

*Включает LINQ, .NET 3.5 CLR
и библиотеки базовых классов .NET*

**Третье
издание**



C# 3.0

СПРАВОЧНИК

O'REILLY® 

Джозеф Албахари
Бен Албахари

C# 3.0

IN A NUTSHELL

C# 3.0

IN A NUTSHELL

Third Edition

Joseph Albahari and Ben Albahari

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Джозеф Албахари
Бен Албахари

C# 3.0

СПРАВОЧНИК

Санкт-Петербург
«БХВ-Петербург»
2009

УДК 681.3.068+800.92С#
ББК 32.973.26-018.1
А45

Албахари, Дж.

А45 С# 3.0. Справочник: Пер. с англ. / Дж. Албахари, Б. Албахари. —
3-е изд. — СПб.: БХВ-Петербург, 2009. — 944 с.: ил.

ISBN 978-5-9775-0245-0

Книга представляет собой подробный справочник по программированию на языке С# 3.0, реализованном в Microsoft Visual Studio 2008. Первые главы целиком посвящены языку С#, начиная с основ синтаксиса, типов и переменных и заканчивая более сложными темами, такими как небезопасный код или препроцессорные директивы. В последующих главах рассмотрено ядро .NET 3.5 Framework, LINQ, XML, коллекции, ввод/вывод и работа в сети, управление памятью, рефлексия, атрибуты, безопасность, домены приложений и взаимодействие с небезопасным кодом. В отличие от предыдущих изданий, материал книги наряду с теоретическими основами сопровождается наглядными практическими примерами, что позволяет сочетать глубину изложения с легкостью понимания.

Для программистов

УДК 681.3.068+800.92С#
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Перевод с английского	<i>Сергея Иноземцева</i>
Редактор	<i>Ирина Иноземцева</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Authorized translation of the English edition of C# 3.0 in a Nutshell, 3E, ISBN: 978-0-596-52757-0, Copyright © 2007 O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Авторизованный перевод английской редакции книги C# 3.0 in a Nutshell, 3E, ISBN: 978-0-596-52757-0, Copyright © 2007 O'Reilly Media, Inc. Перевод опубликован и продается с разрешения O'Reilly Media, Inc., собственника всех прав на публикацию и продажу издания.

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.06.09.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 76,11.

Тираж 2000 экз. Заказ №

"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.003650.04.08 от 14.04.2008 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-0-596-52757-0 (англ.)
ISBN 978-5-9775-0245-0 (рус.)

© 2007 O'Reilly Media, Inc.
© Перевод на русский язык "БХВ-Петербург", 2009

Оглавление

Об авторах	1
Предисловие	3
Целевая аудитория	3
Как организована эта книга.....	4
Что необходимо для чтения этой книги.....	4
Соглашения, использованные в книге.....	5
Использование фрагментов кода	5
Благодарности	6
Глава 1. Введение в C# и .NET Framework.....	9
Объектная ориентированность	9
Безопасность типов.....	10
Управление памятью.....	10
Работа на разных платформах.....	11
Язык C# и среда CLR.....	11
Среда CLR и платформа .NET Framework	11
Что нового в C# 3.0.....	13
Глава 2. Основы языка C#.....	17
Первая программа на C#.....	17
Компиляция	19
Синтаксис	20
Идентификаторы и ключевые слова	20
Как избежать конфликтов.....	21
Контекстные ключевые слова	21
Литералы, пунктуаторы и знаки операций.....	21
Комментарии	22
Основные сведения о типах.....	22
Примеры стандартных типов.....	23
Примеры пользовательских типов	23
Члены типа.....	24
Сходство стандартных и пользовательских типов	24
Конструкторы и создание экземпляров	24
Члены экземпляра и статические члены.....	25
Ключевое слово <i>public</i>	26
Преобразование типов	26
Типы значений и ссылочные типы.....	26
Типы значений.....	27
Ссылочные типы	27
Значение <i>null</i>	28
Расход памяти.....	29
Таксономия стандартных типов	29

Числовые типы	30
Числовые литералы	31
Распознавание типов числовых литералов	31
Числовые суффиксы.....	31
Числовые преобразования	32
Преобразование целого в целое	32
Преобразование числа с плавающей точкой в число с плавающей точкой	32
Преобразование целого в число с плавающей точкой и обратно.....	33
Десятичные преобразования	33
Арифметические операции.....	33
Операции инкремента и декремента.....	33
Специальные операции над целыми числами	34
Целочисленное деление	34
Целочисленное переполнение	34
Проверка на переполнение при целочисленных арифметических операциях	34
Проверка на переполнение выражений с константами.....	35
Побитовые операции.....	35
Восьми- и шестнадцатибитовые целые	36
Специальные значения типов <i>float</i> и <i>double</i>	36
Типы <i>double</i> и <i>decimal</i>	37
Ошибки округления вещественных чисел.....	37
Тип <i>bool</i> и булевы операторы	38
Преобразование типа <i>bool</i>	38
Равенство и операции сравнения	38
Условные операции.....	39
Строки и символы	40
Преобразование типа <i>char</i>	41
Тип <i>string</i>	41
Конкатенация строк	42
Сравнение строк.....	42
Массивы.....	42
Инициализация элементов по умолчанию.....	43
Типы значений и ссылочные типы	43
Многомерные массивы	44
Прямоугольные массивы	44
Ступенчатые массивы	44
Упрощенные выражения для инициализации массива	45
Проверка границ массива	46
Переменные и параметры.....	46
Стек и куча.....	46
Стек	46
Куча.....	47
Определенное присваивание	48
Значения по умолчанию.....	48
Параметры.....	49
Передача аргументов по значению	49
Модификатор <i>ref</i>	50
Модификатор <i>out</i>	51
Особенности передачи аргументов по ссылке	52
Модификатор <i>params</i>	52
Ключевое слово <i>var</i> : неявная типизация локальных переменных (C# 3.0).....	53
Выражения и операции.....	54
Первичные выражения.....	54
Выражения, не имеющие значения	54

Выражения присваивания	54
Приоритет и ассоциативность операций	55
Приоритет	55
Операции с левой ассоциативностью	55
Операции с правой ассоциативностью	55
Таблица операций	56
Операторы	58
Операторы объявления	58
Локальные переменные	58
Операторы выражений	59
Операторы выбора	60
Оператор <i>if</i>	60
Конструкция <i>else</i>	60
Изменение хода выполнения программы с помощью операторных скобок	60
Оператор <i>switch</i>	61
Операторы цикла	63
Циклы <i>while</i> и <i>do-while</i>	63
Цикл <i>for</i>	63
Цикл <i>foreach</i>	64
Операторы перехода	64
Оператор <i>break</i>	64
Оператор <i>continue</i>	65
Оператор <i>goto</i>	65
Оператор <i>return</i>	66
Оператор <i>throw</i>	66
Прочие операторы	66
Пространства имен	66
Ключевое слово <i>namespace</i>	67
Полностью квалифицированные имена	67
Директива <i>using</i>	67
Глобальное пространство имен	68
Правила для пространств имен	68
Область видимости имен	68
Соккрытие имен	69
Повторяющиеся пространства имен	70
Вложенные директивы <i>using</i>	70
Псевдонимы типов и пространств имен	71
Дополнительные возможности пространств имен	71
Внешние пространства имен	71
Квалификаторы псевдонимов для пространств имен	72
Глава 3. Создание типов в C#	75
Классы	75
Поля	75
Модификатор <i>readonly</i>	76
Инициализация поля	76
Совместное объявление нескольких полей	76
Методы	76
Перегрузка методов	77
Передача параметров по значению и ссылке	77
Конструкторы экземпляров	77
Перегрузка конструкторов	78
Неявные конструкторы без параметров	78
Порядок выполнения конструктора и инициализации полей	78
Закрытые конструкторы	79

Инициализаторы объектов (C# 3.0)	79
Ссылка <i>this</i>	80
Свойства	80
Свойства, доступные только для чтения, и вычисляемые свойства	81
Автоматические свойства (C# 3.0).....	82
Обращение к свойству методами <i>get</i> и <i>set</i>	82
Реализация свойств в CLR.....	83
Индексаторы	83
Реализация индексатора	83
Реализация индексатора в CLR.....	85
Константы	85
Статические конструкторы.....	86
Порядок инициализации статических полей.....	87
Недетерминизм статических конструкторов.....	87
Статические классы.....	87
Финализаторы.....	87
Частичные классы и методы.....	88
Частичные методы (C# 3.0).....	88
Наследование.....	89
Полиморфизм	90
Приведение типов.....	91
Приведение "вверх".....	91
Приведение "вниз"	91
Операция <i>as</i>	92
Операция <i>is</i>	92
Виртуальные функции	92
Абстрактные классы и абстрактные члены	93
Сокрытие унаследованных членов.....	94
Модификаторы <i>new</i> и <i>virtual</i>	94
Запечатывание функций и классов.....	95
Ключевое слово <i>base</i>	95
Конструкторы и наследование	96
Неявный вызов конструктора базового класса без параметров	96
Порядок инициализации конструкторов и полей	97
Перегрузка и разрешение конфликтов.....	97
Тип <i>object</i>	98
Упаковка и распаковка.....	98
Семантика копирования при упаковке и распаковке	99
Статическая и динамическая проверка типа	99
Метод <i>GetType</i> и операция <i>typeof</i>	100
Метод <i>ToString</i>	100
Список членов объекта	101
Структуры.....	101
Конструирование структуры	102
Модификаторы доступа.....	102
Примеры.....	103
Понижение уровня доступности	103
Ограничения, накладываемые на модификаторы доступа.....	104
Интерфейсы.....	104
Расширение интерфейса	105
Явная реализация интерфейса	105
Виртуальная реализация членов интерфейса.....	106
Новая реализация интерфейса в подклассе	107
Альтернативы реализации интерфейса	108

Интерфейсы и упаковка	108
Выбор между классом и интерфейсом	109
Перечисления	110
Преобразования перечислений	110
Перечисление флагов	111
Операции над перечислениями	112
Безопасность типов	112
Вложенные типы	113
Обобщенные типы	114
Объявление обобщенных типов	115
Для чего существуют обобщенные типы	116
Обобщенные методы	117
Объявление обобщенных параметров	118
Операция <i>typeof</i> для обобщенных типов	118
Значение по умолчанию для обобщенного типа	119
Ограничения для обобщенных типов	119
Обобщенные типы и ковариантность	120
Сравнение обобщенных типов и массивов	121
Создание подклассов обобщенных классов	121
Обобщенные объявления, ссылающиеся на себя	122
Статические данные	122
Обобщенные типы C# и шаблоны C++	122
Глава 4. Более сложные элементы C#	125
Делегаты	125
Написание подключаемых методов	126
Групповые делегаты	126
Пример группового делегата	127
Экземплярные методы-цели	128
Обобщенные типы делегатов	128
Делегаты и интерфейсы	129
Совместимость делегатов	130
Совместимость по типам	130
Совместимость по параметрам	131
Совместимость по типу возвращаемого значения	131
События	132
Стандартная модель событий	133
Методы обращения к событиям	136
Модификаторы событий	138
Лямбда-выражения (C# 3.0)	138
Явное указание типов параметров лямбда-выражения	139
Обобщенные лямбда-выражения и делегаты <i>Func</i>	140
Внешние переменные	140
Анонимные методы	142
Операторы <i>try</i> и исключения	142
Конструкция <i>catch</i>	144
Блок <i>finally</i>	146
Оператор <i>using</i>	146
Возбуждение исключений	147
Повторное возбуждение исключения	147
Важнейшие свойства класса <i>System.Exception</i>	148
Распространенные типы исключений	149
Распространенные способы программирования	149
Применение метода <i>try</i>	149
Атомарность операций	150
Альтернативы исключениям	151

