**Поколения памет**

**Поколенията (generations)** са механизъм в garbage collector, чиято

единствена цел е подобряването на производителността. Основната идея

е, че почистването на част от динамичната памет винаги е по-бързо от

почистването на цялата памет. Вместо да обхожда всички обекти от

хийпа, garbage collector обхожда само част от тях, класифицирайки ги по

определен признак. В основата на механизма на поколенията стоят

следните предположения:

- колкото по-нов е един обект, толкова по-вероятно е животът му да е

кратък. Типичен пример за такъв случай са локалните променливи,

които се създават в тялото на даден метод и излизат от обхват при

неговото напускане.

- колкото по-стар е обектът, толкова по-големи са очакванията той да

живее дълго. Пример за такива обекти са глобалните променливи.

- обектите, създадени по едно и също време обикновено имат връзка

помежду си и имат приблизително еднаква продължителност на

живота.

Много изследвания потвърждават валидността на изброените твърдения

за голям брой съществуващи приложения. Нека разгледаме по-подробно

поколенията памет и това как те се използват за оптимизация на произво-

дителността на .NET garbage collector.

**Поколение 0**

Когато приложението се стартира, първоначално динамичната памет не

съдържа никакви обекти. Всички обекти, които се създават, стават част от

Поколение 0. Казано накратко Поколение 0 съдържа новосъздадените

обекти – тези, които никога не са били проверявани от garbage collector.

При инициализацията на CLR се определя праг за размера на Поколение

0.Да предположим, че приложението иска да създаде нов обект, F. Добавя-

нето на този обект би предизвикало препълване на Поколение 0. В този

момент трябва да започне събиране на отпадъци и се стартира garbage

collector.

**Почистване на Поколение 0**

Garbage collector процедира по описания по-горе алгоритъм и установява

че обекти B и D са отпадъци. Тези обекти се унищожават и оцелелите

обекти A, C и E се пренареждат в долната (или лява) част на managed

heap. Динамичната памет непосредствено след приключването на събира-

нето на отпадъци изглежда по следния начин:

Сега оцелелите при преминаването на garbage collector обекти стават част

от Поколение 1 (защото са оцелели при едно преминаване на garbage

collector). Новият обект F, както и всички други новосъздадени обекти ще

бъдат част от Поколение 0.

Нека сега предположим, че е минало още известно време, през което

приложението е създавало обекти в динамичната памет. Managed heap

сега изглежда по следния начин:

Добавянето на нов обект J, би предизвикало препълване на Поколение 0,

така че отново трябва да се стартира събирането на отпадъци. Когато

garbage collector се стартира, той трябва да реши кои обекти от паметта

да прегледа. Както Поколение 0, така и Поколение 1 има праг за своя

размер, който се определя от CLR при инициализацията. Този праг е по-

голям от този на Поколение 0. Да предположим че той е 2MB.

В случая Поколение 1 не е достигнало прага си, така че garbage collector

ще прегледа отново само обектите от Поколение 0. Това се диктува от

правилото, че по-старите обекти обикновено имат по-дълъг живот и сле-

дователно почистването на Поколение 1 не е вероятно да освободи много

памет, докато в Поколение 0 е твърде възможно много от обектите да са

отпадъци. И така, garbage collector почиства отново Поколение 0, оцеле-

лите обекти преминават в Поколение 1, а тези, които преди това са били в

Поколение 1, просто си остават там.

Забележете, че обект C, който междувременно е станал недостъпен и сле-

дователно подлежи на унищожение, в този случай остава в динамичната

памет, тъй като е част от Поколение 1 и не е проверен при това премина-

ване на garbage collector.

Следващата фигура показва състоянието на динамичната памет след това

почистване на Поколение 0.

Както вероятно се досещате, с течение на времето Поколение 1 бавно ще

расте. Идва момент, когато след поредното почистване на Поколение 0,

Поколение 1 достига своя праг от 2 MB. В този случай приложението

просто ще продължи да работи, тъй като Поколение 0 току-що е било

почистено и е празно. Новите обекти, както винаги, ще се добавят в

Поколение 0.