# 关于垃圾收集

垃圾收集就是GC

1. Java 和 C++语言的区别，就在于垃圾收集技术和内存动态分配上，C语言没有垃圾收集技术，需要我们手动的收集。
2. 垃圾收集，不是Java语言的伴生产物。早在1960年，第一门开始使用内存动态分配和垃圾收集技术的Lisp语言诞生。
3. 关于垃圾收集有三个经典问题
   1. 哪些内存需要回收？
   2. 什么时候回收？
   3. 如何回收？
4. 垃圾收集机制是Java的招牌能力，极大地提高了开发效率。如今，垃圾收集几乎成为现代语言的标配，即使经过如此长时间的发展，Java的垃圾收集机制仍然在不断的演进中，不同大小的设备、不同特征的应用场景，对垃圾收集提出了新的挑战，这当然也是面试的热点。

## 瞅瞅大厂面试题

### 蚂蚁金服

1. 你知道哪几种垃圾回收器，各自的优缺点，重点讲一下CMS和G1？
2. JVM GC算法有哪些，目前的JDK版本采用什么回收算法？
3. G1回收器讲下回收过程GC是什么？为什么要有GC？
4. GC的两种判定方法？CMS收集器与G1收集器的特点

### 百度

1. 说一下GC算法，分代回收说下
2. 垃圾收集策略和算法

### 天猫

1. JVM GC原理，JVM怎么回收内存
2. CMS特点，垃圾回收算法有哪些？各自的优缺点，他们共同的缺点是什么？

### 滴滴

1. Java的垃圾回收器都有哪些，说下G1的应用场景，平时你是如何搭配使用垃圾回收器的

### 京东

1. 你知道哪几种垃圾收集器，各自的优缺点，重点讲下CMS和G1，
2. 包括原理，流程，优缺点。垃圾回收算法的实现原理

### 阿里

1. 讲一讲垃圾回收算法。
2. 什么情况下触发垃圾回收？
3. 如何选择合适的垃圾收集算法？
4. JVM有哪三种垃圾回收器？

### 字节跳动

1. 常见的垃圾回收器算法有哪些，各有什么优劣？
2. System.gc()和Runtime.gc()会做什么事情？
3. Java GC机制？GC Roots有哪些？
4. Java对象的回收方式，回收算法。
5. CMS和G1了解么，CMS解决什么问题，说一下回收的过程。
6. CMS回收停顿了几次，为什么要停顿两次?

# 什么是垃圾

**垃圾是指在运行程序中没有任何指针指向的对象，这个对象就是需要被回收的垃圾。**外文： An object is considered garbage when it can no longer be reached from any pointer in the runningprogram.  
 如果不及时对内存中的垃圾进行清理，那么，这些垃圾对象所占的内存空间会一直保留到应用程序结束，被保留的空间无法被其他对象使用。甚至可能导致内存溢出。

# 为什么需要GC

1. **回收垃圾。**一个基本认知是如果不进行垃圾回收，内存迟早都会被消耗完，因为不断地分配内存空间而不进行回收，就好像不停地生产生活垃圾而从来不打扫一样。
2. **除了释放没用的对象，垃圾回收也可以清除内存里的记录碎片。**碎片整理将所占用的堆内存移到堆的一端，以便JVM 将整理出的内存分配给新的对象。
3. 随着应用程序所应付的业务越来越庞大、复杂，用户越来越多，**没有GC就不能保证应用程序的正常进行。**而经常造成STW（Stop-The-World）的GC又跟不上实际的需求，所以才会不断地尝试对GC进行优化。

# 早期垃圾回收行为

1. **在早期的C/C++时代，垃圾回收基本.上是手工进行的**。开发人员可以使用 new关键字进行内存申请，并使用delete关键字进行内存释放。比如以下代码

|  |
| --- |
| MibBridge \*pBridge= new cmBaseGroupBridge（）；  //如果注册失败，使用Delete释放该对象所占内存区域  if（pBridge->Register（kDestroy）！=NO ERROR）  delete pBridge； |

