# 2020.5.11

## Jdk,jre,jvm

Jdk《——jre《——jvm

Jdk：java开发工具

Jre：运行环境

Javac：编译

Jvm：java虚拟机（字节码运行在次）

## 内存溢出

VM运行参数：

-XX:+HeapDumpOnOutOfMemoryError 内存溢出时产生快照

-Xss20m规定了每个线程堆栈的大小。一般情况下256K是足够了。影响了此进程中并发线程数大小。

-Xms20m初始的Heap的大小。

-Xmx最大Heap的大小。

在很多情况下，-Xms和-Xmx设置成一样的。这么设置，是因为当Heap不够用时，会发生内存抖动，影响程序运行稳定性。

## JVM区域heap区

Eden Space（伊甸园）、Survivor Space(幸存者区)、Tenured Gen（老年代-养老区）

1、一个人（对象）出来（new 出来）后会在Eden Space（伊甸园）无忧无虑的生活，直到GC到来打破了他们平静的生活。GC会逐一问清楚每个对象的情况，有没有钱（此对象的引用）啊，因为GC想赚钱呀，有钱的才可以敲诈嘛。然后富人就会进入Survivor Space（幸存者区），穷人的就直接kill掉。

2、并不是进入Survivor Space（幸存者区）后就保证人身是安全的，但至少可以活段时间。GC会定期（可以自定义）会对这些人进行敲诈，亿万富翁每次都给钱，GC很满意，就让其进入了Genured Gen(养老区)。万元户经不住几次敲诈就没钱了，GC看没有啥价值啦，就直接kill掉了。

3、进入到养老区的人基本就可以保证人身安全啦，但是亿万富豪有的也会挥霍成穷光蛋，只要钱没了，GC还是kill掉。

# 2020.5.12

## VM

Sun Classic VM

第一款商用Java虚拟机

只能使用纯解释器的方式来执行java代码

Exact VM

只在Solaris平台发布，编译器和解释器混合工作

HotSpot VM

（现在使用的）

KVM

Kilobyte 简单，轻量，高度可移植

在手机平台运行

Jrockit

最快的java虚拟机

专注服务器端应用

优势：垃圾收集器。MissionControl套件

TaoTaoVm

## 内存管理

### 程序计数器

程序计数器(Program Counter Register)是一块很小的内存空间，它可以看着是当前线程所执行的字节码的行号指示器。

在虚拟机的概念模型里，字节码解释器工作时就是通过改变这个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完成。

在多线程的情况下，为了线程切换能恢复到正确的执行位置，每条线程都需要一个独立的计数器各条线程之间计数器互不影响，独立存储，我们称这类内存为"线程私有"的内存。

### 方法区（共享内存）

该区域用于存储已被虚拟区加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

### Java堆(共享内存)

对于大多数应用来说，Java堆(Java Heap)是Java虚拟机所管理的内存中最大的一块。Java堆是被所以线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对象实例。几乎所有的对象实例都是在这里分配内存。实例变量会在对象实例化时随着对象一起分配在Java堆中。Java堆是垃圾收集器管理的主要区域，因此也叫“GC堆”。

### Java虚拟机栈(独立内存)

与程序计数器一样，Java虚拟机栈(Java Virtual Machine Stacks)也是线程私有的，它的生命周期与线程相同。

虚拟机栈描述的是Java方法执行的内存模型：每个方法在执行时都会创建一个栈帧(Stack Frame)用于存储局部变量表、操作数栈、动态链接、方法出口等信息。(栈帧也叫活动记录：所存储的信息)。每个方法从调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。

扩展栈模型信息:

局部变量表存放了编译期可知的各种基本数据类型(boolean、byte、char、short、int、float、long、double)、对象引用(refreence类型，它不等同于对象本身，可能是一个指向对象起始地址的引用指针，也可以是指向一个代表对象的句柄或其他与此对象相关的位置)和returnAddress类型(指向了一条字节码指令的地址).

