**Java虚拟机编译过程：**

①.从磁盘读取到内存中

        ②.经过JDK编译器,解释器等编译工具进行编译生成\*.Class文件(或者类库中的Class文件)

        ③.将加载的Class文件,通过JVM(Java虚拟机)的跨平台性,可以在不同的操作系统进行运行被初始化

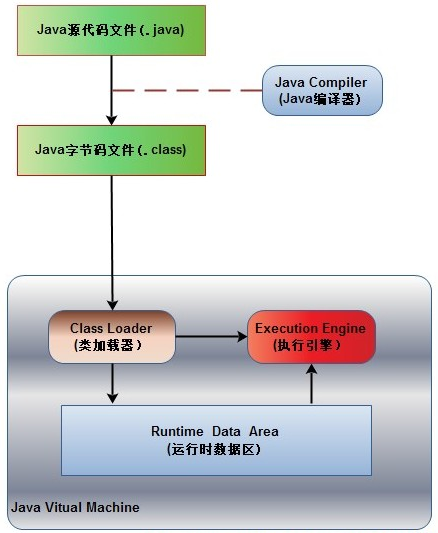
        ④.Class文件首先在内存的(方法区)类加载区中被加载的同时,系统会自动生成一个Class类的对象(反射,针对这个对象,可以获取加载类的所有成员即任何一切)

        ⑤.被加载的同时,也是被实例化(创建对象),

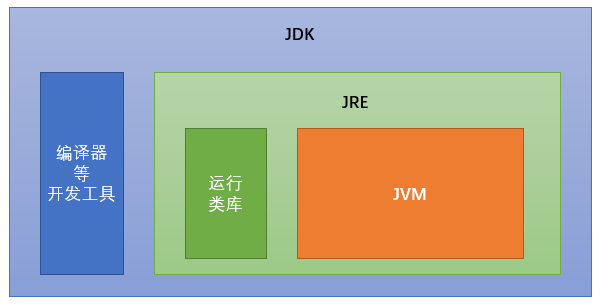
        ⑥.Class文件的静态变量会优先在类被加载的同时给加载到方法区的类加载区