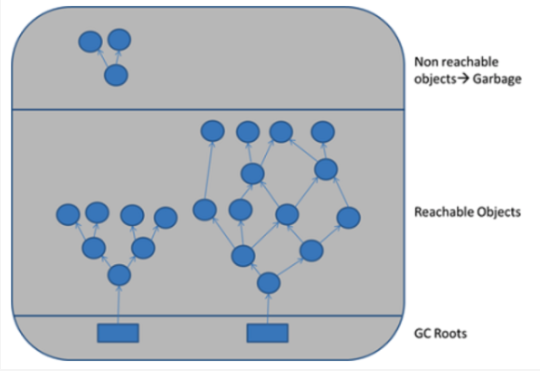
1. 这种方式可以灵活控制内存释放的时间，但是会给开发人员带来频繁申请和释放内存的管理负担。倘若有一处内存区间由于程序员编码的问题忘记被回收，那么就会产生**内存泄漏**，垃圾对象永远无法被清除，随着系统运行时间的不断增长，垃圾对象所耗内存可能持续上升，直到出现内存溢出并造成应用程序崩溃。
2. 在有了垃圾回收机制后，上述代码块极有可能变成这样：

|  |
| --- |
| MibBridge \*pBridge = new cmBaseGroupBridge()；  pBridge 一> Register（kDestroy）； |

现在，除了Java以外，C#、Python、Ruby等语言都使用了自动垃圾回收的思想，也是未来发展趋势，可以说这种自动化的内存分配和来及回收方式已经成为了线代开发语言必备的标准。

## 什么是内存泄漏？

上图：



解释一下这张图，每个Android（或Java）应用程序都有一个起点（GC Root），从这个点中实例化对象、调用方法。一些对象直接引用GC Root，这些对象又引用了另一些对象。因此，形成了引用链，就像上图一样。因此垃圾收集器从GC Root开始并遍历直接或间接链接到GC Root的对象。在此过程结束时，脱离GC Root的对象/对象链将被回收。

有了上图，理解内存泄漏的概念就很简单，说白了就是：**长生命周期对象A持有了短生命周期的对象B，那么只要A不脱离GC Root的链，那么B对象永远没有可能被回收，因此B就泄漏了。**

# Java的垃圾回收机制

oracle官 网关于垃圾回收的介绍

https：[//docs.oracle.com/javase/8/docs/](https://docs.oracle.com/javase/8/docs/" \t "https://www.cnblogs.com/yanl55555/p/_blank)technotes/guides/vm/gctuning/toc.html

## 自动内存管理

内存管理： 分配和回收

**优点：**

1. 自动内存管理，无需开发人员手动参与内存的分配与回收，这样降低内存泄漏和内存溢出的风险
2. 没有垃圾回收器，java也会和cpp一样，各种悬垂指针，野指针，泄露问题让你头疼不已。
3. 自动内存管理机制，将程序员从繁重的内存管理中释放出来，可以更专心地专注于业务开发

**担忧：**

对于Java开发人员而言，自动内存管理就像是一个黑匣子，如果过度依赖于 “自动”，那么这将会是一场灾难，最严重的就会弱化Java开发人员在程序出现内存溢出时定位问题和解决问题的能力。

此时，了解JVM的自动内存分配和内存回收原理就显得非常重要，只有在真正了解JVM是如何管理内存后，我们才**能够在遇见OutOfMemoryError时，快速地根据错误异常日志定位问题和解决问题。**

当需要排查各种内存溢出、内存泄漏问题时，当垃圾收集成为系统达到更高并发量的瓶颈时，我们就必须对这些“自动化”的技术实施必要的监控和调节。

## 垃圾回收区域

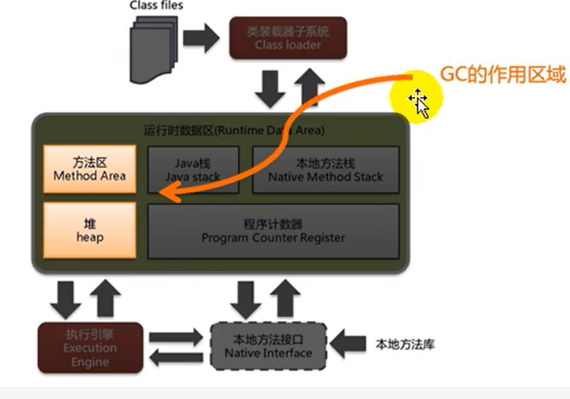
垃圾回收器可以对年轻代回收，也可以对老年代回收，甚至是全堆和方法区的回收。其中，Java堆是垃圾收集器的工作重点。

* 从次数上讲：

**频繁收集Young区**

**较少收集old区**

**基本不动Perm区(方法区/元空间)**



JVM运行时数据区中，程序计数器、虚拟机栈、本地方法栈这三个区域都是线程私有的，也就是说随线程生而生，死而死，因此这三个区域不用考虑如何回收，当方法结束或者线程结束，内存自然就跟着回收了。

堆和方法区具有不确定性，只有处于运行期间，我们才知道程序究竟会创建哪些对象，创建多少个对象，这部分的内存的分配和回收都是动态的，因此需要考虑如何回收。

