> 了解java虚拟机内存划分和各模块作用使得我们更加清楚如何使用内存，防止内存溢出和泄露。

一、Java内存区域的划分

1.程序计数器（Program Counter Register）

这是一块**线程私有**区域，可以看做是当前线程执行的的字节码的行号指示器。它里面存储的是线程将要执行的下一条指令的地址。 **唯一一个JVM中没有规定任何OOM情况的区域**

2.java虚拟机栈

同样也是**线程私有**区域它的单位（即栈内元素）是栈帧，栈的生命周期随线程调用方法而起（入栈），随线程方法调用结束而结束（出栈）。整个虚拟机栈的生命周期跟线程保持一致。

栈帧到底是什么？ 它存储着以下：

1. 局部变量表：编译期可知的各种基本数据类型，对象引用类型。**编译期确定大小**，不会改变。
2. 操作数栈：指令的执行本质是一系列运算，入栈出栈的过程。
3. 动态链接
4. 方法返回地址：方法执行完毕需要回到调用方法的入口处。

虚拟机栈会发生两种异常：

1. OOM：动态扩展虚拟机内存空间时不够时会导致OOM
2. 线程调用方法层级过深，超出虚拟机所允许的最大长度导致StackOverflow

3.本地方法栈

1. 与虚拟机栈相似，但是是为Native方法服务的。
2. 会抛出Stack Overflow和OOM两种异常

4.Jva堆

1. 作用：存放对象实例
2. 是否线程私有：否，线程共享。因此存在并发安全问题
3. 是GC回收的主要作用区域。
4. 从内存回收看可以分为新生代和老年代，从内存分配看堆中可以划分出多个线程私有分分配缓冲区
5. 可以不要求物理上连续的空间
6. 存在OOM

5.方法区

1. 线程共享
2. 存储已被虚拟机加载的类信息、常亮、静态变量、即时编译器编译后的代码等数据。
3. GC时主要是针对常量池的回收和对类型的卸载

> 类卸载条件：
>
> 1. 堆中不存在该类的任何实例
> 2. 加载该类的classLoader已经被回
> 3. 该类对应的java.lang.class类对象没有在任何地方被引用
> 4. 无法在任何地方通过反射访问该类的任何方法

6.运行时常量池

1. 本质是属于方法区的运行时常量池。
2. 存放编译期生成的各种字面量和符号引用
3. 并非编译期预置于class的常量才能进入常量池，通过String的intern（）方法也能将常量动态加入常量池
4. 会存在OOM

二、对象的访问定位

Java程序需要通过栈上的**reference**数据访问堆上的对象，如何通过这个**reference**得到堆上的对象呢?

1. 使用句柄的方式

在java堆中开辟一个区域专门存储对象实例地址和类型数据信息地址的内存块叫做句柄池

> 优点：
 reference中存储的是稳定的句柄地址，对象被移动的时候只会改变句柄中的实例数据指针，reference本身不需修改

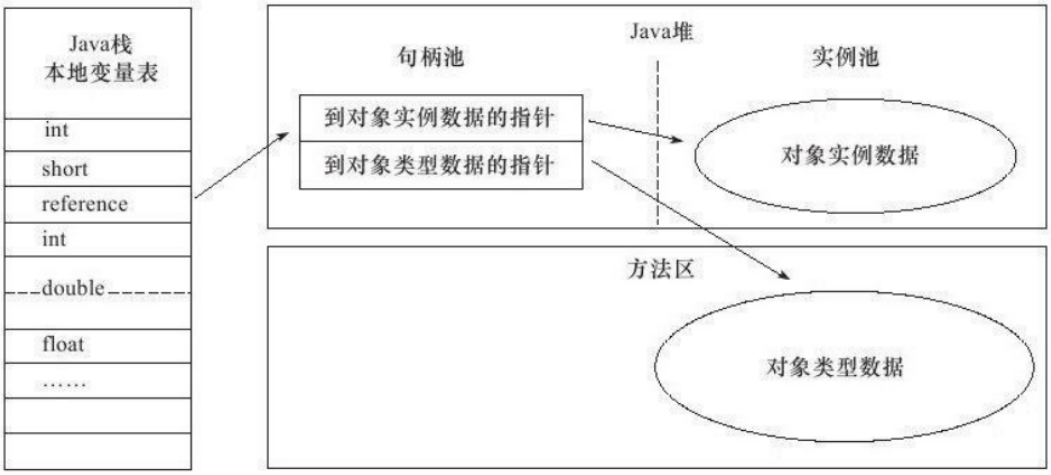

图　2-2　通过句柄访问对象

2. 使用直接指针的方式

reference中存储的直接是对象地址，再通过对象找到类型信息的地址

> 优点：减少一次hash定位的过程，速度更快

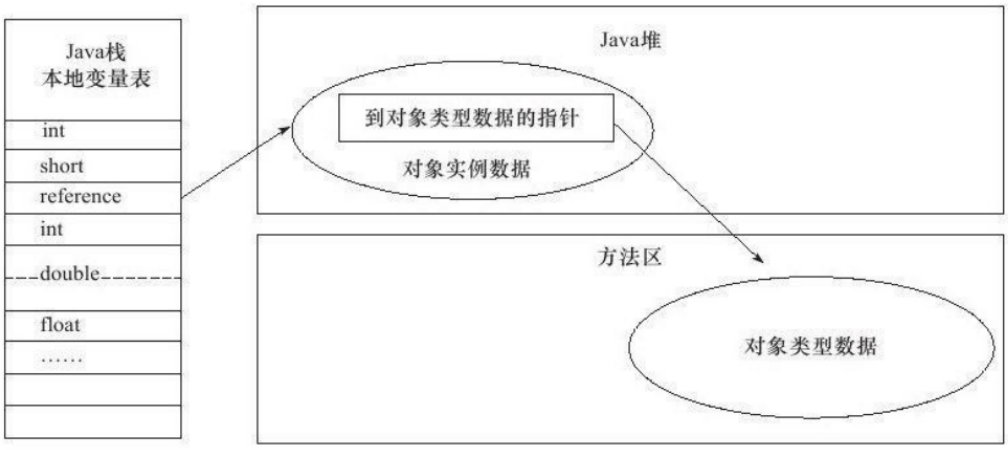

图　2-3　通过直接指针访问对象