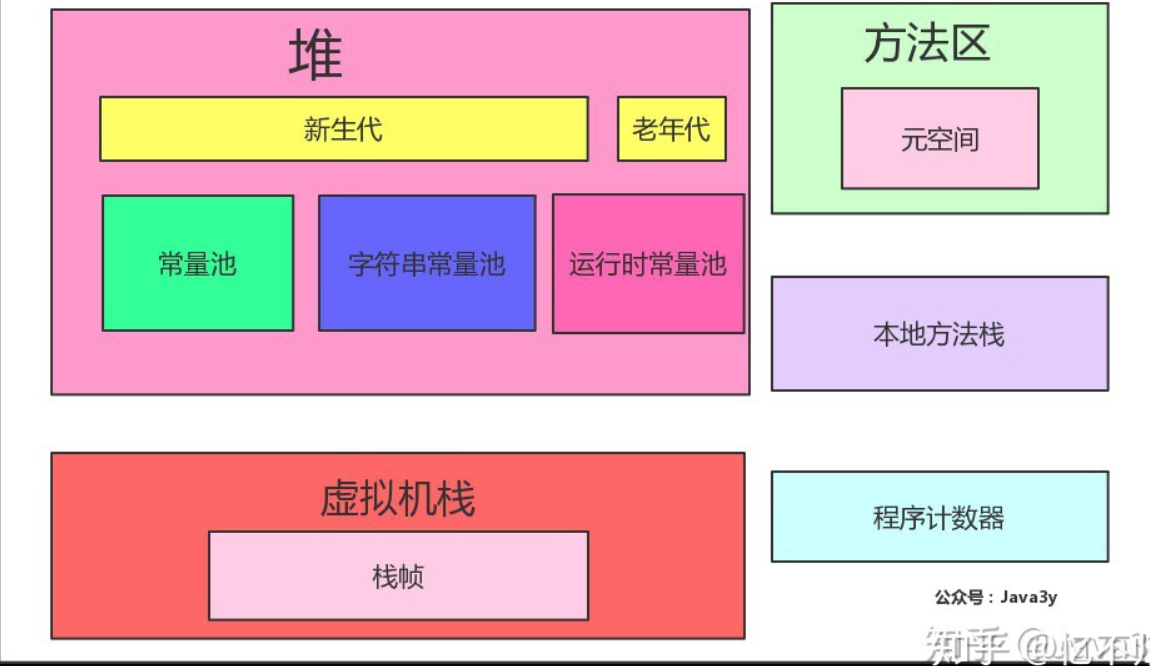
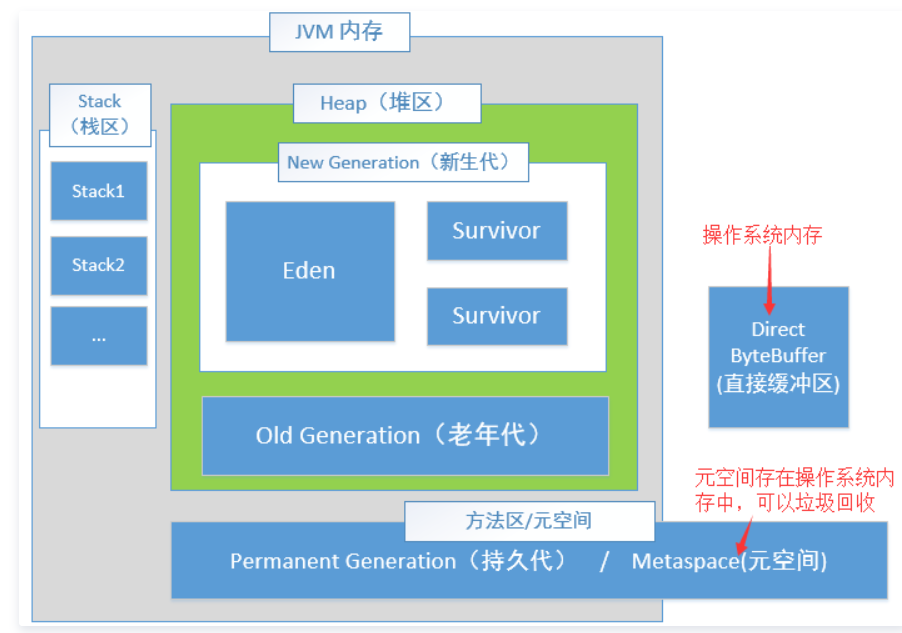


jdk1.8





**Heap（堆）内存大小设置**

-Xms512m

设置JVM堆初始内存为512M

-Xmx1g

设置JVM堆最大可用内存为1G

**New Generation（新生代）内存大小设置**

-Xmn256m

设置JVM的新生代内存大小（－Xmn 是将NewSize与MaxNewSize设为一致。256m）,同下面两个参数 -XX:NewSize=256m -XX:MaxNewSize=256m

