, серверные скрипты проще и надежнее.

Таким образом, создание Web-приложений требует во многом иных подходов, чем создание обычных настольных приложений. Разработка Web-приложений основана на использовании таких основных элементов, как теги HTML, протокол HTTP, клиентские браузерные скрипты и серверные скрипты классических ASP. При создании приложений ASP.NET необходимо учитывать то, что каждому шаблону HTML приложения (файлу ASPX) соответствует в ASP.NET класс, производный от System.Web.UI.Page. Работать с этим классом можно средствами привычных языков программирования .NET, например C#. Поэтому в ASP.NET теперь можно использовать технологии объектно-ориентированного программирования, создавая код, пригодный для повторного использования. Основные свойства объекта Page (Session, Application, Request и Response) обеспечивают доступ к внутренним объектам класса, производного от Page. Элементы управления WebForm во многом аналогичны стандартным элементам управления Windows Forms, с помощью которых можно избежать трудоемкой и утомительной обязанности создавать теги HTML и клиентские скрипты вручную [1, 7–10].

Задание. Создать новый проект C# на основе шаблона ASP.NET Web Site. В приложении в качестве стартовой страницы, вызывающей страницу ASPX с реализованной проверкой вводимых пользователем данных (см. рис. 3.47), использовать страницу HTML, на которой реализовать вызов страницы ASPX посредством нажатия кнопки. Хранение пароля реализовать в статическом виде на сервере. При неправильном указании пароля вывести новую страницу ASPX с сообщением об ошибке. В случае правильного ввода пароля вывести страницу ASPX с содержимым таблицы базы данных Microsoft Access, состоящей из двух полей (символьного и текстового). При этом должен производиться и выводиться на страницу подсчет среднего значения в текстовом поле. При подключении к базе данных можно воспользоваться либо непосредственным указанием источника данных, либо методом MapPath объекта Server, возвращающим физический путь к файлу, который соответствует определенному виртуальному пути (каталогу) на Web-сервере:

```
string connStr = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" +
Server.MapPath("База данных.mdb") + ";Persist Security Info=False";
```

4. НАЗНАЧЕНИЕ И ИСПОЛЬЗОВАНИЕ WEB-СЛУЖБ

4.1. Понятие, роль и инфраструктура Web-служб. Протокол подключения и службы описания и обнаружения. Обзор пространств имен. Пространство имен System.Web.Services. Исходный файл объектно-ориентированного языка программирования .NET. Реализация методов и особенности работы клиента. Тип WebMethodAttribute. Базовый класс System.Web. Services.WebService. Язык описания WSDL

Понятие, роль и инфраструктура Web-служб. Во многих отношениях эта глава объединяет многое из того, что рассмотрено в предыдущих главах. Web-службы ASP.NET – модули кода .NET (обычно установленные на сервере IIS), к которым возможно удаленное обращение по протоколу HTTP.

Как станет очевидным, Web-службы строятся на основе трех взаимосвязанных технологий:

- Web Service Description Language (WSDL, язык описания Web-служб);
- протоколов подключения (HTTP-GET, HTTP-POST и SOAP – Simple Object Access Protocol);
- служб обнаружения (Discovery service) и описания (Description service).

Вначале следует поработать с элементарным примером, создав Web-службу, которая выполняет роль калькулятора, а затем создать более изощренный пример Web-службы, связанной с базой данных, которая будет возвращать объекты ADO.NET DataSet, массивы ArrayList и пользовательские типы.

После того как будет освоено создание Web-службы, следует обратиться к созданию прокси-классов (при помощи утилиты wsdl.exe и среды Microsoft Visual Studio). Эти прокси-классы позволят обращаться к Web-службе как клиентам Web, так и другим клиентам, в том числе с помощью консольных приложений и обычных приложений Windows Forms.

Если посмотреть на Web-службу «снаружи», то это всего лишь блок кода, к которому можно обратиться по протоколу HTTP. Однако сама по себе эта формулировка значит уже очень многое. Подавляющее большинство используемых в настоящий момент технологий удаленной активации кода привязаны к конкретным протоколам (при этом требующих постоян-

ных и надежных соединений), платформам и языкам программирования. В DCOM для обращения к удаленным типам COM используется требующий высокоскоростных надежных соединений RPC. В CORBA используется несколько протоколов, но все они также требуют постоянного подключения и надежных соединений. EJB (Enterprise Java Beans) требует использования определенного протокола и языка программирования Java.

.NET сильно отличается от всех этих технологий. Прежде всего, как это уже очевидно, .NET обеспечивает большую степень языковой независимости, чем что-либо другое. Можно создавать при помощи любого языка программирования для работы с .NET типы, к которым можно будет обращаться из клиента на любом .NET-совместимом языке. Кроме того, для обращения к Web-службам ASP.NET нужно, чтобы на данной конкретной платформе был реализован протокол HTTP – и этого достаточно. При всем существующем разнообразии платформ и операционных систем вряд ли найдется платформа, на которой не был бы реализован HTTP.

Таким образом, Web-разработчик может обнаружить, что для создания Web-службы ASP.NET можно использовать определенный и привычный язык программирования. Клиента Web-служб должен удовлетворить тот факт, что для вызова методов типов Web-служб вполне можно обойтись стандартным HTTP. Кроме того, как вскоре обнаружится, во взаимодействии с протоколом HTTP можно также использовать стандартные XML и SOAP, что также немаловажно.

Web-служба строится из тех же типов, что и любая сборка .NET: классов, интерфейсов, перечислений и структур, которые играют для клиента роль «черного ящика», отвечающего на запросы. Единственное важное ограничение, о котором необходимо постоянно помнить, связано с тем, что Web-службы предназначены для обработки удаленных вызовов и поэтому в них следует избегать применения типов для работы с графическим интерфейсом. Web-службы в отличие от других приложений предназначены для другого: они должны уметь выполнить какое-либо действие по запросу пользователя (произвести вычисления, считать данные из источника и т. д.) и ждать следующего запроса.

Еще один важный момент, связанный с Web-службами, который необходимо осознать, состоит в том, что для них вовсе не обязательно использовать клиентов, работающих через браузер. К Web-службе вполне могут обращаться и обычные консольные или Windows-клиенты (локальные, терминальных служб и т. п.). Для этого в .NET предусмотрены специальные средства, которые позволяют генерировать так называемые прокси-сборки. Обращение к типам этой прокси-сборки происходит, как к обычному типу .NET, а она уже перенаправляет запрос в Web-службу (при по-

мощи HTTP или сообщений SOAP) и возвращает клиенту полученные результаты.

Web-службе (как и обычному приложению ASP.NET) обычно соответствует виртуальный каталог на сервере IIS. Однако для Web-службы потребуется также реализовать дополнительную поддерживающую инфраструктуру. К ней относятся:

- протокол подключения (HTTP-GET, HTTP-POST или SOAP);
- служба обнаружения (Discovery service) для возможности получения клиентом информации о том, что Web-служба существует;
- служба описания (Description service) для возможности получения клиентом информации о том, что делает определенная Web-служба.

Перед рассмотрением реализации каждой части инфраструктуры в примерах следует их кратко описать [1, 7].

Протокол подключения и службы описания и обнаружения. В Web-службах ASP.NET (как и в ADO.NET) стандартный формат передачи информации между службой и клиентом – это формат XML. Сама передача происходит при помощи протокола HTTP. Можно использовать различные протоколы подключения (методы) передачи данных – HTTP-GET, HTTP-POST и SOAP. Следует отметить, что ориентироваться целесообразнее на SOAP, поскольку при помощи этого протокола можно обеспечить передачу очень сложных типов (пользовательских классов, объектов ADO.NET DataSet, массивов объектов и т. п.).

При обращении к удаленной Web-службе клиент должен обладать полной информацией о членах типов Web-службы, которые предоставлены в его распоряжение. Например, клиент должен иметь информацию о том, что он может вызвать метод Foo(), а также все необходимые параметры этого метода: какие параметры этот метод принимает и что он возвращает. За предоставление клиенту этой информацци ответственна служба описания Web-службы. Как обычно, сама информация, которая используется для этого, предоставляется в формате XML (XML Schema) и называется WSDL.

Служба обнаружения позволяет клиенту обнаруживать Web-службы по адресу URL. Для этой службы используются файлы с расширением DISCO (от Discovery), опять-таки в формате XML. Синтаксис этих файлов будет приведен далее [1, 7].

Обзор пространств имен. Разработчики .NET заготовили множество типов, которые могут быть использованы как для построения самих Web-служб, так и для создания необходимой инфраструктуры. Эти типы определены в пространствах имен, представленных в табл. 4.1 [1, 7].

Таблица 4.1

Пространства имен для Web-служб

Пространство имен	Описание
System.Web.Services	В этом пространстве имен определен минимально достаточный набор типов для построения Web-службы
System.Web.Services.Description	Набор типов для программного взаимодействия с WDSL
System.Web.Services.Discovery	Эти типы обеспечивают клиенту Web-служб возможность программно обнаруживать Web-службы, установленные на конкретном компьютере, и используются вместе с файлами DISCO
System.Web.Services.Protocols	В этом пространстве имен определены типы, которые предназначены для работы с протоколами подключения HTTP-GET, HTTP-POST и SOAP

Пространство имен System.Web.Services. В большинстве проектов по созданию Web-служб единственные типы, с которыми придется взаимодействовать напрямую, – это типы пространства имен System.Web.Services. Этот набор типов не так уж велик и представлен в табл. 4.2.

Таблица 4.2

Типы пространства имен System.Web.Services

Тип	Описание
WebMethodAttribute	Добавление атрибута WebMethod в определение метода Web-службы означает, что этот метод может быть вызван удаленным клиентом по HTTP
WebService	Определяет необязательный базовый класс для Web-службы
WebServiceAttribute	Этот атрибут может быть использован для размещения информации о Web-службе (например, для строки, описывающей ее возможности) и является необязательным
WebServiceBindingAttribute	Объявляет связывающий протокол, который реализован методом Web-службы

Пример элементарной Web-службы. Перед тем как приступать к подробностям, следует сформировать мысленный образ того, о чем идет речь, и создать очень простую Web-службу. Для этого надо запустить Mi-

crosoft Visual Studio и создать новый шаблон ASP.NET Web Service, указав для него соответствующие местонахождение и язык программирования (рис. 4.1).