64位长度的long和double类型的数据会占用2个局部变量空间(Slot),其余的数据类型只占用一个。

局部变量表所需的内存空间在编译期间完成分配，当进行一个方法时，这个方法需要的帧中分配多大的局部变量空间是已经确定的，在方法运行期间不会改变局部变量表的大小。

#### 局部变量表

**局部变量表(Local Variable Table)**是一组变量值存储空间，用于存放方法参数和方法内定义的局部变量。局部变量表的容量以变量槽(Variable Slot)为最小单位，Java虚拟机规范并没有定义一个槽所应该占用内存空间的大小，但是规定了一个槽应该可以存放一个32位以内的数据类型。

一个局部变量可以保存一个类型为boolean、byte、char、short、int、float、reference和returnAddress类型的数据。reference类型表示对一个对象实例的引用。returnAddress类型是为jsr、jsr\_w和ret指令服务的，目前已经很少使用了。

虚拟机通过索引定位的方法查找相应的局部变量，索引的范围是从0~局部变量表最大容量。如果Slot是32位的，则遇到一个64位数据类型的变量(如long或double型)，则会连续使用两个连续的Slot来存储。

#### 操作数栈

**操作数栈(Operand Stack)**也常称为操作栈，它是一个后入先出栈(LIFO)。同局部变量表一样，操作数栈的最大深度也在编译的时候写入到方法的Code属性的max\_stacks数据项中。

操作数栈的每一个元素可以是任意Java数据类型，32位的数据类型占一个栈容量，64位的数据类型占2个栈容量,且在方法执行的任意时刻，操作数栈的深度都不会超过max\_stacks中设置的最大值。

当一个方法刚刚开始执行时，其操作数栈是空的，随着方法执行和字节码指令的执行，会从局部变量表或对象实例的字段中复制常量或变量写入到操作数栈，再随着计算的进行将栈中元素出栈到局部变量表或者返回给方法调用者，也就是出栈/入栈操作。一个完整的方法执行期间往往包含多个这样出栈/入栈的过程。

#### 动态链接

在一个class文件中，一个方法要调用其他方法，需要将这些方法的符号引用转化为其在内存地址中的直接引用，而符号引用存在于方法区中的运行时常量池。

Java虚拟机栈中，每个栈帧都包含一个指向运行时常量池中该栈所属方法的符号引用，持有这个引用的目的是为了支持方法调用过程中的**动态连接(Dynamic Linking)**。

这些符号引用一部分会在类加载阶段或者第一次使用时就直接转化为直接引用，这类转化称为**静态解析**。另一部分将在每次运行期间转化为直接引用，这类转化称为动态连接。

#### 方法返回地址

**当一个方法开始执行时，可能有两种方式退出该方法：**

* **正常完成出口**
* **异常完成出口**

**正常完成出口**是指方法正常完成并退出，没有抛出任何异常(包括Java虚拟机异常以及执行时通过throw语句显示抛出的异常)。如果当前方法正常完成，则根据当前方法返回的字节码指令，这时有可能会有返回值传递给方法调用者(调用它的方法)，或者无返回值。具体是否有返回值以及返回值的数据类型将根据该方法返回的字节码指令确定。

**异常完成出口**是指方法执行过程中遇到异常，并且这个异常在方法体内部没有得到处理，导致方法退出。

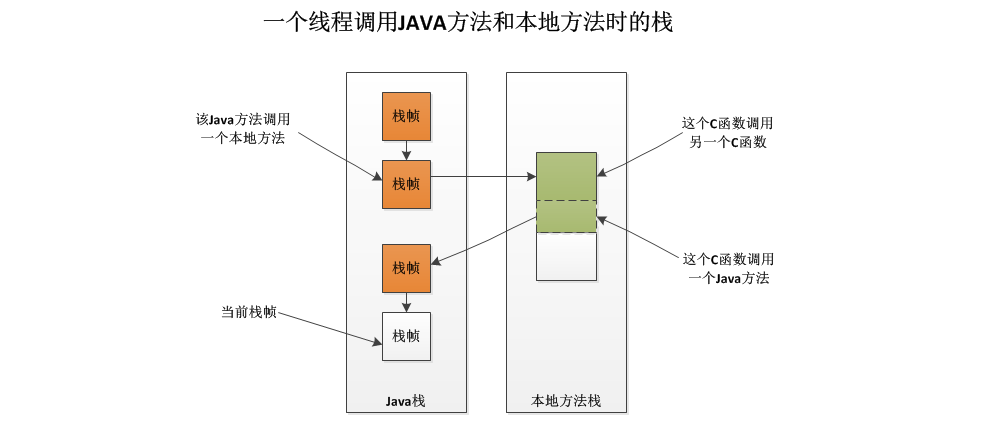
无论是Java虚拟机抛出的异常还是代码中使用athrow指令产生的异常，只要在本方法的异常表中没有搜索到相应的异常处理器，就会导致方法退出。

