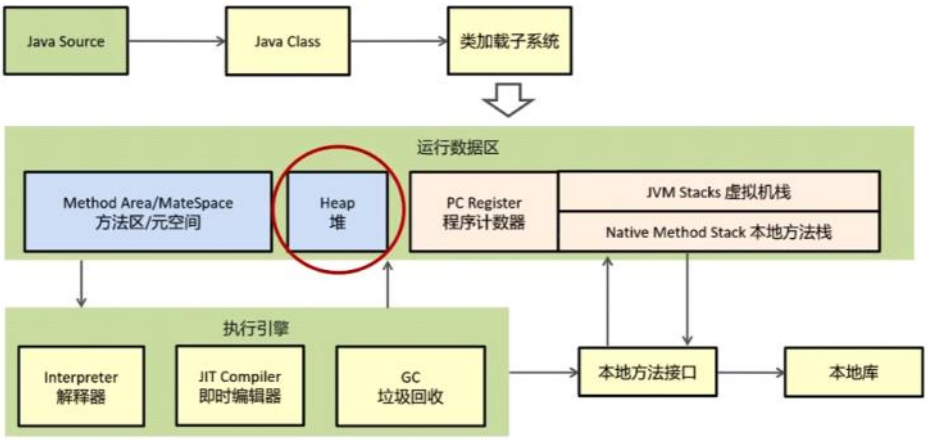
垃圾回收、强软弱虚引用

2024年4月1日 17:35

对象什么时候可以被垃圾器回收

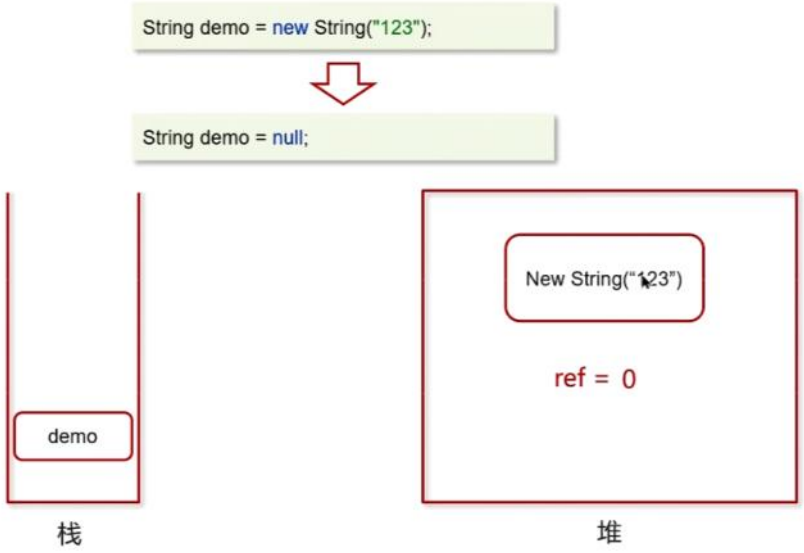


简单一句就是：如果一个或多个对象没有任何的引用指向它了，那么这个对象现在就是垃圾，如果定位了垃圾，则有可能会被垃圾回收器回收。

如果要定位什么是垃圾，有两种方式来确定，第一个是引用计数法，第二个是可达性分析算法

引用计数法

一个对象被引用了一次，在当前的对象头上递增一次引用次数，如果这个对象的引用次数为0，代表这个对象可回收



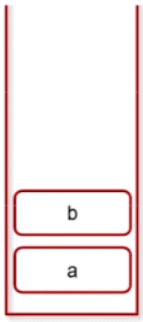
引用计数法

当对象间出现了循环引用的话，则引用计数法就会失效

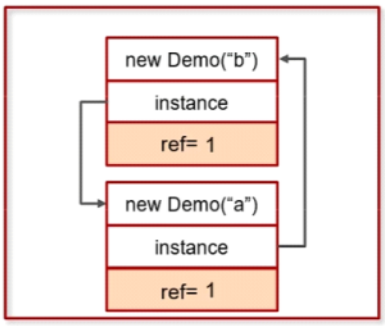
```
public class Demo {
   Demo instance;
   String name;
   public Demo(String name){
      this.name = name;
   }
}
```

```
Demo a = new Demo("a");
Demo b = new Demo("b");
a.instance = b;
b.instance = a;

a = null;
b = null;
```



