

Содержание

Об авторе	26
Предисловие	27
Предполагаемая читательская аудитория	27
Как организована эта книга	28
Что требуется для работы с этой книгой	28
Соглашения, используемые в этой книге	29
Использование примеров кода	30
Ждем ваших отзывов!	30
Благодарности	31
Глава 1. Введение в C# и .NET	33
Объектная ориентация	33
Безопасность в отношении типов	34
Управление памятью	35
Поддержка платформ	35
Общезыковые исполняющие среды, библиотеки базовых классов и исполняющие среды	36
Общезыковая исполняющая среда	36
Библиотека базовых классов	37
Исполняющие среды	37
Унаследованные и нишевые исполняющие среды	40
Краткая история языка C#	41
Нововведения версии C# 9.0	41
Нововведения версии C# 8.0	44
Нововведения версий C# 7.x	48
Нововведения версии C# 6.0	52
Нововведения версии C# 5.0	53
Нововведения версии C# 4.0	54
Нововведения версии C# 3.0	54
Нововведения версии C# 2.0	55
Глава 2. Основы языка C#	57
Первая программа на C#	57
Компиляция	59
Синтаксис	60
Идентификаторы и ключевые слова	60
Литералы, знаки пунктуации и операции	61
Комментарии	62
Основы типов	62
Примеры предопределенных типов	62
Специальные типы	63
Типы и преобразования	68
Типы значений и ссылочные типы	68
Классификация предопределенных типов	72

Числовые типы	72
Числовые литералы	74
Числовые преобразования	75
Арифметические операции	76
Операции инкремента и декремента	76
Специальные операции с целочисленными типами	76
8- и 16-битные целочисленные типы	78
Целочисленные типы с собственным размером (C# 9)	78
Специальные значения <code>float</code> и <code>double</code>	79
Выбор между <code>double</code> и <code>decimal</code>	81
Ошибки округления вещественных чисел	81
Булевский тип и операции	82
Булевские преобразования	82
Операции сравнения и проверки равенства	82
Условные операции	83
Строки и символы	83
Символьные преобразования	84
Строковый тип	84
Массивы	86
Стандартная инициализация элементов	87
Индексы и диапазоны	88
Многомерные массивы	89
Упрощенные выражения инициализации массивов	90
Проверка границ	91
Переменные и параметры	91
Стек и куча	91
Определенное присваивание	93
Стандартные значения	93
Параметры	94
Локальные ссылочные переменные	99
Возвращаемые ссылочные значения	100
Объявление неявно типизированных локальных переменных с помощью <code>var</code>	101
Выражения <code>new</code> целевого типа (C# 9)	101
Выражения и операции	102
Первичные выражения	102
Пустые выражения	102
Выражения присваивания	103
Приоритеты и ассоциативность операций	103
Операции для работы со значениями <code>null</code>	107
Операция объединения с <code>null</code>	107
Операция присваивания с объединением с <code>null</code>	107
<code>null</code> -условная операция	108
Операторы	109
Операторы объявления	109
Операторы выражений	110
Операторы выбора	110
Операторы итераций	115
Операторы перехода	117
Смешанные операторы	118

Пространства имен	119
Директива <code>using</code>	120
Директива <code>using static</code>	120
Правила внутри пространства имен	121
Назначение псевдонимов типам и пространствам имен	122
Дополнительные возможности пространств имен	123
Глава 3. Создание типов в языке C#	125
Классы	125
Поля	125
Константы	126
Методы	128
Конструкторы экземпляров	130
Деконструкторы	131
Инициализаторы объектов	133
Ссылка <code>this</code>	134
Свойства	135
Индексаторы	139
Статические конструкторы	141
Статические классы	142
Финализаторы	142
Частичные типы и методы	143
Операция <code>nameof</code>	144
Наследование	145
Полиморфизм	146
Приведение и ссылочные преобразования	146
Виртуальные функции-члены	149
Абстрактные классы и абстрактные члены	150
Соккрытие унаследованных членов	151
Запечатывание функций и классов	152
Ключевое слово <code>base</code>	152
Конструкторы и наследование	153
Перегрузка и распознавание	154
Тип <code>object</code>	155
Упаковка и распаковка	156
Статическая проверка типов и проверка типов во время выполнения	157
Метод <code>GetType</code> и операция <code>typeof</code>	157
Метод <code>ToString</code>	157
Список членов <code>object</code>	158
Структуры	158
Семантика конструирования структуры	159
Ссылочные структуры	160
Модификаторы доступа	161
Примеры	162
Дружественные сборки	162
Установление верхнего предела доступности	163
Ограничения, накладываемые на модификаторы доступа	163
Интерфейсы	163
Расширение интерфейса	165
Явная реализация членов интерфейса	165

Реализация виртуальных членов интерфейса	166
Повторная реализация члена интерфейса в подклассе	166
Интерфейсы и упаковка	168
Стандартные члены интерфейса	168
Перечисления	169
Преобразования перечислений	170
Перечисления флагов	171
Операции над перечислениями	172
Проблемы безопасности типов	172
Вложенные типы	173
Обобщения	174
Обобщенные типы	175
Для чего предназначены обобщения	176
Обобщенные методы	176
Объявление параметров типа	177
Операция <code>typeof</code> и несвязанные обобщенные типы	178
Стандартное значение для параметра обобщенного типа	178
Ограничения обобщений	179
Создание подклассов для обобщенных типов	180
Самоссылающиеся объявления обобщений	181
Статические данные	181
Параметры типа и преобразования	181
Ковариантность	182
Контравариантность	185
Сравнение обобщений C# и шаблонов C++	186
Глава 4. Дополнительные средства языка C#	187
Делегаты	187
Написание подключаемых методов с помощью делегатов	188
Целевые методы экземпляра и статические целевые методы	189
Групповые делегаты	189
Обобщенные типы делегатов	191
Делегаты <code>Func</code> и <code>Action</code>	191
Сравнение делегатов и интерфейсов	192
Совместимость делегатов	193
События	195
Стандартный шаблон событий	197
Средства доступа к событию	200
Модификаторы событий	201
Лямбда-выражения	202
Явное указание типов параметров лямбда-выражения	203
Захватывание внешних переменных	203
Сравнение лямбда-выражений и локальных методов	206
Анонимные методы	207
Операторы <code>try</code> и исключения	208
Конструкция <code>catch</code>	209
Блок <code>finally</code>	211
Генерация исключений	212
Основные свойства класса <code>System.Exception</code>	214

Общие типы исключений	214
Шаблон методов TryXXX	215
Альтернативы исключениям	216
Перечисление и итераторы	216
Перечисление	216
Инициализаторы коллекций	217
Итераторы	218
Семантика итератора	219
Компоновка последовательностей	221
Типы значений, допускающие null	222
Структура Nullable<T>	222
Подъем операций	223
Тип bool? и операции & и	224
Типы значений, допускающие null, и операции для работы с null	225
Сценарии использования типов значений, допускающих null	225
Альтернативы типам значений, допускающим null	226
Ссылочные типы, допускающие null	227
null-терпимая операция	228
Разъединение контекстов с заметками и с предупреждениями	229
Трактовка предупреждений о допустимости значения null как ошибок	229
Расширяющие методы	230
Цепочки расширяющих методов	230
Неоднозначность и распознавание	231
Анонимные типы	232
Кортежи	233
Именованное элементов кортежа	234
Метод ValueTuple.Create	236
Деконструирование кортежей	236
Сравнение эквивалентности	236
Классы System.Tuple	237
Записи (C# 9)	237
Подоплека	238
Определение записи	238
Неразрушающее изменение	241
Проверка достоверности свойств	242
Вычисляемые поля и ленивая оценка	243
Первичные конструкторы	245
Записи и сравнение эквивалентности	247
Шаблоны	248
Шаблон var	248
Шаблон константы	249
Реляционные шаблоны (C# 9)	249
Комбинаторы шаблонов (C# 9)	250
Шаблоны кортежей и позиционные шаблоны	250
Шаблоны свойств	251
Атрибуты	252
Классы атрибутов	253
Именованные и позиционные параметры атрибутов	253
Применение атрибутов к сборкам и поддерживаемым полям	254

