ASP.NET Core. Веб-приложения

Организация памяти в .NET Core



# На этом уроке

1. Закрепим знания по организации памяти в .NET
2. Реализуем консольное приложение для отработки навыков по работе с памятью
3. Познакомимся с фреймворком ASP.NET Core
4. Создадим WebApi-прилоложение с помощью ASP.NET Core для проекта Timesheets
5. Закрепим базовые понятия HTTP-протокола

# 

# Организация памяти в .NET

В языке C#, как нам уже известно, типы подразделяются на два вида: типы значений (value types) и ссылочные типы (reference types).

Значимые типы:

1. Целочисленные типы (byte, sbyte, short, ushort, int, uint, long, ulong)
2. Типы с плавающей точкой (float, double, decimal)
3. Символ (char), логический (bool)
4. Перечисление (enum)
5. Структуры (struct)

Ссылочные типы:

1. Классы (class) и интерфейсы (interface), записи (record) для С# 9 и выше.
2. Делегаты (delegate)
3. Базовый тип (object)
4. Динамический тип (dynamic)
5. Строки (string)

Значимые типы хранятся в быстрой, но ограниченной области памяти под названием стек. В операционных системах x32 размер стека ограничен 1 МБ; для систем x64 это значение — 4 МБ.

Ссылочные же типы, внутри которых могут храниться значительные массивы информации, находятся на управляемой куче. Это не такая быстрая, но зато обширная область памяти на платформе .NET. Размер управляемой кучи может достигать 1.5 ГБ в системах x32 и неограничен в системах x64.

При инициализации переменных значимых типов их значения записываются на стек. А при инициализации объектов ссылочных типов их значения записываются в куче. При этом на стек помещается указатель на область памяти в куче, где и хранится заданное значение.

Объекты значимых типов живут от момента объявления и до конца охватывающего блока. Объекты ссылочных типов живут от момента инициализации до момента, когда они будут очищены сборщиком мусора.

Во время запуска .NET-приложения ему выделяется область в динамической памяти операционной системы. Это и есть Managed Heap, внутри которой аллоцируются объекты внутри программы. При этом сама управляемая куча располагается рядом с другими объектами в памяти операционной системы.



Что касается самой управляемой кучи, то она для лучшей оптимизации разделена на область памяти мелких объектов (Small Object Heap, SOH) и область памяти крупных объектов (Large Object Heap, LOH). Причем в последнюю категорию попадают объекты размером более 85 000 байт.

# Сборщик мусора (Garbage Collector)

Для помещения ссылочного объекта на управляемую кучу используется уже знакомое ключевое слово *new* вместе с конструктором типа:

|  |
| --- |
| var client = new HttpClient(); |

В нашем случае мы создаем экземпляр класса *HttpClient,* помещаем его на кучу, а переменную *client* инициализируем ссылкой на данный объект на куче.

Нетрудно догадаться, что бесконечно создавать новые объекты в куче и не чистить их попросту нельзя, иначе в какой-то момент, когда на ней попросту не останется места.

В некоторых языках программирования, таких как C++, программисты сами должны заботиться о своевременном удалении неиспользуемых объектов. В платформе .NET существует встроенный механизм автоматического управления памятью под названием Garbage Collector (сборщик мусора).

Он отвечает за использование динамической области памяти (называемой кучей). В частности, GC отвечает за автоматическое освобождение памяти от неиспользуемых объектов.

# Деструктор

Перед уничтожением объекта сборщику мусора необходимо вызвать деструктор класса, чтобы освободить используемые им ресурсы системы. Сам деструктор с помощью знака тильды определяется следующим образом:

|  |
| --- |
| public class SomeClass {   ~SomeClass()  {  // destruct instructions  } } |

При очистке мусора произойдет вызов метода *Finalize(),* потому что компилятор преобразует деструктор в такую конструкцию:

|  |
| --- |
| Finalize() {  try  {  // destruct instructions  }  finally  {  base.Finalize();  } } |

Иными словами, все инструкции из деструктора класса будут помещены в блок *try* метода *finalize*.