栈



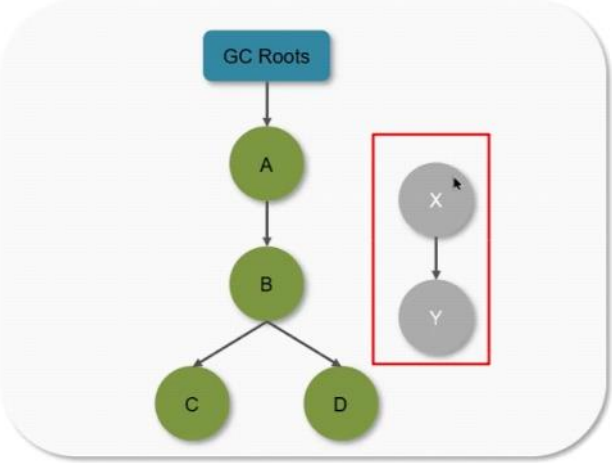
堆

循环引用，会引发内存泄露

高级软件人才培训专家

可达性分析算法

现在的虚拟机采用的都是通过可达性分析算法来确定哪些内容是垃圾。



X,Y这两个节点是可回收的

- Java 虚拟机中的垃圾回收器采用可达性分析来探索所有存活的对象
- 扫描堆中的对象，看是否能够沿着 GC Root 对象 为起点的引用链找到该对象，找不到，表示可以回收

哪些对象可以作为 GC Root？

- 虚拟机栈（栈帧中的本地变量表）中引用的对象
- 方法区中类静态属性引用的对象
- 方法区中常量引用的对象
- 本地方法栈中 JNI（即一般说的 Native 方法）引用的对象

```
public static  void main(String[] args) {
    Demo demo = new Demo();
    demo = null;
}
```

```
public static Demo a;
public static  void main(String[] args) {
    Demo b = new Demo();
    b.a = new Demo();
    b = null;
}
```

```
public static final Demo a = new Demo();
public static  void main(String[] args) {
    Demo demo = new Demo();
    demo = null;
}
```