Указание нескольких атрибутов	254
Атрибуты информации о вызывающем компоненте	254
Динамическое связывание	256
Сравнение статического и динамического связывания	256
Специальное связывание	257
Языковое связывание	258
Исключение <code>RuntimeBinderException</code>	259
Представление типа <code>dynamic</code> во время выполнения	259
Динамические преобразования	260
Сравнение <code>var</code> и <code>dynamic</code>	260
Динамические выражения	261
Динамические вызовы без динамических получателей	261
Статические типы в динамических выражениях	262
Невызываемые функции	263
Перегрузка операций	264
Функции операций	265
Перегрузка операций эквивалентности и сравнения	265
Специальные неявные и явные преобразования	266
Перегрузка операций <code>true</code> и <code>false</code>	267
Небезопасный код и указатели	268
Основы указателей	268
Небезопасный код	268
Оператор <code>fixed</code>	268
Операция указателя на член	269
Ключевое слово <code>stackalloc</code>	270
Буферы фиксированных размеров	270
<code>void*</code>	271
Указатели на функции (C# 9)	271
Атрибут <code>[SkipLocalsInit]</code> (C# 9)	272
Директивы препроцессора	273
Условные атрибуты	275
Директива <code>#pragma warning</code>	275
XML-документация	276
Стандартные XML-дескрипторы документации	276
Дескрипторы, определяемые пользователем	278
Перекрестные ссылки на типы или члены	278
Глава 5. Обзор .NET	281
.NET Standard	282
.NET Standard 2.0	283
.NET Standard 2.1	283
Более старые стандарты .NET Standard	284
Совместимость .NET Framework и .NET 5	284
Версии исполняющих сред и языка C#	285
Ссылочные сборки	285
Среда CLR и библиотека BCL	285
Системные типы	285
Обработка текста	286
Коллекции	286

Запросы	286
XML и JSON	287
Диагностика	287
Параллелизм и асинхронность	287
Потоки данных и ввод-вывод	287
Работа с сетями	288
Сборки, рефлексия и атрибуты	288
Динамическое программирование	288
Криптография	288
Расширенная многопоточность	289
Параллельное программирование	289
Span<T> и Memory<T>	289
Возможность взаимодействия с собственным кодом и COM	289
Регулярные выражения	289
Сериализация	289
Компилятор Roslyn	290
Прикладные слои	290
ASP.NET Core	290
Windows Desktop	291
Глава 6. Основы .NET	295
Обработка строк и текста	295
Тип char	295
Тип string	297
Сравнение строк	301
Класс StringBuilder	304
Кодировка текста и Unicode	305
Дата и время	308
Структура TimeSpan	308
Структуры DateTime и DateTimeOffset	310
Даты и часовые пояса	316
DateTime и часовые пояса	316
DateTimeOffset и часовые пояса	317
Класс TimeZoneInfo	317
Летнее время и DateTime	321
Форматирование и разбор	321
ToString и Parse	322
Поставщики форматов	323
Стандартные форматные строки и флаги разбора	327
Форматные строки для чисел	327
Перечисление NumberStyles	330
Форматные строки для даты/времени	331
Перечисление DateTimeStyles	334
Форматные строки для перечислений	334
Другие механизмы преобразования	334
Класс Convert	335
Класс XmlConvert	337
Преобразователи типов	337
Класс BitConverter	338

Глобализация	339
Контрольный перечень глобализации	339
Тестирование	340
Работа с числами	340
Преобразования	340
Класс Math	341
Структура BigInteger	341
Структура Half	342
Структура Complex	343
Класс Random	344
Перечисления	345
Преобразования для перечислений	345
Перечисление значений перечисления	347
Как работают перечисления	348
Структура Guid	348
Сравнение эквивалентности	349
Эквивалентность значений и ссылочная эквивалентность	349
Стандартные протоколы эквивалентности	351
Эквивалентность и специальные типы	355
Сравнение порядка	360
Интерфейсы IComparable	360
Операции < и >	361
Реализация интерфейсов IComparable	362
Служебные классы	363
Класс Console	363
Класс Environment	364
Класс Process	365
Класс ApplicationContext	367
Глава 7. Коллекции	369
Перечисление	369
IEnumerable и IEnumerator	370
IEnumerable<T> и IEnumerator<T>	371
Реализация интерфейсов перечисления	374
Интерфейсы ICollection и IList	377
ICollection<T> и ICollection	378
IList<T> и IList	379
IReadOnlyCollection<T> и IReadOnlyList<T>	380
Класс Array	381
Конструирование и индексация	383
Перечисление	385
Длина и ранг	385
Поиск	386
Сортировка	387
Обращение порядка следования элементов	388
Копирование	388
Преобразование и изменение размера	389

Списки, очереди, стеки и наборы	389
List<T> и ArrayList	389
LinkedList<T>	392
Queue<T> и Queue	394
Stack<T> и Stack	395
BitArray	396
HashSet<T> и SortedSet<T>	396
Словари	398
IDictionary<TKey, TValue>	398
IDictionary	400
Dictionary<TKey, TValue> и Hashtable	401
OrderedDictionary	402
ListDictionary и HybridDictionary	403
Отсортированные словари	403
Настраиваемые коллекции и прокси	405
Collection<T> и CollectionBase	405
KeyedCollection<TKey, TItem> и DictionaryBase	407
ReadOnlyCollection<T>	410
Неизменяемые коллекции	410
Создание неизменяемых коллекций	412
Манипулирование неизменяемыми коллекциями	412
Построители	412
Неизменяемые коллекции и производительность	413
Подключение протоколов эквивалентности и порядка	414
IEqualityComparer и EqualityComparer	415
IComparer и Comparer	417
StringComparer	419
IStructuralEquatable и IStructuralComparable	420
Глава 8. Запросы LINQ	421
Начало работы	421
Текущий синтаксис	423
Выстраивание в цепочки операций запросов	424
Составление лямбда-выражений	426
Естественный порядок	429
Другие операции	429
Выражения запросов	430
Переменные диапазона	432
Сравнение синтаксиса запросов и синтаксиса SQL	432
Сравнение синтаксиса запросов и текучего синтаксиса	433
Запросы со смешанным синтаксисом	433
Отложенное выполнение	434
Повторная оценка	435
Захваченные переменные	435
Как работает отложенное выполнение	437
Построение цепочки декораторов	438
Каким образом выполняются запросы	439
Подзапросы	440
Подзапросы и отложенное выполнение	443

