



ОГЛАВЛЕНИЕ

Предисловие	13
Об авторах	16
О научных редакторах	18
Благодарности	19
Введение	20
ГЛАВА 1.	
Характеристики производительности	23
Требования к производительности	24
Характеристики производительности	28
В заключение	31
ГЛАВА 2.	
Измерение производительности	32
Подходы к измерению производительности	32
Встроенные инструменты Windows	33
Счетчики производительности	34
Механизм трассировки событий для Windows	42
Профилировщики времени	58
Дискретный профилировщик Visual Studio	59
Инструментированный профилировщик Visual Studio	64
Дополнительные приемы использования профилировщиков времени	67
Профилировщики выделения памяти	71
Профилировщик выделения памяти Visual Studio	72
CLR Profiler	75
Профилировщики памяти	81
Другие профилировщики	86
Профилировщики доступа к данным и базам данных	87
Профилировщики конкуренции	88

Профилировщики ввода/вывода	91
Микрохронометраж	92
Пример неправильного микрохронометража	92
Рекомендации по проведению хронометража	96
В заключение	99

ГЛАВА 3.

Внутреннее устройство типов 102

Пример.....	102
Семантические отличия между ссылочными типами и типами значений.....	104
Хранение, размещение и удаление	105
Внутреннее устройство ссылочных типов	108
Таблица методов.....	109
Вызов методов экземпляров ссылочных типов.....	114
Блоки синхронизации и ключевое слово lock.....	122
Внутреннее устройство типов значений.....	128
Ограничения типов значений	130
Виртуальные методы типов значений.....	132
Упаковка	133
Предотвращение упаковки типов значений с помощью метода Equals	136
Метод GetHashCode	140
Эффективные приемы использования типов значений	144
В заключение	144

ГЛАВА 4.

Сборка мусора 145

Назначение сборщика мусора	146
Управление свободным списком	146
Сборка мусора на основе подсчета ссылок	148
Сборка мусора на основе трассировки	150
Фаза маркировки	151
Фазы чистки и сжатия	158
Закрепление	161
Разновидности сборщиков мусора	163
Приостановка потоков для сборки мусора.....	163
Сборщик мусора для сервера	170
Выбор разновидности сборщика мусора.....	172
Поколения	175
Предположения в основе модели поколений.....	176
Реализация поколений в .NET	177

Куча больших объектов	183
Ссылки между поколениями	185
Фоновый сборщик мусора	188
Сегменты сборщика мусора и виртуальная память	189
Финализация	194
Детерминированная финализация вручную	194
Автоматическая недетерминированная финализация	195
Ловушки недетерминированной финализации	198
Шаблон реализации метода Dispose	202
Слабые ссылки	205
Взаимодействие со сборщиком мусора	208
Класс System.GC	209
Взаимодействие с применением интерфейсов размещения CLR	213
Триггеры сборщика мусора	215
Эффективные приемы повышения производительности сборки мусора	216
Модель поколений	216
Закрепление	218
Финализация	219
Разные советы и рекомендации	220
В заключение	226

ГЛАВА 5.

Коллекции и обобщенные типы 230

Обобщенные типы	230
Обобщенные типы в .NET	234
Ограничения обобщенных типов	236
Реализация обобщенных типов в CLR	239
Коллекции	249
Параллельные коллекции	252
Проблемы, связанные с кешем	254
Собственные коллекции	261
Система непересекающихся множеств	261
Список с пропусками	263
Одноразовые коллекции	265
В заключение	269

ГЛАВА 6.

Конкуренция и параллелизм 270

Перспективы и преимущества	270
----------------------------------	-----

Зачем использовать приемы параллельного программирования?	272
От потоков к пулам потоков и задачам	273
Параллелизм задач	281
Параллелизм данных	290
Асинхронные методы в C# 5	295
Дополнительные шаблоны в TPL	300
Синхронизация	302
Код без блокировок	304
Механизмы синхронизации Windows	311
Вопросы оптимального использования кеша	314
Использование GPU для вычислений	318
Введение в C++ AMP	318
Умножение матриц	322
Моделирование движения частиц	323
Мозаики и разделяемая память	325
В заключение	331

ГЛАВА 7.

