JVM中的栈

Oracle关于栈和栈帧提供了如下描述：

每个JVM线程拥有一个私有的 Java虚拟机栈，创建线程的同时栈也被创建。一个JVM栈由许多帧组成，称之为"栈帧"。JVM中的栈和C等常见语言中的栈比较类似，都用于保存局部变量和部分计算结果，同时也参与方法调用和返回。

如Oracle官方说明，每个线程拥有自己的私有栈，因此在多线程应用中将有多个栈，每个栈有自己的栈帧。

Java中的栈

- **当一个新的线程创建时，JVM会为这个线程创建一个新的Stack。一个Java Stack在一个个独立的栈帧中存储了线程的状态。JVM只会在Java Stack中做两个操作：push 和 pop.**
- 一个线程当前正在执行的方法称之为线程的 当前方法，当前方法对应的栈帧称为 当前帧，当前方法所属的类称为 当前类，当前类的常量池称为 当前常量池。 在执行一个方法时，JVM会保存当前类和当前常量池的轨迹。当JVM执行 需要操作栈帧中数据的指令时，JVM会在当前栈帧进行处理。
- 当一个线程执行一个Java方法时，JVM将创建一个新的栈帧并且把它push到栈顶。此时新的栈帧就变成了当前栈帧，**方法执行时，使用栈帧来存储参数、局部变量、中间指令以及其他数据。**

栈内存，主管程序的运行，生命周期和线程同步

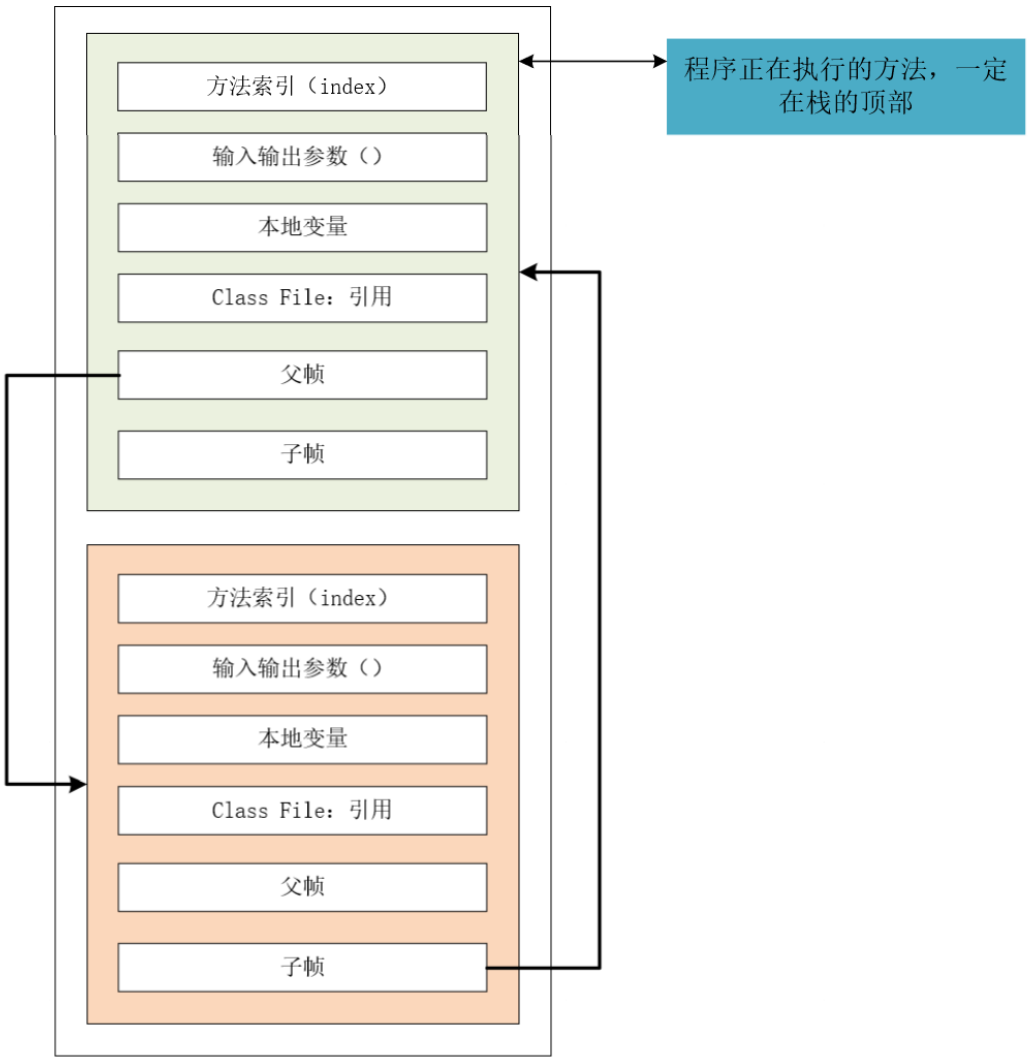
线程结束，栈内存释放了，对于栈来说，不存在垃圾回收，一旦线程结束，栈就Over了！