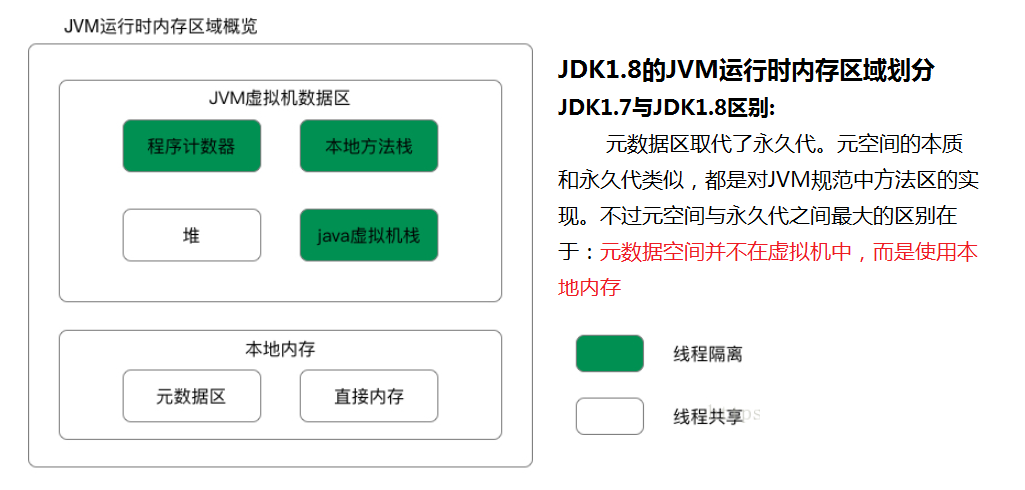
        ⑦.成员变量,初始化方法,也会随后在堆内存中进行加载

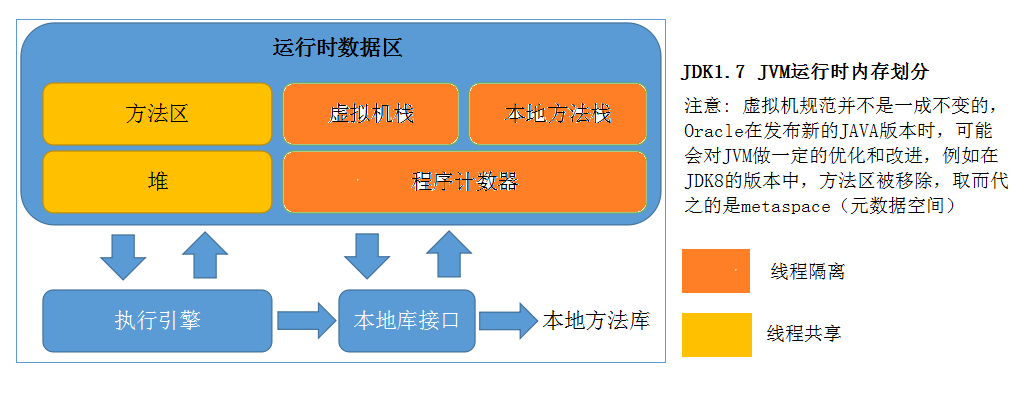
        ⑧.成员方法,局部变量,会再栈内存中被进行加载运行



**JDK与JRE区别：**







**程序计数器：**

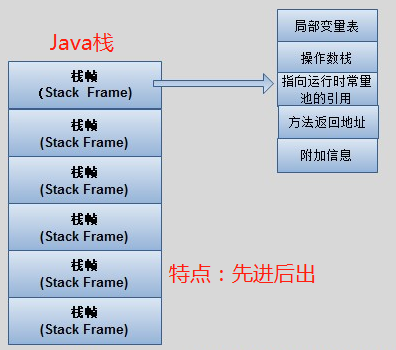
**理解：**程序计数器（Program Counter Register），也有称作PC寄存器，CPU中的寄存器，它保存的是程序当前执行的指令的地址（也可以说保存下一条指令的所在存储单元的地址），当CPU需要执行指令时，需要从程序计数器中得到当前需要执行的指令所在存储单元的地址，然后根据得到的地址获取到指令，在得到指令之后，程序计数器便自动加1或者根据转移指针得到下一条指令的地址，如此循环，直至执行完所有的指令。

**JVM中：** 在JVM中，多线程情况下是通过线程抢占CPU资源，轮流切换来获得CPU执行时间的，在任意时刻，一个CPU的内核只会执行一条线程中的指令，因此为了能够使得每个线程都在线程切换后能够恢复在切换之前的程序执行位置，每个线程都需要有自己独立的程序计数器，并且不能互相被干扰，否则就会影响到程序的正常执行次序。因此，程序计数器是每个线程所私有的

**本地方法栈：**

理解：本地方法栈和JAVA栈的作用和原理非常相似，区别是Java栈是为执行Java方法服务的，而本地方法栈则是为执行本地方法（Native Method）服务的。在HotSpot虚拟机中直接把本地方法栈和Java栈合二为一。

**Java栈：**



**理解：**JVM执行Java代码所使用的栈。

1. Java栈中存放的是一个个的栈帧，每个栈帧对应一个被调用的方法，在栈帧中包括局部变量表(Local Variables)、操作数栈(Operand Stack)、指向当前方法所属的类的运行时常量池（运行时常量池的概念在方法区部分会谈到）的引用(Reference to runtime constant pool)、方法返回地址(Return Address)和一些额外的附加信息。
2. 当线程执行一个方法时，就会随之创建一个对应的栈帧，并将建立的栈帧压栈。当方法执行完毕之后，便会将栈帧出栈。因此可知，线程当前执行的方法所对应的栈帧必定位于Java栈的顶部。（在使用递归方法的时候容易导致栈内存溢出的现象）
   1. **局部变量表：**

就是用来存储方法中的局部变量（包括在方法中声明的非静态变量以及函数形参）。对于基本数据类型的变量，则直接存储它的值，对于引用类型的变量，则存的是指向对象的引用。局部变量表的大小在编译器就可以确定其大小了，因此在程序执行期间局部变量表的大小是不会改变的。

1. **操作数栈**:

栈最典型的一个应用就是用来对表达式求值。当一个线程执行方法的过程中，实际上就是不断执行语句的过程，而归根到底就是进行计算的过程。因此可以这么说，程序中的所有计算过程都是在借助于操作数栈来完成的。

1. **指向运行时常量池的引用**：

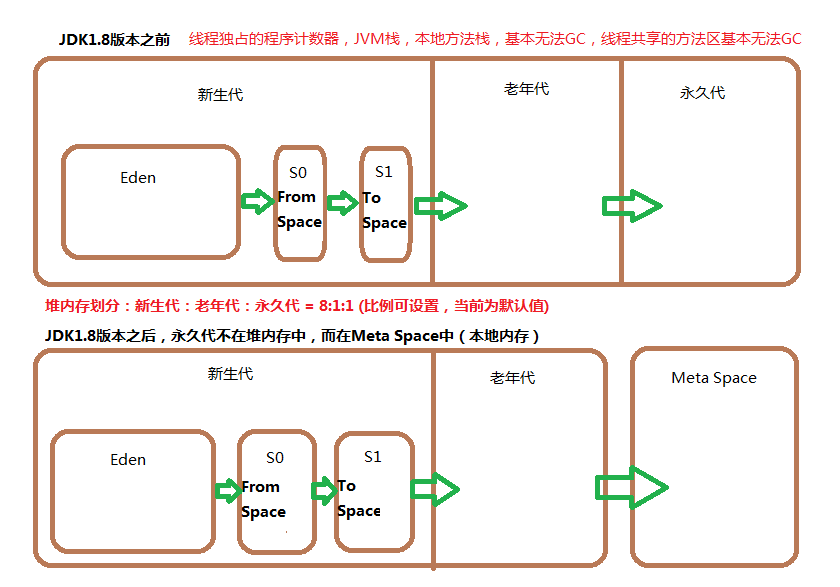
因为在方法执行的过程中有可能需要用到类中的常量，所以必须要有一个引用指向运行时常量。