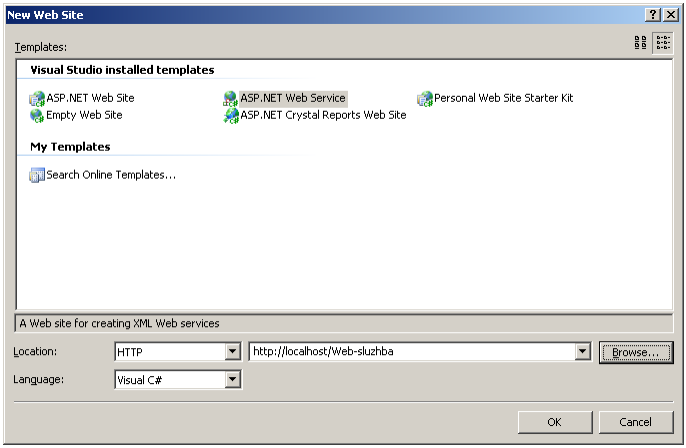


Рис. 4.1. Создание нового шаблона ASP.NET Web Service

Как для любого проекта (решения) ASP.NET, при создании проекта для работы с Web-службами Microsoft Visual Studio может автоматически сформировать виртуальный каталог для него на сервере IIS (если такой каталог не был создан), а ненужные в этом каталоге файлы проекта с расширениями SLN и SUO разместит в подкаталоге пользователя для хранения проектов.

Если необходимо использовать уже готовый исходный код, проще всего создать новый проект и импортировать туда готовые классы.

Среда разработки сгенерировала автоматически файлы с расширениями ASMX и CS. Кроме них могут быть созданы файлы Global.asax, предназначенный для организации реагирования на события глобального уровня (общие для всех сеансов подключения), и web.config, позволяющий в формате XML определить основные параметры приложения ASP.NET (в данном случае Web-службы). Вся реальная работа (как и в большинстве реальных проектов) будет производиться исключительно с тремя типами файлов: ASMX, CS и DISCO (табл. 4.3) [1, 7].

Таблица 4.3

Наиболее важные типы файлов проекта для работы с Web-службами

Тип файла	Описание
ASMX	Файл в XML-совместимом формате WSDL, в котором содержится информация о методах, предоставляемых пользователю, и на основе которого среда выполнения ASP.NET автоматически генерирует код HTML
CS	Обычный исходный файл C#, в котором хранятся определения методов, предоставляемых пользователям Web-службы. Файл ASMX ссылается на соответствующий ему файл CS при помощи атрибута CodeBehind
DISCO	XML-совместимый файл с описанием Web-служб, которые можно найти по указанному адресу URL

Исходный файл объектно-ориентированного языка программирования .NET. Как уже упоминалось ранее, одно из наиболее важных преимуществ ASP.NET – это возможность создавать Web-приложения при помощи полнофункциональных объектно-ориентированных языков программирования, а не ограничиваться скриптами (как в классических ASP). Файл ASMX можно представить как шаблон, на основе которого среда выполнения ASP.NET генерирует код HTML, передаваемый в браузер клиента. Для этого файла при помощи атрибута CodeBehind можно определить файл, в котором и будет реализована вся программная логика Web-службы на определенном языке программирования. В данном случае для Web-службы был выбран шаблон на C#, поэтому и файл фонового кода является исходным файлом C#:

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
        public Service () {

                //Uncomment the following line if using designed components
                //InitializeComponent();
        }
```

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

}
```

Для класса C# в качестве базового был выбран класс System.Web.Services.WebService. Следует заметить, что это совершенно не обязательно. Вполне можно создать Web-службу, обойдясь и без этого класса. Определение будет тем же самым, только класс будет теперь производиться напрямую от System.Object. Как можно убедиться, функциональности Web-службы это нисколько не повредит. Однако выбор в качестве базового класса WebService автоматически обеспечивает Web-службе очень полезный набор членов, с которым произойдет знакомство далее [1, 7].

Реализация методов и особенности работы клиента. В созданной Web-службе пока можно ничего не усложнять и ограничиться четырьмя методами, при помощи которых пользователь сможет производить элементарные арифметические операции. Все эти методы будут доступны по HTTP, но чтобы среда выполнения ASP.NET поняла, какие методы нужно предоставлять по HTTP для пользователей, эти методы нужно пометить атрибутом WebMethod. В общем, определение класса Service можно сделать таким:

```
public class Service : System.Web.Services.WebService
{

        public Service()
        {

                //Uncomment the following line if using designed components
                //InitializeComponent();
        }

        [WebMethod]

        public int Slozhenie(int x, int y)
        {
                return x + y;
        }

        [WebMethod]
```

```
        public int Vichitanie(int x, int y)
        {
                return x - y;
        }

        [WebMethod]

        public int Umnozhenie(int x, int y)
        {
                return x * y;
        }

        [WebMethod]

        public int Delenie(int x, int y)
        {
                if (y == 0)
                {
                        throw new DivideByZeroException("Деление на
ноль.");
                }
                return x / y;
        }

}
```

После того как Web-служба будет откомпилирована, ее следует запустить на выполнение (можно прямо в Microsoft Visual Studio). По умолчанию в качестве клиента будет открыто окно браузера, а в нем откроется страница HTML со списком всех методов, которые помечены атрибутом WebMethod (рис. 4.2).

Можно не только просматривать список методов Web-службы, но и вызывать их прямо из браузера – это уже реализовано в среде выполнения ASP.NET. Например, можно перейти по гиперссылке на Slozhenie и ввести в текстовые поля значения (рис. 4.3).

Осталось только нажать кнопку Invoke, и среда выполнения ASP.NET вызовет метод, передаст ему введенные значения и вернет результат в формате XML (рис. 4.4).

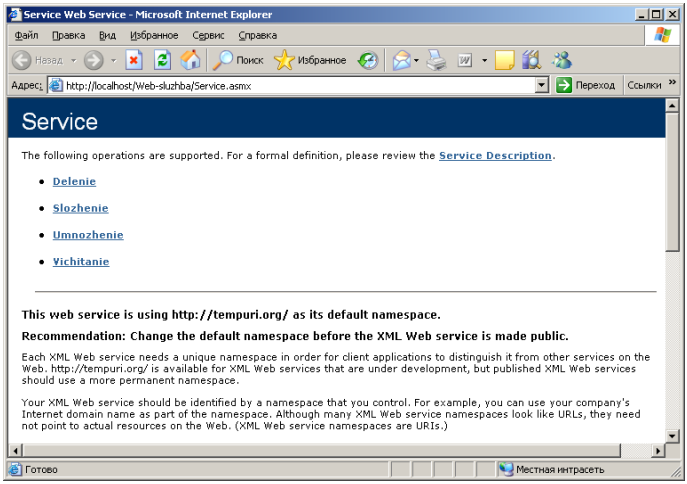


Рис. 4.2. Подключение клиента к Web-службе

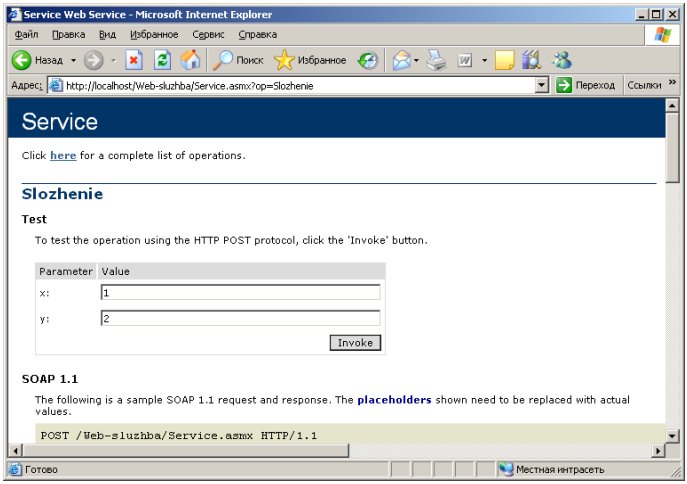


Рис. 4.3. Автоматическая генерация текстовых полей для ввода параметров метода и прочего необходимого кода средой выполнения ASP.NET

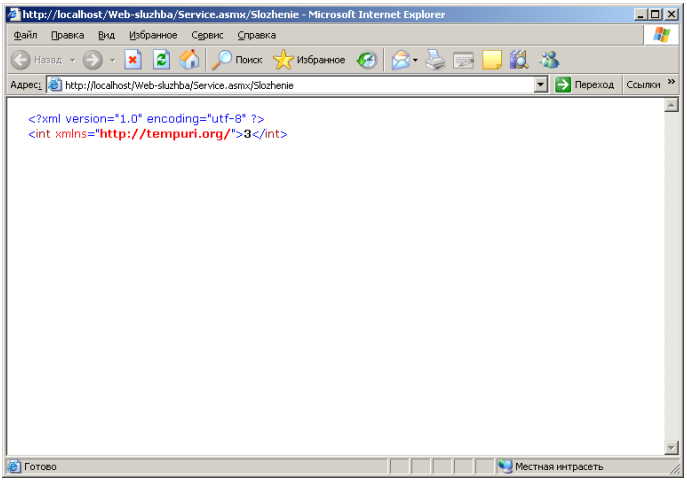


Рис. 4.4. Результат вычислений, возвращаемый в формате XML

Конечно же, метод можно вызывать и не используя графический интерфейс. Например, если посмотреть, какой запрос был отправлен в Web-службу, то он будет выглядеть так:

http://localhost/Web-sluzhba/Service.asmx/Slozhenie

Как видно, запрос состоит из адреса страницы ASMX с добавлением имени метода. Пары «имя-значение», представляющие параметры метода, передаются неявно, так как используется метод POST, при котором для данных формы выделяется отдельная строка в поле заголовка HTTP.

Таким образом, очевидно, что создать Web-службу в ASP.NET – это очень просто. Следует также поработать с примером более серьезной службы, однако перед этим целесообразно обратиться к некоторым особенностям архитектуры Web-служб ASP.NET [1, 7].

Тип WebMethodAttribute. Атрибут WebMethod, представленный типом WebMethodAttribute, должен обязательно быть указан для каждого метода Web-службы, предоставляемого в распоряжение клиента. Как большинство других атрибутов .NET, для атрибута WebMethod можно использовать дополнительные параметры, которые на самом деле являются параметрами конструктора WebMethodAttribute. Например, можно предоставить клиенту дополнительную информацию о методе Web-службы:

```
[WebMethod(Description = "Сложение двух чисел (целых).")]
```

```
public int Slozhenie(int x, int y)
{
        return x + y;
}
```

Параметр Description атрибута WebMethod очень похож на атрибут Helpstring в IDL. А результатом его применения будет дополнительная информация о методе на Web-странице (рис. 4.5).

