EBook Gratis

APRENDIZAJE garbage-collection

Free unaffiliated eBook created from
Stack Overflow contributors.

#garbage-collection

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: garbage-collection

It is an unofficial and free garbage-collection ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official garbage-collection.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con la recolección de basura

Observaciones

La recolección de basura (GC) es una forma de reclamar automáticamente la memoria ocupada por objetos que ya no son necesarios para un programa. Esto contrasta con la administración de memoria manual, donde el programador especifica explícitamente qué objetos deben ser desasignados y devueltos a la memoria. Las buenas estrategias de GC pueden ser más eficientes que la administración de memoria manual, pero pueden depender del tipo de software.

Las principales ventajas de la recolección de basura son:

- Libera al programador de tener que hacer el manejo manual de la memoria.
- Evita ciertos errores difíciles de encontrar que pueden surgir de la administración manual de la memoria (por ejemplo, punteros colgantes, doble liberación, ciertos tipos de fugas de memoria).
- Los lenguajes que usan recolección de basura son usualmente menos complejos.

Las principales desventajas son:

- La recolección de basura tiene algunos gastos generales en comparación con la administración de memoria manual.
- Potencialmente, puede afectar el rendimiento, especialmente cuando la recolección de basura se activa en momentos indeseables.
- Es indeterminista, el programador no sabe cuándo se realiza la recolección de basura y si los objetos se liberan o no.

La mayoría de los lenguajes de programación "más nuevos" tienen una recolección de basura integrada, por ejemplo, Java, C #, .NET, Ruby y JavaScript. Los lenguajes más antiguos como C y C ++ no tienen recolección de basura aunque hay implementaciones disponibles con la recolección de basura. También hay idiomas que le permiten usar una combinación de recolección de basura y administración de memoria manual, por ejemplo, Modula-3 y Ada.

Las estrategias de recolección de basura difieren, pero muchos utilizan una (variación de) el enfoque de marcar y barrer. En la fase de marca todos los objetos accesibles son encontrados y marcados. En la fase de barrido, el montón se escanea en busca de objetos inaccesibles y sin marcar que luego se limpian. Los recolectores de basura modernos también utilizan un enfoque generacional donde se mantienen dos o más regiones de asignación de objetos (generaciones). La generación más joven contiene los objetos asignados más nuevos y se limpia con más frecuencia. Los objetos que "sobreviven" durante un cierto período de tiempo se promueven a una generación anterior.

Muchos lenguajes con GC permiten a los programadores ajustarlo (consulte, por ejemplo, la Guía de ajuste de recolección de basura de la máquina virtual Java 8 o la documentación de

recolección de basura de .Net)

Examples

Introducción

Los objetos se vuelven elegibles para la recolección de basura (GC) si ya no son accesibles por los puntos de entrada principales en un programa. Por lo general, el usuario no realiza el GC explícitamente, pero para que el GC sepa que ya no se necesita un objeto, el desarrollador puede:

Desreferencia / asignación nula

```
someFunction {
     var a = 1;
     var b = 2;
     a = null; // GC can now free the memory used for variable a
     ...
} // local variable b not dereferenced but will be subject to GC when function ends
```

Usa referencias débiles

La mayoría de los idiomas con GC le permiten crear referencias débiles a un objeto que no cuentan como referencia para el GC. Si solo hay referencias débiles a un objeto y no hay referencias fuertes (normales), entonces el objeto es elegible para GC.

```
WeakReference wr = new WeakReference(createSomeObject());
```

Tenga en cuenta que después de este código es peligroso usar el objetivo de la referencia débil sin verificar si el objeto aún existe. Los programadores principiantes a veces cometen el error de usar un código como este:

```
if wr.target is not null {
    doSomeAction(wr.target);
}
```

Esto puede causar problemas porque GC puede haber sido invocado después de la comprobación nula y antes de la ejecución de doSomeAction. Es mejor crear primero una referencia fuerte (temporal) al objeto de esta manera:

```
Object strongRef = wr.target;
if strongRef is not null {
    doSomeAction(strongRef);
}
strongRef = null;
```

Habilitando el registro gc detallado en Java

Normalmente, la recolección de basura de jvm (gc) es transparente para el usuario (desarrollador

/ ingeniero).

Normalmente no se requiere la sintonización del GC a menos que el usuario se enfrente a una pérdida de memoria o tenga una aplicación que requiera una gran cantidad de memoria, lo que eventualmente conduce a una excepción de falta de memoria que obliga al usuario a investigar el problema.

El primer paso suele ser aumentar la memoria (ya sea el montón o el perm-gen / meta-espacio dependiendo de si se debe cargar en tiempo de ejecución o si la base de bibliotecas de la aplicación es grande o si hay una fuga en la carga de clases o en el subproceso mecanismo de manipulación). Pero cuando esto no sea factible, el siguiente paso es tratar de entender qué está mal.

Si uno quiere solo la instantánea en un instante particular en el tiempo, entonces la utilidad `jstat` que es parte del jdk sería suficiente.

