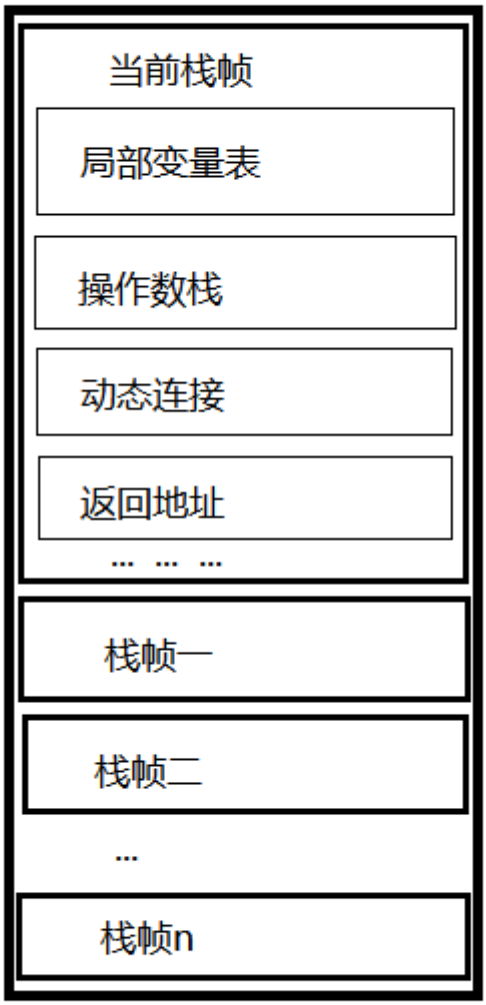
运行时数据区域

一、程序计数器(记录程序运行的位置)

- 程序计数器是一块较小的内存空间，可以看作当前线程所执行的字节码的行号指示器
- 字节码解释器工作室就是通过改变计数器的值来选择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计数器来完成
- java多线程是通过时间切片来分配线程执行的，当一个线程的时间片耗尽时，就需要有一个记录线程的运行记录，方便下一次轮转到此线程执行时可以从上一次挂起的位置继续执行，因此，每一个线程都需要有一个计数器，且线程中计数器是线程隔离的，互不影响，为线程私有

- 执行Java方法时，计数器记录的是正在执行的字节码指令的地址，执行native方法时，计数器为空。

二、java虚拟机栈



线程虚拟机栈

1. Java虚拟机栈
    - 是线程私有的，生命周期于线程相同。虚拟机栈描述的时java方法执行的内存模型，每个方法在执行的同时都会同时都会创建一个栈帧(Stack Frame)用于储存局部变量表、操作数栈、动态链接、方法出口等信息。每一个方法的调用到完成就相当于一个栈帧在虚拟机中入栈到出栈的过程

2. 局部变量表
    - 存放编译期可知的各种基本数据类型、对象引用(对象引用指针或对象句柄)、returnAddress类型(指向了一条字节码指令的地址)
    - long和double类型的数据会占据两个局部变量空间(Slot),其余的数据类型都只占用一个，局部变量表所虚空见在编译期间完分配，当进入一个方法时，这个方法在帧中需要分配的局部变量空间是确定的，方法运行期间不会改变
3. 会抛异常：StackOverflowError和OutOfMemoryError

三、本地方法栈(Native Method Stack)

1. 为虚拟机使用到的Native方法服务，虚拟机规范中对本地方法栈中方法使用的语言、使用方式、与数据结构没有强制规定。
2. 有的虚拟机(如hotspot)直接将本地方法栈于虚拟机栈合二为一
3. 会抛异常：StackOverflowError和OutOfMemoryError

四、java堆(Java Heap)

1. java堆是java虚拟机所管理的内存中最大的一块，所有线程共享，在虚拟机启动时创建
2. java堆的唯一作用就是存放对象实例
3. java堆是垃圾收集器管理的主要区域

五、方法区(Non-Heap)

1. 线程共享的区域，储存已被虚拟机加载的**类信息【全限定类(包名.类名）、直接父类。全限定类名、直接接口的有序列表、修饰符】、常量、静态变量、即时编译器编译后的代码等数据**等
2. 运行时常量池
    - 存放编译器生成的各种字面量和符号引用，在类加载后进入方法区运行时常量池存放