Стратегии композиции	443
Постепенное построение запросов	444
Ключевое слово <code>into</code>	445
Упаковка запросов	446
Стратегии проекции	447
Инициализаторы объектов	447
Анонимные типы	448
Ключевое слово <code>let</code>	449
Интерпретируемые запросы	449
Каким образом работают интерпретируемые запросы	451
Комбинирование интерпретируемых и локальных запросов	454
<code>AsEnumerable</code>	455
Инфраструктура EF Core	456
Сущностные классы EF Core	456
Объект <code>DbContext</code>	456
Отслеживание объектов	461
Отслеживание изменений	463
Навигационные свойства	463
Ленивая загрузка	466
Отложенное выполнение	466
Построение выражений запросов	468
Сравнение делегатов и деревьев выражений	468
Деревья выражений	470
Глава 9. Операции LINQ	473
Обзор	474
Последовательность → последовательность	475
Последовательность → элемент или значение	476
Ничего → последовательность	477
Выполнение фильтрации	477
<code>Where</code>	478
<code>Take</code> и <code>Skip</code>	480
<code>TakeWhile</code> и <code>SkipWhile</code>	480
<code>Distinct</code>	481
Выполнение проекции	481
<code>Select</code>	481
<code>SelectMany</code>	486
Выполнение соединения	493
<code>Join</code> и <code>GroupJoin</code>	493
Операция <code>Zip</code>	501
Упорядочение	501
<code>OrderBy</code> , <code>OrderByDescending</code> , <code>ThenBy</code> и <code>ThenByDescending</code>	502
Группирование	504
<code>GroupBy</code>	504
Операции над множествами	508
<code>Concat</code> и <code>Union</code>	508
<code>Intersect</code> и <code>Except</code>	509

Методы преобразования	509
OfType и Cast	509
ToArray, ToList, ToDictionary, ToHashSet и ToLookup	511
AsEnumerable и AsQueryable	512
Операции над элементами	512
First, Last и Single	512
ElementAt	513
DefaultIfEmpty	514
Методы агрегирования	514
Count и LongCount	514
Min и Max	515
Sum и Average	515
Aggregate	516
Квантификаторы	518
Contains и Any	519
All и SequenceEqual	519
Методы генерации	520
Empty	520
Range и Repeat	520
Глава 10. LINQ to XML	521
Обзор архитектуры	521
Что собой представляет DOM-модель	521
DOM-модель LINQ to XML	522
Обзор модели X-DOM	522
Загрузка и разбор	524
Сохранение и сериализация	525
Создание экземпляра X-DOM	525
Функциональное построение	526
Указание содержимого	527
Автоматическое глубокое копирование	528
Навигация и запросы	528
Навигация по дочерним узлам	528
Навигация по родительским узлам	532
Навигация по равноправным узлам	532
Навигация по атрибутам	533
Обновление модели X-DOM	533
Обновление простых значений	533
Обновление дочерних узлов и атрибутов	534
Обновление через родительский элемент	534
Работа со значениями	536
Установка значений	536
Получение значений	537
Значения и узлы со смешанным содержимым	538
Автоматическая конкатенация XText	538
Документы и объявления	539
XDocument	539
Объявления XML	541

Имена и пространства имен	542
Пространства имен в XML	543
Указание пространств имен в X-DOM	545
Модель X-DOM и стандартные пространства имен	546
Префиксы	547
Аннотации	548
Проецирование в модель X-DOM	549
Устранение пустых элементов	551
Потоковая передача проекции	552
Глава 11. Другие технологии XML и JSON	553
XmlReader	553
Чтение узлов	554
Чтение элементов	556
Чтение атрибутов	559
Пространства имен и префиксы	560
XmlWriter	561
Запись атрибутов	562
Запись других типов узлов	562
Пространства имен и префиксы	563
Шаблоны для использования XmlReader/XmlWriter	563
Работа с иерархическими данными	563
Смешивание XmlReader/XmlWriter с моделью X-DOM	566
Работа с JSON	568
Utf8JsonReader	568
Utf8JsonWriter	570
JsonDocument	571
Глава 12. Освобождение и сборка мусора	575
IDisposable, Dispose и Close	575
Стандартная семантика освобождения	576
Когда выполнять освобождение	577
Очистка полей при освобождении	579
Анонимное освобождение	579
Автоматическая сборка мусора	581
Корневые объекты	582
Финализаторы	583
Вызов метода Dispose из финализатора	585
Восстановление	586
Как работает сборщик мусора	588
Приемы оптимизации	589
Принудительный запуск сборки мусора	593
Настройка сборки мусора во время выполнения	594
Нагрузка на память	594
Организация пула массивов	594
Утечки управляемой памяти	595
Таймеры	596
Диагностика утечек памяти	598

Слабые ссылки	598
Слабые ссылки и кеширование	599
Слабые ссылки и события	600
Глава 13. Диагностика	603
Условная компиляция	603
Сравнение условной компиляции и статических переменных-флагов	604
Атрибут Conditional	605
Классы Debug и Trace	607
Fail и Assert	607
TraceListener	608
Сброс и закрытие прослушивателей	609
Интеграция с отладчиком	610
Присоединение и останов	610
Атрибуты отладчика	611
Процессы и потоки процессов	611
Исследование выполняющихся процессов	611
Исследование потоков в процессе	612
StackTrace и StackFrame	612
Журналы событий Windows	614
Запись в журнал событий	615
Чтение журнала событий	616
Мониторинг журнала событий	616
Счетчики производительности	617
Перечисление доступных счетчиков производительности	617
Чтение данных счетчика производительности	619
Создание счетчиков и запись данных о производительности	620
Класс Stopwatch	621
Межплатформенные инструменты диагностики	622
dotnet-counters	622
dotnet-trace	624
dotnet-dump	625
Глава 14. Параллелизм и асинхронность	627
Введение	627
Многопоточная обработка	628
Создание потока	628
Join и Sleep	630
Блокирование	630
Локальное или разделяемое состояние	632
Блокировка и безопасность потоков	634
Передача данных потоку	635
Обработка исключений	637
Потоки переднего плана или фоновые потоки	638
Приоритет потока	639
Передача сигналов	639
Многопоточность в обогащенных клиентских приложениях	640
Контексты синхронизации	642
Пул потоков	643

Задачи	645
Запуск задачи	646
Возвращение значений	647
Исключения	648
Продолжение	649
TaskCompletionSource	651
Task.Delay	653
Принципы асинхронности	654
Сравнение синхронных и асинхронных операций	654
Что собой представляет асинхронное программирование	654
Асинхронное программирование и продолжение	656
Важность языковой поддержки	657
Асинхронные функции в C#	659
Ожидание	659
Написание асинхронных функций	666
Асинхронные лямбда-выражения	670
Асинхронные потоки	671
Асинхронные методы в WinRT	673
Асинхронность и контексты синхронизации	674
Оптимизация	675
Асинхронные шаблоны	679
Отмена	679
Сообщение о ходе работ	681
Асинхронный шаблон, основанный на задачах	683
Комбинаторы задач	684
Асинхронное блокирование	688
Устаревшие шаблоны	688
Модель асинхронного программирования	688
Асинхронный шаблон на основе событий	689
BackgroundWorker	690
Глава 15. Потоки данных и ввод-вывод	691
Потоковая архитектура	691
Использование потоков	693
Чтение и запись	695
Поиск	696
Закрытие и сбрасывание	696
Тайм-ауты	697
Безопасность в отношении потоков управления	697
Потоки с опорными хранилищами	697
FileStream	698
MemoryStream	701
PipeStream	702
BufferedStream	706
Адаптеры потоков	706
Текстовые адаптеры	707
Двоичные адаптеры	712
Закрытие и освобождение адаптеров потоков	713

