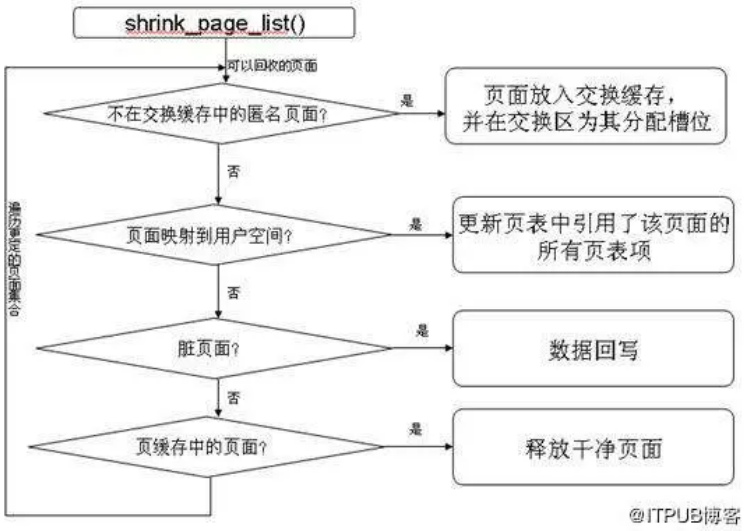
`shrink_slab()`

该函数用来回收磁盘缓存所占用的页面的。Linux 操作系统并不清楚这类页面是如何使用的，所以如果希望操作系统回收磁盘缓存所占用的页面，那么必须要向操作系统内核注册 shrinker 函数，shrinker 函数会在内存较少的时候主动释放一些该磁盘缓存占用的空间。函数 shrink_slab() 会遍历 shrinker 链表，从而对所有注册了 shrinker 函数的磁盘缓存进行处理。Android内核的lowmemorykiller机制就是注册了shrinker，内存过低时选择性杀死**进程**来回收内存。

`shrink_page_list()`

逻辑流程图：



shrink_page_list()执行逻辑

六、用户空间内存管理

6.1 Android用户空间**进程**划分

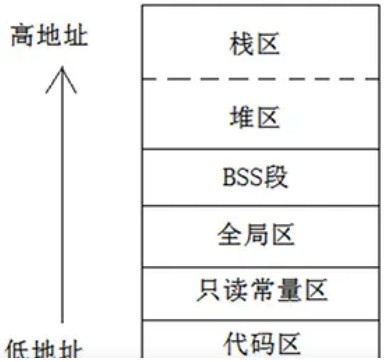
- `Native进程`：不包含虚拟机实例的linux**进程**。
- `Java进程`：包含了虚拟机实例的linux**进程**。

6.2 内存管理

6.2.1 Natvie**进程**

1）内存区域划分：

Native**进程**与Linux**进程**一样，虚拟内存区域分为：代码区、只读常量区、全局区、BSS段、堆区、栈区



`代码区`：存放函数体的二进制代码。
`只读常量区`：存放字符串常量，以及const修饰的全局变量 。
`全局区/数据区`：存放已经初始化的全局变量和已经初始化用static修饰的局部变量。
`BSS段`：存放没有初始化的全局变量和未初始化静态局部变量，该区域会在main函数执行前进行自动清零。
`堆区`：一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表。
`栈区`：由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

注意：栈区和堆区之间并没有严格分割线，可以进行微调，并且堆区分配一般从低地址到高地址分配，而栈区分配一般从高地址到低地址分配。

以上是标准划分，但是对于一个**进程**的内存空间，逻辑上可以分为以下三部分:
`程序区`： 程序的二进制文件。
`静态存储区`： （只读常量区、全局区、BSS段）全局变量和静态变量。
`动态存储区`： （堆区、栈区）本地变量。

