垃圾回收知识

笔记本:	我的笔记		
创建时间:	2019/2/25 17:32	**更新时间:**	2019/3/1 9:42
作者:	liuhouer		
标签:	jvm, 垃圾回收		

被判定为垃圾的标准

没有引用的对象

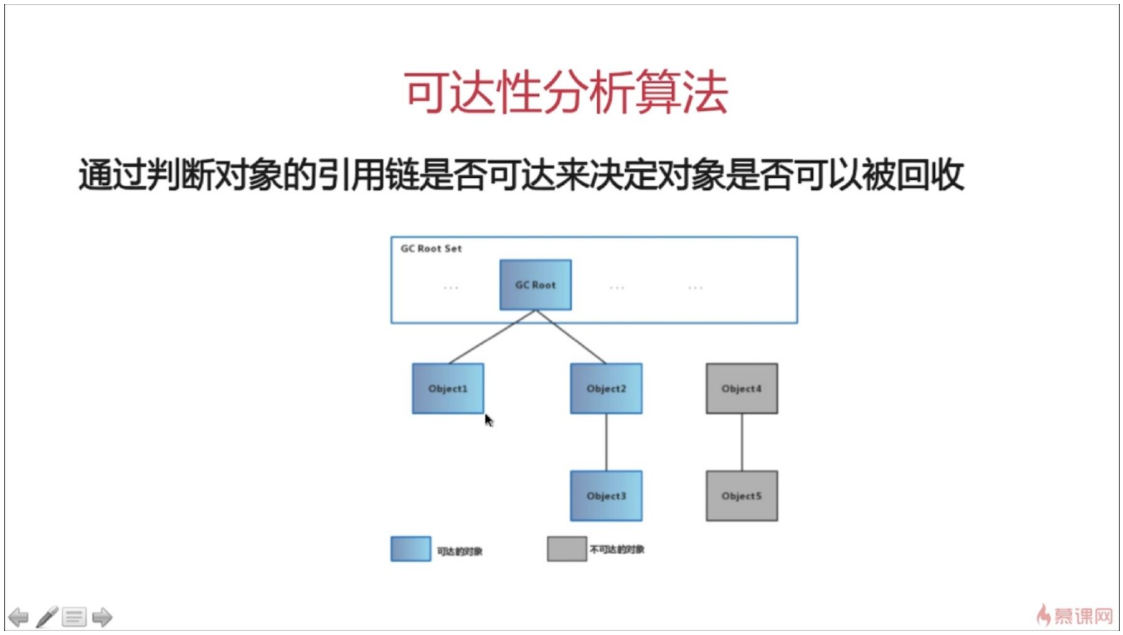
标记垃圾的算法

- 引用计数算法

判断对象的引用数量
➢通过判断对象的引用数量来决定对象是否可以被回收
➢每个对象实例都有一个引用计数器,被引用则+1 ,完成引用则-1
➢任何引用计数为0的对象实例可以被当作垃圾收集

引用计数算法优缺点
➢优点:执行效率高,程序执行受影响较小
➢缺点:无法检测出循环引用的情况,导致内存泄露

- 可达性分析算法（图、树root节点往下遍历）



可以作为GC Root的对象
➢虚拟机栈中引用的对象(栈帧中的本地变量表)
➢方法区中的常量引用的对象
➢方法区中的类静态属性引用的对象

➢本地方法栈中JNI（Native方法)的引用对象
➢活跃线程的引用对象

常见的垃圾回收算法

- 1.标记-清除算法(Mark and Sweep)