六、直接内存

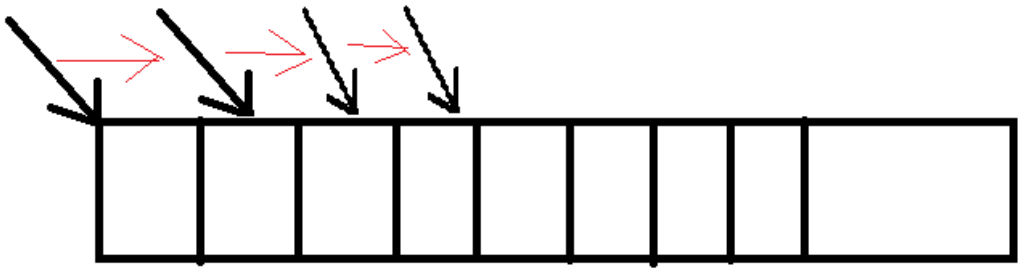
对象

一、创建对象时的内存分配

内存分配

1. 指针碰撞

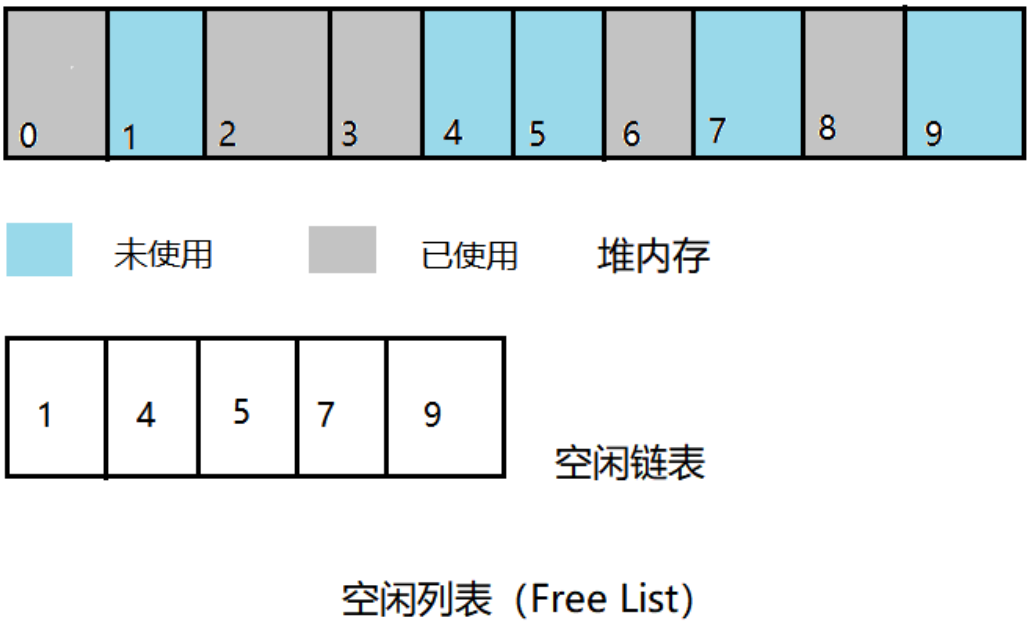
> 绝对规整的堆内存，已使用的内存与未使用的内存通过一个指针分隔开，两种内存不交叉，分配仅将指针移动与对象大小相等的距离即可



指针碰撞

2. 空闲列表

> 内存不规整，已使用与未使用内存相互交叉，使用一个链表记录未使用的内存空间，分配对象内存时从链表中寻找一块足够大的内存进行划分



空闲列表（Free List）

二、对象的内存布局(HotSpot)

1. 对象头

1. 储存对象自身的运行时数据(Mark Word)，如哈希码、GC分代年龄、锁状态标志、线程持有的锁、偏向线程ID、偏向时间戳等
2. 类型指针，即对象指向它的类元数据的指针，虚拟机由此确定该对象属于哪个类

