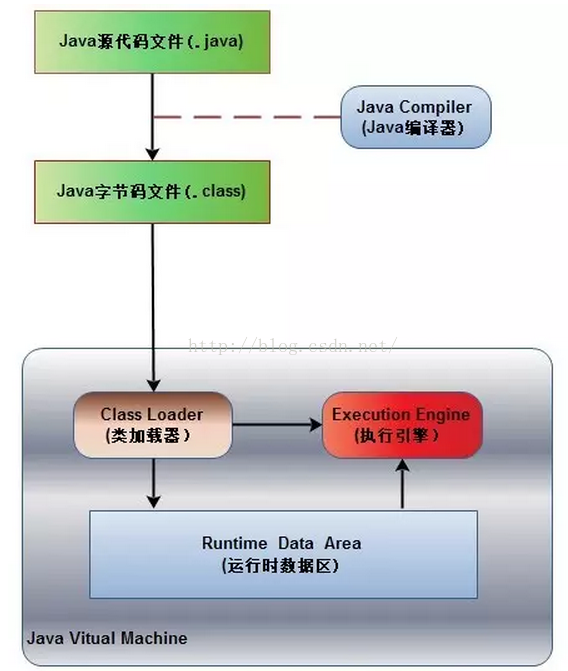
**JVM——Java虚拟机架构**

1. **前言**

Java虚拟机（Java virtualmachine）实现了Java语言最重要的特征：即平台无关性。

平台无关性原理：编译后的Java程序（.class文件）由JVM执行。JVM屏蔽了与具体平台相关的信息，使程序可以在多种平台上不加修改地运行。Java虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。因此实现Java平台无关性。

1. JVM结构图



JVM=

类加载器classloader+执行引擎executionengine+运行时数据区域runtime data area

1. 首先Java源代码文件被Java编译器便以为字节码文件，
2. 然后JVM中的**类加载器**加载完毕之后，交由JVM**执行引擎**执行，
3. 在整个程序执行过程中，JVN中的**运行时数据区**（内存）会用来存储程序执行期间需要用到的数据和相关信息。

因此，在Java中我们常常说到的内存管理就是针对这段空间进行管理（如何分配和回收空间）。

1. **ClassLoader**

classloader把硬盘上的class文件加载到JVM中的运行时数据区域，但是它不负责这个类文件能否执行，而这个是执行引擎负责的。

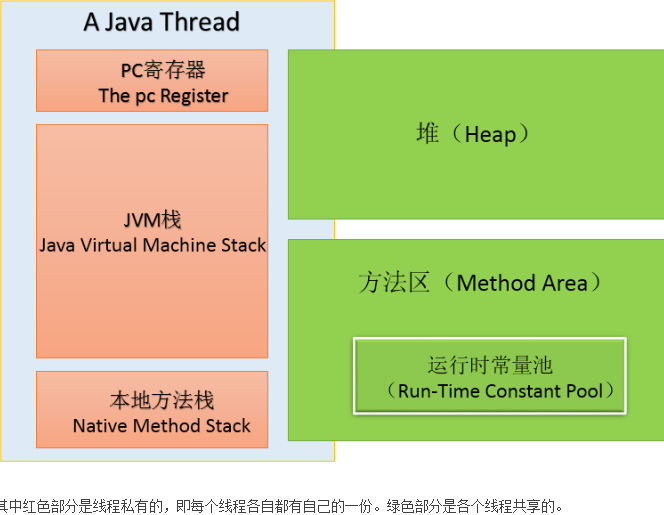
（类加载器的组织结构，加载类的机制原理会在JVM——类加载器总结一文中描述。双亲委派模型以及自定义类加载器会在JVM——自定义类加载器一文中描述。）

1. **执行引擎**

作用：执行字节码，或者执行本地方法。

1. **Runtime Data Area**

JVM在运行期间，在运行时数据区对JVM内存空间的划分和分配，划分为了一下几个区域来存储。



注：JDK1.7已经把常量池移到堆里面了！

1. PC计数器（The pc Register）
2. 每一个Java线程都有一个PC寄存器，用以记录比如在线程切换回来后恢复到正确的执行位置。
3. 如该线程正在执行一个Java方法，则计数器记录的是正在执行的虚拟机字节码地址，如执行native原生方法，则计数器值为空。
4. 此内存区域是唯一一个在JVM中没有规定任何OutOfMemoryError情况的区域。