➢标记:从根集合进行扫描,对存活的对象进行标记
➢清除:对堆内存从头到尾进行线性遍历,回收不可达对象内存

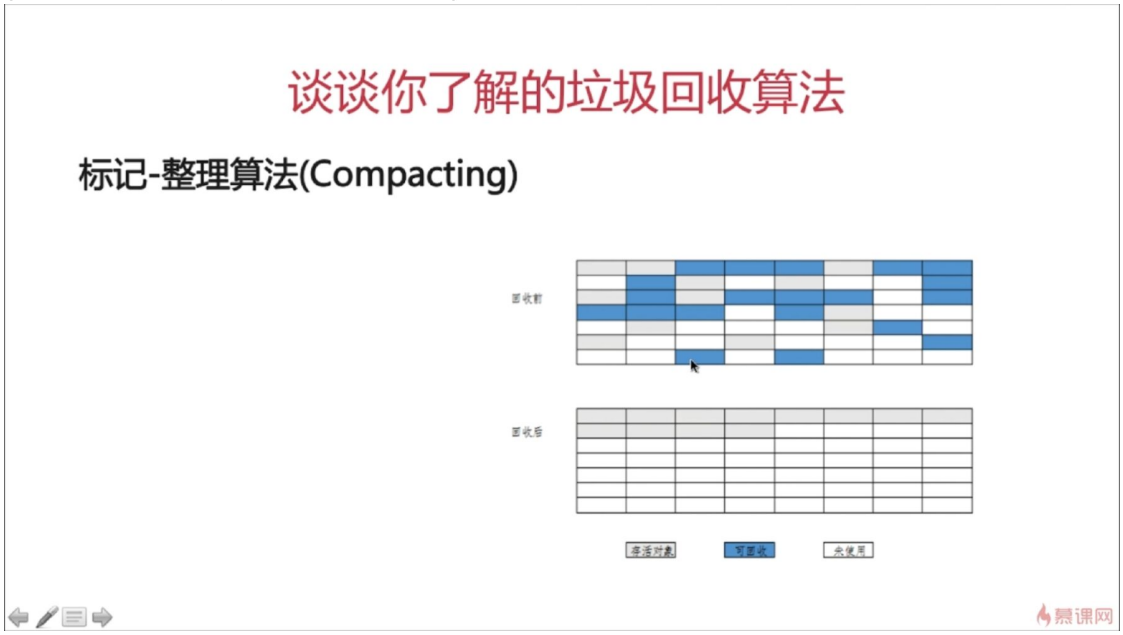
➢缺点：容易造成碎片化

- 2.复制算法(Copying)

➢分为对象面和空闲面
➢对象在对象面上创建
➢存活的对象被从对象面复制到空闲面
➢将对象面所有对象内存清除
➢解决碎片化的问题
➢顺序内存分配，简单高效
➢适用于对象存活率低的场景（新生代Eden）

- 3.标记-整理算法(Compacting)

➢标记:从根集合进行扫描,对存活的对象进行标记
➢清除:移动所有存活的对象,且按照内存地址次序依次排列,然后
将末端内存地址以后的内存全部回收。



- 4.分代收集算法(Generational Collector)（组合使用）

➢垃圾回收算法的组合拳
➢按照对象生命周期的不同划分区域以采用不同的垃圾回收算法
➢目的:提高JVM的回收效率

JDK 6.7 新生代、持久代、老年代
JDK 8　新生代、老年代（移除了持久代）

GC的分类

➢Minor GC（年轻代用复制算法）

年轻代：尽可能快速地收集掉那些生命周期短的对象

- Eden区
- 两个Survivor区

➢Full GC（）

老年代：存放生命周期较长的对象

- 标记-清理算法
- 标记-整理算法

如何从新生代晋升到老年代

➢经历一定Minor次数依然存活的对象
➢Survivor区 中存放不下的对象

常用调优参数

➢-XX:SurvivorRatio：Eden和Survivor的比值,默认8：1
➢-XX:NewRatio :老年代和年轻代内存大小的比例
➢-XX:MaxTenuring Threshold :对象从年轻代晋升到老生代经过GC次数的最大阈值

触发fullGC的条件
➢老年代空间不足
➢永久代空间不足
➢CMS GC时出现promotion failed , concurrent mode failure
➢Minor GC晋升到老年代的平均大小大于老年代的剩余空间
➢调用System.gc()
➢使用RMI来进行RPC或管理的JDK应用,每小时执行1次Full GC

Stop the world
>JVM由于要抗行GC而停止了立用程序的抗行
>任何一神GC算法中都会友生
>多数GC代化通辻减少Stop-the-world友生的吋囘来提高程序性能

