# ОГЛАВЛЕНИЕ

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



## ГЛАВА 9.
## Оптимизация алгоритмов ... 400

# 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 (требующими 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 потребуется 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: @goldshtn

**Дима Зурбалев (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 Феррачиати (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 общеязыковой 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 укладывающиеся 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;
- общеязыковая 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 Зурбалев (Dima Zurbalev)*
*Идо Фламов (Ido Flatow)*

# ГЛАВА 1. Характеристики производительности

Прежде чем приступать к исследованию проблем производительности в мире .NET, необходимо понять, какие характеристики производительности существуют и в чем заключаются требования к производительности. В главе 2 мы исследуем более десятка профилировщиков и инструментов мониторинга, однако, чтобы эффективно использовать их, необходимо знать, какие показатели представляют интерес.

К разным приложениям предъявляются разные требования в смысле производительности, которые определяются различными потребностями. В одних случаях характеристики производительности явно определяются архитектурой приложения: например, чтобы веб-сервер мог обслуживать миллионы пользователей, необходимо создать распределенную систему из нескольких серверов, обеспечивающую поддержку кеширования и распределения нагрузки. В других, результаты тестирования производительности могут потребовать изменения самой архитектуры приложения: нам не раз приходилось видеть системы, перепроектировавшиеся до самого основания после проведения нагрузочных испытаний, как и системы, не выдерживавшие эксплуатационной нагрузки.

По своему опыту можем сказать, что выяснение требуемых параметров производительности и ограничений среды выполнения часто составляют половину успеха. Ниже приводится несколько примеров проблем, которые мы смогли выявить и исправить за последние несколько лет.

- Исследуя серьезные проблемы производительности веб-сервера в одном из центров обработки данных, мы обнаружили, что они были вызваны использованием для тестирования 4 Мбитного канала с низкой задержкой. Не понимая, какой из показателей производительности играет наиболее важную

роль, инженеры потратили несколько десятков дней на оптимизацию самого веб-сервера, который и без того показывал отличную производительность.

- Мы сумели улучшить скорость прокрутки в приложении с графическим интерфейсом за счет тонкой настройки механизма сборки мусора – программного компонента, не имеющего очевидной связи с проблемой. Точный хронометраж операций распределения памяти и настройка поведения сборщика мусора помогли установить и устранить причины задержки в работе графического интерфейса, так раздражавшие пользователей.
- Нам удалось увеличить скорость компиляции в десять раз, подключив жесткий диск к порту SATA и тем самым устранив влияние ошибки в драйвере SCSI-дисков от Microsoft.
- Мы смогли уменьшить размеры сообщений для обмена данными со службой WCF на 90%, заметно улучшив ее масштабируемость и использование CPU за счет настройки механизма сериализации WCF.
- Мы сократили время запуска крупного приложения, насчитывающего 300 сборок, с 35 до 12 секунд на устаревшем оборудовании, за счет сжатия кода приложения и тщательного исследования его зависимостей, с целью выявления тех, что не требуются немедленно на этапе запуска.

Эти примеры наглядно иллюстрируют, что любые системы, от мобильных устройств с низким энергопотреблением до высокопроизводительных рабочих станций с мощными графическими возможностями и распределенных систем, обладают уникальными характеристиками производительности, на которые влияет бесчисленное множество разнообразных факторов. В этой главе мы коротко исследуем разные характеристики производительности и требования к ней в типичном современном программном обеспечении. В следующей главе мы расскажем, как точно измерять эти характеристики, а в остальной части книги – как повышать их.

# Требования к производительности

Требования к производительности во многом зависят от назначения приложения и его архитектуры. Выяснив круг общих требований к

приложению, необходимо определить требования к производительности. В зависимости от процесса разработки программного обеспечения, эти требования возможно придется скорректировать с учетом изменения прежних и появления новых потребностей. Мы приведем несколько примеров и рекомендаций по определению требований к производительности, но вы должны понимать, что эти рекомендации, как и все, что касается производительности, необходимо адаптировать к конкретным условиям, с учетом назначения приложения.

Для начала перечислим несколько примеров *неправильно* сформулированных требований:

- приложение должно сохранять высокую отзывчивость при одновременном доступе нескольких пользователей;
- приложение не должно использовать большой объем памяти при небольшом количестве посетителей;
- единственный сервер базы данных должен быстро обслуживать запросы, даже при наличии нескольких серверов приложений, действующих под высокой нагрузкой.

Основная проблема этих требований в том, что они являются слишком общими и субъективными. Если попытаться вникнуть в их суть, можно обнаружить, что в разных системах отсчета они могут интерпретироваться по-разному. Бизнес-аналитик может считать, что 100 000 одновременно работающих пользователей – это «немного», а технический специалист может с уверенностью сказать, что такое количество пользователей невозможно обслужить, имея единственную машину. Аналогично, разработчик может считать пользовательский интерфейс с задержками 500 мсек вполне «отзывчивым», а эксперт в пользовательских интерфейсах оценивать это как серьезный недостаток.

Требования к производительности должны выражаться *в показателях, которые можно измерить* некоторым способом. Кроме того, требования к производительности должны содержать некоторую информацию об *окружении* – общую или конкретную для этой цели. Ниже перечислены некоторые примеры правильно сформулированных требований:

- приложение должно обслуживать каждую страницу в категории «Важные» не дольше 300 мсек (не включая задержки в сети), при условии одновременного обслуживания не более 5000 пользователей;
- приложение должно потреблять не более 4 Кбайт памяти на каждый неактивный сеанс с пользователем;