对象什么时候可以被垃圾器回收

如果一个或多个对象没有任何的引用指向它了，那么这个对象现在就是垃圾，
如果定位了垃圾，则有可能会被垃圾回收器回收。

定位垃圾的方式有两种

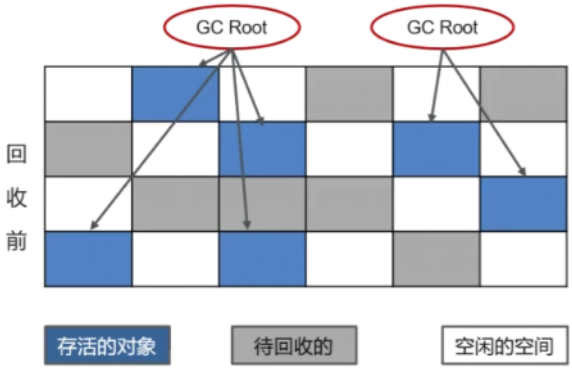
- 引用计数法
- 可达性分析算法

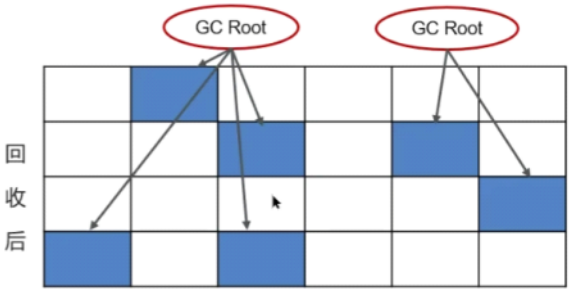
标记清除算法

标记清除算法，是将垃圾回收分为2个阶段，分别是标记和清除。

1.根据可达性分析算法得出的垃圾进行标记

2.对这些标记为可回收的内容进行垃圾回收

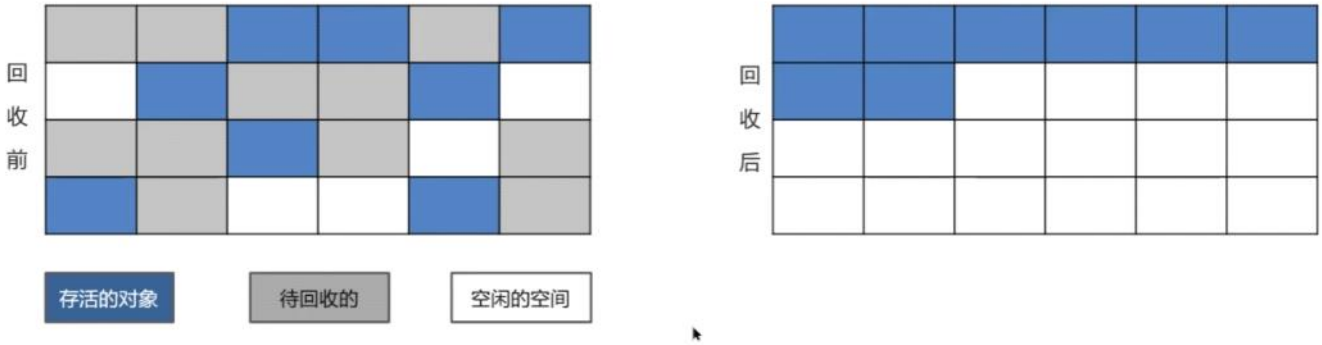




优点：标记和清除速度较快

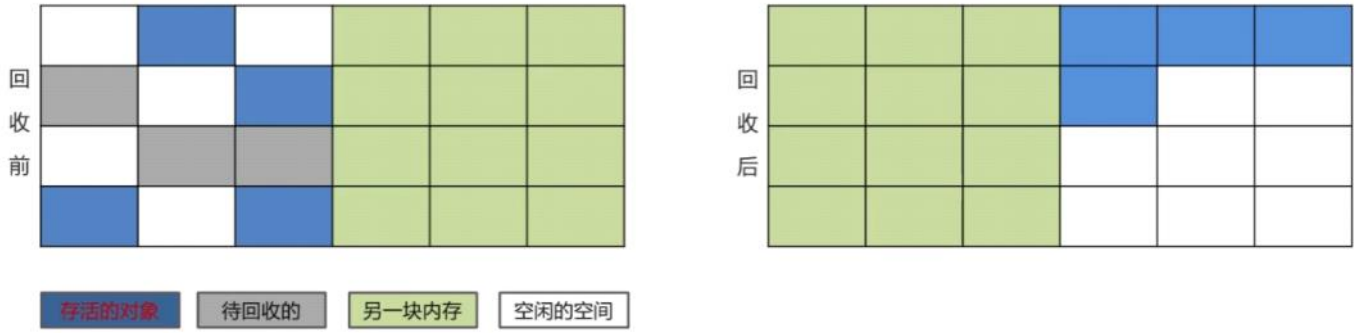
缺点：碎片化较为严重，内存不连贯的

标记整理算法



优缺点同标记清除算法，解决了标记清除算法的碎片化的问题，同时，标记压缩算法多了一步，对象移动内存位置的步骤，其效率也有有一定的影响。

复制算法



优点：

- 在垃圾对象多的情况下，效率较高
- 清理后，内存无碎片

缺点：

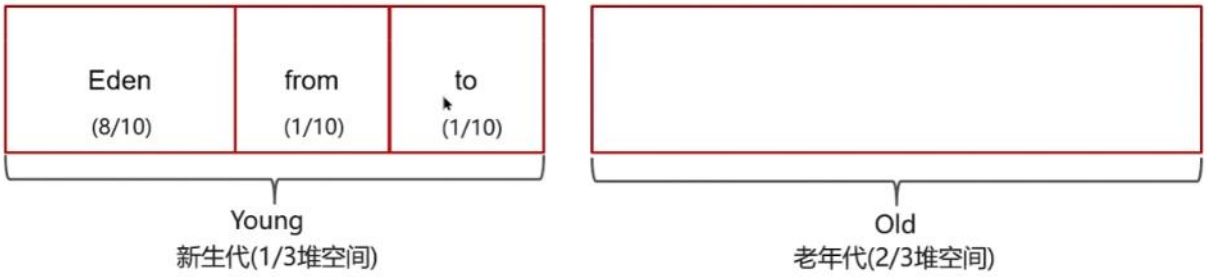
- 分配的2块内存空间，在同一个时刻，只能使用一半，内存使用率较低

JVM 垃圾回收算法有哪些?

