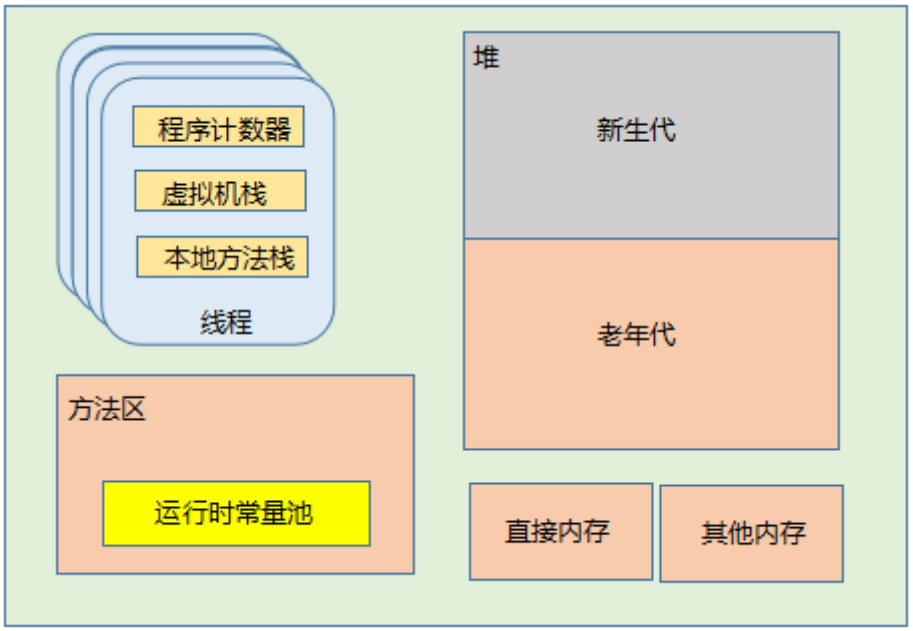
ps：**直接内存**（Direct Memory）：Java的NIO可以使用Native方法直接在java堆外分配内存，使用DirectByteBuffer对象作为这个堆外内存的引用。



OOM可能发生在哪些区域上？

OOM是指JVM的内存不够用了，同时垃圾收集器也无法提供更多的内存。不再会被使用的对象的内存不能被回收，就是内存泄露。从描述中可以看出，在JVM抛出OutOfMemoryError之前，垃圾收集器一般会出马先尝试回收内存。

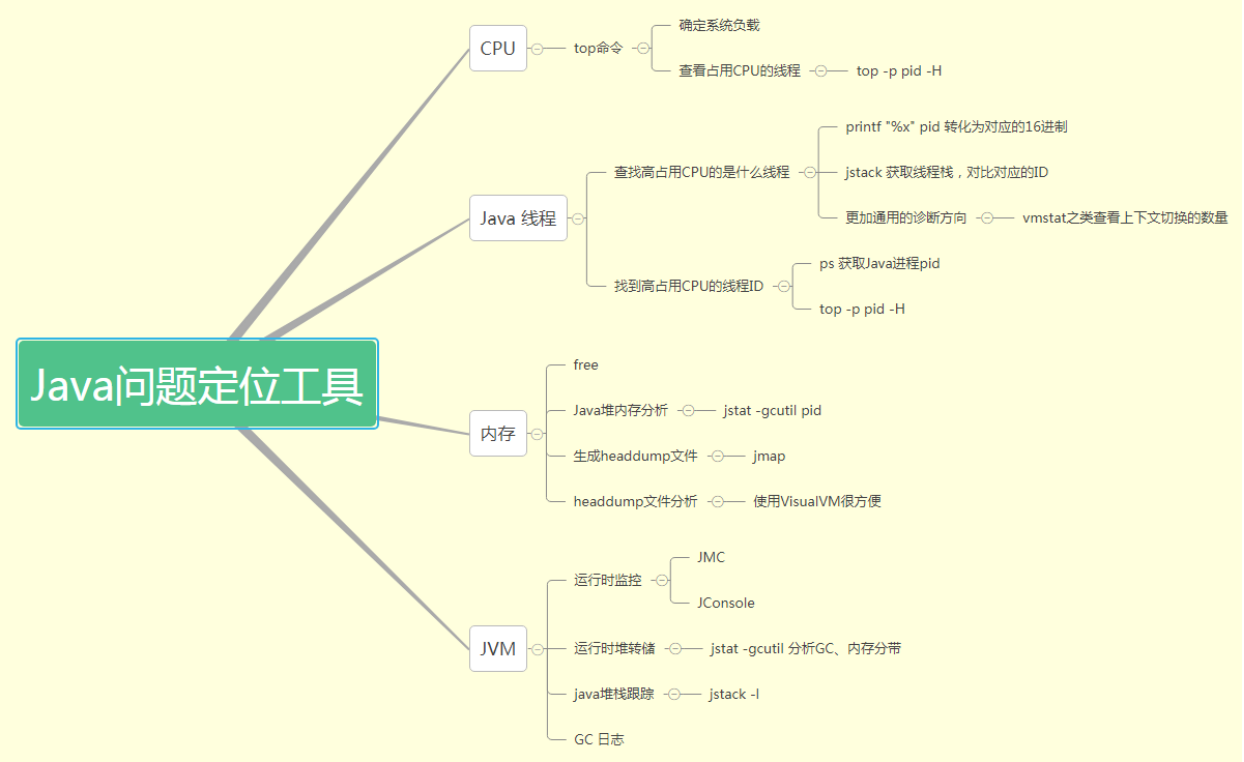
- **堆内存：堆内存不足是最常见的发送OOM的原因之一，如果在堆中没有内存完成对象实例的分配，并且堆无法再扩展时，将抛出OutOfMemoryError异常，抛出的错误信息是"java.lang.OutOfMemoryError:Java heap space"。当前主流的JVM可以通过-Xmx和-Xms来控制堆内存的大小，发生堆上OOM的可能是存在内存泄露，也可能是堆大小分配不合理。**