Сети, ввод/вывод и сериализация 332

Общие понятия	333
Синхронный и асинхронный ввод/вывод	333
Порты завершения ввода/вывода	335
Пул потоков в .NET	340
Копирование памяти	341
Чтение вразброс и запись со слиянием	342
Файловый ввод/вывод	343
Управление кешированием	343
Небуферизованный ввод/вывод	344
Сети	345
Сетевые протоколы	346
Сетевые сокеты	348
Сериализация и десериализация данных	351
Тестирование производительности средств сериализации	352
Сериализация объектов DataSet	354
Windows Communication Foundation	356
Пороговые значения	356
Модель обработки	357
Кеширование	359
Асинхронные клиенты и серверы WCF	359
Привязки	361
В заключение	362

ГЛАВА 8.**Небезопасный код и взаимодействие с ним ... 364**

Небезопасный код.....	365
Закрепление объектов в памяти и дескрипторы сборщика мусора	366
Управление жизненным циклом	368
Выделение неуправляемой памяти	368
Использование пулов памяти	368
P/Invoke.....	370
PInvoke.net и P/Invoke Interop Assistant	372
Привязка.....	374
Заглушки маршалера	375
Двоично совместимые типы	380
Направление маршалинга, ссылочные типы и типы значений	382
Code Access Security	383
Взаимодействие с COM-объектами.....	384
Управление жизненным циклом	386
Маршалинг через границы подразделений	386
Импортирование библиотек типов и Code Access Security.....	389
NoPIA	390
Исключения	391
Расширения языка C++/CLI	392
Вспомогательная библиотека marshal_as	395
Код на языке IL и неуправляемый код	397
Взаимодействие со средой выполнения WinRT в Windows 8... ..	397
Эффективные приемы взаимодействий	398
В заключение	399

ГЛАВА 9.**Оптимизация алгоритмов 400**

Систематизация сложности.....	401
Большое O	401
Машины Тьюринга и классы сложности.....	403
Мемоизация и динамическое программирование	409
Расстояние Левенштейна.....	411
Кратчайший путь между всеми парами вершин	413
Аппроксимация	416
Задача коммивояжера	417
Задача о максимальном разрезе.....	418
Вероятностные алгоритмы	419
Вероятностное решение задачи о максимальном разрезе	419

Тест простоты Ферма	420
Индексирование и сжатие	421
Кодировка переменной длины	421
Сжатие индексов	423
В заключение	425

ГЛАВА 10.

Шаблоны оптимизации производительности ... 426

Оптимизации JIT-компилятора	426
Стандартные оптимизации	427
Встраивание методов	428
Отключение проверки границ	430
Хвостовые вызовы	432
Производительность на этапе запуска	436
Предварительная JIT-компиляция с помощью NGen (Native Image Generator)	438
Фоновая JIT-компиляция в многопроцессорных системах	441
Упаковщики образов	442
Управляемая оптимизация на основе профилирования	443
Различные советы по оптимизации времени запуска	445
Аппаратно-зависимые оптимизации	447
Единственный поток команд и множество потоков данных	448
Распараллеливание инструкций	452
Исключения	457
Механизм рефлексии	458
Генерация кода	459
Генерация из исходного кода	460
Генерация кода с использованием легковесного генератора кода	462
В заключение	467

ГЛАВА 11.

Производительность веб-приложений 468

Измерение производительности веб-приложений	469
Тестирование производительности и нагрузочное тестирование веб-приложений в среде Visual Studio	469
Инструменты мониторинга HTTP	471
Инструменты анализа веб-взаимодействий	473
Увеличение производительности веб-сервера	473
Кеширование часто используемых объектов	474
Использование асинхронных страниц, модулей и контроллеров	476

Настройка окружения ASP.NET	481
Отключение механизмов трассировки и отладки в ASP.NET	481
Отключение механизма ViewState	483
Кеш вывода на стороне сервера	485
Предварительная компиляция приложений ASP.NET	488
Тонкая настройка модели процесса в ASP.NET	488
Настройка IIS	491
Кеширование вывода	491
Настройка пула приложения.....	493
Оптимизация сети	496
Включение HTTP-заголовков кеширования	496
Включение сжатия в IIS	501
Минификация и объединение	504
Использование сетей доставки содержимого (CDN)	507
Масштабирование приложений ASP.NET	509
Горизонтальное масштабирование	510
Механизмы масштабирования в ASP.NET	511
Ловушки горизонтального масштабирования.....	512
В заключение	513
Предметный указатель	514



ПРЕДИСЛОВИЕ

Платформа Desktop .NET Framework недавно (в феврале 2012) отметила свое 10-летие. Я входил в состав команды с самого ее образования и больше половины этого времени занимался проблемами оптимизации платформы, поэтому 10-й день рождения навел меня на мысли о том, где была платформа .NET, куда она пришла, и какие направления в улучшении производительности можно считать «правильными». Возможность написать предисловие к книге, посвященной производительности приложений на платформе .NET, дала мне шанс изложить свои мысли.