Sin embargo, para una comprensión más detallada, es útil tener un registro que contenga la instantánea del montón antes y después de cada evento de gc. Para eso, el usuario debe habilitar el registro detallado de gc mediante el uso de `-verbose:gc` como parte de los parámetros de inicio de jvm e incluir los `-XX:+PrintGCDetails` y `-XX:+PrintGCTimeStamp` .

Para aquellos que `jvisualvm` perfil proactivo de su aplicación, también hay herramientas como `jvisualvm` que también forma parte del jdk a través del cual pueden obtener información sobre el comportamiento de las aplicaciones.

A continuación se muestra un programa de ejemplo, la configuración de gc y la salida del registro verbose-gc:

```
package com.example.so.docs.gc.logging;

import java.util.Arrays;
import java.util.Random;

public class HelloWorld {

    public static void main(String[] args) {
        sortTest();
    }

    private static void sortTest() {
        System.out.println("HelloWorld");

        int count = 3;
        while(count-- > 0) {
            int size = 1024*1024;
            int[] numbers = new int[size];
            Random random = new Random();
            for(int i=0;i<size;i++) {
                numbers[i] = random.nextInt(size);
            }

            Arrays.sort(numbers);
        }
        System.out.println("Done");
```

```
    }


}
```

Opciones de GC:

```
-server -verbose:gc  -XX:+PrintGCDetails -XX:+PrintGCTimeStamps  -Xmx10m  -XX:-
PrintTenuringDistribution  -XX:MaxGCPauseMillis=250 -Xloggc:/path/to/logs/verbose_gc.log
```

Salida:

```
Java HotSpot(TM) 64-Bit Server VM (25.72-b15) for windows-amd64 JRE (1.8.0_72-b15), built on
Dec 22 2015 19:16:16 by "java_re" with MS VC++ 10.0 (VS2010)
Memory: 4k page, physical 6084464k(2584100k free), swap 8130628k(3993460k free)
CommandLine flags: -XX:InitialHeapSize=10485760 -XX:MaxGCPauseMillis=250 -
XX:MaxHeapSize=10485760 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:-
PrintTenuringDistribution -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:-
UseLargePagesIndividualAllocation -XX:+UseParallelGC
0.398: [GC (Allocation Failure) [PSYoungGen: 483K->432K(2560K)] 4579K->4536K(9728K), 0.0012569
secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
0.400: [GC (Allocation Failure) [PSYoungGen: 432K->336K(2560K)] 4536K->4440K(9728K), 0.0008121
secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
0.401: [Full GC (Allocation Failure) [PSYoungGen: 336K->0K(2560K)] [ParOldGen: 4104K-
>294K(5632K)] 4440K->294K(8192K), [Metaspace: 2616K->2616K(1056768K)], 0.0056202 secs] [Times:
user=0.00 sys=0.00, real=0.01 secs]
0.555: [GC (Allocation Failure) [PSYoungGen: 41K->0K(2560K)] 4431K->4390K(9728K), 0.0004678
secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
0.555: [GC (Allocation Failure) [PSYoungGen: 0K->0K(2560K)] 4390K->4390K(9728K), 0.0003490
secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
0.556: [Full GC (Allocation Failure) [PSYoungGen: 0K->0K(2560K)] [ParOldGen: 4390K-
>293K(5632K)] 4390K->293K(8192K), [Metaspace: 2619K->2619K(1056768K)], 0.0060187 secs] [Times:
user=0.00 sys=0.00, real=0.01 secs]
Heap
 PSYoungGen      total 2560K, used 82K [0x00000000ffd00000, 0x0000000100000000,
0x0000000100000000)
  eden space 2048K, 4% used [0x00000000ffd00000,0x00000000ffd14938,0x00000000fff00000)
  from space 512K, 0% used [0x00000000fff80000,0x00000000fff80000,0x0000000100000000)
  to   space 512K, 0% used [0x00000000fff00000,0x00000000fff00000,0x00000000fff80000)
 ParOldGen       total 5632K, used 4389K [0x00000000ff600000, 0x00000000ffb80000,
0x00000000ffd00000)
  object space 5632K, 77% used [0x00000000ff600000,0x00000000ffa49670,0x00000000ffb80000)
 Metaspace       used 2625K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 282K, capacity 386K, committed 512K, reserved 1048576K
```

A continuación se presentan algunos enlaces útiles en GC:

1. Una página archivada que explica los conceptos de gc (jdk7)
2. Tutorial Colector G1
3. Opciones útiles de VM
4. JDK 5 - GC Ergonomics (los conceptos siguen siendo relevantes)
5. Afinación JDK 6 (los conceptos siguen siendo relevantes)

Lea Empezando con la recolección de basura en línea: https://riptutorial.com/es/garbage-collection/topic/8171/empezando-con-la-recoleccion-de-basura

Creditos

S. No	Capítulos	Contributors
1	Empezando con la recolección de basura	Community, Ravindra HV, THelper