无论方法采用何种方式退出，在方法退出后都需要返回到方法被调用的位置，程序才能继续执行，方法返回时可能需要在当前栈帧中保存一些信息，用来帮他恢复它的上层方法执行状态。

方法退出过程实际上就等同于把当前栈帧出栈，因此退出可以执行的操作有：恢复上层方法的局部变量表和操作数栈，把返回值(如果有的话)压如调用者的操作数栈中，调整PC计数器的值以指向方法调用指令后的下一条指令。

一般来说，方法正常退出时，调用者的PC计数值可以作为返回地址，栈帧中可能保存此计数值。而方法异常退出时，返回地址是通过异常处理器表确定的，栈帧中一般不会保存此部分信息。

### 本地方法栈(独立内存)



本地方法栈(Native Method Stock)与虚拟机栈所发挥的作用是非常相似的，区别是虚拟机栈为虚拟机执行Java方法(也就是字节码)服务，而本地方法栈则为虚拟机使用到的Native方法(百度说是Java中声明的可调用的C/C++实现的方法)服务

### 运行时常量池

运行时常量池(Runtime Constant Pool)是方法区的一部分。

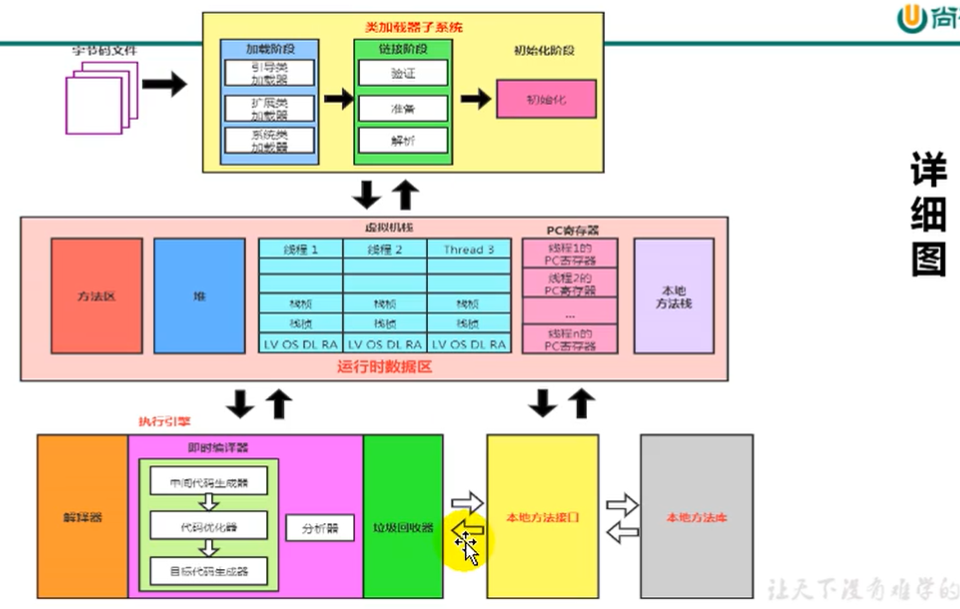
Class文件中除了有类的版本、字段、方法、接口等描述信息外，还有一项信息是常量池，用于存放编译期生成的各种字面量和符号引用，这部分内容将在类加载后进入方法区的运行时常量池中存放。

扩展:

字面量：比较接近于Java语言层面的常量概念，如文本字符串、声明为final的常量值等。

符号引用：属于编译原理方法的概念，包括以下三类常量：

1.类和接口的全限定名 2.字段的名称和描述符 3.方法的名称和描述符



## 类加载子系统