Продуктивность программиста всегда была и будет главной ценностью платформы .NET. Механизм сборки мусора является самой важной особенностью, обеспечивающей продуктивность. Но не только потому, что он устраняет целый класс ошибок (при работе с памятью), но и потому, что позволяет писать библиотеки классов, не загромождая их различными соглашениями о выделении ресурсов (требуемыми передавать временные буферы или устанавливающими правила о том, кто должен освобождать память). Еще одной важной особенностью является строгий контроль типов (которому теперь подчинены и обобщенные типы (Generics)), позволяющий выявлять намерения программиста и находить многие распространенные ошибки еще до запуска программы. Он также обеспечивает строгое соблюдение контрактов между программными компонентами, что очень важно для библиотек классов и больших проектов. Отсутствие контроля типов в таких языках как JavaScript всегда будет расцениваться как недостаток. Поверх этих двух особенностей мы создали библиотеку классов (весьма однородную, с простыми интерфейсами, непротиворечивыми соглашениями об именовании и так далее), простота использования которой была не последней целью. Я очень горжусь полученным результатом – мы создали систему, дающую возможность разрабатывать программный код с высочайшей продуктивностью.

Однако, одной продуктивности программиста недостаточно, особенно для такой зрелой платформы, как среда выполнения .NET. Нам

также необходима высочайшая производительность приложений, о чем и рассказывает данная книга.

Как нельзя ожидать, что программа будет работать без ошибок с самого первого запуска, так же нельзя ожидать, что программа «просто так» начнет показывать высокую производительность. Существуют не только инструменты (сборщик мусора и контроль типов) и приемы (проверки, интерфейсы), снижающие вероятность появления программных ошибок, но и инструменты (различные профилировщики) и приемы (планирование, создание прототипов, отложенная инициализация), снижающие вероятность появления проблем с производительностью.

Но все не так плохо – производительность приложений подчиняется правилу 90%/10%. Обычно более 90% программного кода не оказывает существенного влияния на производительность приложения в целом и может создаваться программистом с максимальной продуктивностью (в виде коротких строк простейшего кода). Однако другие 10% требуют пристального внимания. Для этого нужен план, и этот план должен быть составлен еще до того как код будет написан. Именно об этом рассказывается в главе 1. Для подобного планирования нужны исходные данные (сведения о скорости выполнения различных операций и библиотечных вызовов), а для этого необходимы инструменты измерения (профилировщики), о которых рассказывается в главе 2. И то, и другое является основой любого проекта высокопроизводительного программного обеспечения. Уделите этим главам особое внимание. Если вы примете их близко к сердцу, это поможет вам писать весьма эффективные программы.

Остальная часть книги посвящена деталям, знание которых требуется при составлении плана (главы 3, 4 и 5). Но они не заменят базовых знаний характеристик производительности используемой платформы. Если в ходе работы над прототипом выяснится, что простая и прямолинейная реализация не дает желаемой производительности, следует заняться усовершенствованием алгоритма (глава 9) или подумать о возможности включения параллельной обработки (глава 6), потому что они позволяют получить самый большой эффект. Если и после этого производительность оказывается неудовлетворительной, можно прибегнуть к некоторым хитростям .NET (глава 10). Если все вышеперечисленное не дало желаемого эффекта, остается только пожертвовать малой толикой продуктивности программиста и переписать наиболее критичные фрагменты приложения в виде небезопасного и неуправляемого кода (глава 8).

Ключевым моментом, который я хотел бы подчеркнуть, является план (глава 1), потому что именно с него все начинается. В этом плане вы определите наиболее критичные участки программы и выделите дополнительное время на тщательное исследование прототипов этих участков. Затем, вооруженные результатами исследований прототипов и информацией из этой книги, вы без особого труда сможете добиться желаемой производительности. Иногда для этого достаточно будет просто избежать распространенных ловушек, а иногда – применив простейшие оптимизации. Реже вам может понадобиться организовать параллельную обработку данных или написать небезопасный, неуправляемый код. Как бы то ни было, в результате вы сможете поднять производительность 10% кода (потратив дополнительные усилия) и добиться высокой продуктивности, создавая остальные 90%. Это и есть главная цель платформы .NET: высокая продуктивность программиста и высокая производительность кода.

Итак, помните, что высокая производительность не дается бесплатно – для ее достижения требуется составить план, но это не так сложно, и взяв в руки данную книгу вы уже сделали первый и самый важный шаг на пути к созданию *высокопроизводительных приложений* для .NET.