- 标记清除算法：垃圾回收分为2个阶段，分别是标记和清除,效率高,有磁盘碎片，内存不连续
- 标记整理算法：标记清除算法一样，将存活对象都向内存另一端移动，然后清理边界以外的垃圾，无碎片，对象需要移动，效率低
- 复制算法：将原有的内存空间一分为二，每次只用其中的一块,正在使用的对象复制到另一个内存空间中，然后将该内存空间清空，交换两个内存的角色，完成垃圾的回收;无碎片，内存使用率低

分代收集算法

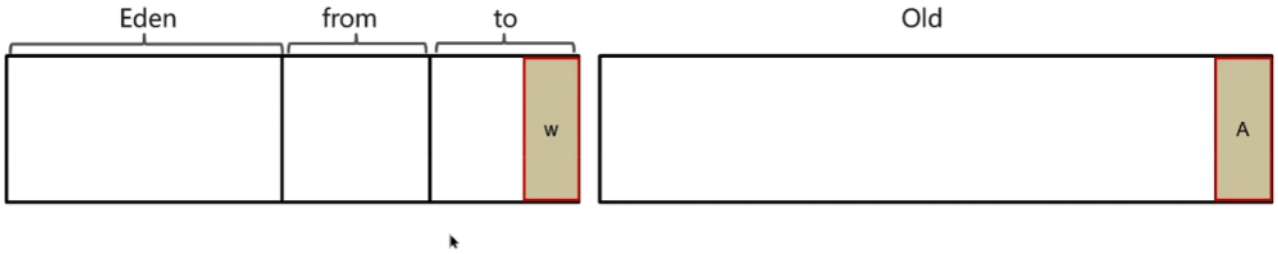
在java8时，堆被分为了两份：新生代和老年代【1：2】



对于新生代，内部又被分为了三个区域。

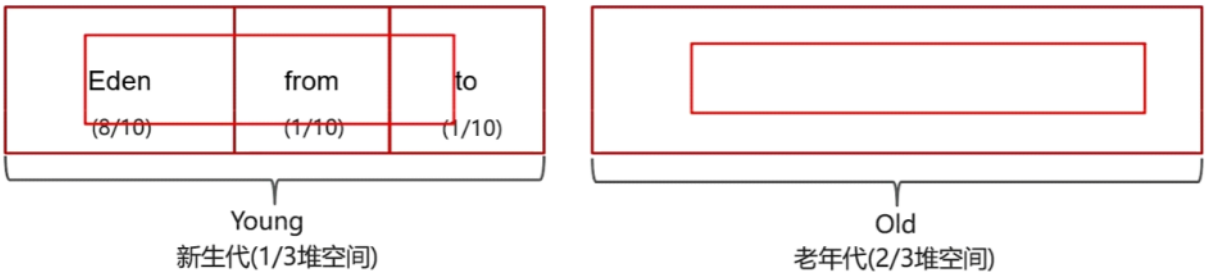
- 伊甸园区Eden，新生的对象都分配到这里
- 幸存者区survivor(分成from和to)
- Eden区，from区，to区【8：1：1】

分代收集算法-工作机制



- 新创建的对象，都会先分配到eden区
- 当伊甸园内存不足，标记伊甸园与 from（现阶段没有）的存活对象
- 将存活对象采用复制算法复制到 to 中，复制完毕后，伊甸园和 from 内存都得到释放
- 经过一段时间后伊甸园的内存又出现不足，标记eden区域to区存活的对象，将存活的对象复制到from区
- 当幸存区对象熬过几次回收（最多15次），晋升到老年代（幸存区内存不足或大对象会导致提前晋升）

MinorGC、 Mixed GC 、 FullGC的区别是什么



- MinorGC【young GC】发生在新生代的垃圾回收，暂停时间短（STW）
- Mixed GC 新生代 + 老年代部分区域的垃圾回收，G1 收集器特有
- FullGC：新生代 + 老年代完整垃圾回收，暂停时间长（STW），应尽力避免

名词解释

STW（Stop-The-World）：暂停所有应用程序线程，等待垃圾回收的完成

说一下JVM中的分代回收

一、堆的区域划分

1. 堆被分为了两份：新生代和老年代【1：2】
2. 对于新生代，内部又被分为了三个区域。Eden区，幸存者区survivor(分成from和to)【8：1：1】

二、对象回收分代回收策略

1. 新创建的对象，都会先分配到eden区
2. 当伊甸园内存不足，标记伊甸园与 from（现阶段没有）的存活对象
3. 将存活对象采用复制算法复制到to中，复制完毕后，伊甸园和 from 内存都得到释放
4. 经过一段时间后伊甸园的内存又出现不足，标记eden区域to区存活的对象，将其复制到from区
5. 当幸存区对象熬过几次回收（最多15次），晋升到老年代（幸存区内存不足或大对象会提前晋升）

MinorGC、 Mixed GC 、 FullGC的区别是什么

- MinorGC【young GC】发生在新生代的垃圾回收，暂停时间短（STW）
- Mixed GC 新生代 + 老年代部分区域的垃圾回收，G1 收集器特有
- FullGC： 新生代 + 老年代完整垃圾回收，暂停时间长（STW），应尽力避免

说一下 JVM 有哪些垃圾回收器?