- **Java虚拟机栈和本地方法栈：这两个区域的区别不过是虚拟机栈为虚拟机执行Java方法服务，而本地方法栈则为虚拟机使用**

到的Native方法服务，在内存分配异常上是相同的。在JVM规范中，对Java虚拟机栈规定了两种异常：1.如果线程请求的栈大于所分配的栈大小，则抛出StackOverFlowError错误，比如进行了一个不会停止的递归调用；2. 如果虚拟机栈是可以动态拓展的，拓展时无法申请到足够的内存，则抛出OutOfMemoryError错误。

- 直接内存：直接内存虽然不是虚拟机运行时数据区的一部分，但既然是内存，就会受到物理内存的限制。在JDK1.4中引入的NIO使用Native函数库在堆外内存上直接分配内存，但直接内存不足时，也会导致OOM。

- 方法区：随着Metaspace元数据区的引入，方法区的OOM错误信息也变成了“java.lang.OutOfMemoryError:Metaspace”。对于旧版本的Oracle JDK，由于永久代的大小有限，而JVM对永久代的垃圾回收并不积极，如果往永久代不断写入数据，例如String.Intern()的调用，在永久代占用太多空间导致内存不足，也会出现OOM的问题，对应的错误信息为“java.lang.OutOfMemoryError:PermGen space”



开发中容易造成内存泄露的操作

- **创建大量无用对象**

比如，我们在需要大量拼接字符串时，使用了String而不是StringBuilder。

```
String str = "";
for (int i = 0; i < 10000; i++) {
    str += i;      //相当于产生了10000个String对象 ，推荐使用StringBuiler类
}
```

- **静态集合类的使用**

像HashMap、Vector、List等的使用最容易出现内存泄露，这些静态变量的生命周期和应用程序一致，所有的对象Object也不能被释放。

- **各种连接对象(IO流对象、数据库连接对象、网络连接对象)未关闭**

IO流对象、数据库连接对象、网络连接对象（socket）等连接对象属于物理连接，和硬盘或者网络连接，不使用的时候一定要关闭。

- **监听器的使用**

释放对象时，没有删除相应的监听器。

要点：

1. 程序员无权调用垃圾回收器。

2. 程序员可以调用System.gc()，该方法只是通知JVM，并不是运行垃圾回收器。尽量少用，会申请启动Full GC，成本高，影响系统性能。

3. finalize方法，是Java提供给程序员用来释放对象或资源的方法，但是尽量少用。