Перечисление и итераторы.....	152
Перечисление.....	152
Итераторы.....	153
Семантика итераторов.....	154
Формирование последовательностей.....	155
Конструирование перечисляемого объекта.....	155
Типы, допускающие значение <i>null</i>	156
Значение <i>null</i>	156
Структура <i>Nullable<T></i>	157
Явное и неявное преобразование типов, допускающих значение <i>null</i>	157
Упаковка и распаковка типов, допускающих значение <i>null</i>	157
Поднятие операций.....	158
Операции равенства.....	158
Операции отношения.....	159
Все остальные операции.....	159
Смешивание разных типов.....	159
Тип <i>bool?</i>	159
Операция проверки на <i>null</i>	160
Применение типов, допускающих значение <i>null</i>	160
Альтернативы для типов, допускающих значение <i>null</i>	161
Перегрузка операций.....	162
Обзор.....	162
Операторные функции.....	162
Перегрузка операций равенства и сравнения.....	163
Нестандартные явные и неявные преобразования типов.....	164
Перегрузка <i>true</i> и <i>false</i>	165
Методы расширения (C# 3.0).....	166
Цепочки методов расширения.....	166
Устранение неоднозначности.....	166
Пространства имен.....	166
Методы расширения и методы экземпляров.....	167
Разные методы расширения.....	167
Методы расширения и интерфейсы.....	168
Анонимные типы (C# 3.0).....	169
Атрибуты.....	169
Классы атрибутов.....	170
Именованные и позиционные параметры.....	170
Цели атрибутов.....	171
Указание нескольких атрибутов.....	171
Небезопасный код и указатели.....	171
Основы работы с указателями.....	171
Небезопасный код.....	172
Оператор <i>fixed</i>	172
Операция "указатель на элемент".....	173
Массивы.....	173
Ключевое слово <i>stackalloc</i>	173
Буферы фиксированного размера.....	174
Конструкция <i>void*</i>	174
Указатели на управляемый код.....	175
Препроцессорные директивы.....	175
Условные атрибуты.....	176
Прагматичный подход к предупреждениям.....	177
XML-документирование.....	177
Стандартные XML-теги.....	179

Теги, определяемые пользователем	181
Перекрестные ссылки	181
Глава 5. Обзор возможностей платформы Framework.....	183
Что нового в .NET Framework 3.5	184
Среда CLR и "ядро" платформы Framework	185
Системные типы	185
Обработка текста	186
Коллекции	186
Запросы	186
XML	186
Потоки и ввод/вывод	187
Работа в сети	187
Сериализация	187
Сборки, отражение и атрибуты	188
Безопасность	188
Нити выполнения и асинхронные методы	188
Домены приложения	189
Низкоуровневое взаимодействие	189
Диагностика	189
Прикладные технологии	189
Технологии для создания пользовательского интерфейса	189
ASP.NET	190
WPF	190
Windows Forms	191
Технологии работы с базами данных	192
ADO.NET	192
Windows Workflow	193
COM+ и MSMQ	193
Технологии построения распределенных систем	193
WCF	193
Удаленное взаимодействие и веб-службы	194
CardSpace	195
Глава 6. Основы Framework.....	197
Работа со строками и текстом	197
Тип <i>char</i>	197
Строки	199
Конструирование строк	199
Пустые строки и строки со значением <i>null</i>	199
Обращение к символам в строке	200
Поиск в строке	200
Манипулирование строками	200
Разбиение и объединение строк	201
Метод <i>String.Format</i> и строки составного формата	202
Сравнение строк	203
Выяснение порядка и культурные особенности	203
Сравнение для выяснения равенства	204
Сравнение для выяснения порядка	205
Класс <i>StringBuilder</i>	206
Текстовые кодировки и Unicode	206
Получение объекта <i>Encoding</i>	208
Кодировка для файлового и потокового ввода/вывода	208
Кодировка для байтовых массивов	208
Кодировка UTF-16 и строки	209

Дата и время	210
Структура <i>TimeSpan</i>	210
Структуры <i>DateTime</i> и <i>DateTimeOffset</i>	211
Выбор между <i>DateTime</i> и <i>DateTimeOffset</i>	212
Создание структуры <i>DateTime</i>	213
Создание структуры <i>DateTimeOffset</i>	214
Текущие значения структур <i>DateTime</i> и <i>DateTimeOffset</i>	215
Работа с датой и временем	215
Форматирование и разбор даты и времени	216
Значение <i>null</i> у <i>DateTime</i> и <i>DateTimeOffset</i>	217
Даты и часовые пояса	218
Часовые пояса и структура <i>DateTime</i>	218
Часовые пояса и структура <i>DateTimeOffset</i>	218
Классы <i>TimeZone</i> и <i>TimeZoneInfo</i>	219
Класс <i>TimeZone</i>	219
Класс <i>TimeZoneInfo</i>	220
Структура <i>DateTime</i> и переход на летнее/зимнее время	223
Форматирование и разбор строк	225
Методы <i>ToString</i> и <i>Parse</i>	225
Поставщики форматов	226
Поставщики форматов и класс <i>CultureInfo</i>	227
Поставщики <i>NumberFormatInfo</i> и <i>DateTimeFormatInfo</i>	227
Составное форматирование	228
Разбор строк с помощью поставщиков форматов	228
Интерфейсы <i>IFormatProvider</i> и <i>ICustomFormatter</i>	229
Стандартные строки формата и флаги разбора	230
Строки формата для чисел	231
Флаги <i>NumberStyles</i>	233
Строки формата для <i>DateTime</i>	234
Разбор строк <i>DateTime</i> и возможные ошибки	236
Флаги <i>DateTimeStyles</i>	237
Строки формата для перечисления	237
Другие механизмы преобразования	238
Класс <i>Convert</i>	238
Округление при преобразовании вещественных чисел в целые	238
Системы счисления по основанию 2, 8 и 16	239
Динамические преобразования	239
Преобразования base64	240
Класс <i>XmlConvert</i>	240
Преобразователи типов	240
Класс <i>BitConverter</i>	241
Работа с числами	242
Преобразования	242
Класс <i>Math</i>	242
Класс <i>Random</i>	243
Перечисления	244
Преобразования перечислений	244
Преобразование типа <i>enum</i> в целочисленный тип	245
Преобразование целочисленного типа в <i>enum</i>	246
Преобразование строк	246
Перебор элементов перечисления	246
Как работают перечисления	247
Структура <i>Guid</i>	247

Сравнение для выяснения равенства	248
Равенство по значению и ссылочное равенство	248
Стандартные протоколы равенства	249
Операции == и !=	249
Виртуальный метод <i>Object.Equals</i>	249
Статический метод <i>object.Equals</i>	250
Статический метод <i>object.ReferenceEquals</i>	251
Интерфейс <i>IEquatable<T></i>	252
Когда <i>Equals</i> не равняется операции ==	252
Равенство типов, определяемых пользователем	253
Как переопределить семантику равенства	254
Переопределение метода <i>GetHashCode</i>	254
Переопределение метода <i>Equals</i>	255
Перегрузка операций == и !=	255
Реализация интерфейса <i>IEquatable<T></i>	256
Пример: структура <i>Area</i>	256
Подключаемые интерфейсы сравнения	257
Сравнение для выяснения порядка	257
Интерфейсы <i>IComparable</i>	258
Интерфейс <i>IComparable</i> и метод <i>Equals</i>	258
Операции < и >	259
Реализация интерфейсов <i>IComparable</i>	260
Вспомогательные классы	261
Класс <i>Console</i>	261
Класс <i>Environment</i>	262
Класс <i>Process</i>	262
Глава 7. Коллекции	265
Перечисление	265
Интерфейсы <i>IEnumerable</i> и <i>IEnumerator</i>	266
Интерфейсы <i>IEnumerable<T></i> и <i>IEnumerator<T></i>	267
Реализация интерфейсов перечисления	268
Интерфейс <i>IDictionaryEnumerator</i>	272
Интерфейсы <i>ICollection</i> и <i>IList</i>	272
Интерфейсы <i>ICollection</i> и <i>ICollection<T></i>	273
Интерфейсы <i>IList</i> и <i>IList<T></i>	274
Класс <i>Array</i>	276
Конструирование массива и индексация его элементов	278
Перебор элементов массива	279
Длина и размерность массива	280
Поиск в массиве	280
Сортировка массива	282
Изменение порядка элементов на противоположный	283
Копирование, преобразование типа и изменение размера массива	283
Списки, очереди, стеки и множества	284
Классы <i>List<T></i> и <i>ArrayList</i>	285
Класс <i>LinkedList<T></i>	287
Классы <i>Queue</i> и <i>Queue<T></i>	289
Классы <i>Stack</i> и <i>Stack<T></i>	290
Класс <i>BitArray</i>	291
Класс <i>HashSet<T></i>	292
Словари	294
Интерфейсы <i>IDictionary</i> и <i>IDictionary<TKey,TValue></i>	295
Классы <i>Dictionary<TKey,TValue></i> и <i>Hashtable</i>	297

Класс <i>OrderedDictionary</i>	298
Классы <i>ListDictionary</i> и <i>HybridDictionary</i>	299
Отсортированные словари	299
Коллекции, настраиваемые пользователем, и прокси-классы	301
Классы <i>Collection<T></i> и <i>CollectionBase</i>	301
Классы <i>KeyedCollection<TKey,TItem></i> и <i>DictionaryBase</i>	304
Класс <i>ReadOnlyCollection<T></i>	306
Подключение протоколов выяснения равенства и порядка	307
Интерфейсы <i>IEqualityComparer</i> и класс <i>EqualityComparer</i>	308
Интерфейсы <i>IComparer</i> и класс <i>Comparer</i>	310
Класс <i>StringComparer</i>	312
Глава 8. Запросы LINQ.....	315
Основа	315
Лямбда-запросы	317
Цепочки операторов запросов.....	317
Почему важны методы расширения	320
Составление лямбда-выражений	320
Лямбда-выражения и сигнатуры делегатов <i>Func</i>	321
Лямбда-выражения и типы элементов.....	321
Естественный порядок элементов.....	323
Прочие операторы	323
Синтаксис, облегчающий восприятие запроса	324
Переменные итерации	326
Синтаксис, облегчающий восприятие запроса, и SQL-синтаксис	326
Синтаксис, облегчающий восприятие запроса, и лямбда-синтаксис	327
Запросы со смешанным синтаксисом	327
Отложенное выполнение	328
Повторное выполнение	329
Внешние переменные	330
Механика отложенного выполнения	331
Цепочки декораторов	332
Как выполняются запросы.....	333
Подзапросы.....	334
Подзапросы и отложенное выполнение	337
Стратегии построения сложных запросов	338
Последовательное построение запросов	338
Ключевое слово <i>into</i>	339
Правила определения областей видимости.....	340
Создание оболочек для запросов	340
Стратегии проецирования	342
Инициализаторы объектов.....	342
Анонимные типы	342
Ключевое слово <i>let</i>	343
Интерпретируемые запросы.....	344
Как работают интерпретируемые запросы.....	346
Выполнение	347
Комбинирование интерпретируемых и локальных запросов	349
Оператор <i>AsEnumerable</i>	350
Запросы LINQ to SQL	352
Классы сущностей в технологии LINQ to SQL	352
Объект <i>DataContext</i>	353
Автоматическое генерирование сущностей	355
Ассоциирование	355