在jvm中，实现了多种垃圾收集器，包括：

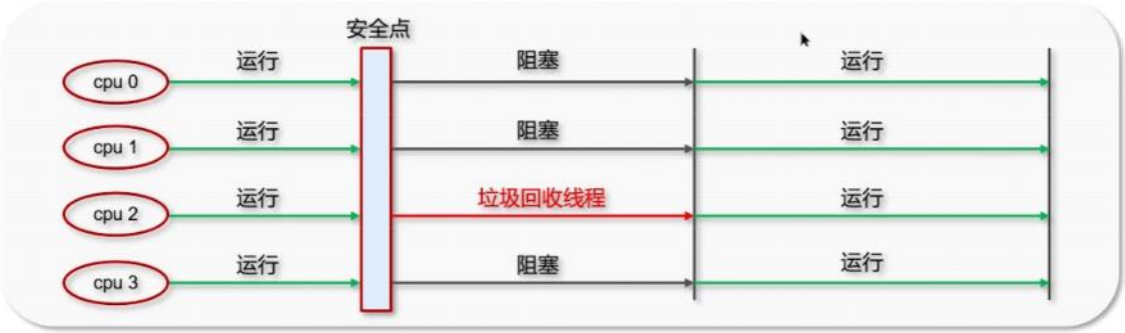
- 串行垃圾收集器
- 并行垃圾收集器
- CMS（并发）垃圾收集器
- G1垃圾收集器

串行垃圾收集器

Serial和Serial Old串行垃圾收集器，是指使用单线程进行垃圾回收，堆内存较小，适合个人电脑

- Serial 作用于新生代，采用复制算法
- Serial Old 作用于老年代，采用标记-整理算法

垃圾回收时，只有一个线程在工作，并且java应用中的所有线程都要暂停（STW），等待垃圾回收的完成。



并行垃圾收集器

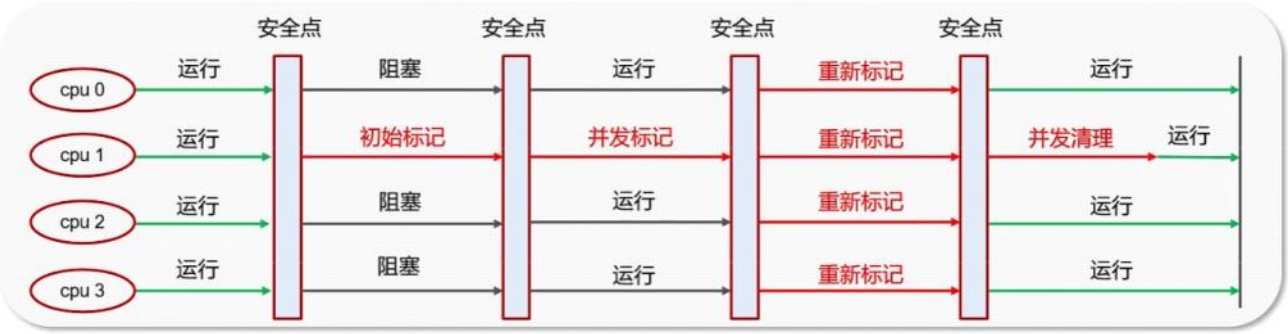
Parallel New和Parallel Old是一个并行垃圾回收器，**JDK8默认使用此垃圾回收器**

- Parallel New作用于新生代，采用复制算法
- Parallel Old作用于老年代，采用标记-整理算法

垃圾回收时，多个线程在工作，并且java应用中的所有线程都要暂停（STW），等待垃圾回收的完成。

CMS（并发）垃圾收集器

CMS全称 Concurrent Mark Sweep，是一款并发的、使用标记-清除算法的垃圾回收器，该回收器是针对老年代垃圾回收的，是一款以获取最短回收停顿时间为目标的收集器，停顿时间短，用户体验就好。其最大特点是在进行垃圾回收时，应用仍然能正常运行。



说一下JVM有哪些垃圾回收器?