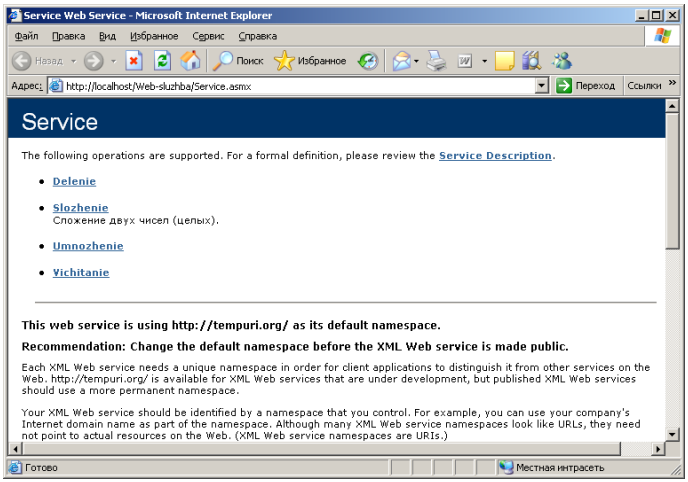


Рис. 4.5. Параметр Description в действии

Если поинтересоваться, что же происходит при добавлении к методу параметра Description, то окажется, что в коде WSDL (об этом – позже) файла ASMX появился дополнительный тег <documentation>:

```
<wsdl:documentation
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Сложение двух
чисел (целых).</wsdl:documentation>
```

Параметр Description – не единственный, который можно использовать для атрибута WebMethod. Наиболее важные параметры этого атрибута представлены в табл. 4.4.

Таблица 4.4

Параметры атрибута WebMethod

Параметр	Описание
Description	Позволяет добавить дружественное описание для метода Web-службы
EnableSession	При установленном значении true (по умолчанию) для метода можно использовать данные сеанса подключения (например, определять, сколько раз в течение сеанса он был вызван пользователем)
MessageName	Этот параметр можно использовать, чтобы определить, как метод Web-службы будет представлен в коде WSDL. Обычно он применяется для того, чтобы не допустить конфликты имен
TransactionOption	Методы Web-службы могут использоваться в качестве корня транзакции COM+. Для этого параметра используются значения из перечисления System.EnterpriseServices.TransactionOption

Следует рассмотреть на примере, для чего нужен параметр MessageName. Предполагается, что в Web-приложении появился дополнительный метод Slozhenie() для сложения двух значений с плавающей запятой:

```
[WebMethod(Description = "Сложение двух чисел (с плавающей запятой).")]

public float Slozhenie(float x, float y)
{
        return x + y;
}
```

Если попробовать откомпилировать проект, то никаких ошибок не будет – с точки зрения C# все в порядке. Однако если попытаться обратиться к Web-службе из браузера, можно увидеть не совсем то, на что рассчитывалось (рис. 4.6).

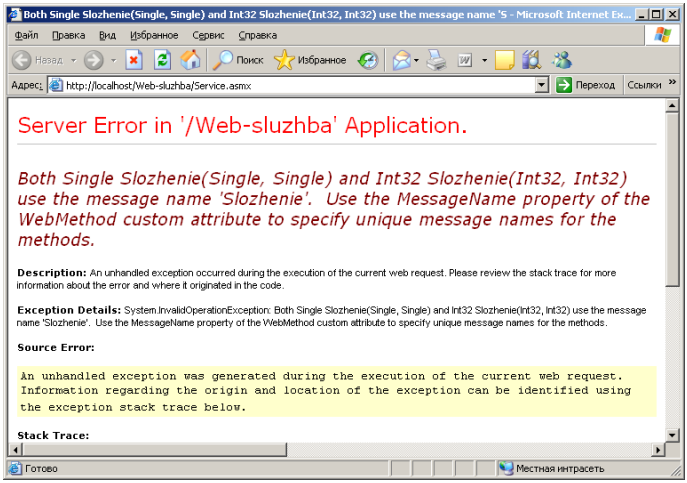


Рис. 4.6. Сообщение об ошибке при использовании одинаковых имен

Конфликт имен произошел на уровне WSDL: необходимо, чтобы для каждого атрибута, который используется для идентификации метода Web-службы, использовалось уникальное значение, т. е. имя Web-метода. Однако по умолчанию значением этого атрибута считается имя метода в его определении в C#. Чтобы решить проблемы с конфликтом имен, достаточно для одного из методов добавить параметр MessageName и его значение:

```
[WebMethod(Description = "Сложение двух чисел (с плавающей запятой).",
MessageName = "Сложение")]

public float Slozhenie(float x, float y)
{
        return x + y;
}
```

При этом необходимо отключить соответствие возможности взаимодействия Web-сервисов разных поставщиков, в частности основной спецификации (профилю):

```
[WebServiceBinding(ConformsTo = WsiProfiles.None)]
```

После этого идентификаторы методов в WSDL станут разными:

```
<wsdl:operation name="Slozhenie">
        <soap12:operation soapAction="http://tempuri.org/Slozhenie"
        style="document" />
        <wsdl:input>
                <soap12:body use="literal" />
        </wsdl:input>
        <wsdl:output>
                <soap12:body use="literal" />
        </wsdl:output>
</wsdl:operation>

<wsdl:operation name="Slozhenie">
        <soap12:operation soapAction="http://tempuri.org/Сложение"
        style="document" />
        <wsdl:input name="Сложение">
                <soap12:body use="literal" />
        </wsdl:input>
        <wsdl:output name="Сложение">
                <soap12:body use="literal" />
        </wsdl:output>
</wsdl:operation>
```

Если потребуется использовать описание для всей Web-службы, а не для отдельного метода, можно использовать для класса C# атрибут WebService с аналогичным параметром Description, например, так:

```
[WebService(Description = "Web-служба")]
```

Если запустить Web-службу после этого добавления, результат будет таким, как показано на рис. 4.7.

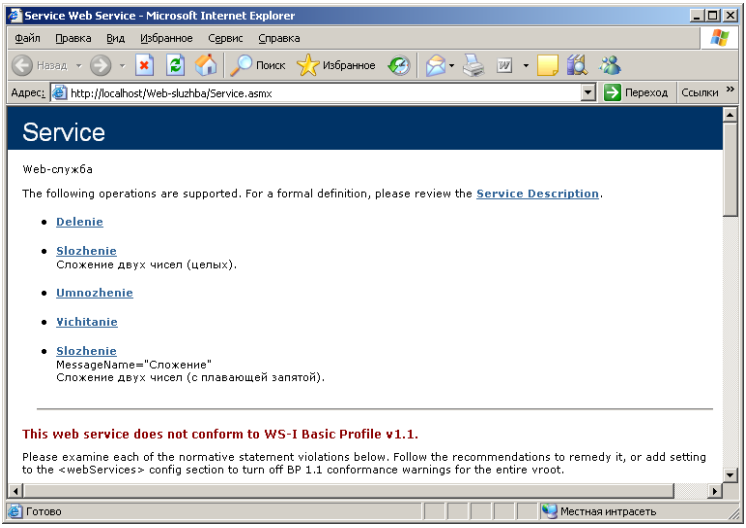


Рис. 4.7. Параметр Description атрибута WebService определяет описание для Web-службы в целом

Задание. Реализовать в Web-службе (в определении класса Service) с использованием объекта Application для текущего запроса HTTP дополнительный метод Chislo_Pi() для вывода на экран значения числа π в формате 64-разрядного значения с плавающей запятой [1, 7].

Базовый класс System.Web.Services.WebService. Как уже можно было заметить, по умолчанию Microsoft Visual Studio производит класс Web-службы от базового класса System.Web.Services.WebService. В принципе Web-служба будет работать и без этого класса в качестве базового, однако в нем предусмотрен набор очень полезных членов и вложенных типов, которые обеспечивают Web-службе те же возможности, которые имеются у стандартного приложения ASP.NET. Обычно в программном коде приходится взаимодействовать со свойствами класса Web-службы, унаследованными от WebService. Наиболее важные свойства класса WebService представлены в табл. 4.5 [1, 7].

Таблица 4.5

Наиболее важные свойства класса WebService

Свойство	Описание
Application	Возвращает ссылку на объект HttpApplicationState для текущего запроса HTTP
Context	Возвращает ссылку на объект HttpContext. Этот объект можно использовать для получения разнообразной информации о контексте выполнения запроса на сервере IIS
Server	Возвращает ссылку на объект HttpServerUtility, который можно использовать для получения информации о сервере, на котором выполняется запрос
Session	Возвращает ссылку на объект HttpSessionState. Используется для получения информации о текущем сеансе подключения
User	Возвращает объект ASP.NET User, который может быть использован для получения информации о пользователе, работающем с Web-службой, или, например, для целей аутентификации

Язык описания WSDL. Во всех программных технологиях, использующих межъязыковое взаимодействие, используются специальные средства для описания программных модулей и типов в них. В COM для этого применяется язык IDL, в обычных приложениях .NET – метаданные сборки (манифест) и типов. Для Web-служб ASP.NET такое специальное средство также предусмотрено: это WSDL.

WSDL – это XML-совместимый язык, который полностью описывает для внешних клиентов возможности Web-служб, методы, которые клиенты могут вызвать, а также поддержку протоколов подключения к Web-службам (HTTP-GET, HTTP-POST и SOAP).

Сразу следует отметить, что код WSDL в ASP.NET генерируется автоматически. Например, в рассмотренном примере WSDL-описание Web-службы можно получить прямо из окна браузера. Если открыть в браузере страницу Service.asmx, то в верхней части страницы будет гиперссылка Service Description, перейдя по которой, можно прочитать код WSDL для соответствующей Web-службы (рис. 4.8).

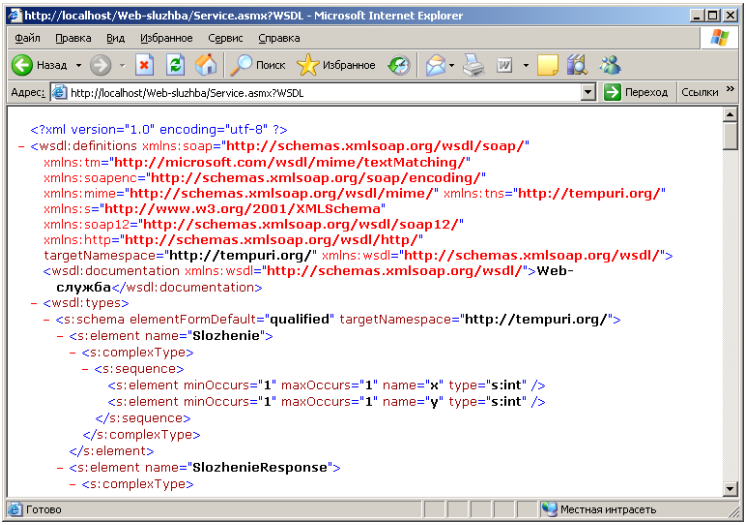


Рис. 4.8. Описание Web-службы в формате WSDL

Поскольку код WSDL генерируется автоматически, досконально разбирать все возможности WSDL не следует, но все же обратить внимание на наиболее принципиальные моменты стоит.