Safepoint
➢分析过程中对象引用关系不会发生变化的点
➢产生Safepoint的地方:方法调用;循环跳转;异常跳转等
➢安全点数量得适中

常见的垃圾收集器
1.Serial【串行】收集器(-XX:+UseSerialGC ,复制算法)
➢单线程收集,进行垃圾收集时,必须暂停所有工作线程
➢简单高效, Client模式下默认的年轻代收集器

2.ParNew收集器(-XX:+UseParNewGC ,复制算法)
➢多线程收集,其余的行为、特点和Serial收集器一 样
➢单核执行效率不如Serial ,在多核下执行才有优势

3.Parallel Scavenge收集器(-XX:+UseParallelGC ,复制算法)
➢比起关注用户线程停顿时间,更关注系统的吞吐量
➢在多核下执行才有优势, Server模式下默认的年轻代收集器

JVM运行模式（server和client)

年轻代垃圾收集器

老年代垃圾收集器

垃圾收集器之间的关系

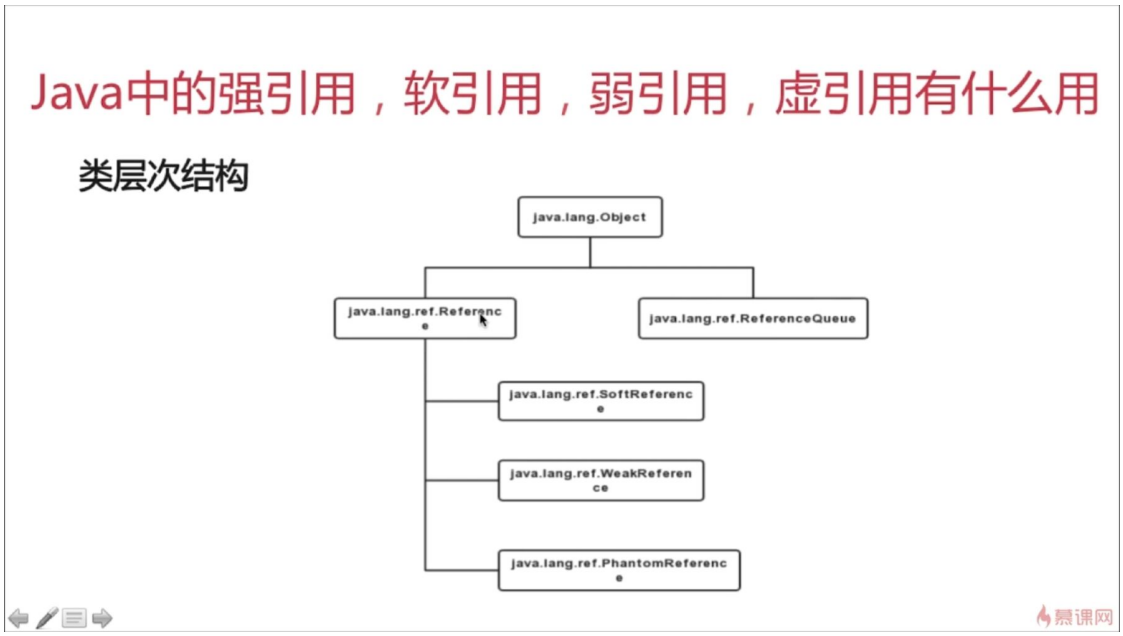
finalize()方法
给与对象一次重生的机会

```
public class Finalization {
    public static Finalization finalization;
    @Override
    protected void finalize(){
        System.out.println("Finalized");
```

```
        finalization = this;
    }

    public static void main(String[] args) {
        Finalization f = new Finalization();
        System.out.println("First print: " + f);
        f = null;
        System.gc();
        try {// 休息一段时间，让上面的垃圾回收线程执行完成
            Thread.currentThread().sleep(1000);
        } catch (InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("Second print: " + f);
        System.out.println(f.finalization);
    }
}
```