在jvm中，实现了多种垃圾收集器，包括：

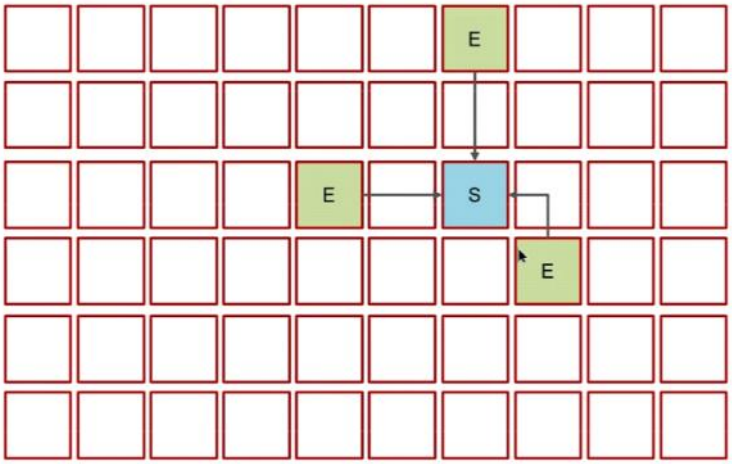
- 串行垃圾收集器：Serial GC、Serial Old GC
- 并行垃圾收集器：Parallel Old GC、ParNew GC
- CMS（并发）垃圾收集器：CMS GC，作用在老年代
- G1垃圾收集器，作用在新生代和老年代

G1垃圾收集器

- 应用于新生代和老年代，**在JDK9之后默认使用G1**
- 划分成多个区域，每个区域都可以充当 eden，survivor，old， humongous，其中 humongous 专为大对象准备
- 采用复制算法
- 响应时间与吞吐量兼顾
- 分成三个阶段：新生代回收、并发标记、混合收集
- 如果并发失败（即回收速度赶不上创建新对象速度），会触发 Full GC

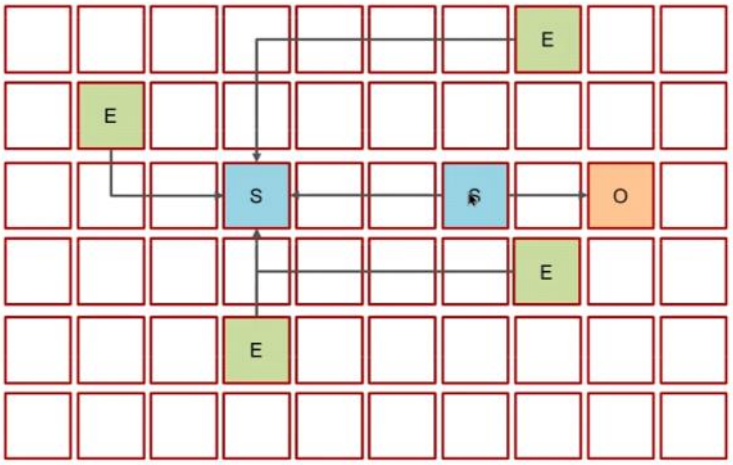
Young Collection(年轻代垃圾回收)

- 初始时，所有区域都处于空闲状态
- 创建了一些对象，挑出一些空闲区域作为伊甸园区存储这些对象
- 当伊甸园需要垃圾回收时，挑出一个空闲区域作为幸存区，用复制算法复制存活对象，需要暂停用户线程



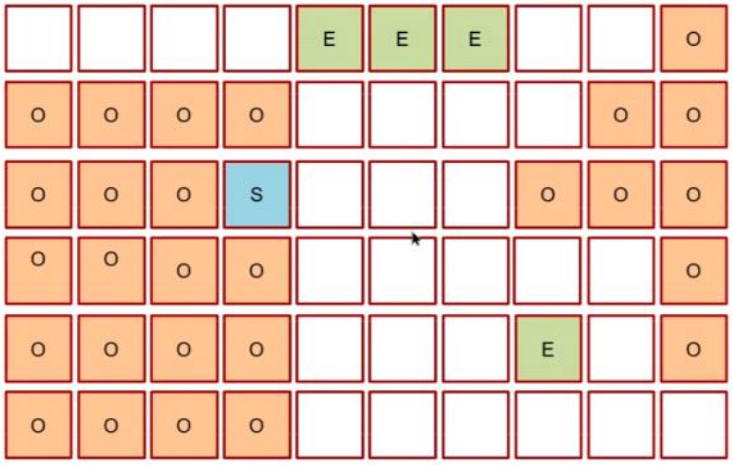
Young Collection(年轻代垃圾回收)