Сам деструктор нельзя вызвать вручную. Он может быть вызван только автоматически сборщиком мусора платформы.

# Механизм сборки мусора

Когда ссылочный объект перестает использоваться, ссылка на место в куче очищается из стека. Однако при этом сам участок памяти на куче попадает под работу инструмента сборщика мусора (GC). Через некоторое время после удаления ссылки из стека GC увидит, что на участок в памяти больше нет ссылок и очистит его. Но это не произойдет одновременно с удалением ссылки из стека (об этом позднее).

Обычно, сборщик мусора запускается в тот момент, когда CLR (Common Language Runtime) испытывает потребность в дополнительной памяти. Его также можно вызвать вручную через метод *GC.Collect()*.

Чтобы оптимизировать работу сборщика мусора, объекты в куче классифицируются по поколениям: 0, 1 и 2.

Поколение 0 присваивается новым объектам, которые не подвергались очищению. Поколение 1 присваивается объектам, прошедшим одну сборку мусора, а поколение 2 — объектам, которые прошли более одного этапа сборки мусора.

Соответственно, алгоритм сборщика мусора построен таким образом, чтобы сначала попытаться очистить объекты нулевого поколения. Поскольку куча сильно фрагментирована и беспорядочна в части хранимых в ней данных, то более «молодые» объекты из поколения 0 очищаются с меньшими затратами, потому что они скорее всего будут расположены близко друг к другу.

Если после очистки объектов с поколением 0 системе все еще требуется память, то сборщик переходит к поколению 1, а затем к поколению 2.

# Введение в ASP.NET Core

ASP.NET Core это кроссплатформенный фреймворк для разработки web-приложений. Он позволяет создавать web-приложения, службы, приложения Интернета вещей (IoT) и серверные части для мобильных приложений.

У этого фреймворка есть довольно много преимуществ:

* Единое решение для создания пользовательского веб-интерфейса и веб-API
* Razor Pages упрощает написание кода для сценариев страниц и повышает его эффективность
* Blazor позволяет использовать в браузере язык C# вместе с JavaScript
* Совместное использование серверной и клиентской логик приложений, написанных с помощью .NET
* Возможность разработки и запуска в ОС Windows, macOS и Linux
* Открытый исходный код и ориентация на сообщество
* Интеграция современных клиентских платформ и рабочих процессов разработки
* Поддержка размещения служб удаленного вызова процедур (RPC) с помощью gRPC
* Облачная система конфигурации на основе среды
* Встроенное введение зависимостей
* Упрощенный высокопроизводительный модульный конвейер HTTP-запросов

# Введение в проект Timesheets

На протяжении курса мы будем разрабатывать собственное приложение учета рабочего времени сотрудников.

Предположим, что существует некая компания, которая оказывает услуги клиентам по договорам. Её сотрудники выполняют отдельные виды работ по каждому из договоров и регистрируют затраченное время. В конце отчетного периода система генерирует счета на оплату для клиентов исходя из работ, которые были зарегистрированы сотрудниками.

Кратко схему основного бизнес-процесса системы можно представить следующим образом:



# 

# Глоссарий

**Стек** — область оперативной памяти, работающая по принципу LIFO. Ограничен по размеру.

**Куча** — область оперативной памяти с динамическим выделением. Ограничена лишь объемом памяти.

**Сборщик мусора** — механизм выделения и освобождения памяти на куче.

# Дополнительные материалы

1. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#, Рихтер

# Используемые источники

1. <https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/reference-types>
2. <https://docs.microsoft.com/ru-ru/dotnet/standard/garbage-collection/>
3. <https://docs.microsoft.com/ru-ru/dotnet/standard/garbage-collection/fundamentals>
4. <https://docs.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0>
5. <https://docs.microsoft.com/ru-ru/aspnet/core/fundamentals/middleware/?view=aspnetcore-5.0>
6. <https://developer.mozilla.org/ru/docs/Web/HTTP/Overview>