Удачи, и получайте удовольствие от книги. Саша, Дима и Идо постарались для этого.

*Вэнс Моррисон (Vance Morrison)
Архитектор производительности, .NET Runtime*

ОБ АВТОРАХ



Саша Голдштейн (Sasha Goldshtein) – наиболее ценный специалист (MVP) в Microsoft Visual C# и директор SELA Group. Руководит направлением производительности и отладки в SELA Technology Center, и консультирует по различным темам, включая подготовку приложений к промышленной эксплуатации, решение проблем производительности и создание распределенных архитектур. Наибольший опыт имеет в области разработки приложений на C# и C++, и создания архитектур масштабируемых и высокопроизводительных систем. Часто выступает на конференциях Microsoft и является автором множества курсов, таких как: «Производительность .NET», «Отладка приложений для .NET», «Внутреннее устройство Windows» и многих других.

Блог: <http://blog.sashag.net>

Twitter: @goldshn



Дима Зурбалеv (Dima Zurbalev) – старший консультант в группе экстренной помощи по вопросам производительности и отладки в SELA Group. Владеет навыками оптимизации производительности и отладки, позволяющими ему решать почти неразрешимые проблемы своих клиентов. Обладает глубокими знаниями внутреннего устройства механизмов CLR и Windows. Большая часть его опыта лежит в плоскости разработки приложений для .NET и инфраструктуры проектов на C++. Является участником ряда открытых проектов на CodePlex.

Блог: <http://blogs.microsoft.co.il/blogs/dimaz>



Идо Флатов (Ido Flatow) – наиболее ценный специалист (MVP) в Microsoft Connected Systems и старший архитектор в SELA Group. Обладает более чем 15-летним опытом и является одним из экспертов по Windows Azure и веб-технологиям в SELA. Специализируется на таких технологиях, как WCF, ASP.NET, Silverlight и IIS. Является сертифицированным преподавателем

Microsoft, соавтор официального курса обучения Microsoft WCF 4.0 (10263A). Часто выступает на конференциях по всему миру.

Блог: <http://blogs.microsoft.co.il/blogs/idof>

Twitter: @IdoFlatow

О НАУЧНЫХ РЕДАКТОРАХ



Тодд Мейстер (Todd Meister) работает в индустрии информационных технологий уже более 15 лет. Исполнял обязанности научного редактора при создании более 75 изданий, посвященных самым разным технологиям, от SQL Server до .NET Framework. Занимает пост старшего архитектора в области информационных технологий в университете Ball State University, в городе Манси, штат Индиана,

США. Живет в центре Индианы со своей женой Кимберли (Kimberly) и пятью детьми.

Фабио Клаудио Ферраччати (Fabio Claudio Ferracchiati) – писатель и научный редактор книг, посвященных самым современным технологиям. Участвовал в создании множества книг на такие темы, как .NET, C#, Visual Basic, SQL Server, Silverlight и ASP.NET. Является сертифицированным разработчиком решений на основе продуктов Microsoft для .NET. Живет в Риме, Италия. Работает в компании Brain Force.



БЛАГОДАРНОСТИ

Создание такой книги, как эта – тяжелый труд, и нам потребовался бы еще как минимум год, если бы не поддержка наших родных и близких, друзей и коллег.

Наш руководитель в SELA Group, Дэвид Басса (David Bassa), оказал нам всю возможную помощь, чтобы мы могли закончить эту книгу, и отнесся с большим пониманием, когда работа над другими проектами затормозилась на заключительных этапах работы над книгой.

Коллектив издательства Apress старался максимально облегчить нам труд и терпел наш скверный английский, не родной для нас. Гвенн (Gwenan), Кевин (Kevin) и Корбин (Corbin): спасибо вам за ваш профессионализм и долготерпение.

И напоследок, но не в последнюю очередь, мы хотим сказать огромное спасибо нашим родным, положившим бесчисленные часы на алтарь этой книги. Без вашей поддержки эта книга никогда не увидела бы свет.



ВВЕДЕНИЕ

Эта книга появилась на свет, потому что на наш взгляд отсутствовало достаточно авторитетное издание, охватывающее все три области, имеющие отношение к производительности приложений на платформе .NET:

- определение показателей производительности и способы их измерения, чтобы можно было проверить, насколько приложение соответствует им или превосходит их;
- приемы улучшения производительности приложений в терминах оптимизации управления памятью, операций ввода/вывода, многопоточного выполнения и так далее;
- полное представление о внутреннем устройстве CLR и .NET для эффективного проектирования высокопроизводительных приложений и исправления проблем с производительностью по мере их появления.