**还可以通过新生代和老年代内存的比值来设置新生代大小** -XX:NewRatio=3

设置新生代（包括Eden和两个Survivor区）与老年代的比值（除去持久代）。设置为3，则新生代与老年代所占比值为1：3，新生代占整个堆栈的1/4

**Survivor内存大小设置**

-XX:SurvivorRatio=8

设置为8,则两个Survivor区与一个Eden区的比值为2:8,一个Survivor区占整个新生代的1/10

**Eden内存大小设置**

新生代减去2\*Survivor的内存大小就是Eden的大小。

## Stack(栈)内存大小设置

-Xss1m 每个线程都会产生一个栈。在相同物理内存下，减小这个值能生成更多的线程。如果这个值太小会影响方法调用的深度。

## Permanent Generation（持久代）内存大小设置

方法区内存分配（JDK8以前的版本使用，JDK8以后没有持久代了，使用的MetaSpace） -XX: PermSize=128m 设置持久代初始内存大小128M -XX:MaxPermSize=512m 设置持久代最大内存大小512M

## Metaspace（元空间）内存大小设置

元空间（Metaspace）(JDK8) -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=512m（JDK8），JDK8的持久代几乎可用完机器的所有内存，同样设一个128M的初始值，512M的最大值保护一下。

1. 默认情况下，类元数据分配受到可用的本机内存容量的限制（容量依然取决于你使用32位JVM还是64位操作系统的虚拟内存的可用性）。
2. 一个新的参数 (MaxMetaspaceSize)可以使用。允许你来限制用于类元数据的本地内存。如果没有特别指定，元空间将会根据应用程序在运行时的需求动态设置大小。



* Java堆是Java虚拟机所管理内存最大、被所有线程共享的一块区域，目的是用来存放对象，基本上所有的对象实例和数组都在堆上分配（不是绝对）。Java堆也是垃圾回收器管理的主要区域。
* 元空间在取代永久代之后，唯一的不同之处在于，元空间所占用的内存为本地内存，不在是JVM申请的内存，所以只要机器还有内存，元空间就可以一直申请使用。由下面的参数可以看出，并没有设置元空间最大内存的参数，就是可以充分利用机器的内存。
* 永久代是HotSpot虚拟机对虚拟机规范中方法区的一种实现方式。
* 对于Java8， HotSpots取消了永久代，那么是不是也就没有方法区了呢？当然不是，方法区是一个规范，规范没变，它就一直在。那么取代永久代的就是元空间。它可永久代有什么不同的？存储位置不同，永久代物理是是堆的一部分，和新生代，老年代地址是连续的，而元空间属于本地内存；存储内容不同，元空间只存储类和类加载器的元数据信息，静态变量和常量池等并入堆中。相当于永久代的数据被分到了堆和元空间中。
* 方法区 - java虚拟机规范的运行时数据区

**注意：**

在JDK1.8中，使用元空间代替永久代来实现方法区，但是方法区并没有改变，所谓"Your father will always be your father"，变动的只是方法区中内容的物理存放位置。正如上面所说，类型信息（元数据信息）等其他信息被移动到了元空间中；但是运行时常量池和字符串常量池被移动到了堆中。但是不论它们物理上如何存放，逻辑上还是属于方法区的。

JDK1.8中字符串常量池和运行时常量池逻辑上属于方法区，但是实际存放在堆内存中，因此既可以说两者存放在堆中，也可以说两则存在于方法区中，这就是造成误解的地方。

1. 常量

在Java程序里，常量用关键字static final修饰，常量又分为：

* 编译期常量
* 运行时常量

1. 运行时常量池：

在Java虚拟机规范中，运行时常量池（Runtime Constant Pool）用于存放编译期生成的各种字面量和符号引用，是方法区的一部分。但是Java虚拟机规范对其没有做任何细节的要求，所以不同虚拟机实现商可以按照自己的需求来实现该区域，比如在 HotSpot 虚拟机实现中，就将运行时常量池移到了堆中。

①、存放字面量、符号引用、直接引用

通常来说，该区域除了保存Class文件中描述的引用外，还会把翻译出来的直接引用也存储在运行时常量池，并且Java语言并不要求常量一定只能在编译器产生，运行期间也可能将常量放入池中，比如String类的intern()方法，当调用intern方法时，如果池中已经包含一个与该String确定的字符串相同equals(Object)的字符串，则返回该字符串。否则，将此String对象添加到池中，并返回此对象的引用。关于该方法的介绍可以看我这篇博客。