Отложенное выполнение запросов LINQ to SQL	356
Класс <i>DataLoadOptions</i>	358
Предварительное указание фильтра	358
"Нетерпеливая" загрузка.....	359
Обновления	359
Построение выражений для запросов	361
Делегаты и деревья выражений.....	361
Компиляция деревьев выражений	362
Оператор <i>AsQueryable</i>	363
Деревья выражений.....	363
Объектная модель документов для выражений.....	363
Глава 9. Операторы LINQ	367
Обзор операторов.....	368
Коллекция → коллекция.....	368
Фильтрация	369
Проецирование.....	369
Объединение	369
Упорядочивание	369
Группирование	369
Операции над множествами.....	370
Методы преобразования: импорт.....	370
Методы преобразования: экспорт.....	370
Коллекция → не коллекция	370
Поэлементные операции.....	370
Методы агрегирования	370
Квантификаторы.....	370
Не коллекция → коллекция	371
Методы генерирования коллекций	371
Фильтрация.....	371
Оператор <i>Where</i>	372
Синтаксис, облегчающий восприятие запроса	372
Реализация метода <i>Enumerable.Where</i>	372
Описание.....	372
Индексированная фильтрация.....	373
Оператор <i>Where</i> в запросе LINQ to SQL	373
Операторы <i>Take</i> и <i>Skip</i>	373
Операторы <i>TakeWhile</i> и <i>SkipWhile</i>	374
Оператор <i>Distinct</i>	375
Проецирование.....	375
Оператор <i>Select</i>	375
Синтаксис, облегчающий восприятие запроса	375
Реализация в классе <i>Enumerable</i>	375
Описание	376
Индексированная проекция.....	376
Подзапросы <i>Select</i> и иерархия объектов	377
Подзапросы и объединения в запросах LINQ to SQL	377
Проецирование в конкретные типы	379
Оператор <i>SelectMany</i>	380
Синтаксис, облегчающий восприятие запроса	380
Реализация в классе <i>Enumerable</i>	380
Описание.....	381
Внешние переменные итерации.....	382
Мышление в соответствии с синтаксисом, облегчающим восприятие запроса	383

Объединение с помощью оператора <i>SelectMany</i>	384
Оператор <i>SelectMany</i> в запросах LINQ to SQL	385
Внешние объединения с помощью оператора <i>SelectMany</i>	386
Объединение	388
Операторы <i>Join</i> и <i>GroupJoin</i>	388
Аргументы оператора <i>Join</i>	388
Аргументы оператора <i>GroupJoin</i>	389
Синтаксис, облегчающий восприятие запроса	389
Описание	389
Оператор <i>Join</i>	390
Объединение по нескольким ключам	392
Объединение в лямбда-синтаксисе	392
Оператор <i>GroupJoin</i>	393
Плоские внешние объединения	394
Объединение и таблицы просмотра	394
Реализации в классе <i>Enumerable</i>	396
Упорядочивание	397
Операторы <i>OrderBy</i> , <i>OrderByDescending</i> , <i>ThenBy</i> и <i>ThenByDescending</i>	397
Аргументы операторов <i>OrderBy</i> и <i>OrderByDescending</i>	397
Аргументы операторов <i>ThenBy</i> и <i>ThenByDescending</i>	397
Синтаксис, облегчающий восприятие запроса	398
Описание	398
Классы, выполняющие сравнение, и сортировка	398
Интерфейсы <i>IOrderedEnumerable</i> и <i>IOrderedQueryable</i>	399
Группирование	400
Оператор <i>GroupBy</i>	400
Синтаксис, облегчающий восприятие запроса	400
Описание	400
Оператор <i>GroupBy</i> в запросах LINQ to SQL	402
Группирование по нескольким ключам	403
Пользовательские классы для выяснения равенства	403
Операции над множествами	403
Операторы <i>Concat</i> и <i>Union</i>	403
Операторы <i>Intersect</i> и <i>Except</i>	404
Методы преобразования	404
Операторы <i>OfType</i> и <i>Cast</i>	405
Операторы <i>ToArray</i> , <i>ToList</i> , <i>ToDictionary</i> и <i>ToLookup</i>	406
Операторы <i>AsEnumerable</i> и <i>AsQueryable</i>	407
Поэлементные операции	407
Операторы <i>First</i> , <i>Last</i> и <i>Single</i>	408
Оператор <i>ElementAt</i>	408
Оператор <i>DefaultIfEmpty</i>	409
Методы агрегирования	409
Операторы <i>Count</i> и <i>LongCount</i>	409
Операторы <i>Min</i> и <i>Max</i>	410
Операторы <i>Sum</i> и <i>Average</i>	411
Оператор <i>Aggregate</i>	412
Квантификаторы	412
Операторы <i>Contains</i> и <i>Any</i>	412
Операторы <i>All</i> и <i>SequenceEqual</i>	413
Методы генерирования коллекций	413
Метод <i>Empty</i>	413
Методы <i>Range</i> и <i>Repeat</i>	414

Глава 10. Запросы LINQ to XML.....	415
Обзор архитектуры	415
Что такое DOM?	415
DOM в технологии запросов LINQ to XML	416
Обзор модели X-DOM	416
Загрузка и синтаксический анализ	418
Сохранение и сериализация	419
Создание экземпляра дерева X-DOM.....	420
Функциональное конструирование	420
Указание содержимого	421
Автоматическое глубокое клонирование	422
Навигация и отправка запросов	423
Навигация по узлам-потомкам	423
Методы <i>FirstNode</i> , <i>LastNode</i> и <i>Nodes</i>	424
Чтение элементов	424
Чтение одного элемента.....	425
Рекурсивные функции.....	426
Навигация по родительским элементам	426
Навигация по элементам одного уровня	427
Навигация по атрибутам	428
Редактирование дерева X-DOM.....	428
Обновление простых значений.....	428
Редактирование узлов-потомков и атрибутов.....	429
Обновление узла через его родителя	430
Удаление последовательности узлов и атрибутов	430
Работа со значениями	431
Установка значений.....	432
Чтение значений	432
Значения и узлы со смешанным содержимым	433
Автоматическая конкатенация элементов <i>XText</i>	434
Документы и объявления.....	434
Класс <i>XDocument</i>	434
XML-объявления	436
Запись объявления в строку	437
Имена и пространства имен	438
Пространства имен в XML	438
Префиксы	439
Атрибуты	440
Указание пространства имен в модели X-DOM	440
X-DOM и пространства имен по умолчанию.....	441
Префиксы.....	443
Аннотации	444
Проецирование в модель X-DOM.....	445
Исключение пустых элементов	447
Проецирование в поток.....	448
Преобразование дерева X-DOM.....	449
Более сложные преобразования	450
Глава 11. Другие XML-технологии	453
Класс <i>XmlReader</i>	453
Чтение узлов	455
Чтение элементов	457
Необязательные элементы	458
Произвольный порядок элементов.....	458
Пустые элементы.....	459
Другие методы семейства <i>ReadXXX</i>	459

Чтение атрибутов.....	461
Узлы-атрибуты	461
Пространства имен и префиксы	462
Класс <i>XmlWriter</i>	463
Запись атрибутов	464
Запись узлов других типов	464
Пространства имен и префиксы	465
Примеры использования классов <i>XmlReader</i> и <i>XmlWriter</i>	465
Работа с иерархическими структурами	465
Комбинирование класса <i>XmlReader</i> или <i>XmlWriter</i> с моделью X-DOM	468
Использование <i>XmlReader</i> с типом <i>XElement</i>	468
Использование <i>XmlWriter</i> с типом <i>XElement</i>	469
Класс <i>XmlDocument</i>	470
Загрузка и сохранение объекта <i>XmlDocument</i>	471
Обход дерева <i>XmlDocument</i>	471
Свойства <i>InnerText</i> и <i>InnerXml</i>	472
Создание узлов и манипуляции с ними	472
Пространства имен	473
Язык XPath.....	474
Самые распространенные операторы XPath	475
Класс <i>XPathNavigator</i>	476
Выдача запросов с учетом пространств имен	477
Класс <i>XPathDocument</i>	478
XSD и проверка корректности документа	478
Проверка соответствия схеме	479
Проверка с помощью класса <i>XmlReader</i>	479
Проверка дерева X-DOM или объекта <i>XmlDocument</i>	481
XSLT	481
Глава 12. Удаление объектов и сборка мусора	483
Интерфейс <i>IDisposable</i> и методы <i>Dispose</i> и <i>Close</i>	483
Стандартная семантика удаления объектов	484
Методы <i>Close</i> и <i>Stop</i>	485
Когда следует удалять объекты.....	485
Условное удаление	486
Сборка мусора и финализаторы.....	488
Вызов метода <i>Dispose</i> из финализатора	490
Как работает сборщик мусора.....	491
Приемы оптимизации.....	492
Принудительная сборка мусора	492
Альтернативы сборке мусора.....	493
Глава 13. Поток и ввод/вывод	495
Архитектура потоков	495
Работа с потоками	497
Чтение и запись	498
Перемещение внутри потока	499
Закрытие и очистка буфера	500
Тайм-ауты.....	500
Безопасность работы нитей с потоками	500
Поток с резервным хранилищем	501
Класс <i>FileStream</i>	501
Конструирование объекта <i>FileStream</i>	501
Указание имени файла	502