Мы полагаем, что разработчики приложений на платформе .NET не могут создавать высокопроизводительные программные решения, не имея полного понимания во всех трех областях. Например, управление памятью в .NET – чрезвычайно сложная тема (хотя и облегчаемая механизмом автоматической сборки мусора в CLR) и является важнейшей причиной появления проблем с производительностью, таких как утечки памяти и длинные паузы, вызванные работой механизма сборки мусора. Без понимания особенностей действия этого механизма высокопроизводительное управление памятью в .NET просто невозможно. Точно так же, чтобы осознанно выбрать наиболее подходящий класс коллекций из множества, что может предложить платформа .NET Framework, или принять решение о создании собственного класса, требуется полное понимание механики работы кешей центрального процессора, среды выполнения и средств синхронизации.

Главы в этой книге упорядочены так, чтобы читать их последовательно, но вы можете свободно перемещаться между ними взад и впе-

ред, восполняя пробелы, когда это необходимо. Главы организованы в следующие логические части.

- Главы 1 и 2 описывают показатели производительности и способы их измерения. В них будут представлены инструменты оценки производительности приложения.
- Главы 3 и 4 подробно описывают внутренние особенности общезыковой среды выполнения (Common Language Runtime, CLR). Основное внимание в них уделяется внутренней организации типов и реализации механизма сборки мусора – двум важнейшим темам, знание которых поможет улучшить производительность приложений, где управление памятью имеет особое значение.
- В главах 5, 6, 7, 8 и 11 обсуждаются конкретные темы, касающиеся платформы .NET Framework и CLR, знание которых дает дополнительные возможности оптимизации – правильное использование коллекций, организация параллельного выполнения кода, оптимизация операций ввода/вывода, эффективное применение механизмов взаимодействий и увеличение производительности веб-приложений.
- Глава 9 является кратким введением в теорию сложности и алгоритмы. Ее цель – дать представление об особенностях оптимизации алгоритмов.
- Глава 10 рассматривает самые разные темы, не укладываемые в другие главы, включая приемы оптимизации времени запуска приложения, применение исключений и механизма рефлексии .NET Reflection.

Для лучшего понимания некоторых из этих тем необходимо обладать определенными знаниями. В этой книге мы будем полагать, что читатель обладает существенным опытом разработки приложений на языке C# для платформы .NET Framework, а также знаком с основными понятиями, включая:

- Windows: потоки выполнения, механизмы синхронизации и виртуальная память;
- общезыковая среда выполнения (Common Language Runtime, CLR): динамический компилятор (Just-In-Time, JIT), промежуточный язык Microsoft (Microsoft Intermediate Language, MSIL), сборка мусора;
- устройство компьютера: основная память, кеш, диск, графическая карта, сетевой интерфейс.

В этой книге приводится достаточно много примеров программ, фрагментов кода и эталонных тестов. В интересах экономии места мы часто будем включать лишь часть кода, однако на веб-сайте книги вы можете получить полные исходные тексты примеров.

В некоторых главах мы используем фрагменты на языке ассемблера для микропроцессоров серии x86, чтобы проиллюстрировать некоторые особенности работы CLR или более полно объяснить те или иные приемы оптимизации. Хотя эти пояснения не являются критически важными для понимания обсуждаемых тем, тем не менее, мы рекомендуем читателям найти время и познакомиться с основами языка ассемблера x86. Отличным ресурсом в этом отношении может служить книга Рэндалла Хайда (Randall Hyde) «The Art of Assembly Language Programming», имеющаяся в свободном доступе (<http://www.artofasm.com/Windows/index.html>).

Наконец, в этой книге описывается множество инструментов измерения производительности, дается масса советов и рекомендаций по улучшению потребительских качеств и скорости выполнения приложений, теоретических обоснований, лежащих в основе механизмов CLR, практических примеров кода и случаев из практики авторов. В течение почти десяти лет мы занимались оптимизацией приложений для наших клиентов и проектированием высокопроизводительных систем, что называется «с нуля». За эти годы мы научили сотни разработчиков думать о производительности на каждом этапе разработки программ и активно искать возможности повышения их потребительских качеств. Прочитав эту книгу, вы присоединитесь к разработчикам высокопроизводительных программ для платформы .NET и исследователей в области оптимизации существующих приложений.

Саша Голдштейн (Sasha Goldshtein)

Дима Зурбалеv (Dima Zurbalev)

Идо Флатов (Ido Flatow)