2. 实例数据

- 对象真正储存的有效信息

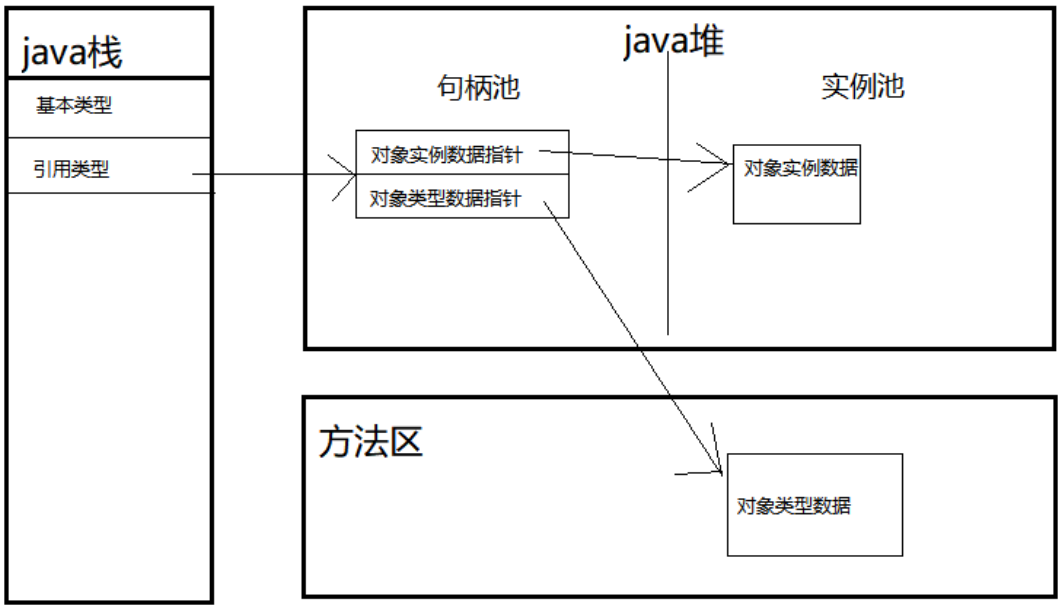
3. 对齐填充

- 使整个对象所占内存为8字节的整数倍，当占位符使用

三、访问定位

1. 句柄访问(稳定)

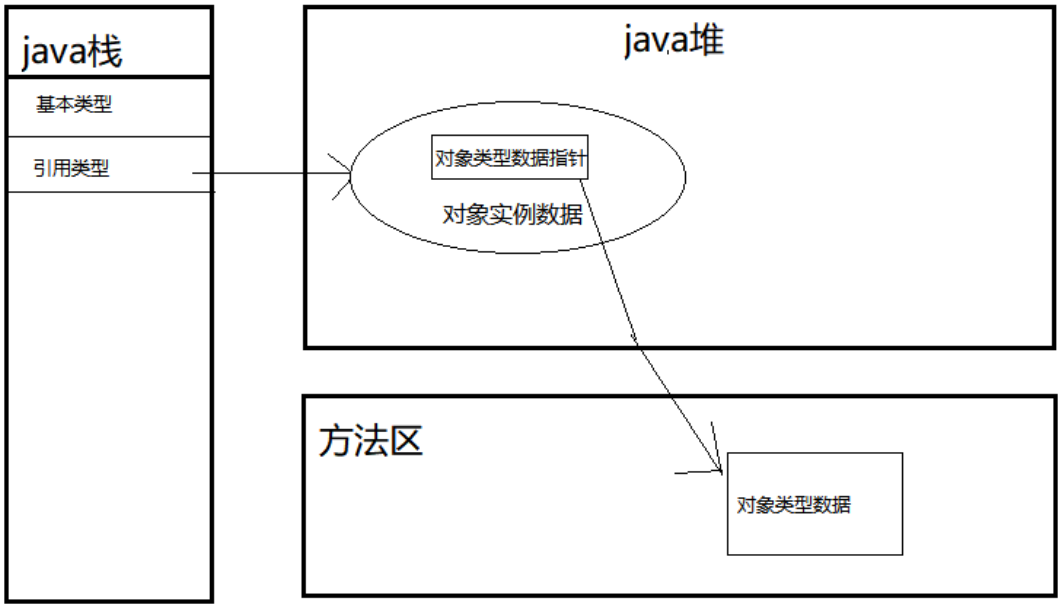
> java堆中将划分一块内存作为句柄池，对象引用储存对象句柄地址，句柄中包含对象实例数据以及类型数据各自的具体地址信息



句柄访问对象

2. 直接指针访问(hotspot使用，定位速度快)

> 引用类型存放Java堆对象地址，堆对象需考虑如何放置对象类型数据



句柄访问对象