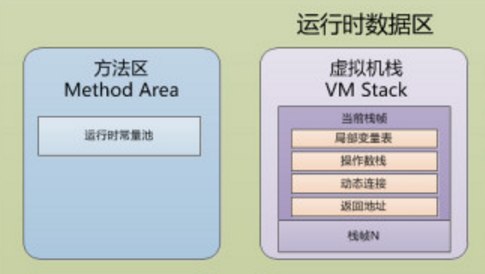
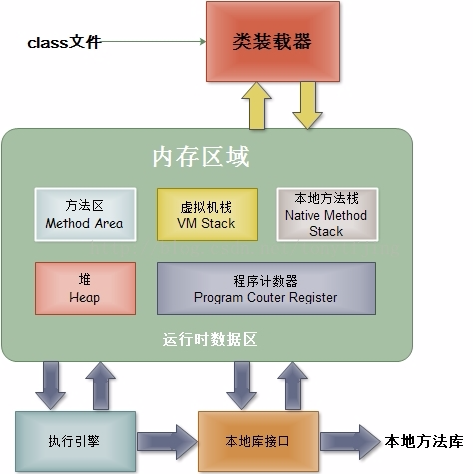
JVM的内存划分：

首先来看一下JVM内存结构，它是由堆、栈、本地方法栈、方法区等部分组成。



JVM中内存

JVM中内存通常划分为三个部分，分别为堆内存与栈内存，程序计数器。

栈内存主要用执行线程方法存放本地临时变量与线程中方法执行时候需要的引用对象地址。

JVM所有的对象信息都存放在堆内存中，相比栈内存，堆内存大的多，所以JVM一直通过对堆内存划分不同的功能区块实现对堆内存中对象管理。

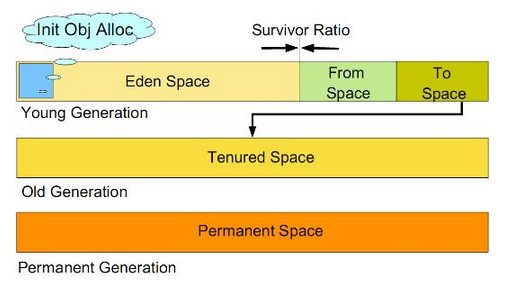
程序计数器：倘若当前执行的是 JVM 的方法，则程序计数器中保存当前执行指令的地址；倘若执行的是native 方法，则程序计数器中为空

堆内存不够最常见的错误就是OOM(OutOfMemoryError)

栈内存溢出最常见的错误就是StackOverflowError，程序有递归调用时候最容易发生

1)堆

所有通过new创建的对象的内存都在堆中分配，其大小可以通过-Xmx和-Xms来控制。堆被划分为新生代和旧生代，新生代又被进一步划分为Eden和Survivor区，最后Survivor由FromSpace和ToSpace组成，结构图如下所示：

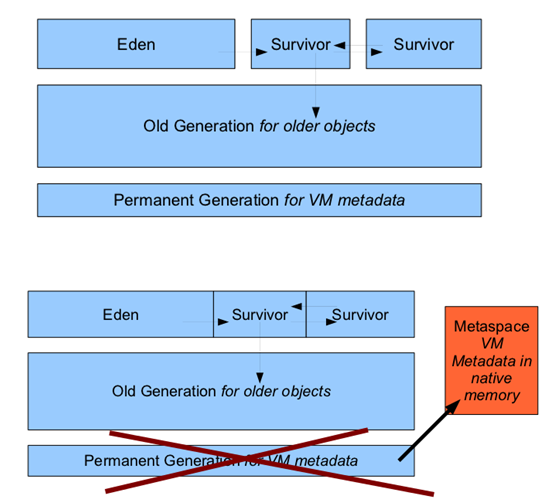


其中最上一层是年轻代，一个对象被创建以后首先被放到年轻代中的Eden内存中，如果存活期超两个Survivor之后就会被转移到老年代中(Old Generation)中。