Потоки со сжатием	715
Сжатие в памяти	716
Сжатие файлов с помощью <code>gzip</code> в Unix	717
Работа с ZIP-файлами	718
Операции с файлами и каталогами	719
Класс <code>File</code>	719
Класс <code>Directory</code>	723
<code>FileInfo</code> и <code>DirectoryInfo</code>	723
<code>Path</code>	724
Специальные папки	726
Запрашивание информации о томе	727
Перехват событий файловой системы	727
Файловый ввод-вывод в UWP	729
Работа с каталогами	729
Работа с файлами	730
Получение доступа к каталогам и файлам	731
Безопасность, обеспечиваемая операционной системой	734
Выполнение под учетной записью стандартного пользователя	735
Повышение административных полномочий и виртуализация	736
Размещенные в памяти файлы	736
Размещенные в памяти файлы и произвольный файловый ввод-вывод	737
Размещенные в памяти файлы и разделяемая память (Windows)	738
Межплатформенная разделяемая память между процессами	738
Работа с аксессуарами представлений	739
Глава 16. Взаимодействие с сетью	741
Сетевая архитектура	742
Адреса и порты	744
Идентификаторы URI	745
Классы клиентской стороны	747
<code>WebClient</code>	748
<code>WebRequest</code> и <code>WebResponse</code>	749
<code>HttpClient</code>	751
Прокси-серверы	756
Аутентификация	757
Обработка исключений	759
Работа с протоколом HTTP	761
Заголовки	761
Строки запросов	761
Выгрузка данных формы	762
Cookie-наборы	763
Реализация HTTP-сервера	765
Использование FTP	768
Использование DNS	770
Отправка сообщений электронной почты с помощью <code>SmtpClient</code>	770
Использование TCP	771
Параллелизм и TCP	774
Получение почты POP3 с помощью TCP	775
TCP в UWP	777

Глава 17. Сборки	779
Содержимое сборки	779
Манифест сборки	780
Манифест приложения (Windows)	781
Модули	782
Класс <code>Assembly</code>	782
Строгие имена и подписание сборок	784
Назначение сборке строгого имени	784
Имена сборок	785
Полностью заданные имена	785
Класс <code>AssemblyName</code>	786
Информационная и файловая версии сборки	787
Подпись <code>Authenticode</code>	787
Подписание с помощью системы <code>Authenticode</code>	788
Ресурсы и подчиненные сборки	790
Встраивание ресурсов напрямую	791
Файлы <code>.resources</code>	792
Файлы <code>.resx</code>	793
Подчиненные сборки	795
Культуры и подкультуры	797
Загрузка, распознавание и изолирование сборок	798
Контексты загрузки сборок	800
Стандартный контекст ALC	805
“Текущий” контекст ALC	806
Метод <code>Assembly.Load</code> и контекстные ALC	807
Загрузка и распознавание неуправляемых библиотек	810
Класс <code>AssemblyDependencyResolver</code>	811
Выгрузка контекстов ALC	812
Унаследованные методы загрузки	812
Реализация системы подключаемых модулей	814
Глава 18. Рефлексия и метаданные	821
Рефлексия и активизация типов	822
Получение экземпляра <code>Type</code>	822
Имена типов	824
Базовые типы и интерфейсы	825
Создание экземпляров типов	826
Обобщенные типы	827
Рефлексия и вызов членов	829
Типы членов	831
Сравнение членов C# и членов CLR	832
Члены обобщенных типов	834
Динамический вызов члена	835
Параметры методов	835
Использование делегатов для повышения производительности	837
Доступ к неоткрытым членам	838
Обобщенные методы	839
Анонимный вызов членов обобщенного интерфейса	839

Рефлексия сборок	842
Модули	842
Работа с атрибутами	842
Основы атрибутов	843
Атрибут AttributeUsage	844
Определение собственного атрибута	845
Извлечение атрибутов во время выполнения	846
Динамическая генерация кода	848
Генерация кода IL с помощью класса DynamicMethod	848
Стек оценки	850
Передача аргументов динамическому методу	850
Генерация локальных переменных	851
Ветвление	852
Создание объектов и вызов методов экземпляра	853
Обработка исключений	854
Выпуск сборок и типов	855
Объектная модель Reflection.Emit	856
Выпуск членов типа	858
Выпуск методов	858
Выпуск полей и свойств	860
Выпуск конструкторов	862
Присоединение атрибутов	863
Выпуск обобщенных методов и типов	863
Определение обобщенных методов	863
Определение обобщенных типов	865
Сложности, связанные с генерацией	865
Несозданные закрытые обобщения	865
Циклические зависимости	866
Синтаксический разбор IL	868
Написание дизассемблера	869
Глава 19. Динамическое программирование	875
Исполняющая среда динамического языка	875
Унификация числовых типов	876
Динамическое распознавание перегруженных членов	878
Упрощение паттерна “Посетитель”	878
Анонимный вызов членов обобщенного типа	882
Реализация динамических объектов	884
DynamicObject	885
ExpandableObject	887
Взаимодействие с динамическими языками	888
Передача состояния между C# и сценарием	889
Глава 20. Криптография	891
Обзор	891
Защита данных Windows	892
Хеширование	893
Алгоритмы хеширования в .NET	894
Хеширование паролей	895

Симметричное шифрование	895
Шифрование в памяти	897
Соединение в цепочку потоков шифрования	898
Освобождение объектов шифрования	899
Управление ключами	900
Шифрование с открытым ключом и подписание	900
Класс RSA	902
Цифровые подписи	903
Глава 21. Расширенная многопоточность	905
Обзор синхронизации	906
Монопольное блокирование	906
Оператор lock	907
Monitor.Enter и Monitor.Exit	908
Выбор объекта синхронизации	909
Когда нужна блокировка	909
Блокирование и атомарность	910
Вложенное блокирование	911
Взаимоблокировки	912
Производительность	913
Mutex	913
Блокирование и безопасность к потокам	914
Безопасность к потокам и типы .NET	916
Безопасность к потокам в серверах приложений	918
Неизменяемые объекты	920
Немонопольное блокирование	921
Семафор	921
Блокировки объектов чтения/записи	923
Сигнализирование с помощью дескрипторов ожидания событий	928
AutoResetEvent	928
ManualResetEvent	932
CountdownEvent	932
Создание межпроцессного объекта EventWaitHandle	933
Дескрипторы ожидания и продолжение	934
WaitAny, WaitAll и SignalAndWait	935
Класс Barrier	936
Ленивая инициализация	937
Lazy<T>	938
LazyInitializer	939
Локальное хранилище потока	940
[ThreadStatic]	940
ThreadLocal<T>	941
GetData и SetData	941
AsyncLocal<T>	942
Таймеры	943
Многопоточные таймеры	944
Однопоточные таймеры	946

Глава 22. Параллельное программирование	947
Для чего нужна инфраструктура PFX?	948
Концепции PFX	948
Компоненты PFX	949
Когда необходимо использовать инфраструктуру PFX?	950
PLINQ	951
Продвижение параллельного выполнения	953
PLINQ и упорядочивание	954
Ограничения PLINQ	955
Пример: параллельная программа проверки орфографии	955
Функциональная чистота	957
Установка степени параллелизма	958
Отмена	959
Оптимизация PLINQ	959
Класс <code>Parallel</code>	965
<code>Parallel.Invoke</code>	965
<code>Parallel.For</code> и <code>Parallel.ForEach</code>	966
Параллелизм задач	971
Создание и запуск задач	972
Ожидание на множестве задач	974
Отмена задач	974
Продолжение	975
Планировщики задач	980
<code>TaskFactory</code>	980
Работа с <code>AggregateException</code>	981
<code>Flatten</code> и <code>Handle</code>	982
Параллельные коллекции	983
<code>IProducerConsumerCollection<T></code>	984
<code>ConcurrentBag<T></code>	985
<code>BlockingCollection<T></code>	986
Реализация очереди производителей/потребителей	987
Глава 23. <code>Span<T></code> и <code>Memory<T></code>	991
Промежутки и нарезание	992
<code>CopyTo</code> и <code>TryCopyTo</code>	994
Работа с текстом	994
<code>Memory<T></code>	995
Однонаправленные перечислители	997
Работа с выделяемой в стеке и неуправляемой памятью	999
Глава 24. Способность к взаимодействию	1001
Обращение к низкоуровневым DLL-библиотекам	1001
Маршализация типов и параметров	1002
Маршализация общих типов	1002
Маршализация классов и структур	1004
Маршализация параметров <code>in</code> и <code>out</code>	1006
Соглашения о вызовах	1006

