

Управление памятью. Сборщик мусора.

№ урока: 9 **Курс:** C# Professional

Средства обучения: Компьютер с установленной Visual Studio

Обзор, цель и назначение урока

Целью данного урока, является разъяснение того, что собой представляет процесс сборки мусора, и какими механизмами руководствуется CLR для управления памятью. Урок позволяет понять, как программно взаимодействовать со сборщиком мусора с помощью класса `System.GC`, и как, используя виртуальный метод `System.Object.Finalize()` и интерфейс `IDisposable`, создавать классы, способные освобождать внутренние неуправляемые ресурсы в определенное время.

Изучив материал данного занятия, учащийся сможет:

- Понимать определение время жизни объекта
- Понимать работу сборщика мусора
- Использовать и создавать деструкторы и финализаторы
- Выполнять освобождение неуправляемых ресурсов
- Реализовывать интерфейс `IDisposable`

Содержание урока

1. Время жизни объекта в памяти
2. Сборщик мусора. Поколения объектов
3. Деструкторы и финализаторы
4. Освобождение неуправляемых ресурсов. Интерфейс `IDisposable`

Резюме

- Как только класс создан, с использованием ключевого слова `new`, поддерживаемого в C#, можно размещать в памяти любое количество его объектов. Ключевое слово `new` возвращает ссылку на объект в куче, а не фактический объект. Если ссылочная переменная создается как локальная переменная в контексте метода, она сохраняется в стеке для дальнейшего использования в приложении.
- После создания объект будет автоматически удален сборщиком мусора тогда, когда в нем отпадет необходимость.
- Сборщик мусора удаляет объект из кучи тогда, когда тот становится недостижимым ни в одной части программного кода.
- Сборщик мусора .NET предназначен в основном для того, чтобы управлять памятью вместо разработчиков.
- В среде CLR сборщик мусора выполняет функции автоматического диспетчера памяти. Это предоставляет следующие преимущества:
 - a) Позволяет разрабатывать приложение без необходимости освобождать память.
 - b) Эффективно выделяет память для объектов в управляемой куче.
 - c) Уничтожает объекты, которые больше не используются, очищает их память и сохраняет память доступной для будущих распределений. Содержимое создаваемых управляемых объектов автоматически оказывается очищенным, чтобы их конструкторам не было нужно инициализировать каждое поле данных.
 - d) Обеспечивает безопасность памяти, гарантируя, что объект не сможет использовать содержимое другого объекта.
- Принудительно запустить сборку мусора с помощью метода `GC.Collect()`, можно в следующих случаях:
 1. Приложение приступает к выполнению блока кода, прерывание которого возможным процессом сборки мусора является недопустимым.

2. Приложение только что закончило размещать чрезвычайно большое количество объектов и нуждается в как можно скорейшем освобождении большого объема памяти.
- После инициализации средой CLR сборщик мусора выделяет сегмент памяти для хранения объектов и управления ими. Эта память называется **управляемой кучей** в отличие от собственной кучи операционной системы.
 - Куча организована в виде **поколений**, что позволяет ей обрабатывать долгоживущие и короткоживущие объекты. Сборка мусора в основном сводится к уничтожению короткоживущих объектов, которые обычно занимают только небольшую часть кучи. В куче существует три поколения объектов.
 - **Поколение 0.** *Объекты не проверялись сборщиком мусора.* Это самое молодое поколение содержит короткоживущие объекты. Примером короткоживущего объекта является временная переменная. Сборка мусора чаще всего выполняется в этом поколении. Вновь распределенные объекты образуют новое поколение объектов и неявно являются сборками поколения 0, если они не являются большими объектами, в противном случае они попадают в кучу больших объектов в сборке поколения 2.
 - **Поколение 1.** *Объекты, пережившие одну проверку сборщиком мусора (а также объекты, помеченные на удаление, но не удаленные, так как в управляемой куче было достаточно свободного места).* Это поколение содержит коротко живущие объекты и служит буфером между короткоживущими и долгоживущими объектами.
 - **Поколение 2.** *Объекты, которые пережили более чем одну проверку сборщиком мусора.* Это поколение содержит долгоживущие объекты. Примером долгоживущих объектов служит объект в серверном приложении, содержащий статические данные, которые существуют в течение длительности процесса.
 - GC не избавляет от управления ресурсами.
 - Дескриптор файла – это ресурс, который должен быть каким-то образом освобожден.
 - GC имеет дело напрямую лишь с ресурсами памяти. Для обработки ресурсов, не связанных с памятью, таких как подключение к базе данных и дескрипторы файлов, можно использовать **финализаторы**, которые позволяют освобождать ресурсы тогда, когда GC известит об уничтожении объекта.
 - Структурные типы могут иметь конструкторы, но не финализаторы. По умолчанию при передаче методу структурного типа в качестве параметра, метод принимает копию этого значения.
 - При сжати кучи сборщиком мусора все объекты, помеченные для удаления, либо освобождают занятую ими память, либо помещаются в очередь на удаление, если у них есть финализатор. За это отвечает другой поток – поток финализатора, который проходит по очереди объектов и вызывает их финализаторы перед освобождением памяти. По завершении работы финализатора память объекта освобождается при следующем проходе сборщика мусора, и объект окончательно уничтожается, уже безвозвратно.
 - Метод `Finalize` представляет собой переопределение виртуального метода из `System.Object`, однако в C# не разрешается явно переопределять этот метод. Вместо этого должен быть написан деструктор, который выглядит как метод без типа возврата, без модификаторов, без параметров, идентификатором которого служит имя класса с предшествующим символом тильды (~).
 - Деструкторы не могут вызываться явно в C#, они не наследуются.
 - Класс может иметь только один деструктор.
 - Когда вызывается финализатор объекта, то вызывается каждый финализатор в цепочке наследования – от последнего к первому.
 - Финализаторы выполняются в отдельном потоке CLR.
 - Код, который объекты выполняют внутри деструктора, должен быть безопасным в отношении потоков.
 - Нет никакого гарантированного способа узнать точно, когда будет вызван финализатор, или в каком порядке будут вызваны финализаторы двух зависимых или независимых объектов.
 - В финализаторе не должно делаться ничего сложнее простой уборки, если она необходима.
 - Финализатор должен создаваться только тогда, когда объект имеет дело с неуправляемыми ресурсами некоторого рода.

