How iOS Handles Memory with ARC

Livingston

iOS handles memory management with ARC, which allocates chunks of memory to each instance, property, constant, or variable. It keeps track of how many instances are utilizing that piece of memory by how many other instances are referencing it. Once its finally done being implemented, that instance of memory is cleared. If one were to try and use that instance later it would probably just crash the app.

The lifecycle of ARC starts off with memory allocation, initialization,usage, deinitialization, and lastly followed by deallocation. There are many types of reference models, some of which are strong and can’t be cleared by ARC, others are explicitly labeled weak to be easily cleared. However, if a reference object is never deinitialized, then it continuously takes up space in memory.

ARC compared to a MRR (manual-retain-release) is almost the complete opposite of each other. ARC allocates memory in a predictable and efficient way automatically. In a MRR method, the coder is in charge of managing all the memory allocation, deinitialization, and deallocation manually. This can be much more tedious than just letting ARC do that automatically. Its more difficult using MRC because you have to keep careful track of when memory instances are no longer being used. If you accidentally call one after its been deallocated you’ll most likely crash the app.

ARC is similar to whats referred to as a Garbage Collection; used specifically in Java platforms. The main difference between these two is that ARC waits for instances to no longer be called on, while Garbage Collection will randomly dump out something that may still be in use if it sees that memory space is getting low. Garbage Collection has very indeterminate times that it will clear something out. This process could easily lead to problems if specific pieces of memory are still being used elsewhere at the time they get dumped.