1、栈里面存放什么

栈：8大基本类型 + 对象的引用 + 实例的方法

2、栈运行原理

栈帧



栈满了：StackOverflowError

堆(Heap)

一个JVM只有一个堆内存，堆内存的大小是可以调节的，我们可以通过选项"-Xmx" 最大堆内存 和

"-Xms" 初始化堆内存 来进行设置。一旦堆区中的内存大小超过 “-xmx"所指定的最大内存时，将会抛出 outofMemoryError(OOM) 异常。

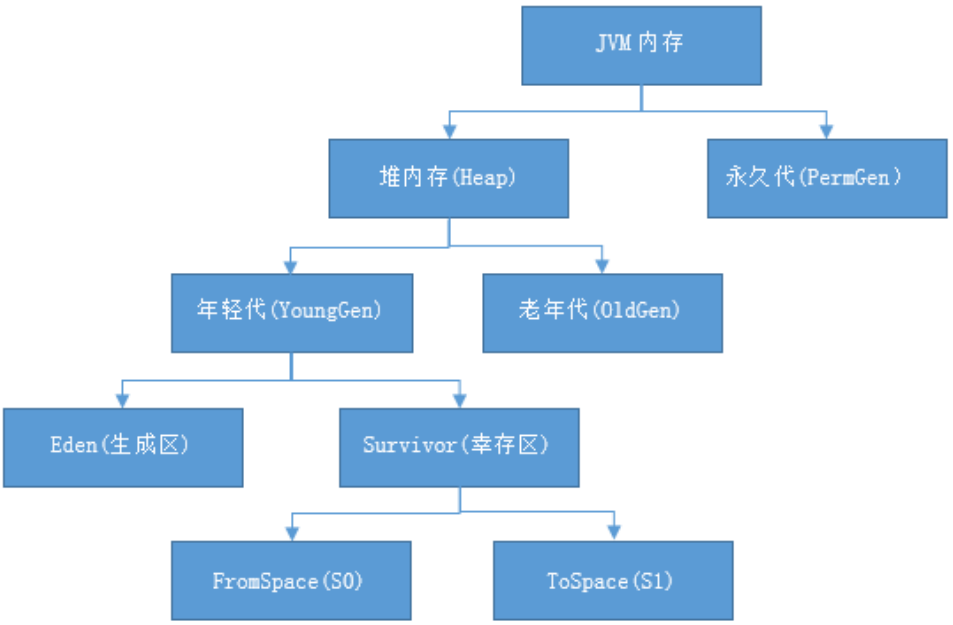
Q：类加载器读取类文件后，一般会把什么东西放到堆中？

A：类，方法，常量，变量~保存我们所有引用类型的真实对象

堆内存中还要细分为三个区域:

