

Exzerpt zu: *Java theory and practice: A brief history of garbage collection*

Preview:

- Java GC
- Wie funktioniert
- Grundlegende Algorithmen:
 - Reference counting
 - Tracing collectors
 - Mark-Sweep collectors
 - Copying collectors
 - Heap compaction
 - Mark-compact collectors

Questions:

1. Was macht ein GC?
2. Welche Art von GC verwendet Java?
3. Wie arbeiten GCs?
4. Welche Arten von GCs gibt es?
5. Wie arbeiten diese?
6. Nach welchen Kriterien kann man ein passendes Verfahren finden?
7. Was sind Vor-/Nachteile der einzelnen Verfahren?

Read:

1. Ein GC identifiziert Speicherbereiche, die zuvor von der Applikation allokiert wurden, jetzt aber nichtmehr zu erreichen sind (es also keine Referenzen mehr gibt, die auf diese Bereiche verweisen) und gibt deren Speicher wieder frei.
2. Java ermöglicht es, viele verschiedene GC-Strategien zu nutzen. Java 1.4 enthält 6 verschiedene GCs und viele Möglichkeiten, diese an die Anforderungen einer Applikation anzupassen.
3. Es gibt viele verschiedene Arbeitsweisen. Manche GCs können inkrementell oder nebenläufig arbeiten, andere müssen das Programm anhalten während sie laufen, ausserdem gibt es Mischformen. GCs mit verschiedenen Generationen (werden in Java seit Version 1.2 eingesetzt) können so z.B. verschiedene Methoden auf verschiedenen Bereiche des Heaps anwenden.
4. Reference Counting, Mark-Sweep, Mark-Compact und Copying.
5. *Reference Counting:*
Die einfachste Methode. Jedes Objekt zählt, wie viele Referenzen auf es verweisen (reference count). Wenn der Referenzzähler 0 wird, wird das Objekt nicht mehr gebraucht und kann freigegeben werden. Das Programm muss sicherstellen, dass jedes mal, wenn sich eine Referenz ändert, der Referenzzähler des entsprechenden Objekts aktualisiert wird. Wird z.B. von der `std::string` Klasse der C++ Standardbibliothek verwendet.
Vorteile: einfach und gute Lokalität (Objekt wird freigegeben, wenn der Zähler 0 wird).
Nachteile: Zyklische Referenzen (Objekt A referenziert Objekt B und Objekt B referenziert gleichzeitig Objekt A) können nicht aufgelöst werden (Folge: Speicherlecks). Reference counting wird selten als richtiger GC eingesetzt, Java verwendet kein Reference Counting.

Tracing Collectors:

Bezeichnet eine Gruppe von Verfahren. Alle von Java verwendeten GCs sind Tracing Collectors. Solche GCs ermitteln, ausgehend vom Root Set, bei jedem Durchlauf selbst, welche Objekte noch erreichbar sind.

Mark-Sweep:

Die einfachste Form eines Tracing Collectors. Eingeführt 1960 mit Lisp. Der Kollektor hält die Applikation an und sucht für jede Referenz im Root Set alle Referenzen, die von dieser aus erreichbar sind. Jedes so erreichte Objekt wird markiert (mark). In der 2. Phase (sweep) werden alle nicht markierten Objekte freigegeben.

Vorteile: Einfach zu Implementieren, aber kann auch zyklische Referenzen verarbeiten und stellt keine Anforderungen an den Compiler.

Nachteile: Bei jedem Durchlauf werden alle Objekte mindestens einmal durchsucht (ob sie markiert sind), was zu langen Pausen führen kann. Ausserdem fragmentiert das Verfahren den Heap (aktive Objekte können über den gesamten Heap verteilt sein).

Copying Collectors:

Eine Untergruppe der Tracing Collectors. Der Heap in 2 Segmente unterteilt, in einem befinden sich die aktiven Elemente, das andere wird ist inaktiv. Beim Kollektordurchlauf werden alle "lebenden" Objekte vom aktive Heapsegment in das inaktive kopiert. Dann tauschen die Segmente die Rollen.

Vorteil: Es werden nur die lebenden Objekte bearbeitet und diese liegen im neuen Heap kompakt zusammen. Dadurch wird der Heap nicht fragmentiert und die Allokation neuer Objekte stark vereinfacht (neue Objekte werden einfach hinter dem letzten Objekt im Heap angefügt).

Nachteile: Verbraucht doppelten Speicher.

Mark-Compact:

Eine Kombination aus Mark-Sweep und Copying Kollektor. Mark-Phase wie gehabt, aber dann werden alle aktiven Objekte in einen kompakten Block am Anfang des Heaps kopiert.

Vorteile: Keine Fragmentierung wie bei Mark-Sweep und Objekte, die viele Kollektordurchläufe überleben sammeln sich am Boden des Heaps und werden daher nur selten kopiert (im Gegensatz zum Copying Collector).

6. Kriterien:

- Wie lange hält der Kollektor bei jedem Durchlauf die Applikation an?
- Wie vorhersehbar sind die Pausen?
- Wieviel Ressourcen werden vom Kollektor benötigt?
- In wieweit stört der Kollektor andere Funktionen (Virtueller Speicher, Cache)?
- Erhöht der Kollektor die Lokalität der Objekte im Speicher?
- Wie viel Unterstützung seitens des Compilers bzw. der Laufzeitumgebung wird benötigt?

7. (siehe 5.)

Reflect:

Liefert einen guten, wenn auch nur sehr groben Überblick über die gängigen Verfahren zur automatischen Speicherbereinigung, der nicht auf Java ausgerichtet, sondern sehr allgemein gehalten ist.

Recite:

Java ist möglicherweise die am weitesten verbreitete Programmiersprache, die Garbage Collection

verwendet. Garbage Collection hat viele Vorteile, es macht das schreiben eines Programms auf jeden Fall einfacher und sicherer. Es gibt aber auch einige Nachteile bei der Ausführungsgeschwindigkeit der Applikationen.

Da es den idealen Garbage Collector nicht gibt, gibt es in Java viele verschiedene GCs. Jeder dieser Kollektoren hat andere Vor- und Nachteile und Einschränkungen.

Die Grundlegenden Algorithmen sind Reference Counting, Tracing Collectors wie Mark-Sweep und Copying Collectors wie Mark-Compact.

Review:

- Sehr kurze aber übersichtliche Beschreibung der Verfahren.
- Es wird nur sehr wenig auf Java direkt eingegangen (man erfährt zwar, dass Java viele verschiedene GCs bereitstellt, aber welche das sind erfährt man hier nicht).
- Es wird auch nicht auf alle Verfahren (Generationen) eingegangen.