1. **方法返回地址**:

当一个方法执行完毕之后，要返回之前调用它的地方，因此在栈帧中必须保存一个方法返回地址。

由于每个线程正在执行的方法可能不同，因此每个线程都会有一个自己的Java栈，互不干扰。

**JAVA堆：**



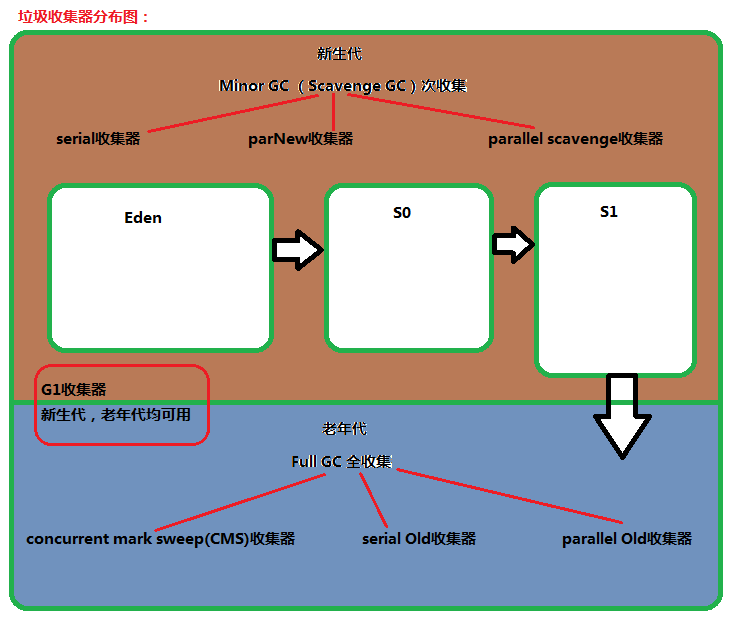
**理解：**Java中堆是用来存储对象本身以及数组（数组及对象的引用是存放在Java栈中的），Java堆基本上不用区关心内存的释放问题，Java的垃圾回收机制会自动进行处理，因此这部分空间也是Java垃圾收集器管理的主要区域，另外，堆是被所有线程共享的，在JVM中只有唯一一个堆。（JVM执行Java代码所使用的堆）

**垃圾回收常用算法：**

1. 引用计数法：
2. 复制算法：
3. 标记清除法：
4. 标记整理法：

**JVM中（scavengeGC）次收集和(FullGC)全收集区别：**

1. scavengeGC指发生在新生代的GC，因为JAva对象大多数都是朝生夕死，所以次收集非常频繁，回收速度也较快，当Eden空间不足以为对象分配内存时，会触发ScavengeGC。
2. FullGC|MajorGC指发生在老年代GC，出现了FullGC一般会伴随着至少一次的MinorGC(老年代的对象大部分是MinorGC过程中从新生代进入老年代的)，FullGC速度一般会比MinorGC慢10倍以上，当老年代内存不足或者显式调用System.gc()方法时，会触发FullGC.



**方法区：**

**理解：**

1. 方法区在JVM中是非常重要的区域，它与堆一样，是被线程所共享的区域，在方法区中，存储了每个类的信息（类的名称，方法信息，字段信息），静态变量，常量以及编译器编译后的代码等。（class文件在内存中存放的位置）
2. 在class文件中除了类的字段，方法，接口等描述信息外，还有一项信息是常量池，用来存储编译期间生成的字面量和符号引用。
3. 在方法区中有个非常重要的部分就是运行时常量池，它是每个类或接口的常量池的运行时表示形式，在类和接口被加载到JVM后，对应的运行时常量池就被创建出来，当然并非class文件常量池中的内容才能进入运行时常量池，在运行期间也可将新的常量放入运行时常量池中，比如String的intern方法。
4. 在JVM规范中，没有强制要求方法区必须实现垃圾回收。也习惯将方法区称为“永久待”，是因为HotSpot虚拟机以永久代来实现方法区，从而JVM的垃圾收集器可以像管理堆区一样管理这部分区域，从而不需要专门为这部分设计垃圾回收机制。不过JDK1.7之后，HotSpot虚拟机便将运行时常量池从永久代移除了。也就是本地内存中。

**本地内存**：（元数据区，直接内存）直接内存是堆外内存，不受JVM限制。