JVM内存模型

```
堆区            ：方法
    线程共享
栈区            ：引用
    线程私有
    生命周期与线程相同
    一个线程对应一个java栈
    每执行一个方法就会往栈中压入一个元素，这个元素叫“栈帧”，而栈帧中包括了方法中的局部变量、用于存放中间状态值的操作栈
    StackOverflowError
方法区
    线程共享
    类信息，常量，静态变量
本地方法栈      ：
    与java栈类似，只不过它是用来表示执行本地方法的，本地方法栈存放的方法调用本地方法接口，最终调用本地方法库，实现与操作系统、硬件交互的目的
PC寄存器
```

垃圾回收机制

```
Garbage Collection（GC）
    由JRE的一个线程对堆内存进行监控和回收
    程序员无法精确控制垃圾回收的时间和顺序
    垃圾信息：没有引用指向的对象
    性能开销（必须跟踪有用的对象）



特点：
    只能回收内存资源（不能回收数据库连接和IO等资源）
    为了更快地让垃圾回收机制回收那些不在使用的资源，将其设置为null
    垃圾回收发生的不可预知性（CPU空闲或内存不足时）
        Runtime.getRuntime().gc()
        System.finalize ()
        只是建议，不能精确控制
    在垃圾回收前总会先调用finalize()方法。可能使之复活
```

几种垃圾回收器

```
1,Serial（连续的）收集器
    单线程的收集器。进行垃圾收集时，必须暂停其他所有的工作线程，直到它收集完成。
    在Client模式下默认新生代收集器
2,ParNew收集器
    Serial收集器的多线程版本
    目前只有ParNew它能与CMS收集器配合工作。
3,Parallel Scavenge（并行回收）收集器
```

新生代收集器，它也是使用复制算法的收集器，又是并行的多线程收集器
4,Serial Old 收集器
Serial的老年代版本
5,Parallel Old 收集器
Parallel的老年代版本
6,CMS收集器
以获取最短回收停顿时间为目标的收集器。
这类应用尤其重视服务器的响应速度，希望系统停顿时间最短，以给用户带来较好的体验。
CMS收集器是基于“标记-清除”算法实现的。

整个过程分为4个步骤：
（1）初始标记
（2）并发标记
（3）重新标记
（4）并发清除

其中，初始标记、重新标记这两个步骤仍然需要“Stop The World”.

优点：并发收集，低停顿。
缺点：
（1）CMS收集器对CPU资源非常敏感。
CPU个数少于4个时，CMS对于用户程序的影响就可能变得很大，为了应付这种情况，虚拟机提供了一种称为“增量式并发收集器”的CMS收集器变种。所做的事情和单CPU年代PC机操作系统使用抢占式来模拟多任务机制的思想
（2）CMS收集器无法处理浮动垃圾，可能出现“Concurrent Mode Failure”失败而导致另一次Full GC的产生。
在JDK1.5的默认设置下，CMS收集器当老年代使用了68%的空间后就会被激活，这是一个偏保守的设置，如果在应用中蓝年代增长不是太快，可以适当调高参数-XX:CMSInitiatingOccupancyFraction的值来提高触发百分比，以便降低内存回收次数从而获取更好的性能，在JDK1.6中，CMS收集器的启动阀值已经提升至92%。

（3）CMS是基于“标记-清除”算法实现的收集器，
手机结束时会有大量空间碎片产生。空间碎片过多，可能会出现老年代还有很大空间剩余，但是无法找到足够大的连续空间来分配当前对象，不得不提前出发FullGC。
为了解决这个问题，CMS收集器提供了一个-XX:+UseCMSCompactAtFullCollection开关参数（默认就是开启的），用于在CMS收集器顶不住要进行FullGC时开启内存碎片合并整理过程，内存整理的过程是无法并发的，空间碎片问题没有了，但停顿时间变长了。虚拟机设计者还提供了另外一个参数-XX:CMSFullGCsBeforeCompaction,这个参数是用于设置执行多少次不压缩的Full GC后，跟着来一次带压缩的（默认值为0，标识每次进入Full GC时都进行碎片整理）

7. G1收集器
优势：
（1）并行与并发
（2）分代收集
（3）空间整理 （标记整理算法，复制算法）
（4）可预测的停顿（G1处处理追求低停顿外，还能建立可预测的停顿时间模型，
能让使用者明确指定在一个长度为M毫秒的时间片段内，消耗在垃圾收集上的时间不得超过N毫秒，
这几乎已经实现Java（RTSJ）的来及收集器的特征）

引用计数器

给每个对象分配一个计数器，当被引用时就加一，引用失效就减一。
计数器为零时，则说明该对象不可能再被使用。

有一个非常明显的缺点，就是无法回收互相引用的对象，从而引起内存泄露。（JVM未使用此算法）

标记-清除法

该算法分两个阶段进行--“标记”和“清除”。首先通过根对象标记所有可达的对象，然后清除所有未被标记的不可达对象。
缺点，就是容易产生内存碎片。

复制算法

基于标记-清除算法（对其产生过多内存碎片的缺点进行了优化）
复制算法将内存空间分成两等份（如下图的A和B），每次只使用其中的一块，
当垃圾回收的时候，将A中的可达对象复制到B中，然后清空A中的所有对象。

标记-压缩法

继承自标记-清除算法
首先将标记所有可达对象，然后将所有可达对象压缩（或者叫移动）到内存的一端，最后将边界以外的空间全部清空。

对象在内存中的状态

可达状态　　：一个对象被创建后，若有一个以上变量引用它。
可恢复状态　　：没有补引用，但finalize()可以使之复活
不可达状态　　：没有被引用，且调用过finzlize()后也没有复活

分代回收

[csdn](https://blog.csdn.net/liushuijinger/article/details/51470379)
新生代
　　新产生的对象，会被分配到新生代的Eden（ 大对象会直接进入老年代）
老年代
永久代
Minor(较小的) Collection（对新生代进行垃圾回收，Hot Spot采用的复制算法）
　　当Eden没有足够空间的时候，就会进行Minor Collection。、
　　在Minor Collection执行的时候，会将存活下来的对象复制到Survivor区。
　　如果Survivor也没有足够空间的时候，将会有一部分对象被迁移到老年代，
　　这个迁移的过程称作晋升（Promotion）
Full Collection
　　（对所有分代进行垃圾回收，也叫Major Collection，Hot Spot采用的是标记-压缩算法）
　　当老年代内存紧张的时候，就会触发Full Collection。
　　通常Full Collection会对整个堆进行回收（CMS收集器除外，它不对新生代进行回收）。
　　回收频率非常低，因为它每一次回收耗时很长。

JVM会通过以下两个参数判断对象是否晋升到老年代：
* 年龄，经历Minor Collection的次数代表对象的年龄 从0开始计 （刚创建的对象）
* 大小，即占用内存空间的大小