**一、方法区**

　　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。

　　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。

　　在方法区中有一个非常重要的部分就是运行时常量池，它是每一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。

　　在JVM规范中，没有强制要求方法区必须实现垃圾回收。很多人习惯将方法区称为“永久代”，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。不过自从JDK7之后，Hotspot虚拟机便将运行时常量池从永久代移除了。

**二、永久代与方法区**

　　涉及到内存模型时，往往会提到永久代，那么它和方法区又是什么关系呢？《Java虚拟机规范》只是规定了有方法区这么个概念和它的作用，并没有规定如何去实现它。那么，在不同的 JVM 上方法区的实现肯定是不同的了。 同时大多数用的JVM都是Sun公司的HotSpot。在HotSpot上把GC分代收集扩展至方法区，或者说使用永久代来实现方法区。因此，我们得到了结论，永久代是HotSpot的概念，方法区是Java虚拟机规范中的定义，是一种规范，而永久代是一种实现，一个是标准一个是实现。其他的虚拟机实现并没有永久带这一说法。在1.7之前在(JDK1.2 ~ JDK6)的实现中，HotSpot 使用永久代实现方法区，HotSpot 使用 GC分代来实现方法区内存回收，可以使用如下参数来调节方法区的大小:

|  |  |
| --- | --- |
| 1  2  3  4  5 | -XX:PermSize  方法区初始大小  -XX:MaxPermSize  方法区最大大小  超过这个值将会抛出OutOfMemoryError异常:java.lang.OutOfMemoryError: PermGen |

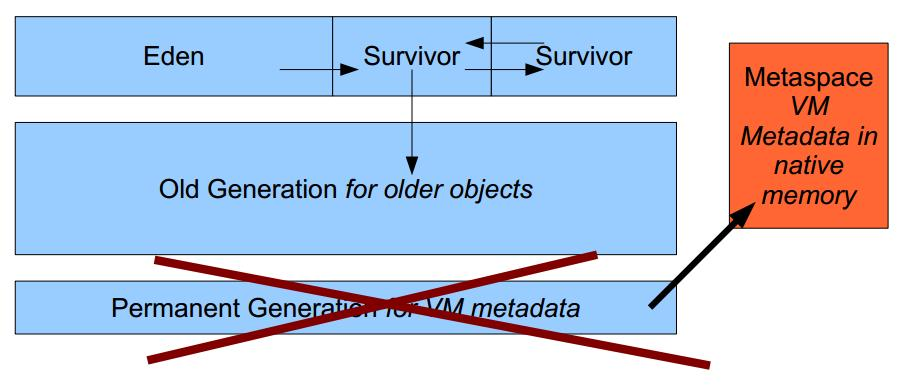
**三、元空间与方法区**

　　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，元空间存储类的元信息，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。

**四、JVM 1.8 永久代—元空间的变动**

JDK8 HotSpot JVM 将移除永久区，使用本地内存来存储类元数据信息并称之为：元空间（Metaspace）

以下是JVM内存模型中方法区的变动



　　1.新生代：Eden+From Survivor+To Survivor

　　2.老年代：OldGen

　　3.永久代（方法区的实现） : PermGen----->替换为Metaspace(本地内存中)

　　方法区和“PermGen space”又有着本质的区别。前者是 JVM 的规范，而后者则是 JVM 规范的一种实现，并且只有 HotSpot 才有 “PermGen space”，而对于其他类型的虚拟机，如 JRockit（Oracle）、J9（IBM） 并没有“PermGen space”。由于方法区主要存储类的相关信息，所以对于动态生成类的情况比较容易出现永久代的内存溢出

　　元空间的本质和永久代类似，都是对JVM规范中方法区的实现。不过元空间与永久代之间最大的区别在于：元空间并不在虚拟机中，而是使用本地内存。因此，默认情况下，元空间的大小仅受本地内存限制，但可以通过以下参数来指定元空间的大小：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | -XX:MetaspaceSize，初始空间大小，达到该值就会触发垃圾收集进行类型卸载，同时GC会对该值进行调整：如果释放了大量的空间，就适当降低该值；如果释放了很少的空间，那么在不超过MaxMetaspaceSize时，适当提高该值。  -XX:MaxMetaspaceSize，最大空间，默认是没有限制的。    　除了上面两个指定大小的选项以外，还有两个与 GC 相关的属性：  -XX:MinMetaspaceFreeRatio，在GC之后，最小的Metaspace剩余空间容量的百分比，减少为分配空间所导致的垃圾收集  -XX:MaxMetaspaceFreeRatio，在GC之后，最大的Metaspace剩余空间容量的百分比，减少为释放空间所导致的垃圾收集 |