②、抛出 OutOfMemoryError 异常

运行时常量池是方法区的一部分，会受到方法区内存的限制，当常量池无法申请到内存时，会抛出该异常。

1. 本地方法栈（Native Method Stacks）作用和虚拟机栈类型，虚拟机栈执行的是Java方法，本地方法栈执行的是 Native 方法，本地方法栈也会抛出抛出 StackOverflowError 和 OutOfMemoryError 异常。注意：由于虚拟机规范并没有对本地方法栈中的方法使用语言、使用方式和数据结构强制规定，因此具体的虚拟机可以自由实现它。上图我们也给出在 HotSpot 虚拟机中，本地方法栈和虚拟机栈合为一体了。
2. Java虚拟机栈（Java Virtual Machine stack），这块区域也是线程私有的，与线程同时创建，用于存储栈帧。Java 每个方法执行的时候都会同时创建一个栈帧（Stack Frame）用于存储局部变量表、操作栈、动态链接、方法出口等信息，每一个方法被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从入栈到出栈的过程。

常量池：

即class文件常量池，是class文件的一部分，用于保存编译时确定的数据

运行时常量池：

　　Java语言并不要求常量一定只能在编译期产生，运行期间也可能产生新的常量，这些常量被放在运行时常量池中。

**类加载后，常量池中的数据会在运行时常量池中存放**！

这里所说的常量包括：基本类型包装类（包装类不管理浮点型，整形只会管理-128到127）和String（也可以通过String.intern()方法可以强制将String放入常量池）

字符串常量池：

　　HotSpot VM里，记录interned string的一个全局表叫做StringTable，它本质上就是个HashSet<String>。注意**它只存储对java.lang.String实例的引用，而不存储String对象的内容**

##### 字符串常量池（string pool）

字符串常量池里的内容是在类加载完成，经过验证，准备阶段之后在堆中生成字符串对象实例，然后将该字符串对象实例的引用值存到string pool中（记住：string pool中存的是引用值而不是具体的实例对象，具体的实例对象是在堆中开辟的一块空间存放的）。string pool在每个HotSpot VM的实例只有一份，被所有的类共享。在jdk1.8后，将String常量池放到了堆中

##### class常量池

当java文件被编译成class文件之后，会在class文件中生成我们所说的class常量池，class文件中除了包含**类的版本、字段、方法、接口**等描述信息外，还有一项信息就是常量池(constant pool table)，用于存放编译器生成的各种**字面量**(文本字符串、被声明为final的常量、基本数据类型的值)和**符号引用**(类和接口的全限定名、字段的名称和描述符、方法的名称和描述符)。

**jvm的方法区里存放着类的版本，字段，方法，接口和常量池。常量池里存储着字面量和符号引用。**

##### 运行时常量池

当类加载到内存中后，jvm就会将class常量池中的内容存放到运行时常量池中，由此可知，运行时常量池也是每个类都有一个。在上面我也说了，class常量池中存的是字面量和符号引用，也就是说他们存的并不是对象的实例，而是对象的符号引用值。而经过解析（resolve）之后，也就是把符号引用替换为直接引用，解析的过程会去查询字符串常量池，也就是我们上面所说的string pool，以保证运行时常量池所引用的字符串与字符串常量池中所引用的是一致

### 静态常量池

Java程序要运行时，需要编译器先将源代码文件编译成字节码（.class)文件，然后在由JVM解释执行。

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(Constant pool table)，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运行时常量池中存放。