Указание режима <i>FileMode</i>	502
Дополнительные характеристики класса <i>FileStream</i>	504
Класс <i>MemoryStream</i>	504
Класс <i>PipeStream</i>	505
Именованные каналы	506
Анонимные каналы	507
Класс <i>BufferedStream</i>	509
Потоковые адаптеры	510
Текстовые адаптеры	510
Классы <i>StreamReader</i> и <i>StreamWriter</i>	512
Кодировки символов	513
Классы <i>StringReader</i> и <i>StringWriter</i>	515
Двоичные адаптеры	515
Закрытие и удаление адаптеров потоков	516
Операции с файлами и каталогами	517
Класс <i>File</i>	518
Атрибуты архивирования и шифрования	519
Безопасность файла	520
Класс <i>Directory</i>	521
Классы <i>FileInfo</i> и <i>DirectoryInfo</i>	522
Класс <i>Path</i>	523
Специальные папки	524
Получение информации о томе	525
Слежение за событиями файловой системы	525
Сжатие файлов	526
Сжатие данных в памяти	528
Изолированная память	528
Виды изоляции	529
Чтение и запись в изолированную память	531
Местоположение изолированной памяти	532
Перебор элементов изолированной памяти	533
Глава 14. Работа в сети.....	535
Архитектура сети	535
Адреса и порты	537
URI	538
Архитектура "запрос/ответ"	540
Класс <i>WebClient</i>	541
Классы <i>WebRequest</i> и <i>WebResponse</i>	542
Прокси-серверы	543
Аутентификация	544
Класс <i>CredentialCache</i>	546
Параллелизм	546
Обработка исключений	548
Поддержка протокола HTTP	549
Заголовки	549
Строки запросов	550
Выгрузка данных формы	550
Cookie	551
Аутентификация с помощью форм	553
SSL	554
Создание HTTP-сервера	554
Работа с протоколом FTP	558
Работа с DNS	560

Отправка почты с помощью класса <i>SmtClient</i>	560
Работа с протоколом TCP	561
Параллелизм и TCP	564
Прием POP3-почты с помощью протокола TCP	565
Глава 15. Сериализация	567
Концепции сериализации	567
Механизмы сериализации	567
Зачем нужны три механизма?	568
Сериализатор с контрактом данных	569
Двоичный сериализатор	569
Класс <i>XmlSerializer</i>	570
Интерфейс <i>IXmlSerializable</i>	570
Форматеры	570
Явная и неявная сериализация	571
Сериализатор с контрактом данных	571
Классы <i>DataContractSerializer</i> и <i>NetDataContractSerializer</i>	572
Применение сериализаторов	573
Использование двоичного форматера	575
Сериализация подклассов	575
Ссылки на объекты	577
Сохранение ссылок на объекты	578
Устойчивость к изменению версии	579
Необходимые члены типа	580
Порядок следования членов	580
Пустое значение и значение <i>null</i>	581
Контракты данных и коллекции	582
Элементы коллекции с подклассами	583
Изменение имен коллекций и элементов	583
Расширение контрактов данных	585
Инструменты сериализации и десериализации	585
Взаимодействие с типом, помеченным как <i>[Serializable]</i>	586
Взаимодействие с типом, реализующим интерфейс <i>IXmlSerializable</i>	588
Двоичный сериализатор	588
С чего начать	589
Атрибуты двоичной сериализации	590
Атрибут <i>[NonSerialized]</i>	590
Атрибуты <i>[OnDeserializing]</i> и <i>[OnDeserialized]</i>	591
Атрибуты <i>[OnSerializing]</i> и <i>[OnSerialized]</i>	592
Атрибут <i>[OptionalField]</i> и сопровождение версий	593
Двоичная сериализация и интерфейс <i>ISerializable</i>	594
Создание подклассов сериализуемых классов	596
XML-сериализация	597
Сериализация на основе атрибутов	598
Атрибуты, имена и пространства имен	599
Порядок следования XML-элементов	600
Подклассы и объекты-потомки	600
Образование подкласса от корневого типа	600
Сериализация объектов-потомков	601
Подклассы объектов-потомков	602
Сериализация коллекций	603
Работа с элементами коллекций, имеющими подклассы	605
Интерфейс <i>IXmlSerializable</i>	605

Глава 16. Сборки	609
Что такое сборка	609
Манифест сборки.....	610
Указание атрибутов сборки.....	611
Манифест приложения.....	611
Развертывание манифеста приложения.....	611
Модули	612
Класс <i>Assembly</i>	613
Подписание сборки	614
Как подписать сборку	615
Отсроченное подписание.....	616
Имена сборок.....	617
Полностью квалифицированные имена.....	618
Класс <i>AssemblyName</i>	618
Глобальный кэш сборок	619
Как устанавливать сборки в глобальном кэше сборок	620
Глобальный кэш сборок и отслеживание версий.....	621
Ресурсы и вспомогательные сборки	622
Непосредственное встраивание ресурсов.....	623
Файлы <i>.resources</i>	624
Файлы <i>.resx</i>	625
Создание <i>.resx</i> -файла из командной строки	625
Чтение файлов <i>.resources</i>	626
Создание URI типа "pack" в Visual Studio.....	627
Вспомогательные сборки.....	627
Построение вспомогательных сборок.....	628
Тестирование вспомогательных сборок	629
Поддержка Visual Studio.....	629
Культуры и субкультуры.....	630
Поиск и загрузка сборок.....	631
Правила поиска сборок и типов	631
Событие <i>AssemblyResolve</i>	632
Загрузка сборок	632
Загрузка из файла	633
Развертывание сборок за пределами базового каталога	634
Упаковка однофайлового приложения.....	636
Избирательное наложение "заплат"	637
Работа со сборками, на которые нет ссылок.....	637
Глава 17. Отражение и метаданные.....	641
Отражение и активизация типов.....	641
Получение экземпляра класса <i>Type</i>	641
Получение типов массивов.....	642
Получение вложенных типов	643
Имена типов	643
Имена вложенных типов.....	644
Имена обобщенных типов	644
Имена массивов и типов указателей.....	644
Имена типов параметров с модификаторами <i>ref</i> и <i>out</i>	644
Базовые типы и интерфейсы.....	645
Создание экземпляров типов.....	645
Обобщенные типы.....	647
Отражение и вызов членов.....	647
Типы-члены.....	649
Члены C# и члены CLR.....	651

Члены обобщенных типов	652
Динамический вызов члена	653
Параметры методов	653
Повышение производительности с помощью делегатов	654
Обращение к закрытым членам	654
Перечисление <i>BindingFlags</i>	655
Обобщенные методы	656
Анонимный вызов членов обобщенного типа	656
Отражение сборок	658
Загрузка сборки в контекст, допускающий только отражение	659
Модули	659
Работа с атрибутами	660
Основы работы с атрибутами	660
Атрибут <i>AttributeUsage</i>	661
Определение собственного атрибута	662
Чтение атрибутов на этапе выполнения	663
Получение атрибутов в контексте отражения	665
Динамическое генерирование кода	665
Генерирование IL-кода с помощью класса <i>DynamicMethod</i>	665
Стек вычислений	667
Передача параметров классу <i>DynamicMethod</i>	668
Генерирование локальных переменных	669
Ветвление	670
Создание объектов и вызов экземплярных методов	670
Обработка исключений	672
Генерирование сборок и типов	673
Сохранение порожденных сборок	674
Объектная модель <i>Reflection.Emit</i>	675
Генерирование членов типа	676
Генерирование методов	677
Генерирование методов экземпляров	678
Атрибут <i>HideBySig</i>	678
Генерирование полей и свойств	679
Генерирование конструкторов	680
Вызов конструкторов базовых классов	681
Добавление атрибутов	682
Генерирование обобщенных методов и типов	682
Определение обобщенных методов	682
Определение обобщенных типов	684
Неудобные цели генерирования	684
Несозданные замкнутые обобщенные типы	684
Циклические зависимости	685
Анализ IL-кода	688
Написание дизассемблера	688
Глава 18. Безопасность	695
Разрешения	695
Классы <i>CodeAccessPermission</i> и <i>PrincipalPermission</i>	696
Интерфейс <i>IPermission</i>	697
Класс <i>PermissionSet</i>	698
Декларативная и принудительная безопасность	699
Безопасность обращения к коду	699
Как CLR выделяет разрешения	701

Работа в изоляции	703
Необязательные разрешения	704
Флаг <i>SecurityAction.RequestOptional</i>	704
Изоляция другой сборки	705
Требования связывания и частичное доверие к вызываемому коду	706
Как утверждать права	707
Безопасность в операционной системе	708
Работа под стандартной пользовательской учетной записью	709
Повышение прав и виртуализация	710
Безопасность, основанная на именах и ролях	711
Указание пользователей и ролей	712
Обзор криптографических возможностей	713
Защита данных Windows	713
Хеширование	714
Симметричное шифрование	716
Шифрование в памяти	717
Образование цепочек из потоков шифрования	719
Уничтожение шифрующих объектов	720
Управление ключами	721
Шифрование с открытым ключом и постановка цифровой подписи	721
Класс <i>RSA</i>	722
Цифровая подпись	723
Глава 19. Нити выполнения	725
Правильное использование нитей	725
Основы	726
Передача данных	729
Совместное обращение к данным	730
Пулы нитей	731
Оптимизация пула	733
Приоритетные и фоновые нити	734
Приоритет нити	735
Обработка исключений	736
Асинхронные делегаты	737
Синхронизация	739
Приостановка выполнения	740
Приостановка и работа вхолостую	741
Метод <i>SpinWait</i>	741
Блокировка	741
Выбор синхронизирующего объекта	743
Вложенные блокировки	744
Когда ставить блокировку	744
Блокировка и атомарность	745
Производительность кода, соперничество блокировок и взаимные блокировки	745
Класс <i>Mutex</i>	746
Класс <i>Semaphore</i>	747
Безопасность нитей	748
Безопасность нитей и типы .NET Framework	750
Блокирование безопасных объектов	751
Статические методы	751
Безопасность нитей в серверах приложений	752
Безопасность нитей в многофункциональных клиентских приложениях	753
Незадерживающая синхронизация	754
Атомарность и класс <i>Interlocked</i>	754
Барьеры памяти и неэкзируемость	756

Сигнализация с помощью дескрипторов ожидания событий	757
Создание и уничтожение дескрипторов ожидания.....	759
Двухсторонняя сигнализация.....	760
Создание объекта <i>EventWaitHandle</i> для нескольких процессов.....	761
Пул дескрипторов ожидания.....	761
Методы <i>WaitAny</i> , <i>WaitAll</i> и <i>SignalAndWait</i>	763
Сигнализация с помощью методов <i>Wait</i> и <i>Pulse</i>	763
Как пользоваться методами <i>Wait</i> и <i>Pulse</i>	764
Очередь "производитель/потребитель".....	767
Тайм-ауты метода <i>Wait</i>	770
Двухсторонняя сигнализация.....	771
Имитация дескрипторов ожидания.....	773
Методы <i>Interrupt</i> и <i>Abort</i>	774
Метод <i>Interrupt</i>	774
Метод <i>Abort</i>	776
Безопасное завершение.....	776
Локальное хранение данных.....	777
Класс <i>BackgroundWorker</i>	778
Создание подкласса для <i>BackgroundWorker</i>	782
Класс <i>ReaderWriterLockSlim</i>	783
Обновляемые блокировки и рекурсия.....	785
Рекурсия блокировок.....	787
Таймеры.....	788
Многонитевые таймеры.....	788
Однонитевые таймеры.....	790
Глава 20. Асинхронные методы.....	793
Для чего существуют асинхронные методы.....	793
Сигнатуры асинхронных методов.....	794
Асинхронные методы и асинхронные делегаты.....	795
Применение асинхронных методов.....	796
Написание асинхронных методов.....	799
Поддельные асинхронные методы.....	802
Альтернативы асинхронным методам.....	803
Асинхронные события.....	804
Глава 21. Домены приложений.....	805
Архитектура доменов приложений.....	805
Создание и уничтожение доменов приложений.....	805
Работа с несколькими доменами приложения.....	808
Метод <i>DoCallBack</i>	810
Домены и нити.....	811
Совместное обращение к данным из разных доменов.....	812
Применение слотов.....	812
Удаленное взаимодействие внутри процесса.....	813
Изолирование типов и сборок.....	815
Раскрытие типа.....	817
Глава 22. Интеграция с DLL.....	819
Вызов кода из DLL.....	819
Маршалинг обычных типов.....	820
Маршалинг классов и структур.....	821
Входной и выходной маршалинг.....	822
Обратные вызовы из неуправляемого кода.....	823

