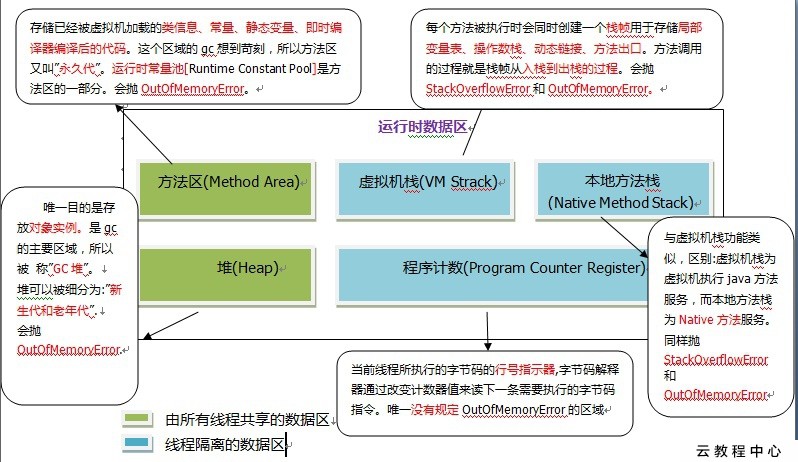
# 运行时数据区域

## 内存模型



线程隔离区：

虚拟区栈:他的生命周期与线程一样，他是描述的是java方法执行的内存模型。

存储的局部变量表存放了编译器可知的各种基本数据类型，64位长度的long与double类型的数据会占用2个局部变量的空间，其余数据类型只占1个。局部变量表的所需的内存空间在编译期间完成分配。

本地方法栈：与虚拟机栈发挥的作用非常相似的，他们之间的区别只不过虚拟机栈执行java方法，而本地方法栈则使用是本地的native方法服务

程序技术器：是一块较小的内存空间，他可以看做是当前线程所执行的字节码的行号指示器。字节码的工作内容就是改变技术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技术器。

线程共享区：

Java堆：对于多程序应用来说java堆是java虚拟机所管理的内存中最大的一块。所有对象的实列与数据都要再这里分配内存。Java堆是垃圾收集器管理的主要区域，因此也称之为GC堆。从内存回收的角度来看。基本上垃圾收集器基本都采用于分代收集算法。大致上年代可分为：新生代与老年代。再细致看，还有Eden空间，From survivor,to Surviovr空间。（从内存分配来看，线程共享的java堆中可能划分多个线程私有的的分配缓存区【TLAB】第三章讲解）

方法区：从程序的部署来看，很多人愿意把方法区称之为‘永久代’。HotSpot的垃圾回收器可以像管理java堆一样管理这部分内存。永久代有 -XX:MaxPermSize 的上限。这部分的区域回收目标主要针对于常量池的回收和对类型的卸载。

运行时常量池是方法区的一部分，用于存放编译期间生成的各种字面量和符号引用

## 对象的创建

1. 虚拟机当遇到一条new指令时，首先将去检查这个指令的参数是否能够在常量池中定位到一个类的符号引用
2. 检查这个符号引用代表的类是否已被加载，解析和初始化过，如果没有那必须先执行相应的类加载过程
3. 类加载检查过后，虚拟机将为新生对象分配内存。对象所需内存的大小在类加载之后便可完全知道。
4. 对象内存划分分为A 指针碰撞【空闲内存连续】 B 空闲列表【空闲内存不连续】。选择那种方案是由java堆是否规则。而决定java堆是否规则所采用的垃圾收集器是否带有压缩整理的功能。
5. 再分配内存时考虑线程安全性。解决方案：a 对分配内存空间的动作进行同步处理（实际上采用CAS配上失败重试的方式保证跟新操作的原子性）b 内存分配的动作按照线程划分再不同的空间中进行，即每个线程再java堆中预先分配小一块内存，称之为本地线程分配缓冲（TLAB）那个线程要分配内存就在那个线程的TLAB上分配，分配之后才需要同步锁定。虚拟机是否用TLAB 可以通过-XX:+/-UseTLAB参数来设定
6. 内存分配之后，虚拟机需要初始化内存值为0（不包括对象头），如果使用TLAB，这一工作可以提前到TLAB
7. 虚拟机要对对象进行必要设置。列如这个对象属于哪个类的实列，如何才能找到类的元数据，对象的哈希码，GC分代年龄信息。这些信息都会存储到对象头中

## 对象的内存布局

对象再内存中存储可分为三块区域：对象头，实列数据，对其填充

对象头：包含两部分信息

第一部分：存储对象吱声运行时的数据，哈希码，GC年龄代，锁状态标志，线程所持有锁，偏向线程id,偏向时间戳等，这部分数据在32与64位的虚拟机中分为32bit 和64bit，官方称之为‘mark word’[注：对象头知识详情请看P47]

