第二章 Java内存区域与内存溢出异常

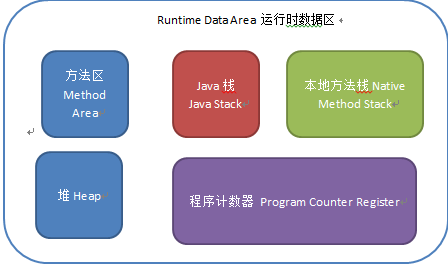
#### 一．概述

1. 对于java程序员来说，不需要手动释放某个java对象的内存。

优点：不容易因为忘记回收内存而出现内存泄漏和内存溢出的问题。

缺点：一切由虚拟机执行，如果出现内存溢出和内存泄漏方面的问题，需要我们了解虚拟机使用内存的原理。

#### 二．运行时数据区域



程序计数器

1. 简介

当前线程所执行的字节码行号指示器。是唯一一个在java虚拟机规范中没有规定任何OutOfMemoryError的区域。

2. 是否线程私有

线程私有。

3. 功能

通过改变计数器的值来选取下一条需要执行的字节码指令，分支、循环、跳转、异常处理、线程恢复等基础功能都需要依赖这个计数器来完成。

其中，线程恢复体现在多线程情况下，多个线程之间轮流切换运行，这时候当一个线程切换回来需要知道自己运行到哪个位置了，而程序计数器就会记录这个位置，以便于切换回来可以继续运行。也因此，需要是线程私有的。

4. 异常

未规定任何OutOfMemoryError。

Java虚拟机栈

1. 简介

Java虚拟机栈描述了Java方法执行的内存模型。

2. 是否线程私有

线程私有。

3. 功能

每个方法运行时都会创建一个栈帧，用于存储局部变量表、操作数栈、动态链接、方法出口等信息。每个方法从调用直至完成的过程，就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。

局部变量表：

存放了编译期可知的各种基本数据类型（byte、int、char…）、对象引用类型，不等同于对象，可能是指向一个代表对象的引用。

其中64位的double和long类型的数据会占用2个局部变量空间，其余数据类型只占用1个。局部变量表所需的空间在编译期完成分配，当进入一个方法时，这个方法需要在栈帧中分配多大的局部变量空间完全是确定的。

4. 异常

·虚拟机不可动态扩展：

如果线程请求的栈深度大于虚拟机所允许的深度，将抛出StackOverflowError。

·虚拟机可以动态扩展：

如果扩展时无法申请到足够内存，将抛出OutOfMemoryError异常。

本地方法栈

1. 简介

本地方法栈与虚拟机栈类似，不过虚拟机栈执行java方法，而本地方法栈执行native方法。

Java堆

1. 简介

仅存放对象的所有实例，是java内存管理中最大的一块。

细分为（默认比例）：

---新生代：1/3

---Eden：8/10

---From Survivor：1/10

---To Survior：1/10

---老年代：2/3

而且java堆还可以划分出多个线程私有的分配缓冲区（TLAB）

2. 是否线程私有

线程共享。

3. 功能

存放对象的所有实例。

4. 异常

堆无法扩展时会抛出OutOfMemoryError异常。

方法区



1. 简介

逻辑上属于堆的一部分，包括Class文件信息和运行时常量池。在jdk1.7中已经把，字符串常量池移出放到堆中。

---静态常量池：

.class文件里的常量池

---运行时常量池：

2. 是否线程私有

线程共享。

3. 功能

存储已被虚拟机加载的类息、常量、静态变量、即时编译后的代码等。

4. 异常

OutOfMemoryError异常。

直接内存

1. 不是虚拟机运行时数据区的一部分，也不是java虚拟机规范中的内存区域。但是这部分内存被频繁的使用，也会导致OutOfMemoryError的出现。

通常基于NIO被使用。

#### 二．HotSpot虚拟机对象

对象的创建

1. 内存分配方式

* 指针碰撞

适用情况：

Java堆中内存是绝对规整的，所有用过的放在一边，空闲的放在另一边，中间放着一个指针作为分界点的指示器。

分配过程：

把指针向空闲空间挪动一段与对象大小相等的距离。

垃圾收集器：

带有压缩整理的功能，如Serial、ParNew等。

* 空闲列表

适用情况：

Java堆中内存是不规整的，已使用的内存和空闲的内存相互交错。虚拟机需要维护一个列表，记录上哪些内存可用。

分配过程：

在分配的时候从列表中找到一个足够大的空间划分给对象实例，并更新列表上的记录。

垃圾收集器：

不具有压缩整理功能，如CMS。

2. 内存分配的线程安全性

内存分配不是线程安全的。

即便是仅仅修改一个指针所指向的位置，在并发情况下也并不是线程安全的，可能出现正在给A对象分配内存，指针还没来得及修改，对象B又同时使用了原来的指针来分配内存的情况。