Обратные вызовы из неуправляемого кода	1007
Обратные вызовы с помощью указателей на функции (C# 9)	1007
Обратные вызовы с помощью делегатов	1009
Эмуляция объединения C	1010
Разделяемая память	1011
Отображение структуры на неуправляемую память	1013
fixed и fixed { . . . }	1016
Взаимодействие с COM	1018
Назначение COM	1018
Основы системы типов COM	1018
Обращение к компоненту COM из C#	1020
Необязательные параметры и именованные аргументы	1021
Неявные параметры ref	1021
Индексаторы	1022
Динамическое связывание	1022
Внедрение типов взаимодействия	1023
Эквивалентность типов	1023
Открытие объектов C# для COM	1024
Включение COM без регистрации	1025
Глава 25. Регулярные выражения	1027
Основы регулярных выражений	1027
Скомпилированные регулярные выражения	1029
RegexOptions	1029
Отмена символов	1030
Наборы символов	1031
Квантификаторы	1032
Жадные или ленивые квантификаторы	1033
Утверждения нулевой ширины	1033
Просмотр вперед и просмотр назад	1034
Привязки	1035
Границы слов	1036
Группы	1036
Именованные группы	1037
Замена и разделение текста	1038
Делегат MatchEvaluator	1039
Разделение текста	1039
Рецептурный справочник по регулярным выражениям	1039
Рецепты	1039
Справочник по языку регулярных выражений	1043
Предметный указатель	1047



Обзор .NET

Почти все возможности исполняющей среды .NET 5 доступны через обширное множество управляемых типов. Типы организованы в иерархические пространства имен и упакованы в набор сборок.

Некоторые типы .NET используются напрямую CLR и являются критически важными для среды управляемого размещения. Такие типы находятся в сборке по имени `System.Private.CoreLib.dll` и включают встроенные типы C#, а также базовые классы коллекций, типы для обработки потоков данных, сериализации, рефлексии, многопоточности и собственной возможности взаимодействия.



Сборка `System.Private.CoreLib.dll` заменяет сборку `microsoft.net.core.targets` из .NET Framework. Во многих местах официальной документации все еще можно встретить ссылки на `microsoft.net.core.targets`.

Уровнем выше находятся дополнительные типы, которые расширяют функциональность уровня CLR, предоставляя такие средства, как XML, JSON, взаимодействие с сетью и LINQ. Они образуют библиотеку базовых классов (Base Class Library — BCL). Выше находятся *прикладные слои*, которые предоставляют API-интерфейсы для разработки определенных видов приложений наподобие веб-приложения или обогащенного клиентского приложения.

Вот что предлагается в настоящей главе:

- краткий обзор библиотеки BCL (которая будет более подробно рассматриваться в оставшихся главах книги);
- высокоуровневый обзор прикладных слоев.

Нововведения версии .NET 5

По сравнению с .NET Core 3 версия .NET 5 предлагает многочисленные улучшения производительности, которые касаются сборщика мусора, компилятора JIT, процессора регулярных выражений, процессора HTTP, LINQ, коллекций и т.д.

.NET 5 также включает поддержку времени выполнения для новых возможностей языка C# 9.

Кроме того, несколько важных усовершенствований было внесено в сериализатор JSON, который раскрывается в дополнительных материалах, доступных для загрузки на веб-сайте издательства. В частности, теперь при сериализации и десериализации объектов он обеспечивает сохранность объектных ссылок.

В составе .NET 5 есть новый 16-битный тип с плавающей точкой по имени `Half` для взаимодействия с процессорами графических плат и новый интерфейс `IReadOnlySet<T>`, который позволяет унифицировать выполнение в отношении коллекций операций над множествами, основанных на чтении.

Взаимодействие с COM теперь поддерживает динамическое связывание (как было в .NET Framework).

API-интерфейс `Windows Forms` теперь содержит класс `TaskDialog`, предоставляющий доступ к версии метода `MessageBox.Show` с более широкими возможностями, которая реализована операционной системой.

Библиотека BCL в .NET 5 теперь на 80% аннотирована для типов, допускающих `null`.

Что касается развертывания, то .NET 5 включает вариант “подрезки приложений”: при создании автономных приложений теперь вы можете выбрать удаление неиспользуемых типов и членов из всех поставляемых сборок (в том числе из исполняющей среды .NET).

Прямая поддержка WinRT из .NET 5 была удалена. Для взаимодействия с API-интерфейсами WinRT вы должны теперь либо применять инструмент *CsWinRT*, который генерирует оболочку C# вокруг библиотеки WinRT, либо ссылаться на пакет NuGet, делающий то же самое. Логическое обоснование удаления поддержки WinRT заключается в том, что главный сценарий использования WinRT — платформа UWP — не запускается под управлением .NET 5, а преемник UWP — WinUI 3 — не будет нуждаться в прямой поддержке WinRT. Причина в том, что WinUI 3 будет поставляться в виде развертываемой исполняющей среды, а не быть частью операционной системы, и потому WinUI 3 откроет доступ к полностью управляемому API-интерфейсу.

.NET Standard

Изобилие публичных библиотек, которые доступны через NuGet, не было бы настолько ценным, если бы они поддерживали только .NET 5. При написании библиотеки вы часто хотите поддерживать различные платформы и версии исполняющей среды. Чтобы достичь такой цели, не создавая отдельные сборки для каждой исполняющей среды, вы должны ориентироваться на наименьший общий знаменатель. Это относительно легко при желании поддерживать только непосредственных предшественников .NET 5: скажем, если вы нацеливаетесь на .NET Core 3.0, то ваша библиотека будет работать под управлением .NET Core 3.0, .NET Core 3.1 и .NET 5+.

Ситуация становится более запутанной, когда вы хотите поддерживать также .NET Framework и Xamarin. Дело в том, что каждая такая исполняющая среда имеет CLR и BCL с перекрывающимися функциональными средствами — ни одна исполняющая среда не является чистым подмножеством остальных.

.NET Standard решает проблему путем определения искусственных подмножеств, которые обеспечивают работу под управлением целого диапазона унаследованных исполняющих сред. Ориентируясь на .NET Standard, вы можете легко писать библиотеки с широким охватом.



.NET Standard — не исполняющая среда, а просто спецификация, описывающая минимальный базовый уровень функциональности (типы и члены), который гарантирует совместимость с определенным набором исполняющих сред. Концепция похожа на интерфейсы C#: стандарт .NET Standard подобен интерфейсу, который конкретные типы (исполняющие среды) могут реализовывать.

.NET Standard 2.0

Наиболее полезной версией является *.NET Standard 2.0*. Библиотека, которая ориентирована на .NET Standard 2.0, а не на специфическую исполняющую среду, будет работать без каких-либо изменений под управлением большинства современных и унаследованных исполняющих сред, все еще используемых в настоящее время, в том числе:

- .NET Core 2.0+
- UWP 10.0.16299+
- Mono 5.4+ (CLR/BCL из более старых версий Xamarin)
- .NET Framework 4.6.1+

Для нацеливания на .NET Standard 2.0 добавьте в свой файл `.csproj` следующие строки:

```
<PropertyGroup>
  <TargetFramework>netstandard2.0</TargetFramework>
</PropertyGroup>
```

Большинство API-интерфейсов, описанных в книге, поддерживаются стандартом .NET Standard 2.0.

.NET Standard 2.1

.NET Standard 2.1 является надмножеством стандарта .NET Standard 2.0, которое поддерживает (только) следующие платформы:

- .NET Core 3+
- Mono 6.4+

.NET Standard 2.1 не поддерживается ни одной из версий .NET Framework (и на момент написания главы даже UWP), что делает его менее полезным, чем .NET Standard 2.0.