- **新生区(伊甸园区)**
- **养老区**

- **永久区**



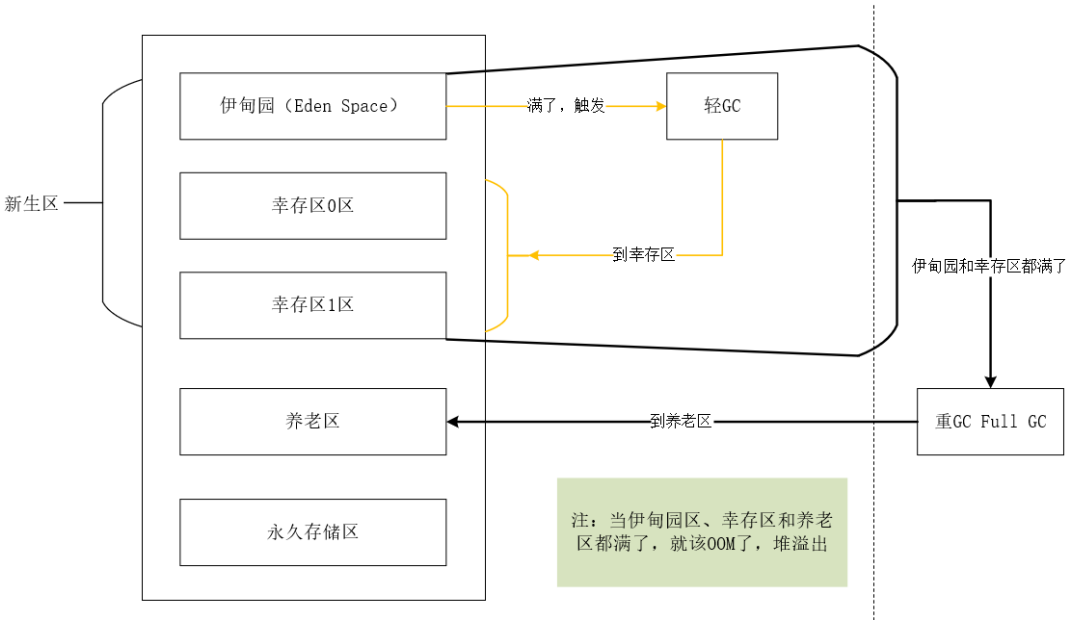
GC垃圾回收主要是在伊甸园区和养老区

1. 堆内存用途：存放的是对象，垃圾收集器就是收集这些对象，然后根据GC算法回收。
2. 非堆内存用途：永久代，也称为方法区，存储程序运行时长期存活的对象，比如类的元数据、方法、常量、属性等。

新生区
年轻代又分为Eden和Survivor区。Survivor区由FromSpace和ToSpace组成。Eden区占大容量，Survivor两个区占小容量，默认比例是8:1:1。