解决方案：

* 同步处理---CAS
* TLAB

3. 对象的内存布局

* 对象头（32位或64位）

存储运行时数据：

哈希码（25bit）、GC分代年龄（4bit）、锁状态标志、线程持有的锁、偏向线程ID、偏向时间戳等。

Java数组还要有一块记录数组长度。

32bit中还有2bit存储锁标志位，1bit固定为0

* 实例数据

对象的有效信息：

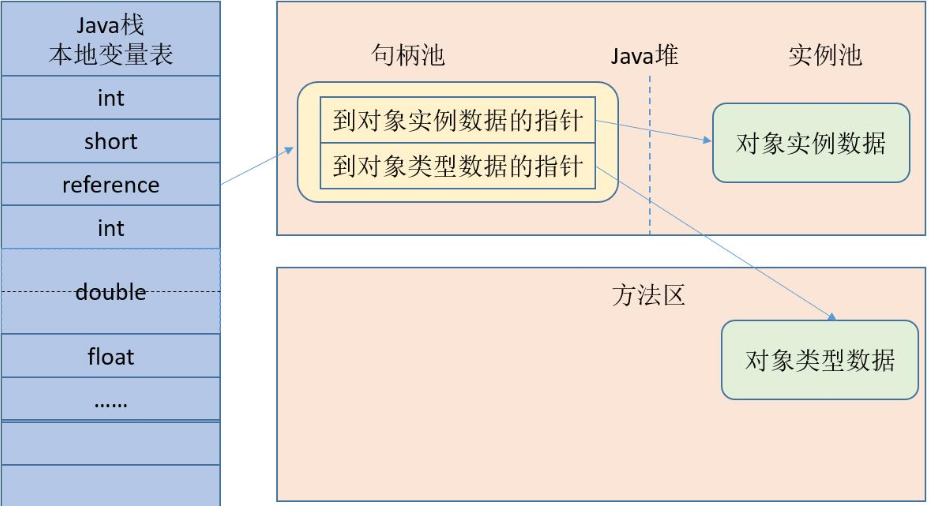
各类型的字段内容

* 对齐填充

非必须。对象必须是8字节的倍数，实例数据不一定多大，所以需要对齐填充。

4. 对象的访问定位

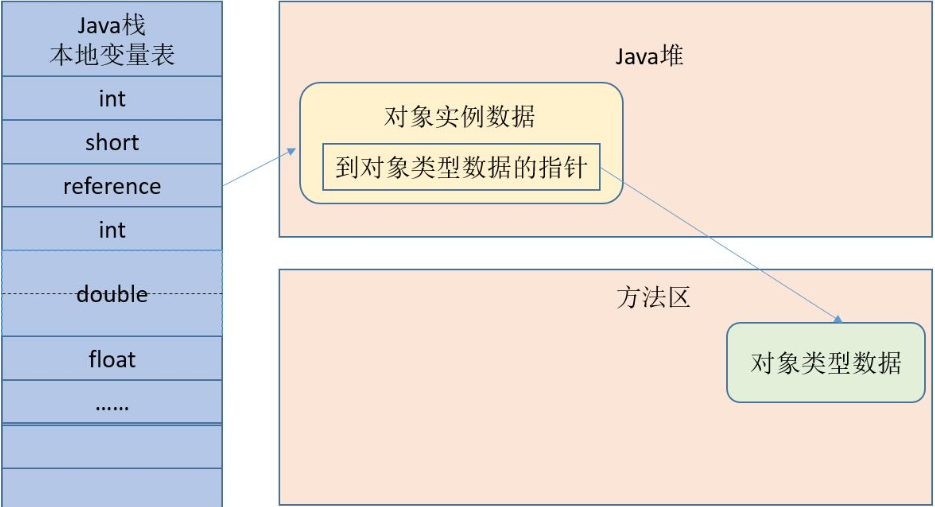
* 句柄池



优势：

reference中存储的是句柄地址，对象被移动时（垃圾收集）指挥改变句柄中的实例数据指针，而reference不需要修改。

* 直接指针



优势：

访问速度更快，节省了一次指针定位的时间开销。