В частности, в .NET Standard 2.1 (но не в .NET Standard 2.0) доступны следующие API-интерфейсы:

- `Span<T>` (глава 23)
- `Reflection.Emit` (глава 18)
- `ValueTask<T>` (глава 14)

Более старые стандарты .NET Standard

Существуют также более старые стандарты .NET Standard, самыми заметными из которых считаются 1.1, 1.2, 1.3 и 1.6. Стандарт с более высоким номером всегда является строгим надмножеством стандарта с меньшим номером. Например, при написании библиотеки, которая нацелена на .NET Standard 1.6, вы будете поддерживать не только последние версии основных исполняющих сред, но также .NET Core 1.0. А если вы нацеливаете библиотеку на .NET Standard 1.3, то будет поддерживаться все, что уже упоминалось, плюс .NET Framework 4.6.0 (табл. 5.1).

Таблица 5.1. Старые стандарты .NET Standard

Если вы нацеливаете библиотеку на...	То будут также поддерживаться...
.NET Standard 1.6	.NET Core 1.0
.NET Standard 1.3	Указанное выше плюс .NET Framework 4.6.0
.NET Standard 1.2	Указанное выше плюс .NET Framework 4.5.1, Windows Phone 8.1, WinRT для Windows 8.1
.NET Standard 1.1	Указанное выше плюс .NET Framework 4.5.0, Windows Phone 8.0, WinRT для Windows 8.0



В стандартах 1.x отсутствуют тысячи API-интерфейсов, которые находятся в стандарте 2.0, в том числе большинство того, что мы описываем в этой книге. В результате нацеливание на какой-то стандарт 1.x становится гораздо более затруднительным, особенно в случае, когда нужна интеграция с существующим кодом или библиотеками.

Вы можете думать о стандарте .NET Standard как о наименьшем общем знаменателе. В случае стандарта .NET Standard 2.0 исполняющие среды, которые ему следуют, имеют похожую библиотеку BCL, поэтому наименьший общий знаменатель является крупным и полезным. Тем не менее, если вас также интересует совместимость с инфраструктурой .NET Core 1.0 (с ее значительно усеченной библиотекой BCL), тогда наименьший общий знаменатель — .NET Standard 1.x — значительно сужается и становится менее полезным.

Совместимость .NET Framework и .NET 5

Поскольку инфраструктура .NET Framework существует настолько долго, нередко можно встретить библиотеки, которые доступны *только* для .NET Framework (без каких-либо эквивалентов для .NET Standard, .NET Core или .NET 5).

Чтобы смягчить такую ситуацию, проектам .NET 5 и .NET Core разрешено ссылаться на сборки .NET Framework при соблюдении описанных ниже условий.

- Если сборка .NET Framework обратится к API-интерфейсу, не поддерживаемому в .NET Core, тогда сгенерируется исключение.
- Распознавание нетривиальных зависимостей может быть безуспешным (и часто таковым оказывается).

Скорее всего, на практике это будет работать в простых случаях, таких как сборка, которая представляет собой оболочку для неуправляемой DLL-библиотеки.

Версии исполняющих сред и языка C#

Компилятор C# автоматически выбирает версию языка в зависимости от исполняющей среды, на которую ориентируется проект:

- для .NET 5 он выбирает C# 9;
- для .NET Core 3.x, Xamarin и .NET Standard 2.1 он выбирает C# 8;
- для .NET Core 2.x, .NET Framework, .NET Standard 2.0 и предшествующих версий он выбирает C# 7.3.

Причина в том, что недавние версии языка C# полагаются на типы, которые доступны только в последних версиях исполняющих сред.

Ссылочные сборки

Когда ваш проект нацелен на .NET Standard, он неявно ссылается на сборку по имени `netstandard.dll`, которая содержит все разрешенные типы и члены для выбранной версии .NET Standard. Она называется *ссылочной сборкой*, потому что существует только в интересах компилятора и не содержит скомпилированный код. Во время выполнения “настоящие” сборки идентифицируются через атрибуты перенаправления сборок (выбор сборок зависит от того, под управлением какой исполняющей среды и платформы в итоге будет производиться запуск).

Интересно отметить, что похожие вещи происходят при нацеливании на .NET 5. Ваш проект неявно ссылается на набор ссылочных сборок, типы которых отражают содержимое сборок времени выполнения для выбранной версии .NET. Это помогает управлять версиями и межплатформенной совместимостью и также позволяет нацеливаться на версию .NET, которая отличается от версии, установленной на вашей машине.

Среда CLR и библиотека BCL

Системные типы

Наиболее фундаментальные типы находятся прямо в пространстве имен `System`. В их состав входят встроенные типы C#, базовый класс `Exception`, базовые классы `Enum`, `Array` и `Delegate`, а также типы `Nullable`, `Type`, `DateTime`, `TimeSpan` и `Guid`. Кроме того, пространство имен `System` вклю-

чает типы для выполнения математических функций (Math), генерации случайных чисел (Random) и преобразования между различными типами (Convert и BitConverter).

Фундаментальные типы описаны в главе 6 вместе с интерфейсами, которые определяют стандартные протоколы, используемые повсеместно в .NET для решения таких задач, как форматирование (IFormattable) и сравнение порядка (Comparable).

В пространстве имен System также определен интерфейс IDisposable и класс GC для взаимодействия со сборщиком мусора; мы рассмотрим их в главе 12.

Обработка текста

Пространство имен System.Text содержит класс StringBuilder (редактируемый или *изменяемый* родственник string) и типы для работы с кодировками текста, такими как UTF-8 (Encoding и его подтипы). Мы раскроем их в главе 6.

Пространство имен System.Text.RegularExpressions содержит типы, которые выполняют расширенные операции поиска и замены на основе образца; такие типы будут описаны в главе 25.

Коллекции

В .NET предлагаются разнообразные классы для управления коллекциями элементов. Они включают структуры, основанные на списках и словарях, и работают в сочетании с набором стандартных интерфейсов, которые унифицируют их общие характеристики. Все типы коллекций определены в следующих пространствах имен, описанных в главе 7:

```
System.Collections           // Необобщенные коллекции
System.Collections.Generic   // Обобщенные коллекции
System.Collections.Specialized // Строго типизированные коллекции
System.Collections.ObjectModel // Базовые типы для создания
                               // собственных коллекций
System.Collections.Concurrent // Коллекции, безопасные в
                               // отношении потоков (глава 22)
```

Запросы

Язык LINQ позволяет выполнять безопасные в отношении типов запросы к локальным и удаленным коллекциям (например, к таблицам SQL Server) и описан в главах 8–10. Крупное преимущество языка LINQ заключается в том, что он предоставляет согласованный API-интерфейс запросов для разнообразных предметных областей. Основные типы находятся в перечисленных ниже пространствах имен:

```
System.Linq                 // LINQ to Objects и PLINQ
System.Linq.Expressions     // Для ручного построения выражений
System.Xml.Linq             // LINQ to XML
```

XML и JSON

XML и JSON поддерживаются в .NET повсеместно. Внимание в главе 10 сосредоточено целиком на LINQ to XML — легковесной модели документных объектов (Document Object Model — DOM) для XML, которую можно конструировать и опрашивать с помощью LINQ. В главе 11 описаны высокопроизводительные низкоуровневые классы для чтения/записи XML, схем и таблиц стилей XML, а также типы для работы с JSON:

```
System.Xml           // XmlReader, XmlWriter и старая модель W3C DOM
System.Xml.Linq      // Объектная модель документа LINQ to XML
System.Xml.Schema    // Поддержка для XSD
System.Xml.Serialization // Декларативная сериализация XML для типов .NET
System.Xml.XPath     // Язык запросов XPath
System.Xml.Xsl       // Поддержка таблиц стилей
System.Text.Json     // Средство чтения/записи JSON и DOM
```

Сериализатор JSON раскрывается в дополнительных материалах, доступных для загрузки на веб-сайте издательства.

