УДК 519.6; 004.02 DOI: 10.26102/2310-6018/2020.30.3.013

Исследование алгоритма Dispose-паттерна при принятии решений управления памятью в клиент-компонентной модели .NET

Е.В. Попова
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина) СПбГЭТУ «ЛЭТИ», Санкт-Петербург, Российская Федерация

Резюме: Представлено исследование алгоритма Dispose-паттерна, который используется в теории принятия решений в задачах управления памятью приложений в .NET. В долго работающих приложениях, с избыточным использованием памяти появляются предпосылки для частого запуска сборщика мусора, который работает с управляемыми ресурсами. Неуправляемые ресурсы без явной очистки могут инициировать проблемы памяти при ликвидации связанных с ними управляемых ресурсов. Составлена блок-схема метода очистки, получен алгоритм использования Dispose-паттерна в клиент-компонентной модели. Получены количественные значения различных характеристик работы сборщика мусора таких, как объем физической памяти, величина процессорного времени, максимальная пауза работы процесса и др. Данные собраны на основе приложения, состоящего из компонента и клиента, написанных на языке C#, запущенных в среде Visual Studio. Рассматриваемый компонент совместим с .NET Framework и не является компонентом Component Object Model. Сравниваются результаты работы приложения с запуском финализатора и без него. Полученные результаты помогут лицу, принимающему решение в выборе критерия оценки различных методов управления памятью, формировании однокритериальной или многокритериальной оптимизационной модели при принятии решения.

Ключевые слова: алгоритм метода управления памятью, принятие решений, Dispose-паттерн, финализатор, клиент-компонентная модель.

Для цитирования: Попова Е.В. Исследование алгоритма Dispose-паттерна при принятии решений управления памятью в клиент-компонентной модели .NET. *Моделирование, оптимизация и информационные технологии.* 2020;8(3). Доступно по: https://moit.vivt.ru/wp-content/uploads/2020/08/Popova_3_20_1.pdf DOI: 10.26102/2310-6018/2020.30.3.013

Investigation of the Dispose-Pattern Algorithm in Making Memory Management Decisions in the .NET Client-Component Model

E.V. Popova
Saint Petersburg Electrotechnical University "LETI", Saint Petersburg, Russian Federation

Abstract: The paper presents a study of the Dispose-pattern algorithm, which is used in decision theory in problems of memory management of applications in .NET. In long running applications with excessive memory usage, there are prerequisites for the frequent launch of the garbage collector, which works with managed resources. Unmanaged resources without explicit cleanup can cause memory problems when the associated managed resources are disposed of. A block diagram of the cleaning method was compiled, an algorithm for using the Dispose-pattern in the client-component model was

obtained. Quantitative values of various characteristics of the garbage collector are obtained, such as the amount of physical memory, the amount of processor time, the maximum pause of the process, etc. The data is collected on the basis of an application consisting of a component and a client written in C #, running in the Visual Studio environment. The component in question is compatible with the .NET Framework and is not a Component Object Model. The results of the application are compared with and without running the finalizer. The results obtained will help the decision-maker in choosing a criterion for assessing various methods of memory management, in the formation of a single-criterion or multi-criteria optimization model when making a decision.

Keywords: memory management method algorithm, decision making, Dispose-pattern, finalizer, client-component model.

For citation: Popova E.V. Investigation of the Dispose-Pattern Algorithm in Making Memory Management Decisions in the .NET Client-Component Model. *Modeling, Optimization and Information Technology*. 2020;8(3). Available from: https://moit.vivt.ru/wp-content/uploads/2020/08/Popova_3_20_1.pdf DOI: 10.26102/2310-6018/2020.30.3.013 (In Russ).

Введение

Управление памятью в .NET – это современная, актуальная проблема. Net Framework – программная платформа, выпущенная в 2002 году с общеязыковой средой исполнения Common Language Runtime (CLR) [1], контролирующая управляемый код. Накопление объектов в памяти может снижать производительность приложений или устройств. Существует ряд методов, управляющих памятью в .NET, и лицо, принимающее решение (ЛПР), опираясь на результаты исследования методов, получает возможность выбрать оптимальный вариант.