Прежде всего любое определение Web-службы на WSDL начинается с тега <definitions>. Далее следуют ссылки на узлы, определяющие протоколы подключения к Web-службе. За ними предусмотрены определения WSDL для каждого метода Web-службы в терминах протоколов подключения HTTP-GET, HTTP-POST и SOAP. Для каждого метода предусмотрены отдельные определения In (для приема данных от клиента) и Out (для передачи данных клиенту), при этом каждая пара перечисляется отдельно для каждого протокола подключения. Например, пара «In-Out» в WSDL-определении метода Slozhenie() для протокола подключения SOAP выглядит следующим образом:

```
<wsdl:message name="SlozhenieSoapIn">
        <wsdl:part name="parameters" element="tns:Slozhenie" />
</wsdl:message>
<wsdl:message name="SlozhenieSoapOut">
        <wsdl:part name="parameters" element="tns:SlozhenieResponse" />
</wsdl:message>
```

В WSDL, как и во многие другие метаязыки, глубоко вникать есть смысл в том случае, если существует необходимость создания чего-нибудь вроде собственного анализатора кода WSDL или просмотрщика типов ASP.NET. Если создание таких программных продуктов действительно входит в планы, можно также посоветовать обратиться к типам пространства имен System.Web.Services.Description. Эти типы библиотеки базовых классов .NET позволяют удобно манипулировать кодом WSDL непосредственно из программы [1, 7].

4.2. Протоколы подключения к Web-службам. Обмен данными при помощи HTTP-GET и HTTP-POST. Обмен данными при помощи SOAP. Работа с прокси-сборками. Сериализация пользовательских типов. Настройка клиента. Протокол обнаружения

Протоколы подключения к Web-службам. Как уже было отмечено, основная задача любой Web-службы ASP.NET – вернуть клиенту запрашиваемые им данные по протоколу HTTP. Однако для обмена данными между клиентом и сервером можно использовать три разных метода (они и называются протоколами подключения – Wire protocols): HTTP-GET, HTTP-POST и SOAP. Для удобства информация о них сведена в табл. 4.6.

Таблица 4.6

Протоколы подключения к Web-службам

Протокол подключения	Характеристика
HTTP-GET	При использовании этого метода данные добавляются к адресной строке URL
HTTP-POST	Данные добавляются в специальное поле заголовка HTTP
SOAP	Данные передаются в XML-совместимом формате SOAP

Выбор протокола подключения определяет то, какими типами (передаваемыми методам Web-служб и возвращаемыми ими клиентам) смогут обмениваться клиент и Web-служба. Можно сделать лишь общее замечание о том, что наибольшие возможности обеспечивает протокол SOAP. Однако вначале следует рассмотреть применение протоколов HTTP-GET и HTTP-POST [1, 7].

Обмен данными при помощи HTTP-GET и HTTP-POST. Как упоминалось ранее, при передаче данных методом HTTP-GET эти данные дописываются к строке запроса в формате URL в виде пар «имя-значение» (сам адрес отделяется от набора значений вопросительным знаком («?»). При применении HTTP-POST данные передаются при помощи тех же пар «имя-значение», только они помещаются в специальное поле заголовка протокола HTTP. При использовании обычных методов возвращаемый клиенту результат всегда формируется в простом формате XML в виде <имя_типа>Значение</имя_типа>.

HTTP-GET и HTTP-POST – методы очень простые и многим разработчикам хорошо знакомые. Однако их возможности оставляют желать лучшего: с их помощью невозможно передавать сложные данные, такие как структуры или экземпляры объектов. Фактически клиент и Web-служба могут обмениваться только типами, представленными в табл. 4.7.

Таблица 4.7

Типы данных, которые можно передавать
при помощи протоколов HTTP-GET и HTTP-POST

Тип	Комментарий
Перечисления	Передача объектов типов, производных от System.Enum, вполне возможна, но следует учитывать, что эти объекты передаются как обычные строковые значения
Простые массивы	Можно передавать только массивы примитивов (но не объектов пользовательских типов)
Строковые значения	Строковые значения передаются без проблем. При помощи строковых значений также передаются и значения других типов данных CLR, таких как Int16, Int32, Int64, Boolean, Single, Double, Decimal, DateTime и многих других

Передача данных при помощи HTTP-GET и HTTP-POST производится очень просто и аналогично рассмотренной ранее. Создается Web-страница с формой HTML, а в качестве получателя для этой формы указывается файл ASMX. Метод передачи данных определяется при помощи атрибута Method тега <form>. Например, содержание Web-страницы, которая будет принимать от пользователя два значения и передавать их методу Slozhenie() Web-службы, может быть следующим:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
<head>
        <title>
                Untitled Page
        </title>
</head>

<body>

        <form name="MainForm"
        action=http://localhost/Web-sluzhba/Service.asmx/Slozhenie
        method="POST">
                <p>
                        <strong>
                                Сложение двух чисел (целых)
                        </strong>
                </p>
                <p>
                        Первое число:
                        <input id="Text1" name="x" type="text">
                </p>
                <p>Второе число:
                        <input id="Text2" name="y" type="text">
                </p>
                <input id="Submit1" type="submit" value="Расчет" />
        </form>

</body>

</html>
```

То, как выглядит эта страница (пользовательский интерфейс Web-службы), показано на рис. 4.9.

Как известно, в качестве получателя данных формы вполне допускается указывать не только адрес страницы ASMX, но и имя вызываемого метода. Кроме того, для каждого из текстовых полей при помощи атрибута Name определено имя параметра, которому оно соответствует.

Результат произведенных Web-службой вычислений по виду аналогичен тому, который возвращает среда выполнения ASP.NET в автоматическом режиме. Следует обратить внимание на формат адресной строки браузера, который доказывает, что при передаче клиентом данных использовался метод HTTP-POST [1, 7].

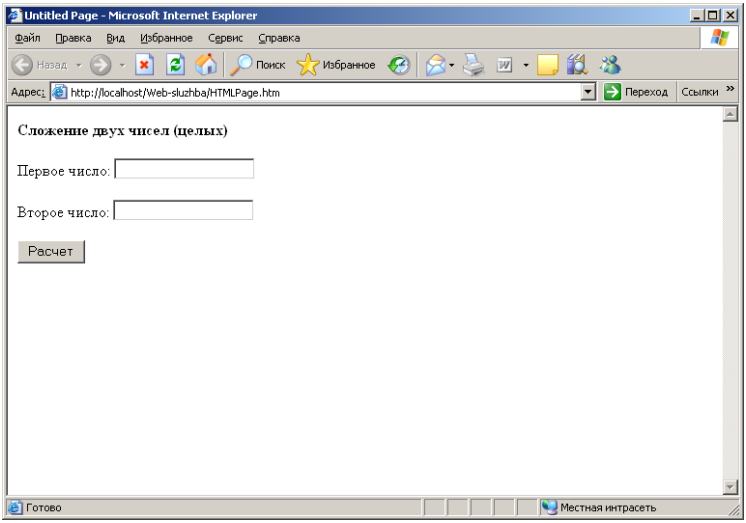


Рис. 4.9. Форма HTML для передачи данных в Web-службу

Обмен данными при помощи SOAP. Гораздо более привлекательная альтернатива методам HTTP-GET и HTTP-POST – использование протокола подключения SOAP. Отличительной особенностью этого протокола является то, что с его помощью можно обмениваться сложными типами данных. Можно передавать как все те же типы данных, что и с помощью HTTP-GET и HTTP-POST, так и дополнительные, которые представлены в табл. 4.8.

Таблица 4.8

Типы, которые можно передавать при помощи SOAP

Тип	Комментарий
Сложные массивы	Можно передавать массивы классов, структур и узлов XML
Пользовательские типы	Можно переносить объекты пользовательских классов (в частности ADO.NET DataSet для обмена данными), структуры и узлы XML
Узлы XML	Можно передавать любые узлы XML

Вся передача информации производится в XML-совместимом формате. Следует отметить наиболее важные моменты при использовании SOAP без подробного его рассмотрения.

В первую очередь необходимо указать на то, что SOAP изначально создавался как очень простой в обращении протокол подключения. Кроме того, он, как и все, что связано с XML, абсолютно независим от платформ, операционных систем, языков программирования и протоколов передачи данных. Например, данные SOAP можно передавать с помощью практически любых Internet-протоколов (HTTP, SMTP и т. п.).

Любое описание в формате SOAP имеет два аспекта: информация, относящаяся к самому сообщению в целом, и данные в формате XML, относящиеся к составным частям данного сообщения.

Например, при использовании протокола SOAP для вызова метода Slozhenie() определение этого метода в SOAP будет выглядеть так, как представлено ранее (с использованием специальных тегов и пар «имя-значение»).

При открытии кода WSDL для Web-службы ближе к концу страницы можно найти записи (узлы XML), которые описывают привязки Web-службы к протоколам подключения HTTP-GET, HTTP-POST и SOAP и означают то, что Web-служба будет работать по каждому из этих протоколов. Они описываются тегами <binding> с атрибутами Name и Type.

Например, если открыть узел для привязки SOAP версии 1.2, там можно будет найти код для метода Slozhenie(), который был приведен ранее при рассмотрении изменения идентификаторов методов в WSDL. Следует обратить внимание на теги <input> и <output>.

Разобраться с тегами и атрибутами SOAP можно, дополнительно изучив его спецификацию, например, посредством Internet (как очевидно, адреса URL с этой информацией помещаются в каждый файл WSDL). Для облегчения работы на практике, как это стало известно, требуемый код SOAP генерируется Microsoft Visual Studio автоматически, и нет необходимости править его вручную [1, 7].

Работа с прокси-сборками. Как можно было убедиться, и передача данных клиентом (при работе по SOAP), и получение их (при работе по любому протоколу подключения) производятся в XML-совместимых форматах. Конечно, можно создавать клиента для Web-служб с модулями компоновки кода в XML и анализа возвращаемых узлов XML, но это довольно трудоемкое занятие. Кроме того, немного странно использовать на клиенте код, к примеру, C#, потом преобразовывать промежуточные результаты в XML, чтобы с их помощью вызывать опять-таки методы C#, но уже на Web-службе. Намного более удобно было бы работать напрямую.

В связи с этим рекомендуется не обращаться на Web-службу из клиента напрямую (хотя это тоже не запрещено), а вначале создать промежуточную прокси-сборку (файл) на привычном C# или другом языке программирования и обращаться именно к ней, а она уже будет сама перенаправлять запро-

сы на Web-службу, получать возвращаемые результаты и передавать их вызывающему клиенту, который может быть каким угодно: обычным консольным клиентом, клиентом Windows Forms или WebForm (ASP.NET). Самое замечательное во всем этом то, что прокси-сборка создается автоматически – при помощи Microsoft Visual Studio или специализированной утилиты wsdl.exe. Эта программа предназначена для генерации кода для клиентов Web-служб XML и непосредственно Web-служб XML, использующих ASP.NET, из файлов контрактов WSDL, схем XSD (XML Schema Document) и документов обнаружения DISCOMAP. Этот инструмент может быть использован в совокупности с утилитой disco.exe, предназначенной для обнаружения адресов URL Web-служб XML, расположенных на Web-сервере, и формирования указанных выше файлов.

Создание прокси-сборки при помощи утилиты wsdl.exe. Один из возможных способов сгенерировать прокси-сборку – воспользоваться утилитой wsdl.exe, входящей в состав .NET SDK. Минимально необходимый синтаксис ее очень прост:

wsdl.exe /out:<имя_устройства>:\<каталог>\<файл_CS> <адрес_URL>

Параметр out – это, конечно, имя создаваемого исходного файла для прокси-сборки. Далее указывается адрес URL, по которому может быть получено описание Web-службы в формате WSDL (например, «http://localhost/Web-sluzhba/Service.asmx»). С полным списком параметров утилиты wsdl.exe можно ознакомиться, запустив ее с параметром «?». Некоторые дополнительные параметры командной строки для указанной утилиты приведены в табл. 4.9.

Таблица 4.9

Параметры командной строки утилиты wsdl.exe

Параметр	Описание
Language	Определяет язык, на котором будет сгенерирован исходный код для прокси-сборки. По умолчанию генерируется код прокси-класса именно на C#, однако по желанию может быть сгенерирован код и на некоторых других языках, например Visual Basic (значение VB), JavaScript, (значение JS) C++ (значение CPP) и т. д.
Namespace	Определяет пространство имен для генерируемой сборки. По умолчанию будет использовано глобальное пространство имен

Продолжение таблицы 4.9

Out	Определяет имя файла, в котором будет сохранен сгенерированный код. По умолчанию используется имя файла, создаваемое на основе имени Web-службы
Protocol	Определяет, по какому протоколу создаваемая прокси-сборка будет обращаться на Web-службу. По умолчанию используется значение SOAP, можно также использовать значения SOAP12, HttpGet и HttpPost. Можно использовать также свой собственный пользовательский протокол, но тогда его надо будет определить в специальном файле конфигурации

Исходный код генерируемой прокси-сборки. В отличие от множества других прокси-сборок, предусмотренных в .NET (для работы с COM, COM+ и т. п.), прокси-сборки для Web-служб создаются в виде файлов с исходным кодом, а не откомпилированным. Поэтому содержимое этих генерируемых файлов можно посмотреть. Например, начало содержимого сгенерированной прокси-сборки может выглядеть следующим образом (следует обратить внимание на атрибут WebServiceBindingAttribute):