Диагностика

В главе 13 мы рассмотрим регистрацию в журнале и утверждения, а также покажем, как взаимодействовать с другими процессами, выполнять запись в журнал событий Windows и проводить мониторинг производительности. Соответствующие типы определены в пространстве имен System.Diagnostics и его подпространствах.

Параллелизм и асинхронность

Многим современным приложениям в каждый момент времени приходится иметь дело с несколькими действиями. Начиная с версии C# 5.0, решение стало проще за счет асинхронных функций и таких высокоуровневых конструкций, как задачи и комбинаторы задач. Все это подробно объясняется в главе 14, которая начинается с рассмотрения основ многопоточности. Типы для работы с потоками и асинхронными операциями находятся в пространствах имен System.Threading и System.Threading.Tasks.

Потоки данных и ввод-вывод

В .NET предоставляется потоковая модель для низкоуровневого ввода-вывода. Потоки данных обычно применяются для чтения и записи напрямую в файлы и сетевые подключения и могут соединяться в цепочки либо помещаться внутри декорированных потоков с целью добавления функциональности сжатия или шифрования. В главе 15 описана потоковая архитектура, а также специфическая поддержка для работы с файлами и каталогами, сжатием, изолированным хранилищем, каналами и файлами, отображенными в память. Тип Stream и типы ввода-вывода определены в пространстве имен System.IO и его подпространствах, а типы WinRT для файлового ввода-вывода — в пространстве имен Windows.Storage и его подпространствах.

Работа с сетями

С помощью типов из пространства имен `System.Net` можно напрямую работать со стандартными сетевыми протоколами, такими как HTTP, FTP, TCP/IP и SMTP. В главе 16 будет показано, как взаимодействовать с использованием каждого из упомянутых протоколов, начиная с простых задач вроде загрузки веб-страницы и заканчивая применением TCP/IP для извлечения сообщений электронной почты POP3. Ниже перечислены пространства имен, которые будут рассмотрены:

```
System.Net
System.Net.Http           // HttpClient
System.Net.Mail           // Для отправки электронной почты через SMTP
System.Net.Sockets        // TCP, UDP и IP
```

Сборки, рефлексия и атрибуты

Сборки, в которые компилируются программы на C#, состоят из исполняемых инструкций (представленных на языке IL) и метаданных, которые описывают типы, члены и атрибуты программы. С помощью рефлексии можно просматривать метаданные во время выполнения и предпринимать действия вроде динамического вызова методов. Посредством пространства имен `Reflection.Emit` можно конструировать новый код на лету.

В главе 17 мы опишем строение сборок и объясним, как их динамически загружать и изолировать. В главе 18 мы раскроем рефлексия и атрибуты — покажем, как инспектировать метаданные, динамически вызывать функции, записывать специальные атрибуты, выпускать новые типы и производить разбор низкоуровневого кода IL. Типы для применения рефлексии и работы со сборками находятся в следующих пространствах имен:

```
System
System.Reflection
System.Reflection.Emit
```

Динамическое программирование

В главе 19 мы рассмотрим несколько паттернов для динамического программирования и работы со средой DLR. Мы покажем, как реализовать паттерн “Посетитель” (Visitor), создавать специальные динамические объекты и взаимодействовать с IronPython. Типы, предназначенные для динамического программирования, находятся в пространстве имен `System.Dynamic`.

Криптография

.NET обеспечивает всестороннюю поддержку для популярных протоколов хеширования и шифрования. В главе 20 мы раскроем хеширование, симметричное шифрование и шифрование с открытым ключом, а также API-интерфейс Windows Data Protection. Типы для этого определены в следующих пространствах имен:

```
System.Security
System.Security.Cryptography
```

Расширенная многопоточность

Асинхронные функции в C# значительно облегчают параллельное программирование, поскольку снижают потребность во взаимодействии с низкоуровневыми технологиями. Тем не менее, все еще возникают ситуации, когда нужны сигнальные конструкции, локальное хранилище потока, блокировки чтения/записи и т.д. Данные вопросы подробно обсуждаются в главе 21. Типы, связанные с многопоточностью, находятся в пространстве имен `System.Threading`.

Параллельное программирование

В главе 22 мы рассмотрим библиотеки и типы для работы с многоядерными процессорами, включая API-интерфейсы для реализации параллелизма задач, императивного параллелизма данных и функционального параллелизма (PLINQ).

`Span<T>` и `Memory<T>`

Для содействия микро-оптимизации в “горячих” точках в плане производительности среда CLR предлагает несколько типов, которые помогают программировать так, чтобы снизить нагрузку на диспетчер памяти. Двумя ключевыми типами подобного рода являются `Span<T>` и `Memory<T>`, которые будут описаны в главе 23.

Возможность взаимодействия с собственным кодом и COM

Вы можете взаимодействовать с собственным кодом и с кодом COM. Возможность взаимодействия с собственным кодом позволяет вызывать функции из неуправляемых DLL-библиотек, регистрировать обратные вызовы, отображать структуры данных и работать с собственными типами данных. Возможность взаимодействия с COM позволяет обращаться к типам COM (на машинах Windows) и открывать для COM доступ к типам .NET. Типы, поддерживающие такую функциональность, определены в пространстве имен `System.Runtime.InteropServices` и рассматриваются в главе 24.

Регулярные выражения

В главе 25 мы покажем, как можно использовать регулярные выражения для сопоставления с символьными шаблонами в строках.

Сериализация

.NET предлагает несколько систем для сохранения и восстановления объектов в двоичном или текстовом представлении. Такие системы могут применяться для передачи данных, а также сохранения и восстановления объектов из файлов. В дополнительных материалах, доступных для загрузки на веб-сайте издательства, раскрываются все четыре механизма сериализации: двоичный сериализатор, (обновленный) сериализатор JSON, сериализатор XML и сериализатор на основе контрактов данных.

Компилятор Roslyn

Сам компилятор C# написан на языке C# — проект называется Roslyn, а библиотеки доступны в виде пакетов NuGet. С помощью этих библиотек вы можете эксплуатировать функциональность компилятора многими способами помимо компиляции исходного кода в сборку, скажем, писать инструменты для анализа и рефакторинга кода. Компилятор Roslyn рассматривается в дополнительных материалах, доступных для загрузки на веб-сайте издательства.

Прикладные слои

Приложения, основанные на пользовательском интерфейсе, можно разделить на две категории: *тонкий клиент*, равнозначный веб-сайту, и *обогащенный клиент*, представляющий собой программу, которую конечный пользователь должен загрузить и установить на компьютере или на мобильном устройстве.

Для разработки приложений тонких клиентов на языке C# предусмотрена инфраструктура ASP.NET Core, которая запускается в среде Windows, Linux и macOS. Кроме того, инфраструктура ASP.NET Core позволяет реализовывать API-интерфейсы для веб-сети.

Для разработки приложений обогащенных клиентов на выбор доступно несколько API-интерфейсов:

- слой Windows Desktop, который включает API-интерфейсы WPF и Windows Forms и функционирует на настольных компьютерах Windows 7/8/10;
- платформа UWP, запускаемая на настольных компьютерах и устройствах Windows 10;
- платформа Xamarin, функционирующая на мобильных устройствах iOS и Android (в следующем крупном выпуске запланирована поддержка настольных компьютеров macOS).

Вдобавок существуют сторонние библиотеки вроде Avalonia, которые предлагают межплатформенную поддержку пользовательских интерфейсов.