- Основное назначение интерфейса **IDisposable** заключается в высвобождении неуправляемых ресурсов. Сборщик мусора автоматически высвобождает память, выделенную для управляемого объекта, если этот объект уже не используется.
- При использовании **IDisposable** ответственность за вызов метода `Dispose` возлагается на клиента.
- У клиента нет никакой возможности поручить системе или компилятору его автоматический вызов.
- При реализации метода `Dispose` класс обычно строится таким образом, что код финализатора повторно использует `Dispose`.
- Ключевое слово **using** было перегружено для поддержки шаблона `Disposable`. Общая идея состояла в том, что оператор **using** должен захватывать ресурсы внутри фигурных скобок, следующих за ключевым словом **using**, в то время как область видимости этих локальных переменных ограничена областью определения следующих далее фигурных скобок.
- Оператор **using** расширяется до конструкции `try/finally`.
- Оператор **using** требует, чтобы все ресурсы, захваченные в процессе, были неявно преобразуемыми к **IDisposable**.

Закрепление материала

- Что такое время жизни объекта?
- Что такое управляемая куча?
- Что такое сборщик мусора?
- Что такое поколения **GC**, и какие поколения вы знаете?
- Что такое финализаторы и деструкторы?
- Назовите основные правила использования деструкторов.
- Для чего используется интерфейс **IDisposable**?

Дополнительное задание

Создайте свой класс, объекты которого будут занимать много места в памяти (например, в коде класса будет присутствовать большой массив) и реализуйте для этого класса, формализованный шаблон очистки.

Самостоятельная деятельность учащегося

Задание 1

Выучите основные конструкции и понятия, рассмотренные на уроке.

Задание 2

Создайте класс, который позволит выполнять мониторинг ресурсов, используемых программой. Используйте его в целях наблюдения за работой программы, а именно: пользователь может указать приемлемые уровни потребления ресурсов (памяти), а методы класса позволят выдать предупреждение, когда количество реально используемых ресурсов приблизиться к максимально допустимому уровню.

Задание 3

Зайдите на сайт MSDN.

Используя поисковые механизмы MSDN, найдите самостоятельно описание темы по каждому примеру, который был рассмотрен на уроке, так, как это представлено ниже, в разделе «Рекомендуемые ресурсы», описания данного урока. Сохраните ссылки и дайте им короткое описание.

Рекомендуемые ресурсы

MSDN: Сборка мусора

<http://msdn.microsoft.com/ru-ru/library/0xy59wtx.aspx>

MSDN: Основы сборки мусора

<http://msdn.microsoft.com/ru-ru/library/ee787088.aspx#generations>

MSDN: класс GC

<http://msdn.microsoft.com/ru-ru/library/system.gc.aspx>

MSDN: интерфейс IDisposable

<http://msdn.microsoft.com/ru-ru/library/system.idisposable.aspx>