Java的4种引用



ReferenceQuene

JVM垃圾收集器

JVM垃圾收集器发展历程

第一阶段，Serial（串行）收集器

在jdk1.3.1之前，java虚拟机仅仅能使用Serial收集器。 Serial收集器是一个单线程的收集器，但它的“单线程”的意义并不仅仅是说明它只会使用一个CPU或一条收集线程去完成垃圾收集工作，更重要的是在它进行垃圾收集时，必须暂停其他所有的工作线程，直到它收集结束。

第二阶段，Parallel（并行）收集器

Parallel收集器也称吞吐量收集器，相比Serial收集器，Parallel最主要的优势在于使用多线程去完成垃圾清理工作，这样可以充分利用多核的特性，大幅降低gc时间。

第三阶段，CMS（并发）收集器

CMS收集器在Minor GC时会暂停所有的应用线程，并以多线程的方式进行垃圾回收。在Full GC时不再暂停应用线程，而是使用若干个后台线程定期的对老年代空间进行扫描，及时回收其中不再使用的对象。

第四阶段，G1（并发）收集器

G1收集器（或者垃圾优先收集器）的设计初衷是为了尽量缩短处理超大堆（大于4GB）时产生的停顿。相对于CMS的优势而言是内存碎片的产生率大大降低。

年轻代的垃圾回收

1.新生代

Serial (第一代)

PraNew (第二代)

Parallel Scavenge (第三代)

G1收集器(第四代)

老年代的垃圾回收

2.老年代

Serial Old (第一代)

Parallel Old (第二代)

CMS (第三代)

G1收集器 (第四代)

G1收集器概述

从JDK(1.3)开始，HotSpot团队一直努力朝着高效收集、减少停顿(STW: Stop The World)的方向努力，也贡献了从串行Serial收集器、到并行收集器Parallerl收集器，再到CMS并发收集器，乃至如今的G1在内的一系列优秀的垃圾收集器。

G1(Garbage First)垃圾收集器是当今垃圾回收技术最前沿的成果之一。早在JDK7就已加入JVM的收集器大家庭中，成为HotSpot重点发展的垃圾回收技术。同优秀的CMS垃圾回收器一样，G1也是关注最小时延的垃圾回收器，也同样适合大尺寸堆内存的垃圾收集，官方也推荐使用G1来代替选择CMS。

1.G1收集器的最大特点

G1最大的特点是引入分区的思路，弱化了分代的概念。

合理利用垃圾收集各个周期的资源，解决了其他收集器甚至CMS的众多缺陷。

2.G1相比较CMS的改进

算法： G1基于标记-整理算法, 不会产生空间碎片，分配大对象时不会无法得到连续的空间而提前触发一次FULL GC。

停顿时间可控： G1可以通过设置预期停顿时间（Pause Time）来控制垃圾收集时间避免应用雪崩现象。

并行与并发：G1能更充分的利用CPU，多核环境下的硬件优势来缩短stop the world的停顿时间。

3.CMS和G1的区别

CMS中，堆被分为PermGen，YoungGen，OldGen；而YoungGen又分了两个survivo区域。

在G1中，堆被平均分成几个区域(region)，在每个区域中，虽然也保留了新老代的概念，但是收集器是以整个区域为单位收集的。

G1在回收内存后会马上同时做合并空闲内存的工作、而CMS默认是在STW（stop the world）的时候做。

G1会在Young GC中使用、而CMS只能在OldGen区使用。

4.G1收集器的应用场景

G1垃圾收集算法主要应用在多CPU大内存的服务中，在满足高吞吐量的同时，尽可能的满足垃圾回收时的暂停时间。

就目前而言、CMS还是默认首选的GC策略、可能在以下场景下G1更适合：

服务端多核CPU、JVM内存占用较大的应用（至少大于4G）

应用在运行过程中会产生大量内存碎片、需要经常压缩空间

想要更可控、可预期的GC停顿周期，防止高并发下应用雪崩现象

1.G1之前的JVM内存模型

新生代：伊甸园区(eden space) + 2个幸存区

老年代

持久代(perm space)：JDK1.8之前

元空间(metaspace)：JDK1.8之后取代持久代