- 随着时间流逝，伊甸园的内存又有不足
- 将伊甸园以及之前幸存区中的存活对象，采用复制算法，复制到新的幸存区，其中较老对象晋升至老年代



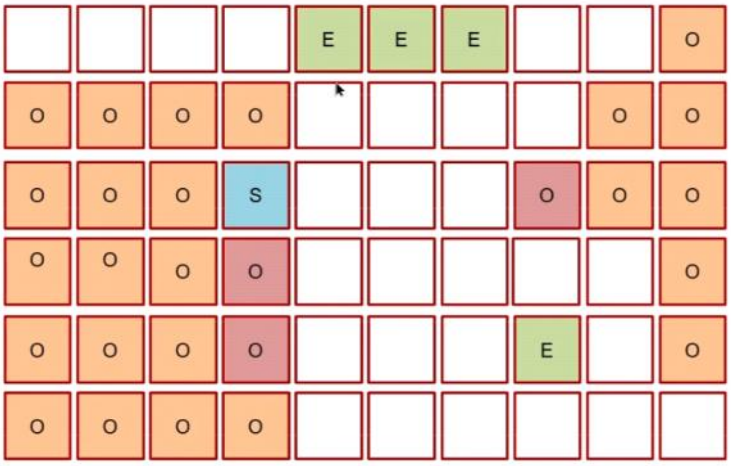
Young Collection + Concurrent Mark (年轻代垃圾回收+并发标记)

当老年代占用内存超过阈值(默认是45%)后，触发并发标记，这时无需暂停用户线程



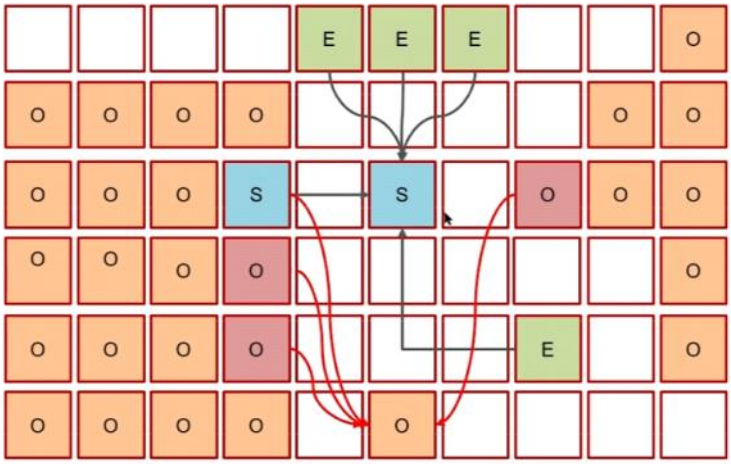
Young Collection + Concurrent Mark (年轻代垃圾回收+并发标记)

- 并发标记之后，会有重新标记阶段解决漏标问题，此时需要暂停用户线程。
- 这些都完成后就知道了老年代有哪些存活对象，随后进入混合收集阶段。此时不会对所有老年代区域进行回收，而是根据暂停时间目标优先回收价值高（存活对象少）的区域（这也是 Gabage First 名称的由来）。



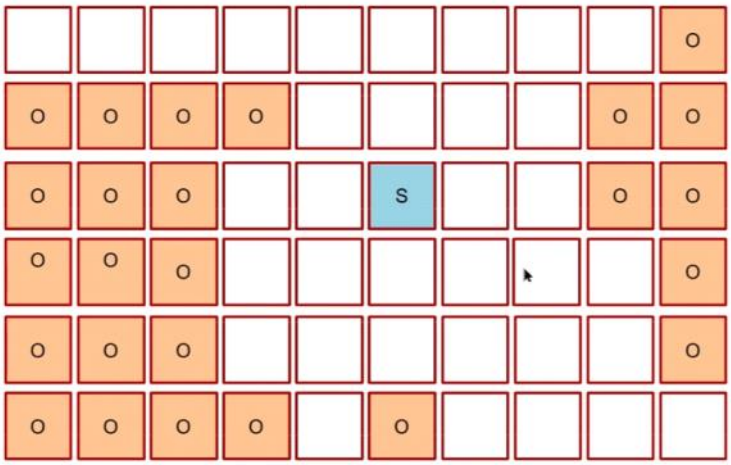
Mixed Collection (混合垃圾回收)