- 类：诞生和成长的地方，甚至死亡；
- 伊甸园
- 幸存者区(0,1) 0 from，1 to

经过研究，99%的对象都是临时对象！



老年区

新生区的幸存者经过多次(15)存入养老区

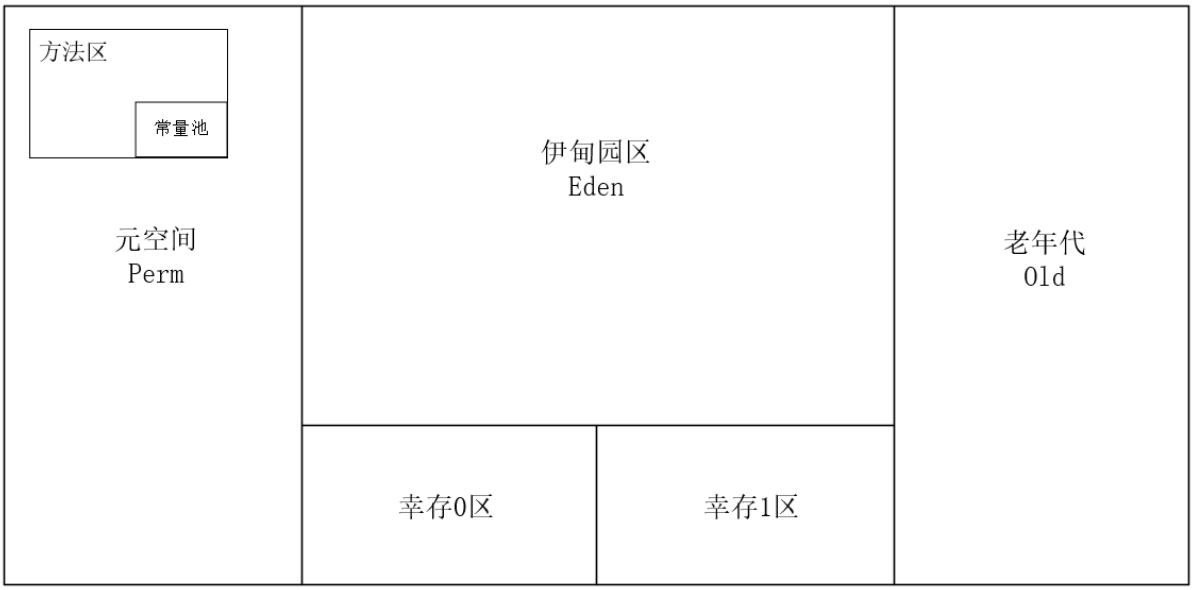
永久区

这个区域是常驻内存的。用来存放JDK自身携带的Class对象。Interface元数据，存储的是Java运行时的一些环境。这个区域不存在垃圾回收！关闭虚拟机就会释放这个区域的内存。

JDK1.6之前：永久代，常量池在方法区中

JDK1.7：永久代，但是慢慢退化了，提出了去永久代的概念，常量池在堆中

JDK1.8之后：无永久代，常量池在元空间,元空间与永久代类似，都是方法区的实现，他们最大区别是：元空间并不在JVM中，而是使用本地内存。



但是，元空间：逻辑上存在，物理上不存在！

```
package com.draco.heapOverflow;

/**
 * 元空间逻辑上存在，物理上不存在
 */
public class SanQu {
    public static void main(String[] args) {
        // 返回jvm试图使用的最大内存
        long max = Runtime.getRuntime().maxMemory();
        // 返回jvm的初始化内存
        long total = Runtime.getRuntime().totalMemory();

        System.out.println("max="+max+"字节\t"+(max/(1024*1024))+"MB");
        System.out.println("total="+total+"字节\t"+(total/(1024*1024))+"MB");

        //默认情况下，试图分配的最大内存是电脑内存的1/4，而初始化的内存是1/64
        // -Xms1024m -Xmx1024m -XX:+PrintGCDetails
    }
}
```

运行结果：

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java.exe" ...
max=1881145344字节	1794MB
total=128974848字节	123MB
```

当修改了VM选项后：-Xms1024m -Xmx1024m -XX:+PrintGCDetails，输出结果：

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java.exe" ...
max=1029177344字节  981MB
total=1029177344字节    981MB
Heap
 PSYoungGen      total 305664K, used 20971K [0x00000000eab00000, 0x0000000100000000, 0x0000000100000000)
  eden space 262144K, 8% used [0x00000000eab00000,0x00000000ebf7afb8,0x00000000fab00000)
  from space 43520K, 0% used [0x00000000fd580000,0x00000000fd580000,0x0000000100000000)
  to   space 43520K, 0% used [0x00000000fab00000,0x00000000fab00000,0x00000000fd580000)
 ParOldGen       total 699392K, used 0K [0x00000000c0000000, 0x00000000eab00000, 0x00000000eab00000)
  object space 699392K, 0% used [0x00000000c0000000,0x00000000c0000000,0x00000000eab00000)
 Metaspace       used 3394K, capacity 4496K, committed 4864K, reserved 1056768K
  class space    used 379K, capacity 388K, committed 512K, reserved 1048576K

Process finished with exit code 0
```

让我们来算一笔账，

新生区：305664k；养老区：699392k

加在一起：1,005,056k，除以1024后 = 981.5MB，等于jvm试图分配的最大内存，所以说元空间逻辑上存在，物理上不存在。

出现OOM

1. 尝试扩大堆内存去查看内存结果

   -Xms1024m -Xmx1024m -XX:+PrintGCDetails

2. 若不行，分析内存，看一下是哪个地方出现了问题（专业工具）
   1. 能够看到代码第几行出错：内存快照分析工具，MAT（eclipse），Jprofiler
   2. Dubug，一行行分析代码！（不现实）

MAT，Jprofiler作用：

- 分析Dump内存文件，快速定位内存泄漏
- 获得堆中的数据
- 获得大的对象
- ...

VM options参数

-Xms 设置初始化堆内存分配大小，默认1/64

-Xmx 设置最大分配堆内存，默认1/4

-XX:+PrintGCDetails 打印GC垃圾回收信息

-XX:+HeapDumpOnOutOfMemoryError 生成oomDump文件