```
//------------------------------------------------------------------------------
// <auto-generated>
//         This code was generated by a tool.
//         Runtime Version:2.0.50727.42
//
//         Changes to this file may cause incorrect behavior and will be lost if
//         the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.ComponentModel;
using System.Diagnostics;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Serialization;

//
// This source code was auto-generated by wsdl, Version=2.0.50727.42.
//

/// <remarks/>
```

```
[System.CodeDom.Compiler.GeneratedCodeAttribute("wsdl", "2.0.50727.42")]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="ServiceSoap",
Namespace="http://tempuri.org/")]
public partial class Service :
System.Web.Services.Protocols.SoapHttpClientProtocol {

        private System.Threading.SendOrPostCallback
SlozhenieOperationCompleted;

        private System.Threading.SendOrPostCallback
Slozhenie1OperationCompleted;

        private System.Threading.SendOrPostCallback
VichitanieOperationCompleted;

        private System.Threading.SendOrPostCallback
UmnozhenieOperationCompleted;

        private System.Threading.SendOrPostCallback
DelenieOperationCompleted;

        private System.Threading.SendOrPostCallback
Chislo_PiOperationCompleted;

        /// <remarks/>
        public Service() {
                this.Url = "http://localhost/Web-sluzhba/Service.asmx";
        }
```

Как видно, информация о том, где находится Web-служба, помещается при помощи свойства Url в объект класса прямо во время работы конструктора. Еще один важный момент: класс прокси-сборки – это класс, производный от базового класса SoapHttpClientProtocol. В этом базовом классе и определены те возможности, с помощью которых прокси-сборка будет взаимодействовать с Web-службой. Некоторые наиболее важные члены, унаследованные от указанного базового класса, представлены в табл. 4.10.

Таблица 4.10

Наиболее важные члены, унаследованные от класса SoapHttpClientProtocol

Член	Описание
BeginInvoke()	Начинает асинхронный вызов метода Web-службы
EndInvoke()	Заканчивает асинхронный вызов метода Web-службы
Invoke()	Производит синхронный вызов метода Web-службы
Proxy	Позволяет получить или установить информацию о настройках прокси-сервера для обращения к Web-службе через брандмауэр
Timeout	Позволяет получить или установить тайм-аут (в миллисекундах), в течение которого прокси-сборка будет ожидать возврата результатов при синхронном вызове метода Web-службы
Url	Позволяет получить или установить адрес URL для Web-службы
UserAgent	Позволяет получить или установить значение заголовка User agent для производимых запросов (т. е. каким браузером будет представляться данная прокси-сборка Web-службе)

В сгенерированной прокси-сборке автоматически создаются определения методов для синхронного и асинхронного вызова каждого метода Web-службы. Очевидно, что при синхронном вызове метода выполнение приостанавливается до возврата результатов с Web-службы, а при асинхронном методе сразу после вызова управление возвращается клиенту. Когда выполнение метода, вызванного асинхронным способом, завершено, среда выполнения производит обратный вызов прокси-сборки. Реализация синхронного вызова метода Slozhenie() в сгенерированной прокси-сборке будет выглядеть так:

```
[System.Web.Services.Protocols.SoapDocumentMethodAttribute
 ("http://tempuri.org/Slozhenie", RequestNamespace="http://tempuri.org/",
ResponseNamespace="http://tempuri.org/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
public int Slozhenie(int x, int y) {
        object[] results = this.Invoke("Slozhenie", new object[] {
                                        x,
                                        y});
        return ((int)(results[0]));
}
```

Как видно, определение каждого метода, который нужно перенаправить на Web-службу, помечается при помощи атрибута SoapDocumentMethodAttribute. У метода Slozhenie() такая же сигнатура, как у исходного метода Slozhenie() на Web-службе. Это очень удобно: как только клиенту потребуется обратиться к методу на Web-службе, он просто привычными средствами вызывает этот метод в прокси-сборке, а та уже передает запрос на Web-службу и возвращает клиенту полученные оттуда результаты.

Скорее всего, единственное изменение, которое может вызвать желание внести в сгенерированный исходный файл прокси-сборки, – поместить все его содержимое в специальное пространство имен. Это можно сделать опять-таки автоматически, указав имя пространства имен при помощи параметра namespace утилиты wsdl.exe.

Компиляция прокси-сборки. Прежде чем создать клиента, работающего через прокси-сборку, необходимо эту прокси-сборку окончательно сформировать, т. е. скомпилировать соответствующий файл. Это можно сделать при помощи Microsoft Visual Studio (через новый проект типа Visual C# с шаблоном Class Library) или компилятора командной строки csc.exe. В любом случае необходимо не забыть добавить ссылки на сборки System.Web.Services.dll и System.XML.dll:

```
csc.exe /out:<имя_устройства>:\<каталог>\<файл_DLL>
/target:library <имя_устройства>:\<каталог><файл_CS>
/reference:System.Web.Services.dll /reference:System.XML.dll
```

В результате будет создана прокси-сборка в виде библиотеки типов с прокси-классом (рис. 4.10).

Создание клиента для работы через прокси-сборку. Основное назначение прокси-сборки – облегчить создание клиента. И действительно, клиент, работающий через прокси-сборку, создается достаточно просто. Он может быть любым – обычным консольным клиентом, клиентом Windows Forms или WebForm (ASP.NET). Например, для создания самого простого консольного клиента следует в решении добавить ссылку на откомпилированную прокси-сборку, проверив в ней ссылку на сборку System.Web.Services.dll, и написать следующий код:

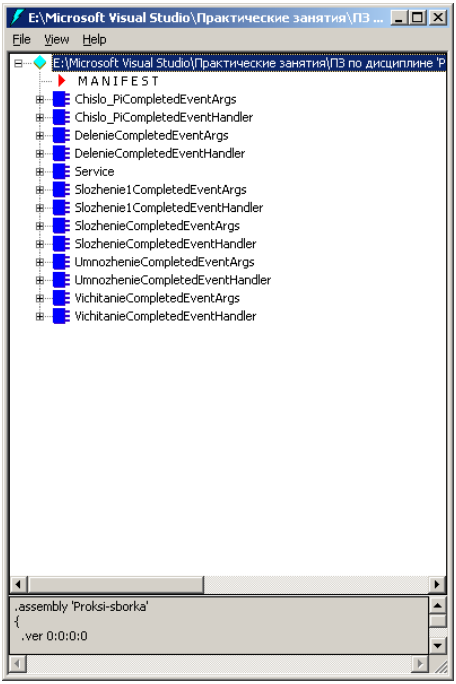


Рис. 4.10. Прокси-сборка в окне Microsoft .NET Framework IL Disassembler

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{

        class Program
        {

                static void Main(string[] args)
                {

                        Service w = new Service();

                        Console.WriteLine("1 + 2 = {0}", w.Slozhenie(1, 2));
                        try
                        {
```

```
                    w.Delenie(0, 0);
                }
                catch (DivideByZeroException e)
                {
                    Console.WriteLine(e.Message);
                }
            }

        }

}
```

Результат выполнения этой программы представлен на рис. 4.11.