Имитация объединения из языка C.....	824
Совместно используемая память	825
Отображение структуры в неуправляемую память	827
Ключевое слово <i>fixed</i>	830
Справочник по атрибутам	831
Атрибут <i>DllImport</i>	831
Атрибут <i>StructLayout</i>	832
Атрибут <i>FieldOffset</i>	833
Атрибут <i>MarshalAs</i>	833
Элементы перечисления <i>UnmanagedType</i>	834
Глава 23. Диагностика.....	837
Условная компиляция.....	837
Условная компиляция как альтернатива статическим переменным-флагам.....	838
Атрибут <i>Conditional</i>	839
Альтернативы атрибуту <i>Conditional</i>	840
Классы <i>Debug</i> и <i>Trace</i>	840
Класс <i>TraceListener</i>	841
Принудительная запись на диск и закрытие слушателей.....	843
Интеграция с отладчиком.....	843
Присоединение отладчика и прерывание выполнения.....	844
Атрибуты отладчика.....	844
Процессы и нити	845
Анализ работающего процесса.....	845
Анализ нитей в процессе.....	845
Классы <i>StackTrace</i> и <i>StackFrame</i>	846
Журнал событий Windows.....	848
Запись в журнал событий.....	848
Чтение журнала событий	849
Мониторинг журнала событий.....	849
Счетчики производительности	850
Просмотр доступных счетчиков.....	851
Чтение данных из счетчиков производительности.....	852
Создание счетчиков и запись данных о производительности	853
Класс <i>Stopwatch</i>	855
Глава 24. Регулярные выражения.....	857
Основы регулярных выражений	857
Откомпилированные регулярные выражения	858
Перечисление <i>RegexOptions</i>	859
Экранирование символов.....	860
Множества символов	861
Квантификаторы	862
"Жадные" и "ленивые" квантификаторы	863
Директивы нулевой длины	863
Просмотр вперед и просмотр назад	864
Якоря	865
Границы слов	866
Группы	866
Именованные группы.....	867
Замещение и разбивка на части	868
Делегат <i>MatchEvaluator</i>	869
Разбивка текста.....	869

Сборник рецептов по регулярным выражениям.....	870
Рецепты	870
Шаблон для номера социальной страховки или номера телефона в США	870
Извлечение пар "имя=значение" (по одной в строчке текста).....	870
Проверка сильного пароля.....	870
Строки длиной не менее 80 символов	870
Разбор даты и времени.....	871
Шаблон для римских чисел.....	871
Удаление повторяющихся слов.....	871
Подсчет слов.....	872
Шаблон для глобального уникального идентификатора.....	872
Разбор XML-тега.....	872
Разбивка на части слова, написанного в стиле camel.....	872
Получение допустимого имени файла.....	872
Экранирование символов Unicode для HTML-кода	873
Справочник по языку регулярных выражений	873
Приложение 1. Ключевые слова языка C#.....	879
Приложение 2. Соответствие сборок пространствам имен.....	889
Предметный указатель.....	899

Об авторах

Джозеф Албахари (Joseph Albahari) работает руководителем программных проектов на C# в австралийской компании Egton Software Services, которая является партнером крупнейшего производителя программных продуктов для здравоохранения в Великобритании. Он более 15 лет занимается разработкой крупных коммерческих приложений на .NET и других платформах для таких областей, как медицина, телекоммуникации и образование. Джозеф специализируется на создании заказных компонентов и написал прикладные платформы для трех компаний.

Бен Албахари (Ben Albahari) ранее работал менеджером программных проектов в Microsoft, где он, в частности, занимался созданием .NET Compact Framework и ADO.NET. Затем он стал соучредителем компании Genamics, поставляющей инструментальные средства для программистов, работающих на C# и J++, а также программные продукты для анализа ДНК и белковых последовательностей. Он был соавтором предыдущих изданий этой книги, а также "C# Essentials" ("Основы C#"), первой книги издательства O'Reilly, посвященной этому языку.



Предисловие

Каждый выпуск C# и .NET Framework содержит новые функциональные возможности и имеет большой потенциал для повышения производительности. Версия C# 3.0 содержит самые значительные изменения в языке за все время его развития, в том числе унифицированный синтаксис запросов, называемый LINQ (Language Integrated Query, запрос, интегрированный в язык). Он наводит мост между программами и их источниками данных, традиционно отделяемым друг от друга. Он также приближает C# к функциональным языкам, таким как LISP и Haskell.

Платой за рост функциональности является необходимость изучить больший объем материала. Хотя такие инструменты, как IntelliSense от Microsoft (а также многочисленные справочные материалы), отлично помогают справиться с этой задачей, все они предполагают наличие у читателя *карты концептуального знания*. Наша книга как раз и предлагает подобную карту в сжатом и унифицированном стиле, без излишней информации и длинных предисловий.

Это издание, в отличие от предыдущих, построено исключительно вокруг теоретических концепций и практических примеров, что делает книгу пригодной как для последовательного чтения, так и для просмотра в произвольном порядке. Кроме того, изложение материала ведется более углубленно, чем ранее, поскольку рассчитано на читателя с более низким уровнем начальных знаний. Поэтому книга стала более понятной, чем предыдущие издания.

В книге описываются язык C#, среда CLR и сборки ядра Framework. Мы решили сосредоточиться на этих темах, чтобы уделить больше внимания таким трудным вопросам, как нити выполнения, безопасность и домены приложений. Мы специально отмечаем функциональные возможности, впервые появившиеся в C# 3.0, и соответствующие нововведения во Framework, так что нашу книгу можно использовать и как справочник по C# 2.0.

Целевая аудитория

Книга рассчитана на аудиторию с уровнем подготовки от среднего до высокого. Предварительное знакомство с языком C# не требуется, но определенный опыт программирования необходим. Для новичка книга может послужить дополнением к учебнику программирования (но не заменит его).

Даже если вы знаете C# 2.0, вы найдете для себя полезную информацию на более чем ста страницах, посвященных LINQ и другим нововведениям в C# 3.0. Кроме

того, во многих главах мы постараемся углубить ваши знания языка и ядра платформы Framework.

Эта книга образует идеальное сочетание с любой из множества книг, посвященных таким прикладным технологиям, как WPF, ASP.NET или WCF. Особенности языка и платформы .NET Framework, которые остались за рамками этих книг, подробно описываются в нашей книге и наоборот.

Если вам нужна книга, где кратко обсуждаются все существующие технологии .NET Framework, то наша книга не для вас. Она также не подходит тому, кто ищет замену для IntelliSense (то есть алфавитные списки типов и их членов, присутствующие в предыдущих изданиях).

Как организована эта книга

Гл. 1 — это введение в C# и .NET Framework. *Гл. 2–4* целиком посвящены языку C#, начиная с основ синтаксиса, типов и переменных и заканчивая более сложными темами, такими как небезопасный код или препроцессорные директивы. Если вы не знакомы с языком, то вам следует прочитать эти главы последовательно, хотя разделы *гл. 4* можно читать в любом порядке.

В остальных главах описывается ядро платформы .NET Framework и затрагиваются такие темы, как LINQ, XML, коллекции, ввод/вывод и работа в сети, управление памятью, отражение, атрибуты, безопасность, нити выполнения, домены приложений и взаимодействие с небезопасным кодом. Порядок чтения большинства глав не принципиален, за исключением *гл. 6, 7 и 13*, в которых закладываются основы для последующих тем. *Гл. 8, 9 и 10*, посвященные LINQ, тоже лучше читать по порядку.

Что необходимо для чтения этой книги

Для примеров кода, приведенных в книге, требуется компилятор C# 3.0 (или 2.0) и Microsoft .NET Framework 3.5 (или 3.0/2.0). Кроме того, вам понадобится документация Microsoft по платформе .NET. Простейший способ получить все сразу (а также интегрированную среду разработки) сводится к установке Microsoft Visual Studio. Для изучения материала, изложенного в книге, подойдет любое издание Visual Studio, в том числе и Visual Studio Express (которое сейчас можно загрузить бесплатно). Кроме того, Visual Studio содержит экспресс-версию SQL Server, которая потребуется для выполнения примеров LINQ to SQL, и инструментальное средство IntelliSense, выводящее на экран список членов типа по мере того, как вы набираете код.

Если вы ничего не имеете против использования обычного текстового редактора и командной строки, альтернативным решением может быть загрузка .NET Framework SDK. В комплект входят компилятор, документация по .NET и дополнительные утилиты командной строки.

Но проще всего загрузить и установить только Microsoft .NET Framework Runtime. Эта среда содержит компилятор, запускаемый из командной строки, однако в ней нет других утилит и документации.

Соглашения, использованные в книге

На рис. 1 приведена примерная схема изображения типов. Абстрактные классы представлены параллелограммами, а интерфейсы — кружками. Наследование обозначается непрерывной стрелкой, идущей от подтипа к базовому типу. Линия с окрашенным ромбом на конце означает любое другое отношение (ассоциирование, агрегирование или композицию).

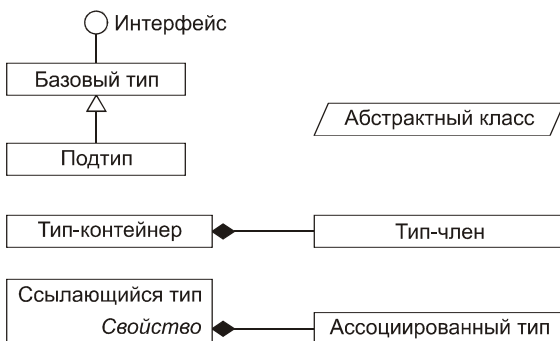


Рис. 1. Примерная схема

В книге использованы следующие типографские соглашения:

- *курсив* выделяет новые термины;
- моноширинный шрифт обозначает код C#, ключевые слова и идентификаторы, а также результат работы программы;
- моноширинный шрифт, полужирный выделяет в коде места, на которые следует обратить внимание;
- моноширинный шрифт, курсив обозначает текст, который должен быть заменен на пользовательский.