　　《Java虚拟机规范(JavaSE7)》中也说了方法区是堆的逻辑组成部分。   
　　实际上JDK1.7中，存储在永久代的部分数据就已经转移到了Java Heap或者是 Native Heap。但永久代在JDK1.8才被移除

**五、移除永久代的影响**

　　由于类的元数据分配在本地内存中，元空间的最大可分配空间就是系统可用内存空间。因此，我们就不会遇到永久代存在时的内存溢出错误，也不会出现泄漏的数据移到交换区这样的事情。最终用户可以为元空间设置一个可用空间最大值，如果不进行设置，JVM会自动根据类的元数据大小动态增加元空间的容量。

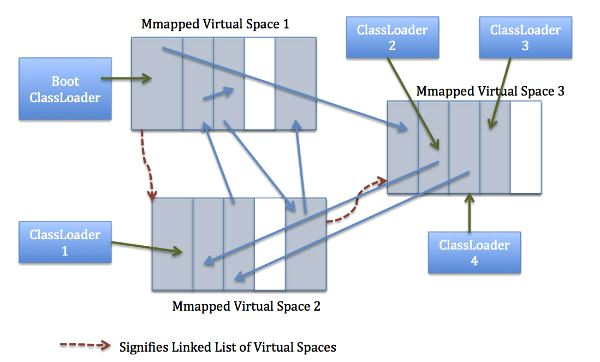
　　注意：永久代的移除并不代表自定义的类加载器泄露问题就解决了。因此，你还必须监控你的内存消耗情况，因为一旦发生泄漏，会占用你的大量本地内存，并且还可能导致交换区交换更加糟糕。

**六、元空间内存管理**

　　元空间的内存管理由元空间虚拟机来完成。先前，对于类的元数据我们需要不同的垃圾回收器进行处理，现在只需要执行元空间虚拟机的C++代码即可完成。在元空间中，类和其元数据的生命周期和其对应的类加载器是相同的。话句话说，只要类加载器存活，其加载的类的元数据也是存活的，因而不会被回收掉。

准确的来说，每一个类加载器的存储区域都称作一个元空间，所有的元空间合在一起就是我们一直说的元空间。当一个类加载器被垃圾回收器标记为不再存活，其对应的元空间会被回收。在元空间的回收过程中没有重定位和压缩等操作。但是元空间内的元数据会进行扫描来确定Java引用。

　　元空间虚拟机负责元空间的分配，其采用的形式为组块分配。组块的大小因类加载器的类型而异。在元空间虚拟机中存在一个全局的空闲组块列表。当一个类加载器需要组块时，它就会从这个全局的组块列表中获取并维持一个自己的组块列表。当一个类加载器不再存活，那么其持有的组块将会被释放，并返回给全局组块列表。类加载器持有的组块又会被分成多个块，每一个块存储一个单元的元信息。组块中的块是线性分配（指针碰撞分配形式）。组块分配自内存映射区域。这些全局的虚拟内存映射区域以链表形式连接，一旦某个虚拟内存映射区域清空，这部分内存就会返回给[操作系统](http://lib.csdn.net/base/operatingsystem)。



　　运行时常量池在JDK1.6及之前版本的JVM中是方法区的一部分，而在HotSpot虚拟机中方法区放在了”永久代(Permanent Generation)”。所以运行时常量池也是在永久代的。   
但是JDK1.7及之后版本的JVM已经将字符串常量池从方法区中移了出来，在Java 堆（Heap）中开辟了一块区域存放字符串常量池。

　　String.intern()是一个Native方法，它的作用是：如果运行时常量池中已经包含一个等于此String对象内容的字符串，则返回常量池中该字符串的引用；如果没有，则在常量池中创建与此String内容相同的字符串，并返回常量池中创建的字符串的引用。

**七、变动原因**

字符串存在永久代中，现实使用中易出问题, 由于永久代内存经常不够用或发生内存泄露，爆出异常    java.lang.OutOfMemoryError: PermGen类及方法的信息等比较难确定其大小，因此对于永久代的大小指定比较困难，太小容易出现永久代溢出，太大则容易导致老年代溢出。

永久代会为 GC 带来不必要的复杂度，并且回收效率偏低。

 This is part of the JRockit and Hotspot convergence effort. JRockit customers do not need to configure the permanent generation (since JRockit does not have a permanent generation) and are accustomed to not configuring the permanent generation.

 即：移除永久代是为融合HotSpot JVM与 JRockit VM而做出的努力，因为JRockit没有永久代，不需要配置永久代。