Развитие теории принятия решений (ТПР) получило новый толчок с использованием информационных технологий [2]. Решение многих задач упростилось с применением языков программирования, но и методы программирования, при сравнении из конечного числа допустимых, могут выбираться на основе наработок ТПР для определенных задач. Например, принятие решений оптимального выбора методов среди набора вариантов при онлайн деятельности, работе веб-сервисов, дистанционной работе. Все эти задачи затрагивают проблемы управления памятью нагруженных приложений в .NET.

Цель данной работы - исследовать алгоритм использования Dispose-паттерна в клиент-компонентной модели, верифицировать теоретические наработки метода очистки. Полученные результаты должны помочь ЛПР в выборе конкретного метода в задачах управления памятью в .NET.

CLR – это общеязыковая среда, которая автоматически управляет памятью посредством garbage collector (GC) [3]. Но GC не выделяет и не освобождает неуправляемую память. В нагруженных приложениях, устройствах с ограниченной памятью накопление объектов в памяти может привести к дефициту ресурсов. Неуправляемые ресурсы, связанные с управляемыми могут зависать после обработки GC управляемых ресурсов, что приводит к утечке памяти с выданными исключениями OutOfMemoryException, замедлению отклика.

Сборщик мусора освобождает память, когда отмирают все действующие ссылки на управляемый объект. Выделение памяти производится в управляемых кучах [4]. Чтобы отделить долгоживущие объекты от вновь образующихся и быстро уничтожающихся, сборка мусора основана на поколениях (gen0, gen1, gen2) [5]. При каждой сборке GC продвигает живые объекты в следующее поколение. Поколения соответствуют логическому представлению кучи GC, и чем больше номер поколения,

тем сборка мусора происходит реже, инициирует сборку в предшествующих поколениях и является самой затратной.

Материалы и методы

Существуют несколько методов управления памятью приложений в .NET, такие как объединение больших объектов в пулы, принудительная сборка мусора с использованием GC.Collect. Рассматриваемый в работе метод очистки – метод, реализующий Dispose-паттерн. Действие Dispose-паттерна направлено на уменьшение обращения к финализатору. Финализатор подразумевает спорадическое освобождение ресурсов. Он вызывается перед тем, как происходит уничтожение объекта GC. Внутри финализатора можно задать действия, которые будут выполняться перед уничтожением объекта [6]. Если объект с указателем на финализацию переживает сборку мусора, то попадает в следующее поколение и тянет за собой граф связанных с ним объектов. Так как финализаторы выполняются в отдельном потоке параллельно с другими потоками приложения, возникновение новых объектов, требующих финализации, будет конкурировать со старыми объектами в очереди, и это может привести к проблемам с памятью.

Метод очистки реализуется в клиент-компонентной модели. Компонентная модель в .NET – это набор правил, прописывающих форму и поведение компонента. Клиентом может выступать любой класс, поддерживающий этот набор правил, реализующийся через интерфейс System.ComponentModel.IComponent. Интерфейс IComponent является производным от System.IDisposable, а класс, содержащий управляемые и неуправляемые ресурсы реализует интерфейс IDisposable [7]. Функциональность компонента обеспечивается через открытые члены [8].

Открытый метод Dispose компонента блокирует вызов метода Finalize с помощью GC.SuppressFinalize [9], передает закрытому методу параметр со значением true, и задействует определенную ветку логического метода. Метод protected Dispose(Boolean) выполняет основную работу по освобождению ресурсов и вызывает метод Dispose(Boolean) базового класса. На Рисунке 1 приведена блок-схема [10] Dispose-паттерна клиент-компонентной модели.

Алгоритм использования Dispose-паттерна представляет следующие шаги:

Шаг 1. В клиенте циклически создаются объекты класса компонента.

Шаг 2. С помощью этих объектов происходит обращение к открытой функции Fn класса компонента с циклически изменяющимися параметрами.

Шаг 3. С помощью рекурсивной функции Fn компонента создается большое количество объектов.

Шаг 4. Если в клиенте происходит обращение к открытому методу Dispose компонента, то в закрытый логический метод Dispose передается параметр со значением true, и освобождаются управляемые ресурсы.

Иначе в компоненте срабатывает финализатор, который передает в закрытый логический метод Dispose параметр со значением false, и освобождаются неуправляемые ресурсы.