ASP.NET Core

ASP.NET Core — это легковесный модульный преемник ASP.NET, поддерживающий популярный паттерн MVC (Model-View-Controller — модель-представление-контроллер). ASP.NET Core подходит для создания веб-сайтов, API-интерфейсов на основе REST и микрослужб. Кроме того, ASP.NET Core может работать в сочетании с двумя популярными фреймворками для одностраничных приложений: React и Angular.

ASP.NET Core функционирует под управлением Windows, Linux и macOS и может самостоятельно размещаться в специальном процессе. В отличие от своего предшественника из .NET Framework (ASP.NET) архитектура ASP.NET Core не зависит от System.Web и от исторического багажа веб-форм.

Как и любая архитектура тонкого клиента, ASP.NET Core обладает следующими общими преимуществами по сравнению с обогащенными клиентами:

- отсутствует развертывание на клиентской стороне;
- клиент может запускаться на любой платформе, которая поддерживает веб-браузер;
- легко развертывать обновления.

Windows Desktop

Прикладной слой Windows Desktop предлагает на выбор два API-интерфейса для реализации пользовательских интерфейсов в приложениях обогащенных клиентов: WPF и Windows Forms. Оба API-интерфейса работают в среде Windows Desktop/Server 7–10.

WPF

Инфраструктура WPF появилась в 2006 году и с тех пор неоднократно расширялась. В отличие от своей предшественницы, Windows Forms, она явно визуализирует элементы управления с использованием DirectX, обеспечивая перечисленные ниже преимущества.

- WPF поддерживает развитую графику, включая произвольные трансформации, трехмерную визуализацию, мультимедиа-возможности и подлинную прозрачность. Оформление поддерживается через стили и шаблоны.
- Основная единица измерения не базируется на пикселях, поэтому приложения корректно отображаются при любой настройке DPI (dots per inch — точек на дюйм).
- Она располагает обширной и гибкой поддержкой динамической компоновки, означающей возможность локализации приложения без опасности того, что элементы будут перекрывать друг друга.
- Применение DirectX делает визуализацию быстрой и способной извлекать преимущества от аппаратного ускорения графики.
- Она предлагает надежную привязку к данным.
- Пользовательские интерфейсы могут быть описаны декларативно в XAML-файлах, которые можно сопровождать независимо от файлов отделенного кода, что помогает отделить внешний вид от функциональности.

Инфраструктура WPF требует некоторого времени на изучение из-за своего размера и сложности. Типы, предназначенные для написания WPF-приложений, находятся в пространстве имен `System.Windows` и во всех его подпространствах за исключением `System.Windows.Forms`.

Windows Forms

Windows Forms — это API-интерфейс обогащенного клиента, который поставлялся с первой версией .NET Framework в 2000 году. По сравнению с WPF она является относительно простой технологией, которая предлагает большинство возможностей, необходимых во время разработки типового Windows-приложения. Она также играла важную роль в сопровождении унаследованных приложений.

Тем не менее, в сравнении с WPF инфраструктура Windows Forms обладает рядом недостатков, большинство из которых объясняется тем, что она является оболочкой для GDI+ и библиотеки элементов управления Win32.

- Несмотря на предоставление механизмов осведомленности о настройках DPI, все же слишком легко писать приложения, которые некорректно отображаются на клиентах с настройками DPI, отличающимися от таких настроек у разработчика.
- Для рисования нестандартных элементов управления используется API-интерфейс GDI+, который вопреки достаточно высокой гибкости медленно визуализирует крупные области (и без двойной буферизации может вызывать мерцание).
- Элементы управления лишены подлинной прозрачности.
- Большинство элементов управления не поддерживают компоновку. Например, поместить элемент управления изображением внутрь заголовка элемента управления вкладкой не удастся. Настройка списковых представлений, полей с раскрывающимися списками и элементов управления с вкладками, которая была бы тривиальной в WPF, требует много времени и сил.
- Трудно корректно и надежно реализовать динамическую компоновку.

Последний пункт является веской причиной отдавать предпочтение WPF перед Windows Forms, даже если разрабатывается бизнес-приложение, которому необходим только пользовательский интерфейс, а не учет “поведенческих особенностей пользователей”. Элементы компоновки в WPF, подобные Grid, упрощают организацию меток и текстовых полей таким образом, что они будут всегда выровненными — даже при смене языка локализации — без запутанной логики и какого-либо мерцания. Кроме того, не придется приводить все к наименьшему общему знаменателю в смысле экранного разрешения — элементы компоновки WPF изначально проектировались с поддержкой изменения размеров.

В качестве положительного момента следует отметить, что инфраструктура Windows Forms относительно проста в изучении и все еще поддерживается в небольшом количестве сторонних элементов управления.

Типы Windows Forms находятся в пространствах имен `System.Windows.Forms` (сборка `System.Windows.Forms.dll`) и `System.Drawing` (сборка `System.Drawing.dll`). Последнее пространство имен также содержит типы GDI+ для рисования специальных элементов управления.

UWP и WinUI 3

Универсальная платформа Windows (Universal Windows Platform — UWP) представляет собой API-интерфейс обогащенного клиента, который предназначен для разработки сенсорных приложений, ориентированных на настольные компьютеры и устройства Windows 10. Слово “универсальная” относится к ее способности функционировать на различных устройствах Windows 10, в том числе Xbox, Surface Hub и HoloLens. Однако платформа UWP не совместима с ранними версиями Windows, включая Windows 7 и Windows 8/8.1.

API-интерфейс UWP использует XAML и кое в чем похож на WPF. Ниже перечислены его ключевые отличия.

- Приложения UWP поставляются главным образом через Магазин Microsoft.
- Приложения UWP работают в песочнице, чтобы уменьшить угрозу со стороны вредоносного программного обеспечения, а это означает, что они не могут выполнять такие задачи, как чтение или запись произвольных файлов, и их нельзя запускать с повышенными административными правами.
- Платформа UWP опирается на типы WinRT, которые являются частью операционной системы (Windows), а не управляемой исполняющей среды. В результате при написании приложений вы обязаны объявлять диапазон версий Windows 10 (скажем, от Windows 10 сборки 17763 до Windows 10 сборки 18362). Таким образом, вам необходимо либо ориентировать приложения на старый API-интерфейс, либо требовать от пользователей установки самого последнего обновления Windows.

Для решения последнего вопроса в Microsoft предлагают библиотеку WinUI 3, которая переносит API-интерфейсы WinRT из операционной системы в исполняющую среду (обеспечивая доступ к полностью управляемому интерфейсу). Библиотека WinUI 3 также поможет преодолеть разрыв между инфраструктурами Windows Desktop и UWP: вместо выбора одной из них появляется возможность смешивания и подгонки компонентов из обеих инфраструктур.

К API-интерфейсу UWP относятся пространства имен `Windows.UI` и `Windows.UI.Xaml`.

Хamarin и Xamarin Forms

Xamarin позволяет писать мобильные приложения на языке C#, которые ориентированы на операционную систему iOS или Android. Xamarin.iOS и Xamarin.Android функционируют под управлением .NET CLR/BCL на базе Mono (производный продукт исполняющей среды с открытым кодом Mono).

В дополнение к API-интерфейсам, специфичным для iOS и Android, имеется продукт под названием *Xamarin Forms*, который действует как верхний слой, позволяя одному проекту пользовательского интерфейса генерировать приложения и для iOS, и для Android. В настоящее время компания Microsoft работает над новым развитием Xamarin Forms, называемым MAUI (Multi-platform App UI — пользовательский интерфейс для многоплатформенных приложений), который даст возможность создавать также межплатформенные настольные приложения (для Windows и macOS). Новый продукт будет поставляться в версии Mono, согласованной с выходом .NET 6.

За дополнительной информацией обращайтесь на веб-сайты <https://www.xamarin.com> и <https://github.com/dotnet/maui>.