持久代中存放着对象的方法、变量等元数据信息。通过如果永久内存不够，我们就会得到如下错误：

Java.lang.OutOfMemoryError: PermGen

而在JDK8中情况发生了明显的变化，就是一般情况下你都不会得到这个错误，原因在于JDK8中把存放元数据中的**永久内存从堆内存中移到了本地内存(native memory)中**，JDK8中JVM堆内存结构就变成了如下：



这样永久内存就不再占用堆内存，它可以通过自动增长来避免JDK7以及前期版本中常见的永久内存错误(java.lang.OutOfMemoryError: PermGen)，也许这个就是你的

JDK升级到JDK8的理由之一吧。当然JDK8也提供了一个新的设置Matespace内存大小的参数，通过这个参数可以设置Matespace内存大小，这样我们可以根据自己项目的实际情况，避免过度浪费本地内存，达到有效利用。

最后看一下详细的各个位置的功能：

首先来看一下JVM内存结构，它是由堆、栈、本地方法栈、方法区等部分组成。

1)堆

所有通过**new创建的对象的内存都在堆中分配**，其大小可以通过-Xmx和-Xms来控制。堆被划分为新生代和旧生代，新生代又被进一步划分为Eden和Survivor区，最后Survivor由FromSpace和ToSpace组成。

新建的对象优先在用Eden分配内存（大对象直接进入老年代），Eden空间不足的时候，会把存活的对象转移到Survivor中。新生代大小可以由-Xmn来控制，也可以用-XX:SurvivorRatio来控制Eden和Survivor的比例.

旧生代:用于存放新生代中经过多次垃圾回收仍然存活的对象

2)栈

虚拟机栈：每个线程执行每个方法的时候都会在栈中申请一个栈帧，每个栈帧包括局部变量区和操作数栈，用于存放此次方法调用过程中的临时变量（基本数据类型，对象引用）、参数和中间结果

本地方法栈：用于支持native方法的执行，存储了每个native方法调用的状态

4)方法区

存放了要加载的类信息、静态变量、final类型的常量、属性和方法信息。JVM用**持久代(PermanetGeneration)来存放方法区**，可通过-XX:PermSize和-XX:MaxPermSize来指定最小值和最大值。

5)程序计数器（线程私有）

当前线程所执行的字节码文件（class）的行号指示器。字节码解释器就是通过改变计数器的值来选取下一条执行的字节码指令，分支、循环、跳转、异常处理、线程恢复

6)运行时常量池（1.7就取消了）

方法区的一部分，用于存放编译器生成的各种字面量和符号引用。

**总结**

　 1、所有线程共享的内存数据区：方法区，堆。而虚拟机栈，本地方法栈和程序计数器都是线程私有的。

　　 2、存放于栈中的东西如下：  
 　　　  2.1 每个线程包含一个栈区,栈中只保存基础数据类型的对象（局部变量）和自定义对象的引用(不是对象,类似一个指针**，reference)**。**对象都存放在堆区中**。其中**基础类型数据放在方法区**，而自定义对象的数据则放到了堆区里了。比如int a = 1；则1在方法区，比如class是person，里面的hand a=“hands”，hands这个是**字段内容，也是有效信息，也是放到了堆中的对象中，为实例数据。然后这些字段所对应的类型，比如hand这些类型数据则放到了方法里，得去方法里找。**

　　　　2.2 每个栈中的数据(基础数据类型和对象引用)都是私有的，其他栈不能访问。  
 　　　  2.3 方法的形式参数，方法调用完后从栈空间回收  
 　　　  2.4 引用对象的地址，引用完后，栈空间地址立即被回收，堆空间等待GC

　　 3、存放于堆中的东西如下：

　　　　3.1 存储的全部是对象，每个对象包含一个与之对应的class信息

　　　　3.2 Jvm只有一个堆区（heap）被所有线程共享，堆区中不存放基本类型和对象引用，只存放对象本身

4、存放于方法区中的东西如下：

　　　　4.1 存放线程所执行的字节码指令

　　　　4.2 跟堆一样.被所有线程共享.方法区包含：所有的class和static变量

　　　　4.3 常量池位于方法区中，见如下图示说明