Возможны два варианта реализации Dispose-паттерна – с финализатором и подавлением его.

Рисунок 1 - Блок-схема Dispose-паттерна клиент-компонентной модели
Figure 1 - Block diagram of the Dispose-pattern of the client-component model

Результаты

Приложение было запущено с обращением к открытому методу Dispose через ссылочный объект [11] и с закомментированием этой строчки кода. Количественные характеристики работы GC получены с помощью утилиты PerfView. Утилита представляет собой open source инструмент от Microsoft, позволяющий создавать файлы журналов трассировки событий. Компонент и клиент были запущены в среде Visual Studio. Алгоритм начинает работу в клиенте, который связан с компонентом правилами и условиями компонентной модели. В компоненте подключена директива using System.ComponentModel, прописан Dispose-паттерн, методы, к которым обращается клиент. Количественные характеристики работы приложения представлены на Рисунках 2 и 3.

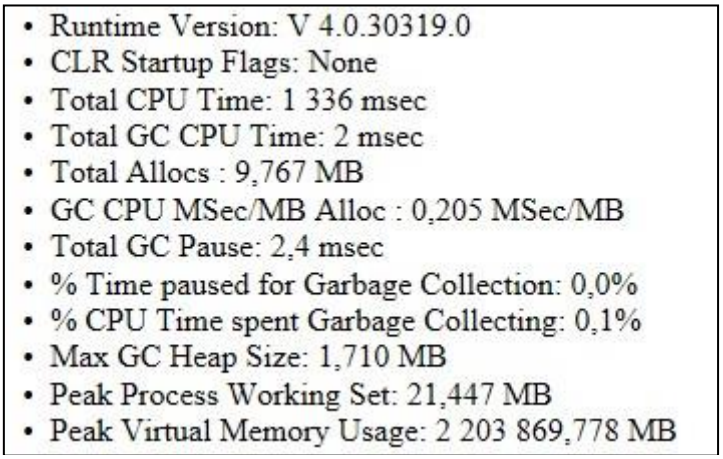

Рисунок 2 - Запуск клиента с методом Dispose
Figure 2 - Launching the client with the Dispose method

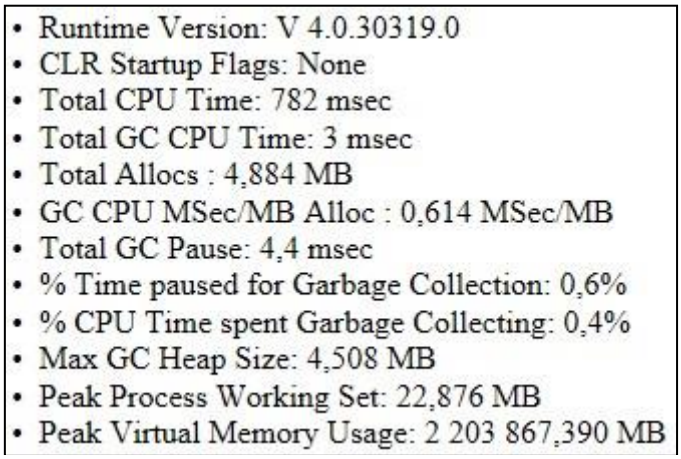

Рисунок 3 - Запуск клиента со срабатыванием финализатора
Figure 3 - Launching the client with the finalizer triggering

В Таблицах 1 и 2 показано сколько раз происходило освобождение памяти внутри каждого поколения во время проведения замеров.

Таблица 1 - GC Rollup By Generation с методом Dispose
Table 1 - GC Rollup By Generation with Dispose method

GC Rollup By Generation										
All times are in msec.										
Gen	**Count**	**Max Pause**	**Max Peak MB**	**Max Alloc MB/sec**	**Total Pause**	**Total Alloc MB**	**Alloc MB/ MSec GC**	**Survived MB/ MSec GC**	**Mean Pause**	**Induced**
ALL	6	0,5	1,7	1,975	2,4	9,8	4	0,03	0,4	0
0	6	0,5	1,7	1,975	2,4	9,8	0,2	0,03	0,4	0
1	0	0	0	0	0	0	0	не число	не число	0
2	0	0	0	0	0	0	0	не число	не число	0