静态常量池就是上面说的class文件中的常量池。class常量池是在编译时每个class文件中都存在。不同的符号信息放置在不同标志的常量表中。

### 运行时常量池

运行时常量池就是将编译后的类信息放入方法区中，也就是说它是方法区的一部分。

运行时常量池用来动态获取类信息，包括：class文件元信息描述、编译后的代码数据、引用类型数据、类文件常量池等。

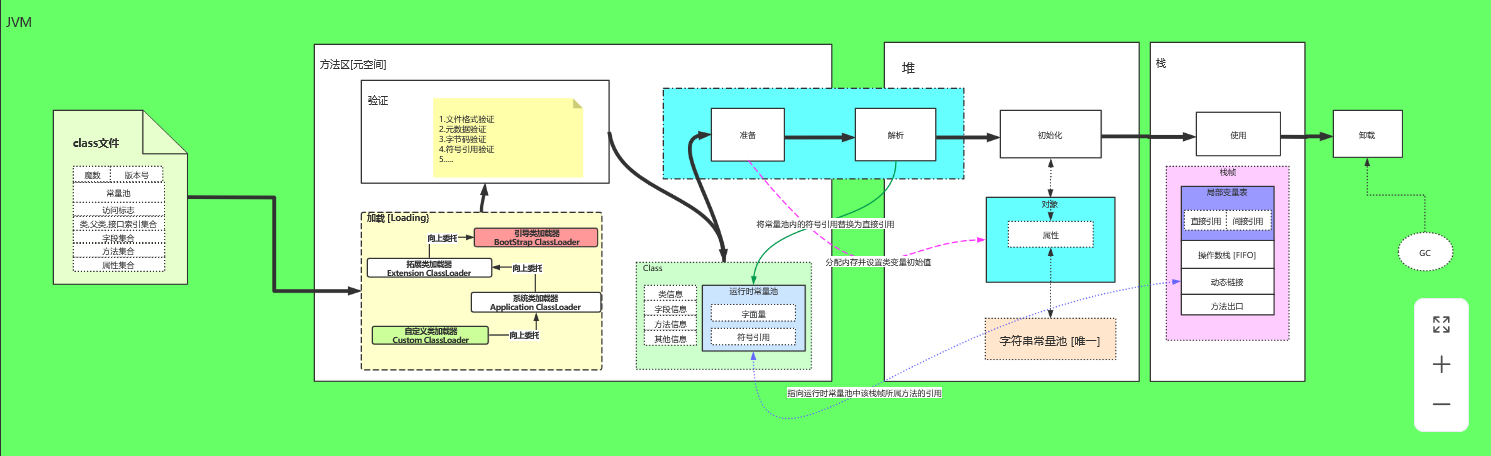
运行时常量池是在类加载完成之后，将每个class常量池中的符号引用值转存到运行时常量池中。每个class都有一个运行时常量池，类在解析之后将符号引用替换成直接引用，与全局常量池中的引用值保持一致。

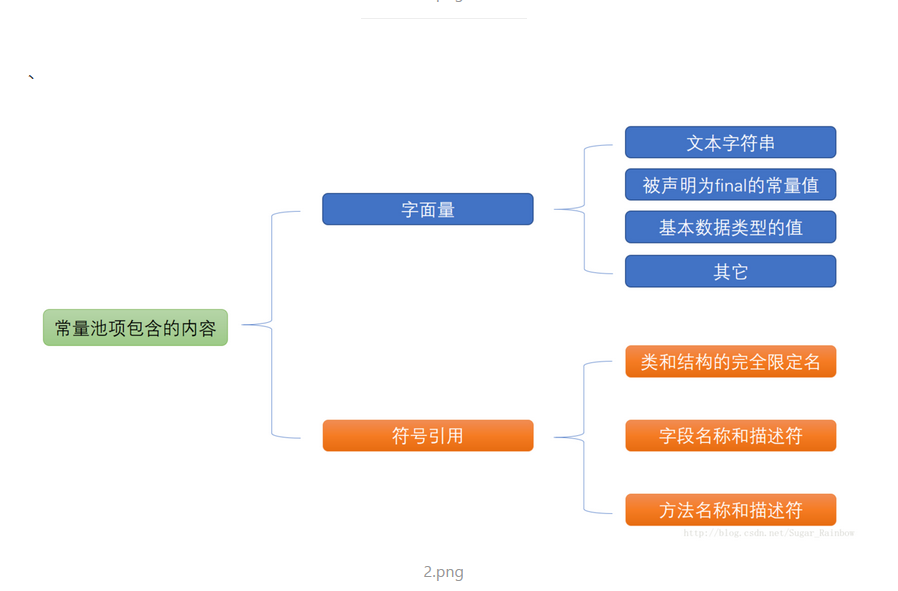
运行时常量池相对于class文件常量池的另外一个特性是具备动态性，java语言并不要求常量一定只有编译器才产生，也就是并非预置入class文件中常量池的内容才能进入方法区运行时常量池，运行期间也可能将新的常量放入池中。

### 字符串常量池

字符串池里的内容是在类加载完成，经过验证、准备阶段之后存放在字符串常量池中。关于字符串常量池的具体实现我们这里先不展开，后面用专门的文章来进行讲解。

字符串常量池的处理机制我们前面文章已经讲到，只会存储一份，被所有的类共享。基本流程是：创建字符串之前检查常量池中是否存在，如果存在则获取其引用，如果不存在则创建并存入，返回新对象引用





方法区在JVM中也是一个非常重要的区域，它与堆一样，是被**线程共享**的区域。 在方法区中，存储了每个类的信息（包括类的名称、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。