ПРИМЕЧАНИЕ

Содержит совет, предложение или замечание общего характера.

! ПРЕДУПРЕЖДЕНИЕ

Содержит предупреждение или предостережение.

Использование фрагментов кода

Эта книга должна помочь вам в работе. Вы можете использовать код, приведенный в этой книге, в своих программах и документах. Вы не обязаны получать наше раз-

решение за исключением тех случаев, когда воспроизводите значительные фрагменты кода. Например, для написания программы, в состав которой входят отрывки кода, приведенные в этой книге, разрешение не требуется. Но продажа или распространение CD-ROM с примерами из книг O'Reilly *требует* разрешения. Когда вы отвечаете на чей-то вопрос, цитируя эту книгу и фрагменты кода из нее, вы не нуждаетесь в разрешении. Если вы включаете значительный объем кода из этой книги в документацию к своему продукту, вы *должны* получить разрешение.

Мы не требуем обязательных ссылок на книгу при цитировании, но будем благодарны, если вы их приведете. Ссылка обычно содержит название книги, автора, издательство и ISBN. Например, "C# 3.0 in a Nutshell, by Joseph Albahari and Ben Albahari. Copyright 2007 Ben Albahari and Joseph Albahari, 978-0-596-52757-0".

Если вы сомневаетесь, не нарушаете ли вы разрешение на использование кода, изложенное выше, не стесняйтесь обратиться по адресу permissions@oreilly.com.

Благодарности

От Джозефа Албахари

Прежде всего я хочу поблагодарить своего брата и соавтора Бена Албахари за то, что он убедил меня участвовать в этом, как оказалось, весьма успешном проекте. Некоторые главы потребовали плотной совместной работы, которая еще больше сблизила нас. Мне было очень приятно работать с Беном, исследуя особо трудные темы. Он разделяет мое стремление подвергать сомнению прописные истины и упорно разбираться в сложных вопросах, пока не станет до конца понятно "как это работает".

Группа научных редакторов, многие из которых являются теперешними или бывшими сотрудниками Microsoft, была выше всяких похвал. Своим высоким качеством книга обязана глубине и широте познаний Николаса Палдино (Nicholas Paldino). То же самое я должен сказать о Джоэле Побаре (Joel Pobar) и о его глубочайшем знании среды CLR. Моей искренней благодарности заслуживают Кшиштоф Цвалина (Krzysztof Cwalina), Мэтт Уоррен (Matt Warren), Глин Гриффитс (Glyn Griffiths), Ион Василиан (Ion Vasilian), Брэд Абрамс (Brad Abrams), Сэм Джентил (Sam Gentile) и Адам Натан (Adam Nathan).

Я также хочу поблагодарить своего редактора Джона Осборна (John Osborn) за то, что он уделил много времени и сил координации этого проекта, и корректора Одри Дойл (Audrey Doyle) за наведение окончательного блеска. Я благодарю членов моей семьи Соню Албахари (Sonia Albahari) и Мири Албахари (Miri Albahari) за поддержку. Моя сестра Мири, которая сама недавно опубликовала книгу, прекрасно понимала, что такое отсутствие свободного времени!

Приношу благодарность Лутцу Рёдеру (Lutz Roeder), чья утилита .NET Reflector была моим основным инструментом анализа кода. Благодарю читателей моей электронной веб-книги "Threading in C#", чьи комментарии и полезные советы помогли мне в работе над настоящим изданием.

В заключение я хочу поблагодарить свою фирму Egton Software Services за использование технологии LINQ, лишь только появилась такая возможность. Я особенно благодарен Джеймсу Майлсу (James Miles) за его энтузиазм и творческий подход. Наша команда получила бесценный опыт реальной работы, необходимый каждому, кто пишет о технологии до ее официального выпуска.

От Бена Албахари

Поскольку мой брат выразил свою благодарность первым, вы легко догадаетесь, что я хочу сказать. ☺ Мы оба программируем практически с детства (у нас был один на двоих компьютер Apple IIe, брат писал свою операционную систему, а я игру "Виселица"), так что уже давно пора написать книгу вдвоем. Я надеюсь, что богатый опыт, который мы получили, работая над книгой, передастся вам при ее чтении.

Я хотел бы также поблагодарить своих бывших коллег по работе в Microsoft. Там работает много замечательных людей, не только умных, но и обладающих прекрасными человеческими качествами, и я скучаю по ним. В частности, я многому научился у Брайана Бекмана (Brian Beckman), которому весьма обязан.



Введение в C# и .NET Framework

C# — это универсальный объектно-ориентированный язык программирования, обеспечивающий безопасность преобразования типов. Целью его создания было повышение производительности труда программистов. Поэтому в языке сбалансированы простота, выразительность и эффективность. Главным разработчиком языка, начиная с первой версии, является Андерс Хейлсберг (Anders Hejlsberg), создатель Turbo Pascal и архитектор Delphi. Язык C# нейтрален к платформе, но в силу некоторых своих характеристик особенно хорошо работает на платформе Microsoft .NET Framework.

Объектная ориентированность

Язык C# является полноценной реализацией объектно-ориентированной парадигмы, включающей в себя *инкапсуляцию*, *наследование* и *полиморфизм*. Инкапсуляция означает создание вокруг *объекта* границы, отделяющей его внешнее поведение от внутренней реализации. С точки зрения объектной ориентированности отличительными чертами языка C# являются:

Унифицированная система типов

Основным строительным "кирпичиком" программы на языке C# является инкапсулированная единица данных и функций, называемая *типом*. C# имеет унифицированную систему типов, в которой все типы, в конечном счете, имеют общий базовый тип. Это означает, что все типы, от простых, таких как целые числа, до типов, представляющих сложные пользовательские объекты, имеют один базовый набор функциональных возможностей. Например, любой тип может быть преобразован в строку методом `ToString`.

Классы и интерфейсы

В чистой объектно-ориентированной парадигме единственным видом типа является *класс*. Однако в C# существуют и другие виды типов, к которым относятся и *интерфейс* (аналогичный интерфейсам Java). Интерфейс подобен классу с тем отличием, что представляет собой только объявление типа, без реализации. Он особенно полезен в тех случаях, когда требуется множественное наследование (в отличие от таких языков, как C++ и Eiffel, язык C# не поддерживает множественное наследование классов).

Свойства, методы и события

В чистой объектно-ориентированной парадигме все функции являются *методами* (яркий пример — язык Smalltalk). В C# методы представляют собой лишь один из нескольких видов функций, наряду со *свойствами*, *событиями* (и другими видами). Свойства — это функции, инкапсулирующие фрагменты состояния объекта, например, цвет кнопки или текст надписи. События — это функции, которые упрощают действия, связанные с изменением состояния объекта.

Безопасность типов

Язык C#, в основном, безопасен в отношении преобразования типов. Это означает, что типы могут взаимодействовать только по протоколам, которые они же и определяют, обеспечивая тем самым внутреннюю непротиворечивость каждого типа. Например, C# не позволяет вам работать со *строковым* типом так, словно он *целочисленный*.

Говоря более конкретно, C# поддерживает *статическую типизацию*, т. е. обеспечивает безопасность типа на этапе компиляции. Это является дополнением к *динамической безопасности* типов, обеспечиваемой средой .NET CLR на этапе выполнения.

Статическая типизация позволяет выявить большое количество ошибок еще до запуска программы. Она избавляет вас от необходимости проводить специальное тестирование, поскольку перекладывает на компилятор проверку согласованности всех типов в программе. В результате большие программы становятся более управляемыми, более предсказуемыми и более устойчивыми. Кроме того, статическая типизация позволяет при написании программы в Visual Studio .NET использовать такие инструменты, как IntelliSense, поскольку всегда известен тип каждой конкретной переменной, и, следовательно, известно, какие именно методы можно вызывать для нее.

Про C# говорят, что это *язык со строгой типизацией*, потому что его правила, действующие в отношении типов (как статически, так и динамически), являются очень строгими. Например, вы не можете передавать число с плавающей точкой в качестве аргумента функции, работающей с целым, если вы предварительно *явно* не преобразуете число с плавающей точкой в целое. Этот принцип позволяет избежать многих ошибок.

Строгая типизация также играет свою роль при работе кода C# в изолированной среде, где каждый аспект безопасности контролируется приложением-хостом. В изолированной среде очень важно то, что вы не можете произвольным образом изменить (и "испортить") состояние объекта, обойдя его правила типизации.

Управление памятью

В программе на языке C# предполагается, что управление памятью производится средой выполнения автоматически. Среда CLR имеет сборщик мусора, работаю-

щий как компонент вашей программы и освобождающий память от объектов, на которые больше нет ссылок. Это избавляет программистов от необходимости явно освобождать память и снимает проблему некорректных указателей, типичную для таких языков, как C++.

Язык C# не исключает использование указателей; он просто делает их ненужными для большинства задач программирования. Вы можете применять указатели в ситуациях, требующих высокой производительности, а также для улучшения взаимодействия с другими программами, но делать это разрешено только в блоках кода, явно помеченных как небезопасные.

Работа на разных платформах

Как правило, язык C# применяется для написания кода, работающего на платформах Windows. Хотя корпорация Microsoft стандартизировала C# и CLR в соответствии с правилами ECMA, существует относительно мало ресурсов (как связанных с Microsoft, так и независимых), предназначенных для поддержки C# на платформах, отличных от Windows. Это означает, что если вашей основной задачей является разработка платформенно-независимого приложения, то разумно сделать выбор в пользу такого языка, как Java. Таким образом, язык C# можно применять при написании кроссплатформенного кода в следующих случаях:

- ❑ код C# работает на сервере и обслуживает код DHTML, который может выполняться на любой платформе. Это как раз случай ASP.NET;
- ❑ код C# выполняется в среде, отличной от Microsoft CLR. Самым известным примером является проект Mono, имеющий собственный компилятор C# и среду выполнения, работающую в Linux, Solaris, Mac OS X и Windows;
- ❑ код C# выполняется на хосте, поддерживающем Microsoft Silverlight (в Windows или Mac OS X). Это новая технология, аналогичная Flash Player от Adobe.

Язык C# и среда CLR

Для выполнения кода C# требуется среда, имеющая такие функциональные возможности, как автоматическое управление памятью и обработка исключений. Структура языка C# соответствует структуре среды CLR, обладающей этими возможностями (хотя в техническом отношении C# не зависит от среды CLR). Более того, система типов C# хорошо ложится на систему типов CLR (например, в обеих используются одни и те же объявления базовых типов).

Среда CLR и платформа .NET Framework

Платформа .NET Framework состоит из среды выполнения CLR (Common Language Runtime, общезыковая среда выполнения) и большого количества библиотек. Библиотеки, в свою очередь, делятся на две группы: библиотеки ядра (которым и посвящена книга) и прикладные, зависящие от библиотек ядра. На рис. 1.1 пред-

ставлен общий вид этих библиотек (который может служить своеобразной картой нашей книги).