Таблица 2 - GC Rollup By Generation со срабатыванием финализатора
Table 2 - GC Rollup By Generation with finalizer firing

GC Rollup By Generation										
All times are in msec.										
Gen	***Count***	**Max Pause**	**Max Peak MB**	**Max Alloc MB/sec**	**Total Pause**	**Total Alloc MB**	**Alloc MB/MSec GC**	**Survived MB/MSec GC**	**Mean Pause**	**Induced**
ALL	*3*	1,7	4,5	10,632	4,4	4,9	1,1	1,304	1,5	0
0	*1*	1,1	2,9	10,632	1,1	1,6	0,2	1,279	1,1	0
1	*2*	1,7	4,5	9,761	3,3	3,3	0,7	1,316	1,7	0
2	*0*	0	0	0	0	0	0	не число	не число	0

Обсуждение

При сопоставлении количественных характеристик, приведенных на Рисунках 2 и 3, можно сделать выводы, что при запуске клиента с методом Dispose время работы GC составило 2 мс, а без метода - 3 мс, что больше на 1 мс; максимальный размер кучи GC с методом Dispose составил 1710 MB, а без метода - 4508 MB, что больше на 2798 MB; использование физической памяти с методом Dispose зафиксировалось на значении 21447 MB, а без метода – 22876 MB, что больше на 1429 MB. Перечисленные показатели имеют более привлекательные значения при вызове метода Dispose (Рисунок 4). Процессорное время, затраченное на сборку мусора составляет в процентах 0,1 и 0,4, что характеризует работу GC в процессах без очистки второго поколения и без утечек памяти.

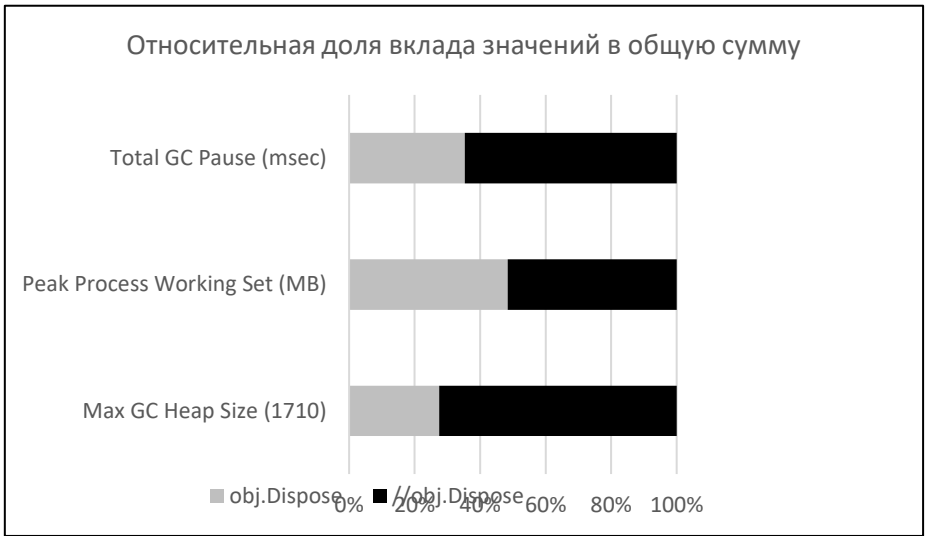

Рисунок 4 - Относительная доля вклада значений, которые имеют улучшенные показатели при вызове метода Dispose

Figure 4 - The relative share of the contribution of values that have improved indicators when calling the Dispose method

В то же время запуск клиента с методом Dispose заняло процессорного времени 1336 мс, а без метода 782 мс, что на 554 мс меньше. Пиковое использование виртуальной памяти с задействованием финализатора меньше на 2388 MB, а Total Allocs меньше на 4883 MB. Перечисленные показатели имеют более привлекательные значения при срабатывании финализатора (Рисунок 5).