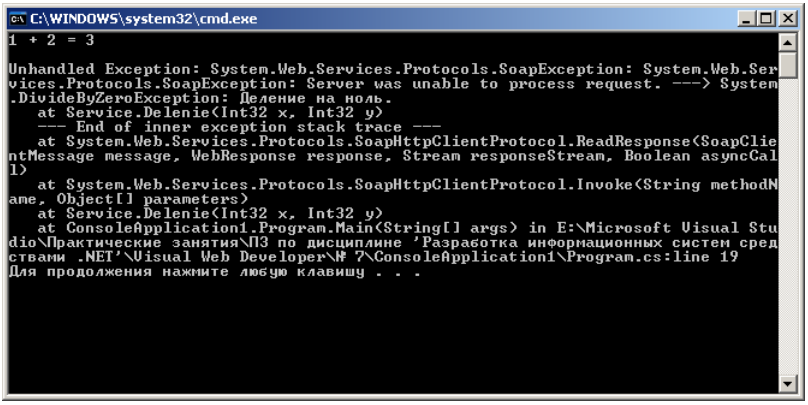


Рис. 4.11. Консольный клиент Web-службы

Следует обратить внимание, что получена полная информация об исключении, возникшем на Web-службе.

Создание прокси-сборки в Microsoft Visual Studio. Создавать прокси-сборки можно и при помощи утилиты wsdl.exe, и прямо в Microsoft Visual Studio. Работа с wsdl.exe более трудоемка, но у этой утилиты есть одно существенное преимущество – можно указать протокол подключения, который будет использоваться прокси-сборкой (HTTP-GET, HTTP-POST или SOAP). Microsoft Visual Studio позволяет генерировать только прокси-сборки, работающие по протоколу SOAP. Впрочем, если создается прокси-сборка, то в подавляющем большинстве случаев будет использоваться для нее именно SOAP. Поэтому прокси-сборки удобнее создавать в Microsoft Visual Studio.

Следует воспользоваться этой возможностью и создать очень простого клиента Windows Forms с элементарным интерфейсом, который позволит пользователю вводить два числа и передавать их какому-нибудь арифметическому методу, например Slozhenie(). Этот метод будет перенаправлять полученные данные через прокси-сборку на Web-службу для вычислений. Создание прокси-сборки производится автоматически, нужно только добавить в проект Web-ссылку (рис. 4.12).

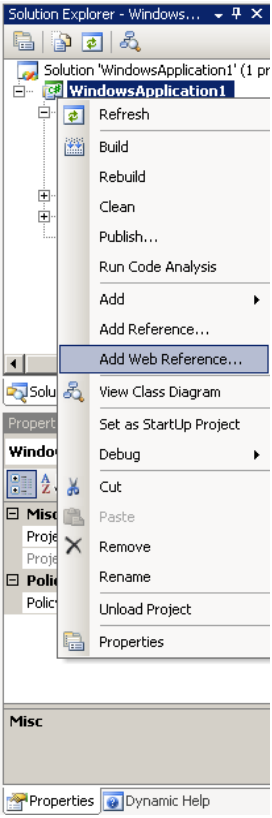
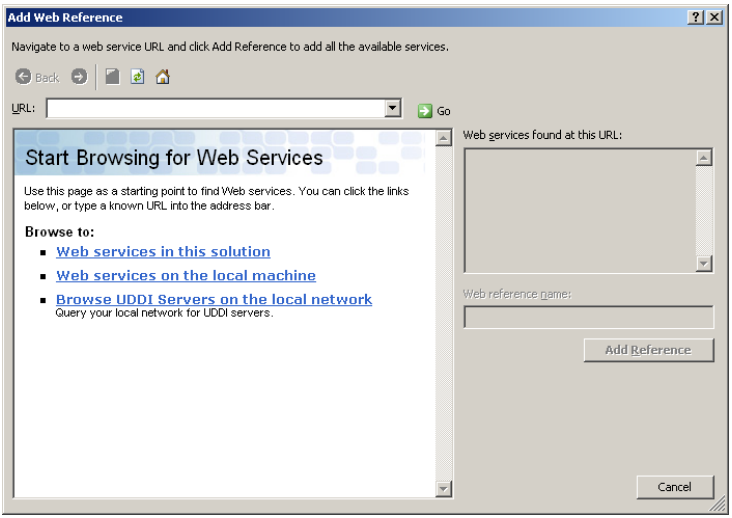


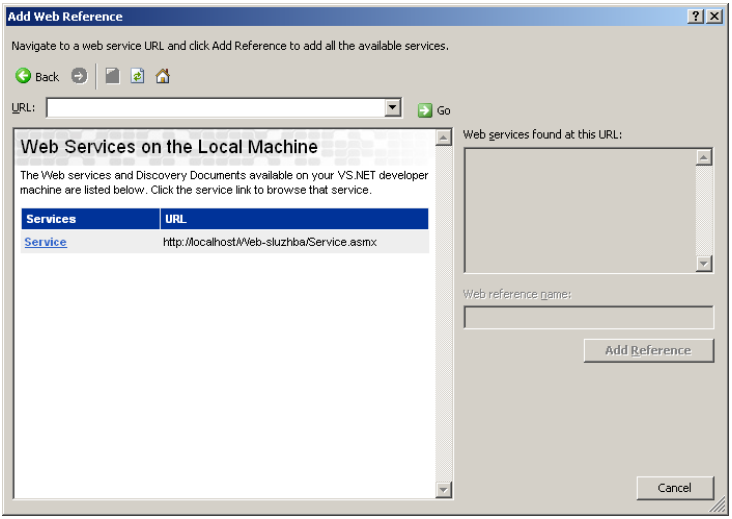
Рис. 4.12. Добавление Web-ссылки

Отроется диалоговое окно Add Web Reference («Добавить Web-ссылку»), в котором можно указать адрес URL к файлу ASMX или воспользоваться возможностью выбора (поиска) Web-служб и просмотреть различную информацию о них. В данном случае следует выполнить следующие действия: 1) выбрать пункт Web services on the local machine (Web-

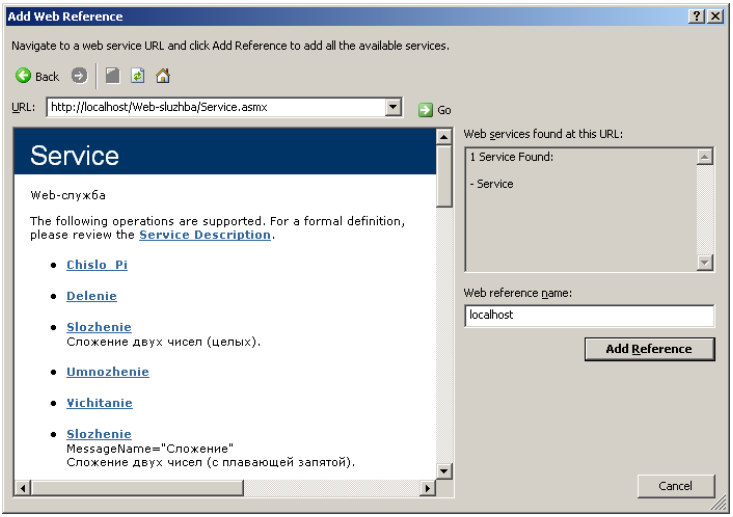
службы на локальной машине); 2) выбрать необходимую Web-службу; 3) добавить ссылку кнопкой Add Reference (рис. 4.13).



а



б



в

Рис. 4.13. Диалоговое окно Add Web Reference:
а – выбор пункта Web services on the local machine;
б – выбор необходимой Web-службы; в – добавление ссылки кропкой Add Reference

В окне Solution Explorer должен появиться новый узел Web-ссылки (рис. 4.14).

После этого уже можно напрямую работать в проекте с классом прокси-сборки (он будет называться Service). Следует обратить внимание, что в качестве названия пространства имен прокси-сборки выбирается имя компьютера, на котором расположена Web-служба:

```
private void button1_Click(object sender, EventArgs e)
{

        localhost.Service w = new localhost.Service();

        int Resultat = w.Slozhenie(int.Parse(textBox1.Text),
int.Parse(textBox2.Text));
        label5.Text = Resultat.ToString();
}
```

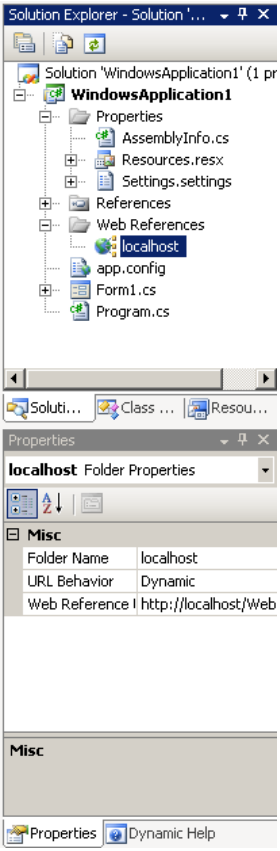


Рис. 4.14. Web-ссылка в окне Solution Explorer

Графический интерфейс разрабатываемого клиентского приложения может быть, например, такой, который изображен на рис. 4.15.

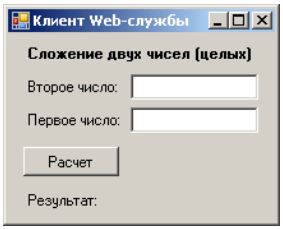


Рис. 4.15. Клиент Windows Forms Web-службы

Web-служба для работы с объектом ADO.NET DataSet и ее клиент. Помимо достаточно элементарного примера Web-службы, выполняющей функции калькулятора, следует рассмотреть более сложный ее пример. Возможности Web-служб можно представить гораздо лучше, если реализовать методы, которые будут возвращать (конечно, только при помощи SOAP) объекты ADO.NET DataSet, пользовательских классов и массивы объектов. Создание такой Web-службы рассмотрено ниже.