混合收集阶段中，参与复制的有 eden、survivor、old



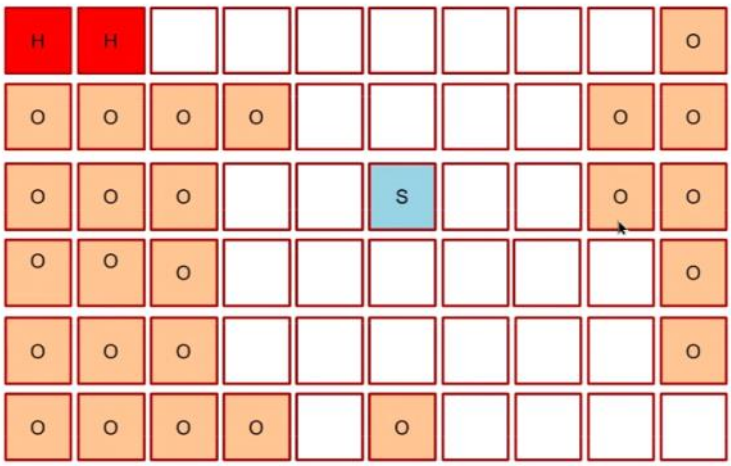
Mixed Collection (混合垃圾回收)

复制完成，内存得到释放。进入下一轮的新生代回收、并发标记、混合收集



Mixed Collection (混合垃圾回收)

复制完成，内存得到释放。进入下一轮的新生代回收、并发标记、混合收集



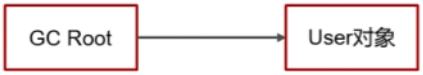
详细聊一下G1垃圾回收器

- 应用于新生代和老年代，**在JDK9之后默认使用G1**
- 划分成多个区域，每个区域都可以充当 eden，survivor，old， humongous ，其中 humongous 专为大对象准备
- 采用复制算法
- 响应时间与吞吐量兼顾
- 分成三个阶段：新生代回收(stw)、并发标记(重新标记stw)、混合收集
- 如果并发失败（即回收速度赶不上创建新对象速度），会触发 Full GC

强引用、软引用、弱引用、虚引用的区别

- 强引用：只有所有 GC Roots 对象都不通过【强引用】引用该对象，该对象才能被垃圾回收

```
User user = new User();
```



- 软引用：仅有软引用引用该对象时，在垃圾回收后，内存仍不足时会再次出发垃圾回收

```
User user = new User();
SoftReference softReference = new SoftReference(user);
```

强引用、软引用、弱引用、虚引用的区别

弱引用：仅有弱引用引用该对象时，在垃圾回收时，无论内存是否充足，都会回收弱引用对象

```
User user = new User();
WeakReference weakReference = new WeakReference(user);
```

延伸话题：ThreadLocal内存泄漏问题

```
static class Entry extends WeakReference<ThreadLocal<?>> {
    Object value;

    Entry(ThreadLocal<?> k, Object v) {
        super(k);
        value = v; //强引用，不会被回收
      }
}
```

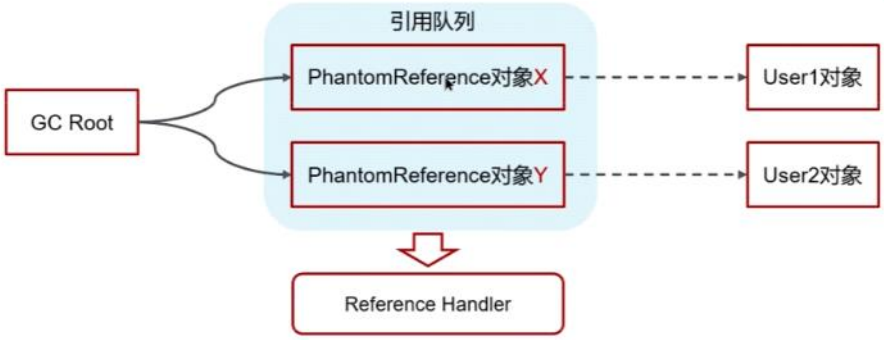
强引用、软引用、弱引用、虚引用的区别

虚引用：必须配合引用队列使用，被引用对象回收时，会将虚引用入队，由 Reference Handler 线程调用虚引用相关方法释放直接内存

```
User user = new User();
ReferenceQueue referenceQueue = new ReferenceQueue();
PhantomReference phantomReference = new PhantomReference(user,queue);
```

软引用

弱引用



强引用、软引用、弱引用、虚引用的区别?

- 强引用：只要所有 GC Roots 能找到，就不会被回收
- 软引用：需要配合SoftReference使用，当垃圾多次回收，内存依然不够的时候会回收软引用对象
- 弱引用：需要配合WeakReference使用，只要进行了垃圾回收，就会把弱引用对象回收
- 虚引用：必须配合引用队列使用，被引用对象回收时，会将虚引用入队，由 Reference Handler 线程调用虚引用相关方法释放直接内存