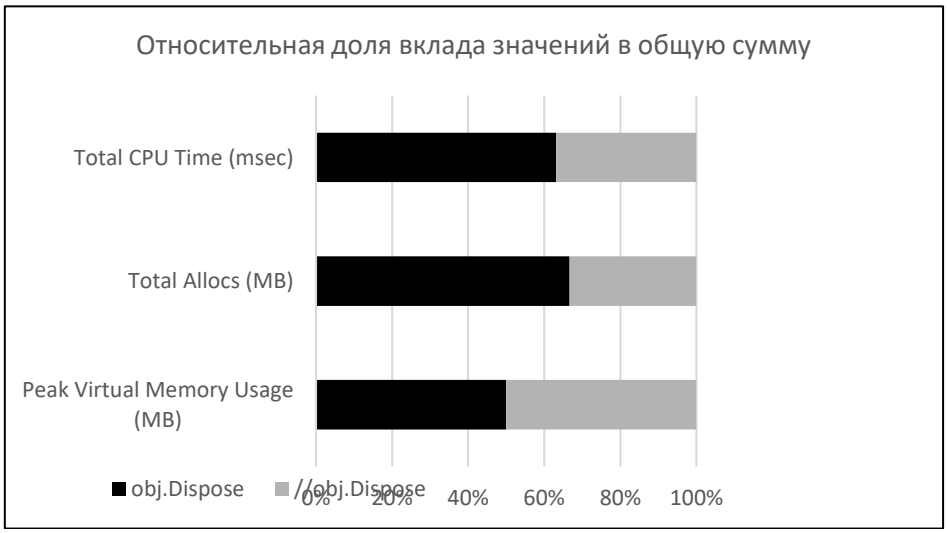

Рисунок 5 – Относительная доля вклада значений, которые имеют улучшенные показатели при задействовании финализатора

Figure 5 - The relative share of the contribution of values that have improved indicators when using the finalizer

В сводных таблицах производительности сборки мусора (Таблицы 1 и 2) при сопоставлении количественных характеристик можно сделать выводы, что при выполнении программы с методом Dispose, GC вызывается 6 раз, а без метода – 3 раза; GC задерживает процесс с методом Dispose на 0,5 мс, а без метода – на 1,7 мс; GC начинает удалять большее число объектов первого поколения чем нулевого без метода Dispose.

При запуске приложения с методом Dispose вся сборка мусора происходит в нулевом поколении, то есть все создаваемые объекты умирают быстро и не доживают до следующего поколения, а при задействовании финализатора появляются объекты, пережившие сборку мусора в нулевом поколении и помещённые GC в первое поколение. При этом количество сборок мусора в первом поколении в два раза больше, чем в нулевом поколении (Таблица 2), а сборка первого поколения в полтора раза длиннее нулевого поколения (Max Pause).

В обеих таблицах среднее время паузы равно максимальной паузе (Max Pause), соответствующей времени на которое останавливаются все потоки .NET приложения. Induced-сборка равна нулю в обеих таблицах, так как в коде нет вызова GC.Collect. По количеству сборок мусора более привлекателен запуск приложения с финализатором, а по максимальным паузам, останавливающим все потоки – приложение с вызовом метода Dispose.

Таким образом, обобщая результаты, представленные на Рисунках 2 и 3 и в таблицах, можно сделать вывод, что запуск приложения с методом Dispose и с

активацией финализатора имеет как положительные, так и отрицательные значения характеристик в сравнении, и многое зависит от выбора ЛПР критерия оптимизации.

Заключение

Принятие решений в задачах управления памятью приложений в .NET ставит ЛПР перед выбором оптимального метода. В работе исследован алгоритм Dispose-паттерна на примере клиент-компонентной модели. Получены количественные характеристики деятельности GC с помощью утилиты PerfView работы приложения с финализатором и без него. Исследование поможет ЛПР в выборе критерия оценки разных методов управления памятью.

В перспективе необходимо исследование других методов управления памятью, фиксирование количественных характеристик с помощью технических средств, выбора однокритериальной или многокритериальной модели.