（2）JVM栈（Java Virtual MachineStacks）

1. JVM栈是线程私有的，并且生命周期与线程相同。并且当线程运行完毕后，相应内存也就被自动回收。
2. 栈里面存放的元素叫栈帧，每个方法从调用到执行结束，其实是对应一个栈帧的入栈和出栈。栈帧用于存储执行方法时的一些数据，如局部变量、操作数栈（执行引擎计算时需要），方法出口等等。
3. 这个区域可能有两种异常：如果线程请求的栈深度大于虚拟机所允许的深度，将抛出StackOverflowError异常（如：将一个函数反复递归自己，最终会出现这种异常）。如果JVM栈可以动态扩展（大部分JVM是可以的），当扩展时无法申请到足够内存则抛出OutOfMemoryError异常。

（3）本地方法栈（Native Method Stacks）

1. 本地方法栈与虚拟机栈所发挥的作用很相似，他们的区别在于虚拟机栈为执行Java代码方法服务，而本地方法栈是为Native方法服务。
2. 和JVM栈一样，这个区域也会抛出StackOverflowError和OutOfMemoryError异常。

（4）方法区（Method Area）

1. 方法区域是全局共享的，比如每个线程都可以访问同一个类的静态变量。在方法区中，存储了已被JVM加载的类的信息、静态变量、编译器编译后的代码等。如，当程序中通过getName、isInterface等方法来获取信息时，这些数据来源于方法区。
2. 由于使用反射机制的原因，虚拟机很难推测哪个类信息不再使用，因此这块区域的回收很难！另外，对这块区域主要是针对常量池回收，值得注意的是JDK1.7已经把常量池转移到堆里面了。
3. 同样，当方法区无法满足内存分配需求时，会抛出OutOfMemoryError。

（5）运行时常量池（Runtime Constant Pool）

1） 存放类中固定的常量信息、方法引用信息等，其空间从方法区域（JDK1.7后为堆空间）中分配。

2） Class文件中除了有类的版本、字段、方法、接口等描述信息外，还有就是常量表，用于存放编译期已可知的常量，这部分内容将在类加载后进入方法区（永久代）存放。但是Java语言并不要求常量一定只有编译期预置入Class的常量表的内容才能进入方法区常量池，运行期间也可将新内容放入常量池（最典型的String.intern()方法）。

3） 当常量池无法在申请到内存时会抛出OutOfMemoryError异常，上面也分析过了。

（6）Java堆

1. Java堆是JVM所管理的最大的一块内存。它是被所有线程共享的一块内存区域，**在虚拟机启动时创建**。
2. 几乎所有的实例对象都是在这块区域中存放。（JIT编译器貌似不是这样的）。
3. Java堆是垃圾收集管理的主要战场。所有Java堆可以细分为：新生代和老年代。再细致分就是把新生代分为：Eden空间、FromSurvivor空间。JVM具体的垃圾回收机制总结请查看另一篇JVM——内存管理和垃圾回收。
4. 根据Java虚拟机规范的规定，Java堆可以处于物理上不连续的内存空间中，只要逻辑上是连续的即可。
5. 如果在堆中没有内存完成实例分配，并且堆也无法再扩展时，将会抛出OutOfMemoryError异常。

**5.堆和栈的区别**

这是一个非常常见的面试题，主要从以下几个方面来回答。

1. 各司其职

最主要的区别就是：

栈内存用来存储局部变量和方法调用信息；

而堆内存用来存储Java中的对象。无论是成员变量、局部变量还是类变量，它们指向的对象都存储在堆内存中。

1. 空间大小

栈的内存要远远小于堆内存，如果你使用递归的话，那么你的栈很快就会很快就会充满并产生StackOverFlowError。

关于如何设置堆栈内存的大小，可以查看JVM——内存管理和垃圾回收。

1. 独有还是独享

栈内存归属于线程的私有内存，每个线程都会有一个栈内存，其存储的变量只能在其所属线程中可见。

而堆内存中的对象对所有线程可见，可以被所有线程访问。

1. 异常错误

如果线程请求的栈深度大于虚拟机所允许的深度，将抛出StackOverFlowError异常。

如果JVM栈可以动态扩展（大部分JVM是可以的），当扩展时无法申请到足够内存则抛出OutOfMemoryError异常。

而堆内存没有可用的空间存储生成的对象，JVM会抛出java.lang.OutOfMemoryError。