# 二、Java内存区域与内存溢出异常

## 1 概述

Jvm虚拟机负责java程序内存的管理（申请与释放）

## 2 运行时数据区域

jvm虚拟机会将其管理的内存分为很多种，来满足程序运行的需要。

### 1 程序计数器

每个线程所执行的字节码的指令位置。通俗讲就是：线程指向到代码的哪个位置。

线程私有的。

Why： 每个线程都需要记录位置。

### 2 栈（java虚拟机栈）

1 每个线程都会有自己独立的栈，存放局部变量。先进后出，入栈出栈。

2 栈是有深度的：如果深度不足，则StackOverflowError

如果栈支持动态拓展，申请不到足够的内存来拓展的时候，则OutOfMerroryError

3 为什么多线程编程的时候局部变量不影响线程同步？

因为每个线程都有自己独立的栈，不共享；但堆不是，多个线程共享同一个堆空间。

4使用 –Xss 调整大小。

### 3 本地方法栈

Native方法使用的栈

### 4 堆

1 程序运行的对象实例和数组分配在堆上。

2 怎么进行垃圾回收呢？

垃圾回收算法。

3 为了更好地垃圾回收，使用分代手机，堆又细分为几块。

新生代： Eden，From Survivor, To Survivor

老生代：

永久代：

### 5 静态方法区

所有线程共享的。

1 存储jvm加载的类信息，编译的代码信息。

类加载器加载的类信息应该放在这里。

2 如果是Hotspot虚拟机，方法区就是永久代。

3 内部可以细分出一块 常量池。 字符串、static变量都放在常量池中。所有线程共享这一份。但是，好像在jdk1.7之后，jvm逐步去永久代，已经把 字符串常量池从永久代中移出了。

### 7 直接内存 （堆外内存）

NIO中新加入的类，不属于JVM管理。但JVM+直接内存不能大于操作系统内存。

总结：

1 程序运行，首先加载类(放入方法区)-》static变量（方法区中）-》字符串（运行时常量池中）-》程序运行-》基本类型的局部变量（放入线程栈中）-》生成引用类型的对象（线程栈中放入引用指针，对象实例占用空间在堆中申请）

2 每块数据区域在什么情况下都报什么异常。

# 三、垃圾收集器与内存分配策略

哪些对象可以回收？（判断对象死亡还是存活）

什么时候回收？

如何回收？ （垃圾收集算法）

## 3.2 对象已死吗？

怎么判断对象是否还在“存活”，哪些算法？

### 3.2.1 引用计数法

给对象添加一个引用计数器，当其为0时，对象不再可用，可以回收了。

缺点： 无法解决对象间相互引用的问题。

### 3.2.2 可达性分析算法

从GC Roots对象出发，向下搜索，没有达到的对象，则是不可用的对象。

1 GC ROOts对象不是一个，而是多个。

2 GC Roots对象怎么挑选？

### 3.2.3 扩展引用几种类型

1 为什么增加几种引用的类型？

刚开始，只有强引用。但我们希望能描述这样的一类对象，当内存空间还足够时，则保留在内存中；如果内存空间在进行垃圾收集后还是很紧张，则抛弃这些对象。这样，能使用这些引用实现缓存功能。

它们的回收时机不同：

强引用：

软引用：在系统即将发生内存溢出异常之前，将这些对象进行二次回收。

弱引用：会延长一个垃圾回收的寿命。在第二次垃圾回收的时候，进行回收。

虚引用：对象被回收时，收到一个系统通知。

### 3.2.5 回收方法区

方法区（HotSpot中的永久代）也可以通过设置jvm参数进行垃圾回收，回收废弃的常量和无用的类，但影响小，我们并不关注。

## 3.3 垃圾回收算法

回收算法的发展历程。

### 3.3.1 标记-清除算法（Mark-Sweep）

最简单，最原始；效率不高，产生大量的空间碎片。

### 3.3.2 复制算法

为了解决上面算法效率低，内存碎片的问题。

思想：把内存分为大小相同的两块，每次只使用其中一块。而用完需要垃圾回收的时候，把存活对象复制到另外一块上。

特点：简单高效；但浪费了一部分内存空间。

现代的商业虚拟机都使用这种算法来对新生代的内存进行回收，但因为新生代中对象98%朝生夕死，并不是1：1比例划分。

Eden：From Survivor：To Survivor = 8 ： 1 ： 1 （使用90%的新生代内存，浪费10%），并不能保证10%的Survivor内存够用，因为需要老年代进行分配担保。

### 3.3.3 标记-整理算法（Mark-Compact）

老年代对象存活率高，复制算法不适用。（拷贝对象大，效率低，浪费空间）

因此，有了 标记-整理算法。

### 3.3.4 分代收集算法

把前面的算法进行整合，根据对象的存活周期不同将内存划分为几块，每块采用合适的算法。

## 3.5 垃圾收集器

讨论的是HotSpot虚拟机使用的哪些垃圾收集器。

STW： 暂停用户工作线程的时间，越少越好

1 Serial 收集器 复制算法 单线程、Client模式适用

2 ParNew 收集器 复制算法 Serial的多线程模式 Server模式适用

3 CMS 收集器 ConcurrentMarkSweep 老年代

缩短STW时间为目的。3次标记，一次清除。

a 初始标记

4 G1 收集器

参考：

<https://ylgrgyq.github.io/jvm/2016/07/03/garbage-first-collector-understanding.html>

<http://www.open-open.com/lib/view/open1482392048975.html>

<http://ifeve.com/%E6%B7%B1%E5%85%A5%E7%90%86%E8%A7%A3g1%E5%9E%83%E5%9C%BE%E6%94%B6%E9%9B%86%E5%99%A8/>