ЛИТЕРАТУРА

1. Анализ проблем с памятью .NET Framework. *Microsoft. Документация*. 2016. Доступно по адресу: https://technet.microsoft.com/ru-ru/evalcenter/dn342825(v=vs.85)/ (дата обращения: 20.06.2020 г.).
2. Костикова А. В. Исторические аспекты развития теории принятия решений. *Философия науки*, 2014;3(62):16-28.
3. Уотсон Бен. *Высокопроизводительный код на платформе .NET*. 2-е изд. - СПб.: Питер, 2019:1-416.
4. Atienza D., Mamagkakis S. Dynamic Memory Management Design Methodology for Reduced Memory Footprint in Multimedia and Wireless Network Applications. *Computer Science Proceedings Design, Automation and Test in Europe Conference and Exhibition*. 2004:1-74.
5. GC.Collect Method. *Microsoft. Документация*. Доступно по адресу: https://docs.microsoft.com/ru-ru/dotnet/api/system.gc.collect?view=netcore-2.0/ (дата обращения: 15.06.2020 г.).
6. Производительность .NET Framework. *Microsoft. Документация*. 2017. Доступно по адресу: https://docs.microsoft.com/ru-ru/dotnet/framework/performance/ (дата обращения: 19.07.2020 г.).
7. Албахари Д., Албахари Б. *C# 5.0. Справочник. Полное описание языка.* :Пер. с англ. - М.:ООО "И. Д. Вильямс", 2014:1-1008.
8. Реализация метода Dispose. *Microsoft. Документация*. 2020. Доступно по адресу: https://docs.microsoft.com/ru-ru/dotnet/standard/garbage-collection/implementing-dispose/ (дата обращения 10.07.2020 г.).
9. Michaelis M. *Essential C# 3.0: For .NET Framework 3.5*, Pearson Education, Inc., 2009:1-335.
10. ГОСТ 19.701-90 (ИСО 5807-85) "Единая система программной документации". М.: Стандартинформ, 2010:1-23.
11. Parkinson M. et al. Project Snowflake: Non-blocking Safe Manual Memory Management in .NET. *Microsoft Research*, 2017:1-40.

REFERENCES

1. Analysis of memory problems.NET Framework. *Microsoft. Documentation*. 2016. Available from: https://technet.microsoft.com/ru-ru/evalcenter/dn342825(v=vs.85)/ [Accessed 20th June 2020]. (In Russ)
2. Kostikova A. V. Historical aspects of the development of decision theory. *Filosofiya nauki* 2014;3(62):16-28.
3. Uotson Ben. *High performance code on the platform.NET*. 2-e izd. - SPb.: Piter, 2019:1-416.
4. Atienza D., Mamagkakis S. Dynamic Memory Management Design Methodology for Reduced Memory Footprint in Multimedia and Wireless Network Applications. *Computer Science Proceedings Design, Automation and Test in Europe Conference and Exhibition*. 2004:1-74.
5. GC.Collect Method. *Microsoft. Documentation*. Available from: https://docs.microsoft.com/ru-ru/dotnet/api/system.gc.collect?view=netcore-2.0/ [Accessed 15th June 2020]. (In Russ)
6. Performance .NET Framework. *Microsoft. Documentation*. 2017. Available from: https://docs.microsoft.com/ru-ru/dotnet/framework/performance. [Accessed 19th July 2020]. (In Russ)
7. Albahari D., Albahari B. *C# 5.0. Directory. Full language description*. :Per. s angl. - M.:OOO "I. D. Vil'yams", 2014:1-1008. (In Russ)
8. Implementation of the method Dispose. *Microsoft. Documentation.* 2020. Available from: https://docs.microsoft.com/ru-ru/dotnet/standard/garbage-collection/implementing-dispose. [Accessed 10th July 2020]. (In Russ)
9. Michaelis M. *Essential C# 3.0: For .NET Framework 3.5*. Pearson Education, Inc., 2009:1-335.
10. GOST 19.701-90 (ISO 5807-85) "Unified system of program documentation." M.: Standartinform, 2010:1-23.
11. Parkinson M. et al. Project Snowflake: Non-blocking Safe Manual Memory Management in .NET. *Microsoft Research*, 2017:1-40.

ИНФОРМАЦИЯ ОБ АВТОРЕ / INFORMATION ABOUT THE AUTHOR

Попова Елена Владимировна, к.т.н.**,** доцент, кафедра МО ЭВМ, СПбГЭТУ «ЛЭТИ», Санкт-Петербург, Российская Федерация.
e-mail: serana5@inbox.ru

Elena V. Popova, Can. Sci. (Tech), Associate Professor, Department Of Computer Science, Saint Petersburg Electrotechnical University "LETI", Saint Petersburg, Russian Federation.