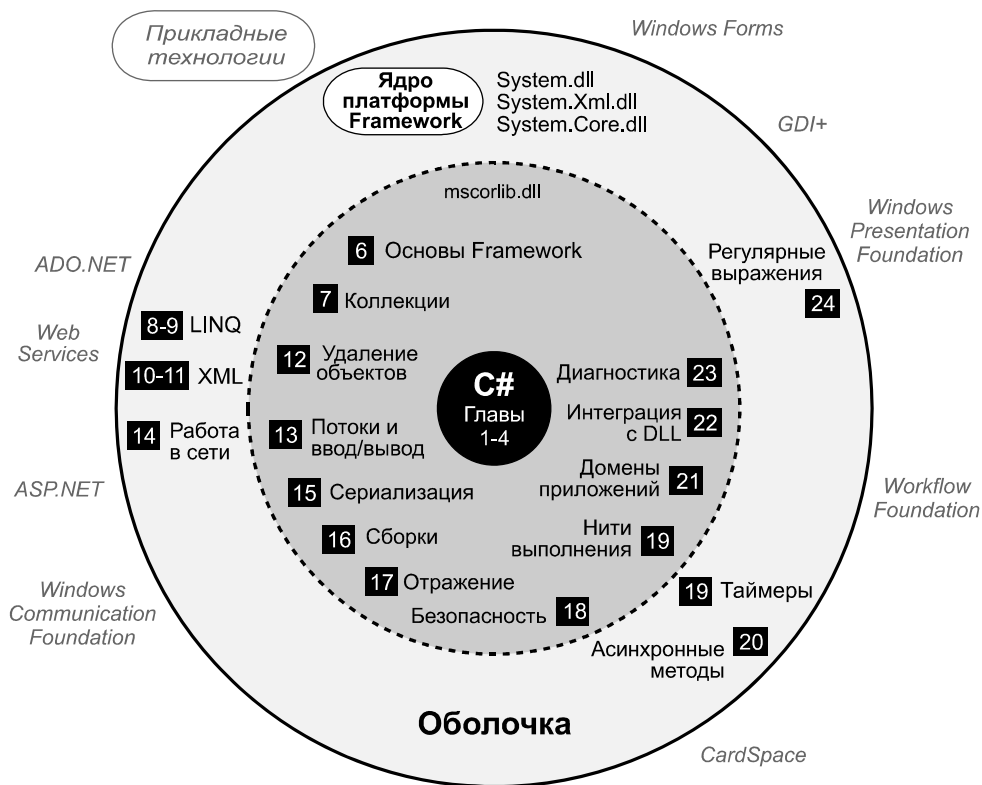


Рис. 1.1. Темы, затрагиваемые в книге, и номера глав, где они обсуждаются. Названия специальных технологий и библиотек, выходящих за рамки этой книги, выделены серым шрифтом и расположены за пределами оболочки

CLR является средой выполнения *управляемого кода*. Язык C# представляет собой один из *управляемых языков*, т. е. он компилируется в управляемый код. Такой код помещается в *сборку*, имеющую форму либо выполняемого файла (с расширением exe), либо библиотеки (с расширением dll) и содержащую также информацию о типах, называемых *метаданными*.

Управляемый код — это код на языке IL (Intermediate Language, промежуточный язык). Когда CLR загружает сборку, IL-код преобразуется в машинный код, например, x86. Это преобразование выполняется компилятором JIT (Just-In-Time, своевременный), являющимся частью CLR. Сборка сохраняет практически все конструкции исходного языка, что облегчает проверку кода и даже его динамическое генерирование.

ПРИМЕЧАНИЕ

Приложение .NET Reflector, написанное Лутцем Рёдером (Lutz Roeder), является прекрасным инструментом для изучения содержимого сборки. Оно также может быть использовано в качестве декомпилятора.

Среда CLR имеет большое количество служб, действующих на этапе выполнения. В их число входят службы управления памятью, загрузки библиотек и обеспечения безопасности.

Среда CLR нейтральна в языковом отношении и позволяет разработчикам создавать приложения на самых разных языках (C#, Visual Basic .NET, Managed C++, Delphi .NET, Chrome .NET и J#).

Платформа .NET Framework содержит библиотеки, позволяющие написать практически любое Windows-приложение. В *гл. 5* дан обзор библиотек .NET.

Что нового в C# 3.0

Новые функциональные возможности версии C# 3.0 относятся, в основном, к технологии LINQ (Language Integrated Query, запрос, интегрированный в язык). Ее создателей вдохновила работа над проектом Omega (ранее известным под именами X# и Xen). Главный архитектор Эрик Мейер (Erik Meijer) тесно сотрудничал с Андерсом Хейлсбергом при включении этой технологии в состав C#.

LINQ предоставляет возможность писать SQL-подобные запросы прямо в программе на языке C#, причем их корректность проверяется *статически*. Архитектура LINQ позволяет выполнять запросы локально или удаленно. Платформа .NET Framework предоставляет API-интерфейсы, использующие LINQ при обращении к коллекциям, удаленным базам данных и XML-документам. В число функциональных возможностей C# 3.0 входят:

- лямбда-выражения;
- методы расширения;
- неявно типизированные локальные переменные;
- синтаксис, облегчающий восприятие запроса;
- анонимные типы;
- инициализаторы объектов;
- неявно типизированные массивы;
- автоматические свойства;
- частичные методы;
- деревья выражений.

Лямбда-выражения представляют собой миниатюрные функции, создаваемые динамически. Они являются результатом естественного развития анонимных методов, впервые появившихся в C# 2.0. Фактически лямбда-выражения обладают всеми функциональными возможностями анонимных методов. Например, следующее лямбда-выражение возводит в квадрат целое число:

```
Func<int,int> square = x => x * x;  
Console.WriteLine (square(3)); // 9
```


Читателям, знакомым с языками *функционального программирования*, такими как Scheme и Haskell, лямбда-выражения не покажутся чем-то необычным. В языке C# лямбда-выражения применяются, в основном, в LINQ-запросах, например:

```
string[] names = { "Tom", "Dick", "Harry" };
IEnumerable<string> filteredNames =           // Выбрать только имена
    Enumerable.Where (names, n => n.Length >= 4); // из 4 и более символов
```

Методы расширения расширяют существующий тип, добавляя в него новые методы, но не меняя его определение. Они действуют как "синтаксический сахар", создавая впечатление, что статические методы ведут себя как методы экземпляра. Поскольку операторы LINQ-запросов реализованы в виде методов расширения, мы можем упростить предыдущий пример:

```
string[] names = { "Tom", "Dick", "Harry" };
IEnumerable<string> filteredNames = names.Where (n => n.Length >= 4);
```

Неявно типизированные локальные переменные разрешают вам не указывать тип переменной при ее объявлении, позволяя компилятору распознать его самостоятельно. Поскольку компилятор способен определить тип переменной `filteredNames`, мы можем еще больше упростить наш запрос:

```
var filteredNames = names.Where (n => n.Length == 4);
```

Синтаксис, облегчающий восприятие запросов, позволяет писать запросы в стиле SQL. Он значительно упрощает некоторые виды запросов и тоже выступает в роли "синтаксического сахара" для запросов, имеющих вид лямбда-выражений. Перепишем предыдущий пример в соответствии с синтаксисом, облегчающим восприятие запросов:

```
var filteredNames = from n in names where n.Length >= 4 select n;
```

Анонимные типы — это простые классы, создаваемые динамически и обычно используемые при окончательном выводе запросов:

```
var query = from n in names where n.Length >= 4
    select new {
        Name = n,
        Length = n.Length
    };
```

Вот еще более простой пример:

```
var dude = new { Name = "Bob", Age = 20 };
```

Неявно типизированные массивы исключают необходимость в объявлении типа массива, когда массив конструируется и инициализируется за один шаг:

```
var dudes = new[]
{
    new { Name = "Bob", Age = 20 },
    new { Name = "Rob", Age = 30 }
};
```

Инициализаторы объектов упрощают конструирование, позволяя устанавливать значения полей одновременно с вызовом конструктора. Инициализаторы объектов работают как с анонимными, так и с именованными типами. В следующем примере демонстрируется эта новая возможность языка C# 3.0 и приводится эквивалентный код на C# 2.0:

```
// C# 3.0
Bunny b1 = new Bunny { Name="Bo", LikesCarrots=true, LikesHumans=false };
// C# 2.0
Bunny b2 = new Bunny( );
b2.Name = "Bo";
b2.LikesHumans = false;
```

Автоматические свойства облегчают работу программиста в тех случаях, когда нужно написать свойство, просто читающее или устанавливающее закрытое поле, содержащее резервное значение для свойства. В следующем примере компилятор автоматически генерирует закрытое резервное поле для свойства x:

```
public class Stock
{
    // C# 3.0:
    public decimal X { get; set; }

    // C# 2.0:
    private decimal y;
    public decimal Y
    {
        get { return y; }
        set { y = value; }
    }
}
```

Частичные методы позволяют автоматически сгенерированному частичному классу предоставлять программисту возможности для ручной настройки. Технология LINQ to SQL использует частичные методы в специализированных классах, отображающих SQL-таблицы. Приведем пример пары частичных методов, один из которых сгенерирован автоматически, а другой написан вручную:

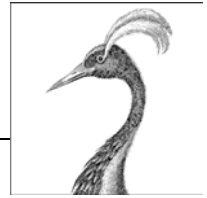
```
// PaymentFormGen.cs – сгенерирован автоматически
partial class PaymentForm
{
    ...
    partial void ValidatePayment (decimal amount);
}
// PaymentForm.cs – написан вручную
partial class PaymentForm
{
    ...
    partial void ValidatePayment (decimal amount)
```

```
{
    if (amount > 100)
        ...
}
```

Деревья выражений являются миниатюрными объектными моделями, описывающими лямбда-выражения. Компилятор C# 3.0 генерирует деревья выражений, когда лямбда-выражение присваивается специальному типу `Expression<TDelegate>`:

```
Expression<Func<string,bool>> predicate = s => s.Length > 10;
```

Деревья выражений обеспечивают удаленное выполнение LINQ-запросов (например, на сервере базы данных), потому что они могут быть проанализированы и оттранслированы на этапе выполнения (например, в SQL-запрос).



ОСНОВЫ ЯЗЫКА C#

В этой главе мы изложим основы языка C#.

Первая программа на C#

Разберем программу, которая умножает число 12 на 30 и выводит результат (360) на экран. Две косых, идущих подряд, указывают, что часть строчки после них является *комментарием*.

```
using System;           // импортирование пространства имен
class Test              // объявление класса
{
    static void Main( ) // объявление метода
    {
        int x = 12 * 30; // оператор 1
        Console.WriteLine(x); // оператор 2
    }                     // конец метода
}                         // конец класса
```

Основу программы составляют два *оператора*. Операторы в C# выполняются последовательно. Каждый оператор заканчивается точкой с запятой:

```
int x = 12 * 30;
Console.WriteLine(x);
```

Первый оператор вычисляет *выражение* $12 * 30$ и сохраняет результат в *локальной переменной* x целого типа (`int`). Второй оператор вызывает *метод* `WriteLine` класса `Console`, чтобы вывести переменную x на экран.