Как обычно, начать новый проект, например на Visual C#, следует на основе шаблона ASP.NET Web Service. Первое, что необходимо сделать, – создать метод Web-службы Organizatsii(), который будет возвращать объект DataSet с полным набором записей из таблицы «Организации» созданной ранее «Базы данных вакансий». Код для этого метода может выглядеть так:

```
[WebMethod]

public DataSet Organizatsii()
{

        SqlConnection connection = new SqlConnection();

        connection.ConnectionString = @"Data Source=-\SQLEXPRESS;
        Initial Catalog=База данных вакансий; Integrated Security=sspi";

        SqlDataAdapter org_TableAdapter = new SqlDataAdapter
         ("SELECT * FROM Организации", connection);

        DataSet Baza_Dannikh = new DataSet();

        org_TableAdapter.Fill(Baza_Dannikh, "Организации");
        return Baza_Dannikh;
}
```

Теперь, если создать клиента Windows Forms и добавить в него Web-ссылку на Web-службу, можно вызывать метод Web-службы Organizatsii(), например, для заполнения элемента управления DataGrid данными из таблицы «Организации»:

```
public Form1()
{
        InitializeComponent();
        CenterToScreen();
```

```
        localhost.Service w = new localhost.Service();

        DataSet Baza_Dannikh = w.Organizatsii();
        dataGrid1.DataSource = Baza_Dannikh.Tables["Организации"];
        w.Dispose();
}
```

Результат работы клиента представлен на рис. 4.16 [1, 7].

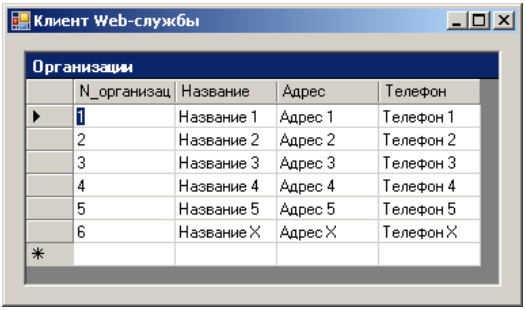


Рис. 4.16. Получение с Web-службы объекта DataSet

Сериализация пользовательских типов. В возможности протокола передачи SOAP входит передача XML-представлений объектов пользовательских типов с сохранением их внутреннего состояния. Чтобы убедиться в этом на примере, следует добавить в разработанное приложение новый класс, например Organizatsiya. Он будет очень простым: две переменные, определенные как Public – для хранения иформации о названии специальности и о ее шифре, и перегруженный конструктор, который позволяет установить эти значения. Также обязательно добавить одну очень важную деталь – атрибут XmlInclude, определенный в пространстве имен System.Xml.Serialization. Определение указанного класса будет выглядеть так:

```
[XmlInclude(typeof(Organizatsiya))]

public class Organizatsiya
{
        public string Nazvanie_spetsialnosti;

        public int Shifr_spetsialnosti;

        public Organizatsiya()
```

```
        {
        }

        public Organizatsiya(string n, int s)
        {
                Nazvanie_spetsialnosti = n;
                Shifr_spetsialnosti = s;
        }

}
```

Процесс сохранения объекта вместе со всем внутренним состоянием (в данном случае в формате XML) называется сериализацией. Однако сериализация объекта возможна лишь в том случае, если класс этого объекта был помечен атрибутом XmlInclude.

А теперь следует определить как Private массив типа ArrayList (он будет называться, например, AL_Organizatsii) и добавить в него несколько объектов Organizatsiya – прямо в конструкторе класса Web-службы:

```
private ArrayList AL_Organizatsii = new ArrayList();

public Service()
{
        //Uncomment the following line if using designed components
        //InitializeComponent();
        AL_Organizatsii.Add(new Organizatsiya("Специальность 1", 1));
        AL_Organizatsii.Add(new Organizatsiya("Специальность 2", 2));
        AL_Organizatsii.Add(new Organizatsiya("Специальность 3", 3));
}
```

После этого можно определить в Web-службе два метода: Vivod_spiska() для возвращения всего массива AL_Organizatsii целиком и Vivod_organizatsii() для возвращения конкретного объекта Organizatsiya по его номеру в массиве. Определения каждого из этих методов следующие [1, 7]:

```
[WebMethod]

public ArrayList Vivod_spiska()
{
        return AL_Organizatsii;
}
```

```
[WebMethod]

public Organizatsiya Vivod_organizatsii(int s)
{
        if (s <= AL_Organizatsii.Count)
        {
                return (Organizatsiya)AL_Organizatsii[s-1];
        }
        throw new IndexOutOfRangeException();
}
```

Настройка клиента. Служба уже создана и осталось создать клиента, который к ней будет обращаться. Рекомендуется воспользоваться уже готовым ранее созданным клиентом с элементом управления DataGrid. Единственное, что обязательно надо сделать, так как Web-служба изменилась,– это обновить Web-сслыку, например, из окна Solution Explorer, выбрав пункт меню Update Web Reference. Следует также добавить к графическому интерфейсу клиента надпись, текстовое поле для ввода номера объекта Organizatsiya в массиве и две кнопки (для вызова каждого из методов Web-службы) (рис. 4.17).

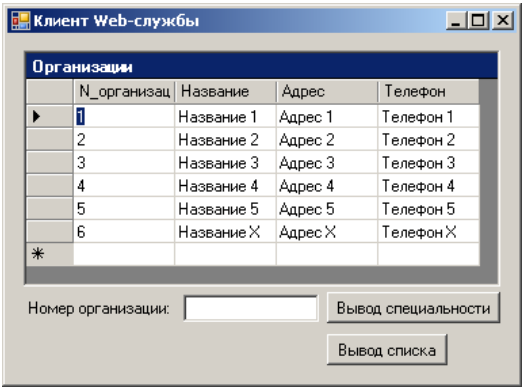


Рис. 4.17. Графический интерфейс клиента Web-службы

Код обработчика события Click для кнопки, которая будет вызывать метод Vivod_spiska(), будет таким:

```
private void button1_Click(object sender, EventArgs e)
{
        try
```

```
        {
                localhost.Organizatsiya s;

                localhost.Service ss = new localhost.Service();

                s = ss.Vivod_organizatsii(int.Parse(textBox1.Text));
                MessageBox.Show(s.Nazvanie_spetsialnosti,
                "Названия специальностей организации № " +
textBox1.Text);
                ss.Dispose();
        }
        catch
        {
                MessageBox.Show("Названия специальностей данной
организации
                отсутствуют");
        }
}
```

Результат запуска клиента представлен на рис. 4.18.

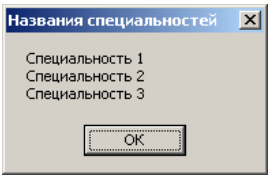


Рис. 4.18. Получение объекта Organizatsiya из массива AL_Organizatsii на Web-службе

Следует помнить, что извлекается информация об объекте Organizatsiya из массива, а не из таблицы «Организации», которая отображается в DataGrid. Обработчик события Click для второй кнопки будет выглядеть так:

```
private void button2_Click(object sender, EventArgs e)
{

        localhost.Service ss = new localhost.Service();

        object[] obj_s = ss.Vivod_spiska();
        string Nazvaniya_spetsialnostey = "";
```

```
        for (int i = 0; i < obj_s.Length; i++)
        {

                localhost.Organizatsiya s = (localhost.Organizatsiya)obj_s[i];

                Nazvaniya_spetsialnostey += s.Nazvanie_spetsialnosti + "\n";
        }

        MessageBox.Show(Nazvaniya_spetsialnostey, "Названия
специальностей");
        ss.Dispose();
}
```

Создание типов для сериализации (некоторые уточнения). Обычно при сериализации объекта в формате XML сохранение его внутреннего состояния является принципиальным, чтобы его можно было потом восстановить (например, на клиенте) и продолжить с ним работу. Чтобы среда выполнения могла сериализовать внутренние данные объекта, ей необходимо получить к ним доступ. Однако если внутренние переменные определены как Private, то получить к ним доступ среда выполнения не сможет. Как известно, в примере обе переменные (Nazvanie_spetsialnosti и Shifr_spetsialnosti) предусмотрительно определены как Public. Предполагается, что они определены как Private (как положено с точки зрения культуры программирования).

Если после этого вызвать метод Vivod_spiska(), то в массиве также обнаружатся три объекта Organizatsiya. Однако ни для одного из этих объектов не сохранится информация о внутреннем состоянии. Проблема решается просто: надо либо определять переменные как Public, либо создать свойства для доступа к переменным, определенным как Private:

```
private string Nazvanie_spetsialnosti;

private int Shifr_spetsialnosti;

public string Nazvanie_spetsialnosti_
{
        get
        {
                return Nazvanie_spetsialnosti;
        }
        set
        {
```

```
                Nazvanie_spetsialnosti = value;
        }
}

public int Shifr_spetsialnosti_
{
        get
        {
                return Shifr_spetsialnosti;
        }
        set
        {
                Shifr_spetsialnosti = value;
        }
}
```

В принципе создание объектов, к которым можно будет обратиться через Web-службу, не сильно отличается от создания обычных объектов C#. Главное – не забыть пометить такие объекты атрибутом XmlInclude и обеспечить возможность доступа к данным, определенным как Private [1, 7].

Протокол обнаружения. Последнее, что будет рассмотрено в этой главе:

- служба обнаружения (Discovery service) для Web-служб, посредством которой клиенты могут получить информацию о существовании Web-службы, ее возможностях и особенностях правильного взаимодействия с ней;
- файлы DISCO, которые содержат информацию (описание) о Web-службах виртуального каталога и используются для их настройки и доступа к ним.

Когда клиент обращается к Web-службе, первое, что он должен сделать, – убедиться, что Web-служба по данному адресу существует. В принципе это можно сделать программным образом – в библиотеке базовых классов .NET предусмотрены для этого соответствующие типы. Однако стандартное средство для этого – использовать службу обнаружения. Кроме того, служба обнаружения также необходима многим средствам разработки (например, она используется при добавлении Web-ссылки в проект).

Информация обо всех Web-службах в конкретном виртуальном каталоге и его подкаталогах хранится в файле DISCO. Этот файл можно создать как вручную при помощи утилиты disco.exe, так и полностью автоматически в Microsoft Visual Studio при добавлении Web-ссылки в проект на основе шаблона ASP.NET Web Service. Синтаксис программы disco.exe следующий:

disco.exe /out:<имя_устройства>:\<каталог> <адрес_URL>

Параметр out определяет путь, по которому будут сохранены файлы форматов DISCO, DISCOMAP и WSDL, сформированные в результате работы утилиты. Например, для созданного ранее проекта исходный код файла DISCO в формате XML будет такой, который представлен на рис. 4.19.