第二部分：类型指正，即对象指向他的元数据的指针，虚拟机可以通过这个指针来确定这个对象是哪个类的实列【并不是所有的虚拟机实现都必须在对象数据上保留类型指正，换句话说，查找对象的元信息并不一定要经过对象本省，2.3.3节讨论】。如果对象是数组的话，那再对象头中还必须要一块用于记录数据的长度的数据。

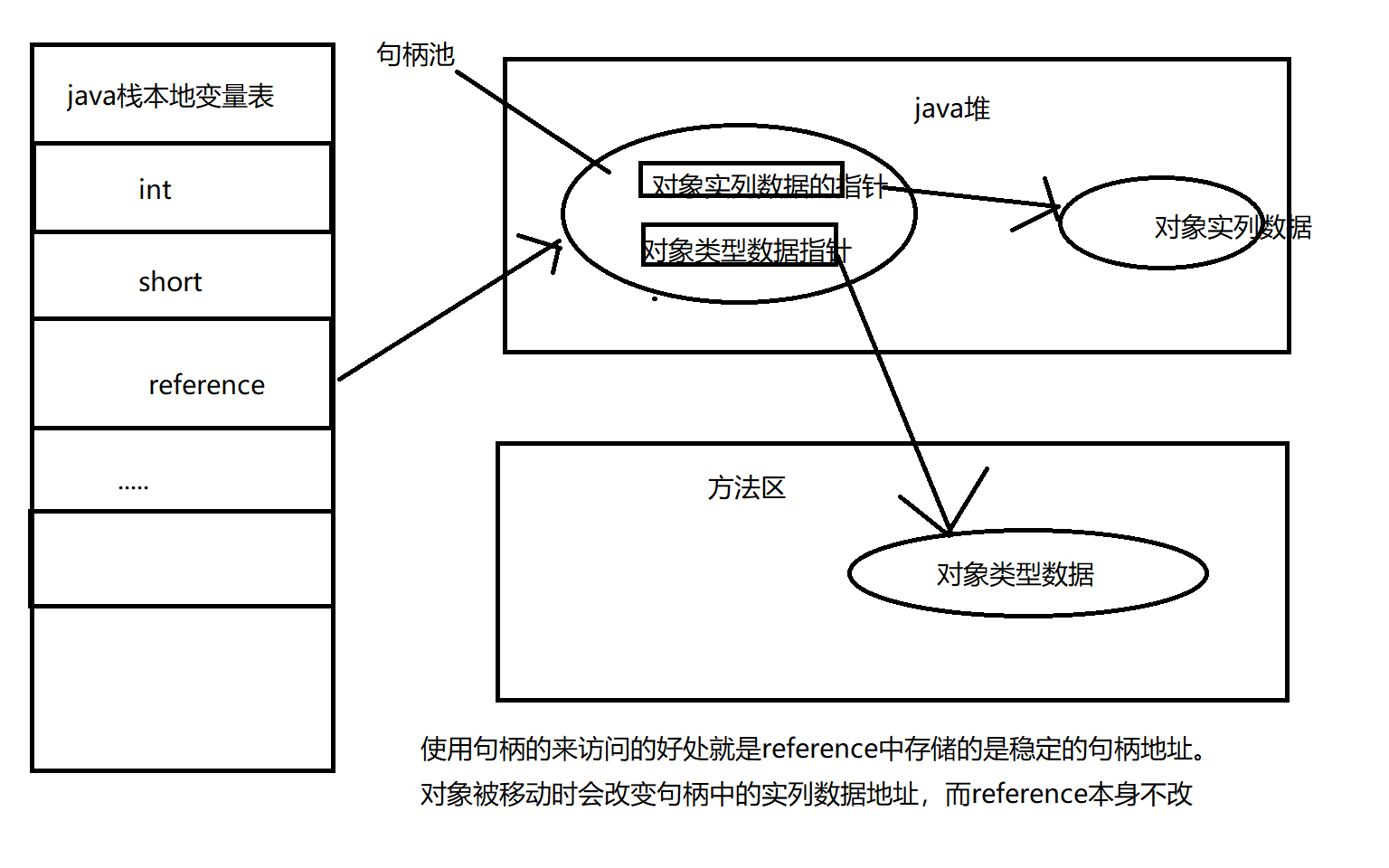
实列数据：真正存储有效信息，程序中定义的各种类型的字段内容

对其填充：并不是必然存在的，没什么特别的含义，起占位符的作用（详情看P48）

## 对象的访问定位

Java程序需要通过栈上的引用数据来操作堆上的具体对象，引用类型再java虚拟机中只规定了一个指向对象的引用。并没有具体去说怎么样定位访问，所以对象的访问方式是取决于虚拟机的目前的主流的访问方式有句柄和直接指正

句柄：java堆会划分一块内存来作为句柄池，栈中的引用就存储的句柄地址，句柄包含的是对象的实列数据与类型数据各自的具体地址信息。如下图所示



直接指针：