Метод выполняет некое действие с помощью последовательности операторов, называемой *операторным блоком* и представляющей собой пару фигурных скобок, содержащую несколько операторов (возможно, ни одного). Мы определили метод по имени `Main`:

```
static void Main( )
{
    ...
}
```

Применение функций высокого уровня, обращающихся к функциям низкого уровня, упрощает код. Мы можем переработать нашу программу так, что она будет содержать метод многократного использования, умножающий целое число на 12:

```
using System;
class Test
{
    static void Main( )
    {
        Console.WriteLine(FeetToInches(30));    // 360
        Console.WriteLine(FeetToInches(100));   // 1200
    }
    static int FeetToInches(int feet)
    {
        int inches = feet * 12;
        return inches;
    }
}
```

Метод может получать *входные* данные от вызывающей функции, для чего в нем определяются *параметры*, и возвращать ей *выходные* данные, и с этой целью в нем определяется *тип возвращаемого значения*. Мы создали метод по имени `FeetToInches` с одним параметром для ввода значения в футах и с типом возвращаемого значения для вывода значения в дюймах:

```
static int FeetToInches(int feet) {...}
```

Литералы 30 и 100 являются *аргументами*, передаваемыми методу `FeetToInches`. В нашем примере метод `Main` ничего не содержит в круглых скобках, потому что у него нет параметров, а поскольку он не возвращает никакого значения, то перед его именем стоит ключевое слово `void`:

```
static void Main( )
```

C# распознает метод по имени `Main` как точку входа в программу, определяемую по умолчанию. Метод `Main`, в принципе, может возвращать целое значение (а не `void`), передавая его вычислительной среде. Кроме того, он может принимать массив строковых аргументов (в который будут занесены аргументы, передаваемые программе). Например:

```
static int Main(string[] args) {...}
```

ПРИМЕЧАНИЕ

Массив (например, `string[]`) представляет собой набор из фиксированного количества элементов некоторого типа. Массивы определяются с помощью пары квадратных скобок, помещенной после типа элемента (см. *разд. "Массивы"* далее в этой главе).

Методы — это один из нескольких видов функций в C#. Другим видом являются *знаки операций*, например, используемый здесь знак `*`, обозначающий умножение. Существуют также *конструкторы*, *свойства*, *события*, *индексаторы* и *деструкторы*.

В приведенном примере два метода сгруппированы в класс. *Класс* объединяет в себе функции и данные, образующие объектно-ориентированный "кирпичик" программы. Члены класса `Console` обрабатывают ввод/вывод, выполняемый через командную строку; примером метода этого класса является `WriteLine`. Класс `Test` имеет два метода, `Main` и `FeetToInches`. Класс аналогичен *типу*, в чем мы убедимся в разд. "Основные сведения о типах" далее в этой главе.

На самом внешнем уровне программы типы организованы в *пространства имен*. Директива `using` использована для того, чтобы пространство имен `System` стало доступным нашему приложению, и чтобы можно было работать с классом `Console`. Мы могли бы определить все свои классы в пространстве имен `TestPrograms` таким способом:

```
using System;
namespace TestPrograms
{
    class Test {...}
    class Test2 {...}
}
```

Платформа .NET Framework организована в виде вложенных пространств имен. Вот пример пространства имен, содержащего типы для обработки текста:

```
using System.Text;
```

Директива `using` поставлена здесь для удобства. Кроме того, вы можете ссылаться на тип по его полному имени, т. е. по имени с префиксом, обозначающим пространство имен, например, `System.Text.StringBuilder`.

Компиляция

Компилятор C# преобразует исходный код, представленный в виде набора файлов с расширением `cs`, в сборку. *Сборка* — это единица компоновки и развертывания в .NET. Сборка может представлять собой либо *приложение*, либо *библиотеку*. Обычное консольное приложение или Windows-приложение имеет метод `Main` и является EXE-файлом. Библиотека — это файл с расширением `dll`; она эквивалентна EXE-файлу без точки входа. Ее предназначение заключается в том, что она вызывается из приложения или других библиотек. (Иногда говорят, что приложение или другая библиотека ссылается на эту библиотеку.) Платформа .NET Framework является набором библиотек.

Компилятор C# представляет собой файл `csc.exe`. Вы можете воспользоваться интегрированной средой разработки, например Visual Studio .NET, для автоматического запуска компилятора или скомпилировать программу вручную из командной строки. Во втором случае вы должны вначале сохранить программу в файле, например, `MyFirstProgram.cs`, а затем вызвать `csc` (расположенный в `<windows>/Microsoft.NET/Framework`) из командной строки следующим образом:

```
csc MyFirstProgram.cs
```

В результате будет создано приложение `MyFirstProgram.exe`.

ПРИМЕЧАНИЕ

Сборки обсуждаются в *гл. 16*.

Синтаксис

Синтаксис C# основан на синтаксисе языков C и C++. В этом разделе мы обсудим элементы синтаксиса C# на примере следующей программы:

```
using System;
class Test
{
    static void Main( )
    {
        int x = 12 * 30;
        Console.WriteLine(x);
    }
}
```

Идентификаторы и ключевые слова

Идентификаторы — это имена, которыми программисты называют классы, методы, переменные и т. д. Перечислим идентификаторы в нашей программе в порядке их появления:

```
System Test Main x Console WriteLine
```

Идентификатор должен быть словом, состоящим из символов Unicode и начинающимся с буквы или символа подчеркивания. Идентификаторы C# чувствительны к регистру. В соответствии с соглашением аргументы, локальные переменные и закрытые поля именованы в стиле *camel* (например, `myVariable`), а все остальные идентификаторы — в стиле *Pascal* (например, `MyMethod`).

Ключевые слова — это имена, зарезервированные компилятором, которые нельзя использовать в качестве идентификаторов. В нашей программе присутствуют следующие ключевые слова:

```
using class static void int
```

Приведем полный список ключевых слов C#:

abstract	As	base	bool	break
byte	Case	catch	char	checked
class	Const	continue	decimal	default
delegate	Do	double	else	enum
event	Explicit	extern	false	finally
fixed	Float	for	foreach	goto
if	Implicit	in	int	interface
internal	Is	lock	long	namespace

new	Null	object	operator	out
override	Params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
while				

Как избежать конфликтов

Если вам действительно необходимо использовать идентификатор, совпадающий с ключевым словом, вы можете квалифицировать его с помощью префикса @. Например:

```
class class {...} // так нельзя
class @class {...} // так можно
```

Символ @ не считается частью идентификатора. Следовательно, @myVariable и myVariable — одно и то же.

Контекстные ключевые слова

В языке C# также присутствуют *контекстные ключевые слова*. Хотя они и распознаются компилятором, их можно использовать в качестве идентификаторов, не квалифицируя. Их список:

add	ascending	by	descending	equals
from	get	global	group	in
into	join	let	on	orderby
partial	remove	select	set	value
var	where	yield		

В контексте применения этих ключевых слов двусмысленность не возникает.

Литералы, пунктуаторы и знаки операций

Литералы — это простейшие элементы данных, статически встроенные в программу. В нашем примере были использованы следующие литералы:

```
12 30
```

Пунктуаторы помогают разметить структуру программы. В нашем примере встречаются такие пунктуаторы:

```
; { }
```

Точка с запятой служит для завершения оператора. Это означает, что сам оператор может занимать несколько строчек:


```
Console.WriteLine
    (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10);
```

Фигурные скобки объединяют несколько операторов в один операторный блок.

Знаки операций участвуют в выражениях. Большинство знаков операций в С# состоит из одного символа, например, для умножения используется символ *. Знаки операций подробно обсуждаются далее в этой главе. В нашей программе были использованы следующие знаки операций:

```
. ( ) * =
```

Точка означает, что конструктивный элемент является членом более общего элемента. Круглые скобки используются при объявлении или вызове метода. Если метод не принимает никаких аргументов, ставятся пустые скобки. Знак равенства служит для *присваивания* (двойной знак равенства, ==, используется при сравнении на равенство, о чем мы поговорим чуть позже).

Комментарии

В С# существуют два разных стиля документирования исходного кода: *однотрочные комментарии* и *многострочные комметарии*. Однотрочный комментарий начинается с двух косых (//) и продолжается до конца строки. Например:

```
int x = 3; // переменной x присваивается значение 3
```

Многострочный комментарий начинается с символов /* и заканчивается символами */. Например:

```
int x = 3; /* этот комментарий
            занимает две строки */
```

Комментарии могут содержать документирующие XML-теги (см. разд. "XML-документирование" гл. 4).

Основные сведения о типах

Тип определяет, каким должно быть значение. *Значение* — это место в памяти, отведенное под *переменную* или *константу*. Переменная представляет значение, которое может измениться, а константа — постоянную величину (о константах мы поговорим далее в этой главе). В нашей первой программе мы создали локальную переменную по имени x:

```
static void Main( )
{
    int x = 12 * 30;
    Console.WriteLine(x);
}
```

Все значения в С# являются *экземплярами* определенного типа. Характер и множество допустимых значений переменной зависят от ее типа. Переменная x имеет тип int.

Примеры стандартных типов

Стандартные типы — это типы, поддерживаемые компилятором. Тип `int` является базовым стандартным типом, представляющим целые числа, попадающие в диапазон от -2^{31} до $2^{31}-1$ и занимающие 32 бита в памяти. С экземплярами типа `int` мы можем выполнять, например, арифметические действия:

```
int x = 12 * 30;
```

Другим стандартным типом C# является тип `string`. Он представляет последовательность символов, например, ".NET" или "http://oreilly.com". Мы можем манипулировать строками, вызывая специальные функции:

```
string message = "Hello world";
string upperMessage = message.ToUpper( );
Console.WriteLine(upperMessage); // ВЫВОДИТСЯ: HELLO WORLD
int x = 2007;
message = message + x.ToString( );
Console.WriteLine(message); // ВЫВОДИТСЯ: Hello world2007
```

Базовый тип `bool` имеет только два значения, `true` и `false`. Этот тип обычно применяется для реализации ветвления в вычислениях с помощью условного оператора `if`. Например:

```
bool simpleVar = false;
if (simpleVar)
    Console.WriteLine("This will not print");
int x = 5000;
bool lessThanAMile = x < 5280;
if (lessThanAMile)
    Console.WriteLine ("This will print");
```

ПРИМЕЧАНИЕ

В языке C# стандартные типы (которые также называются встроенными) обозначаются при помощи ключевых слов. Пространство имен `System` в .NET Framework содержит много важных типов, не являющихся стандартными (например, `DateTime`).

Примеры пользовательских типов

Подобно тому, как мы строим сложные функции из простых, мы можем строить сложные типы из базовых. В следующем примере определяется пользовательский тип `UnitConverter`. Это класс, который используется в качестве основы для преобразования единиц измерения:

```
using System;

public class UnitConverter
{
    int ratio; // поле
    public UnitConverter(int unitRatio) {ratio = unitRatio; } // конструктор
    public int Convert(int unit) {return unit * ratio; } // метод
}
```