Memory Management in .NET

In the common language runtime (CLR), the garbage collector (GC) serves as an automatic memory manager. The garbage collector manages the allocation and release of memory for an application. For developers working with managed code, this means that you don't have to write code to perform memory management tasks. Automatic memory management can eliminate common problems, such as forgetting to free an object and causing a memory leak or attempting to access memory for an object that's already been freed.

Fundamentals of memory

The following list summarizes important CLR memory concepts:

- Each process has its own, separate virtual address space. All processes on the same computer share the same physical memory and the page file, if there is one.
- By default, on 32-bit computers, each process has a 2-GB user-mode virtual address space.
- As an application developer, you work only with virtual address space and never manipulate physical memory directly. The garbage collector allocates and frees virtual memory for you on the managed heap.
- If you're writing native code, you use Windows functions to work with the virtual address space. These functions allocate and free virtual memory for you on native heaps.
- Virtual memory can be in three states:

State	Description
Free	The block of memory has no references to it and is available for allocation.
Reserved	The block of memory is available for your use and cannot be used for any other allocation request. However, you cannot store data to this memory block until it is committed.
Committed	The block of memory is assigned to physical storage.

- Virtual address space can get fragmented. This means that there are free blocks, also known as holes, in the address space. When a virtual memory allocation is requested, the virtual memory manager has to find a single free block that is large enough to satisfy that allocation request. Even if you have 2 GB of free space, an allocation that requires 2 GB will be unsuccessful unless all of that free space is in a single address block.
- You can run out of memory if there isn't enough virtual address space to reserve or physical space to commit.

The page file is used even if physical memory pressure (that is, demand for physical memory) is low. The first time that physical memory pressure is high, the operating system must make room in physical memory to store data, and it backs up some of the data that is in physical memory to the page file. That data is not paged until it's needed, so it's possible to encounter paging in situations where the physical memory pressure is low.

Memory allocation

When you initialize a new process, the runtime reserves a contiguous region of address space for the process. This reserved address space is called the managed heap. The managed heap maintains a pointer to the address where the next object in the heap will be allocated. Initially, this pointer is set to the managed heap's base address. All reference types are allocated on the managed heap. When an application creates the first reference type, memory is allocated for the type at the base address of the managed heap. When the application creates the next object, the garbage collector allocates memory for it in the address space immediately following the first object. As long as address space is available, the garbage collector continues to allocate space for new objects in this manner.

Releasing Memory

The garbage collector's optimizing engine determines the best time to perform a collection based on the allocations being made. When the garbage collector performs a collection, it releases the memory for objects that are no longer being used by the application. It determines which objects are no longer being used by examining the application's roots. Every application has a set of roots. Each root either refers to an object on the managed heap or is set to null. An application's roots include static fields, local variables and parameters on a thread's stack, and CPU registers. The garbage collector has access to the list of active roots that the just-in-time (JIT) compiler and the runtime maintain. Using this list, it examines an application's roots, and in the process creates a graph that contains all the objects that are reachable from the roots.

Objects that are not in the graph are unreachable from the application's roots. The garbage collector considers unreachable objects garbage and will release the memory allocated for them. During a collection, the garbage collector examines the managed heap, looking for the blocks of address space occupied by unreachable objects. As it discovers each unreachable object, it uses a memory-copying function to compact the reachable objects in memory, freeing up the blocks of address spaces allocated to unreachable objects. Once the memory for the reachable objects has been compacted, the garbage collector makes the necessary pointer corrections so that the application's roots point to the objects in their new locations. It also positions the managed heap's pointer after the last reachable object. Note that memory is compacted only if a collection discovers a significant number of unreachable objects. If all the objects in the managed heap survive a collection, then there is no need for memory compaction.

To improve performance, the runtime allocates memory for large objects in a separate heap. The garbage collector automatically releases the memory for large objects. However, to avoid moving large objects in memory, this memory is not compacted.

The managed heap

After the garbage collector is initialized by the CLR, it allocates a segment of memory to store and manage objects. This memory is called the managed heap, as opposed to a native heap in the operating system.

There is a managed heap for each managed process. All threads in the process allocate memory for objects on the same heap.

To reserve memory, the garbage collector calls the Windows `VirtualAlloc` function and reserves one segment of memory at a time for managed applications. The garbage collector also reserves segments, as needed, and releases segments back to the operating system (after clearing them of any objects) by calling the Windows `VirtualFree` function.

The fewer objects allocated on the heap, the less work the garbage collector has to do. When you allocate objects, don't use rounded-up values that exceed your needs, such as allocating an array of 32 bytes when you need only 15 bytes.

When a garbage collection is triggered, the garbage collector reclaims the memory that's occupied by dead objects. The reclaiming process compacts live objects so that they are moved together, and the dead space is removed, thereby making the heap smaller. This ensures that objects that are allocated together stay together on the managed heap to preserve their locality.

The intrusiveness (frequency and duration) of garbage collections is the result of the volume of allocations and the amount of survived memory on the managed heap.

The heap can be considered as the accumulation of two heaps: the large object heap and the small object heap. The large object heap contains objects that are 85,000 bytes and larger, which are usually arrays. It's rare for an instance object to be extremely large.