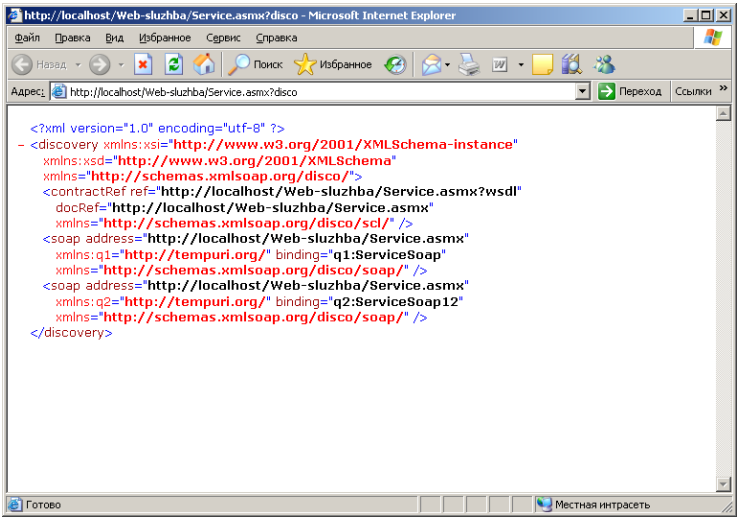


Рис. 4.19. Исходный код файла DISCO Web-службы

Тег <discovery> определяет, что в ответ на запрос клиента к службе обнаружения данный файл DISCO должен быть обработан средой выполнения ASP.NET на сервере и должен быть возвращен ответ на этот запрос (ответ, конечно, будет в формате XML), сгенерированный на основе этого файла.

Добавление новой Web-службы. Для приведенного примера в виртуальном каталоге существует только одна Web-служба, к которой можно обратиться по адресу http://localhost/Web-sluzhba/Service.asmx. Поэтому содержимое файла DISCO вполне очевидно. В виртуальный каталог можно добавить новую Web-службу. Это делается следующим образом.

Следует открыть в Microsoft Visual Studio проект и добавить в него новую Web-службу. Проще всего это сделать при помощи команд меню WebSite → Add New Item… или через контекстное меню для проекта (рис. 4.20).

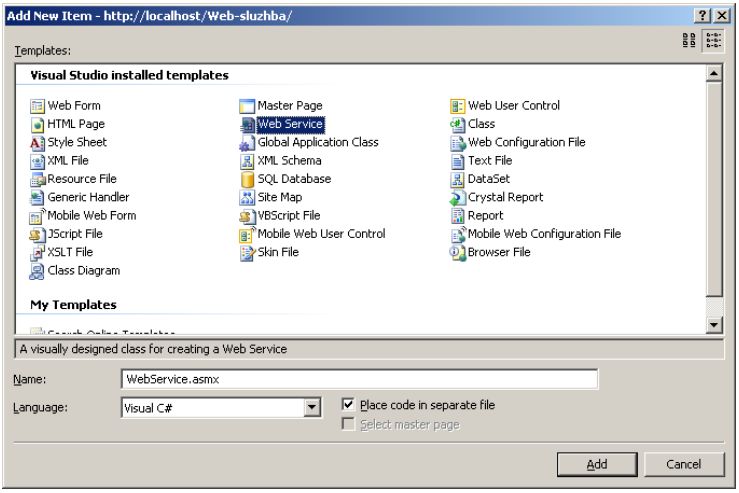


Рис. 4.20. Добавление в проект ASP.NET Web Service новой Web-службы

Таким образом, в одном проекте ASP.NET Web Service может быть несколько Web-служб. Далее, например, можно определить в новом классе WebService единственный метод Metod():

```
[WebMethod]

public string Metod()
{
        return "Сообщение";
}
```

После добавления новой Web-ссылки в проект клиентского приложения будет создан новый файл DISCO, соответствующий сформированной Web-службе и содержащий определенную информацию о ней (рис. 4.21).

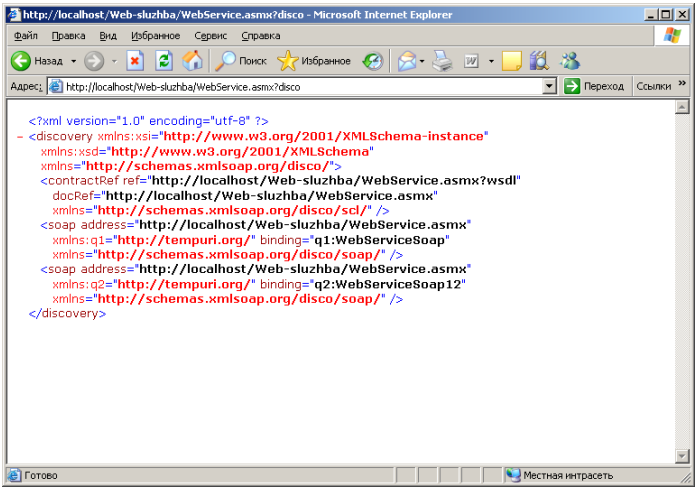


Рис. 4.21. Исходный код файла DISCO новой Web-службы

Конечно, просмотр необработанного кода XML в браузере – это не самый лучший способ получения сведений о Web-службах. В библиотеке базовых классов .NET предусмотрены типы, которые позволяют работать с данными, возвращаемыми службой обнаружения. Кроме того, эта информация необходима мастеру при добавлении Web-ссылки и многим средствам разработки.

Таким образом, рассмотрены основные принципы и приемы построения Web-служб, наиболее важные пространства имен и типы, которые используются для создания Web-служб, а также основные технологии: служба обнаружения, язык описания (WSDL) и протокол подключения (HTTP-GET, HTTP-POST и SOAP).

После того как Web-служба создана и определены в ней доступные для внешних клиентов методы при помощи атрибута WebMethod, проще всего реализовывать клиентов таким образом, чтобы они обращались не к Web-службе напрямую, а к промежуточной прокси-сборке, которая будет перенаправлять вызовы методов на Web-службу. Прокси-сборку можно сгенерировать при помощи утилиты wsdl.exe (для использования любого протокола подключения) либо из Microsoft Visual Studio (только для SOAP). Рассмотрены вопросы передачи объектов пользовательских типов посредством SOAP и определения классов как пригодных для сериализации в формате XML при помощи атрибута XmlInclude. Также рассмотрены особенности протокола обнаружения, с использованием которого возможно получение различной информации о Web-службах [1, 7].

ЗАКЛЮЧЕНИЕ

Таким образом, в рамках дисциплины «Разработка информационных систем средствами .NET» рассмотрены некоторые теоретические и практические вопросы, касающиеся методов проектирования и разработки (программной реализации) информационных систем, основанных на достаточно перспективной в настоящее время технологии .NET корпорации Microsoft, в частности:

- работа с интегрированной средой разработки приложений Microsoft Visual Studio для создания программных приложений для доступа к базам данных и различного рода Web-приложений (Web-служб), а также некоторыми вспомогательными программными продуктами для настройки функционирования информационных систем;
- работа с системой управления базами данных Microsoft SQL Server;
- администрирование служб Internet Information Services и ASP.NET Development Server;
- особенности проектирования и программной реализации информационных систем с использованием доступа к данным при помощи технологии ADO.NET;
- принципы разработки Web-приложений с применением технологии ASP.NET;
- назначение и использование Web-служб;
- методы поиска информации с использованием автоматизированных систем;
- реализация доступа к данным, хранимым в памяти и в файлах систем управления базами данных Microsoft Access и Microsoft SQL Server, на языке программирования Visual C# с использованием технологий ADO.NET и ASP.NET и элементов управления Windows Forms и Web Forms;
- разработка пользовательского интерфейса посредством элементов управления WebForm и клиентских скриптов на языках программирования JavaScript и VBScript для серверных скриптов;
- проектирование и разработка локального ASP.NET Web-сайта;
- использование технологии ADO.NET для взаимодействия с локальными и удаленными базами данных при программировании кода для страниц ASP.NET;
- назначение, использование и программная реализация Web-служб.

СПИСОК ЛИТЕРАТУРЫ

1. Троелсен, Э. C# и платформа .NET. Библиотека программиста / Э. Троелсен; пер. с англ. – СПб. : Питер, 2004. – 796 с.

2. Плат, Д. С. Знакомство с Microsoft .NET / Д. С. Плат; пер. с англ. – М. : Русская редакция, 2001. – 240 с.

3. Просиз, Дж. Программирование для Microsoft .NET / Дж. Просиз; пер. с англ. – М. : Русская редакция, 2003. – 704 с.

4. Microsoft Corporation. Анализ требований и создание архитектуры решений на основе Microsoft .NET: учеб. курс MCSD / пер. с англ. – М. : Русская редакция, 2004. – 416 с.

5. Сеппа, Д. Microsoft ADO.NET / Д. Сеппа; пер. с англ. – М. : Русская редакция, 2003. – 640 с.

6. Вилдермьюс, Ш. Практическое использование ADO.NET. Доступ к данным в Internet / Ш. Вилдермьюс; пер. с англ. – М. : Вильямс, 2003. – 288 с.

7. Альманах программиста: в 4 т. / сост. Ю. Е. Купцевич. – М. : Русская редакция, 2003–2004. – Т. 1: Microsoft ADO.NET, Microsoft SQL Server, доступ к данным из приложений. – 2003. – 400 с.; Т. 2: ASP.NET, Web-сервисы, WEB-приложения. – 2003. – 400 с.; Т. 3: Платформа 2003: Microsoft Windows Server 2003, Microsoft Internet Information Services 6.0, Microsoft Office System. – 2003. – 320 с.; Т. 4: Безопасность в Microsoft .NET. – 2004. – 304 с.

8. Рейли, Д. Создание приложений Microsoft ASP.NET / Д. Рейли; пер. с англ. – М. : Русская редакция, 2002. – 480 с.

9. Оньон, Ф. Основы ASP.NET с примерами на C# / Ф. Оньон; пер. с англ. – М. : Вильямс, 2003. – 304 с.

10. Microsoft Corporation. Разработка Web-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET. Учебный курс MCAD / MCSD / пер. с англ. – М. : Русская редакция, 2003. – 704 с.

Учебное издание

Борис Александрович Тонконогов

Проектирование и разработка информационных систем средствами технологии .NET

Учебно-методическое пособие

Редакторы *С. О. Сараева, О. А. Кучинский*

Корректор *С. О. Сараева*

Компьютерная верстка *М. Ю. Мошкова*

Подписано в печать 03.06.2010. Формат 60×90 $^{1}/_{16}$.
Бумага офсетная. Гарнитура Times. Ризография.
Усл. печ. л. 16,375. Уч.-изд. л. 9,035.
Тираж 15 экз. Заказ № 75.

Издатель и полиграфическое исполнение
учреждение образования «Международный государственный
экологический университет имени А. Д. Сахарова»

ЛИ № 02330/0131580 от 28.07.2005 г.
Республика Беларусь, 220070, г. Минск, ул. Долгобродская, 23

E-mail: info@iseu.by
http://www.iseu.by