> 注:
> `局部变量`：在一个有限的范围内的变量，作用域是有限的，对于程序来说，在一个函数体内部声明的普通变量都是局部变量，局部变量会在栈上申请空间，函数结束后，申请的空间会自动释放。
> `全局变量`：在函数体外申请的，会被存放在全局（静态区）上，知道程序结束后才会被结束，这样它的作用域就是整个程序。
> `静态变量`：和全局变量的存储方式相同，在函数体内声明为static就可以使此变量像全局变量一样使用，不用担心函数结束而被释放。

2）内存分配与回收
内存的静态分配和动态分配的区别主要是两个：

- 时间上：静态分配发生在程序编译和连接时，动态分配则发生在程序调入和执行时。
- 空间上：堆都是动态分配的，没有静态分配的堆。栈有2种分配方式：静态分配和动态分配。静态分配是编译器完成的，比如局部变量的分配。动态分配由函数malloc进行分配。不过栈的动态分配和堆不同，他的动态分配是由编译器进行释放，无需我们手工实现。

动态内存分配与回收
所谓动态内存分配，就是指在程序执行的过程中动态地分配或者回收存储空间的分配内存的方法。动态内存分配不象数组等静态内存分配方法那样需要预先分配存储空间，而是由系统根据程序的需要即时分配，且分配的大小就是程序要求的大小。

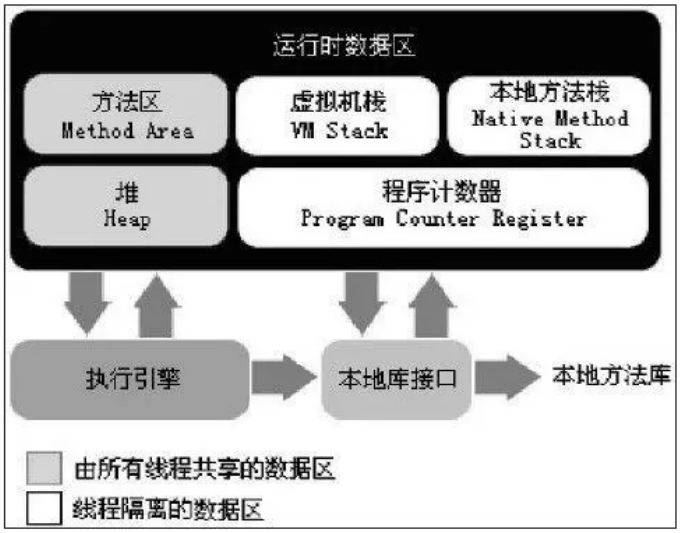
分配和回收操作函数介绍：
`malloc`：动态内存分配，用于在堆上申请一块连续的指定大小的内存区域，但是并没有初始化。
`calloc`:则将初始化这部分的内存,设置为0. calloc = malloc + memset（初始化工作）。
`alloca`:是向栈申请内存,因此无需释放。
`realloc`:则对malloc申请的内存进行大小的调整。
(注：这四个函数都是由free来释放内存。)

6.2.2 Java**进程**

从之前写的系统启动流程中我们了解了，zygote是java**进程**的鼻祖，它通过了Runtime启动了虚拟机，并通过fork，把虚拟机作为环境带给了每一个应用**进程**。虚拟机的设计除了提供跨平台能力之外，也提供了对象生命周期的管理，内存管理，线程管理，安全和异常的管理等统一的处理方案。

1）内存区域划分

这部分之前文章有总结：，如下是JVM内存划分模型，其实Dalvik和ART都一样，就是Heap的space结构会有区别：



JVM内存划分模型

程序计数器：是一块较小的线程私有的内存空间，用来记录正在执行的虚拟机字节码指令，以此来记录当前线程的运行状态；它是一个指针，指向执行引擎正在执行的指令的地址。

虚拟机栈：栈是一块连续的内存区域，大小是由操作系统预定好的（2M左右），它是先进后出的队列，进出一一对应，不产生碎片，运行效率稳定高。局部变量的基本数据类型和引用存储于栈中，因为它们属于方法中的变量，生命周期随方法而结束。

本地方法栈：针对Native方法的，功能与虚拟机栈一致。

静态存储区（方法区）：内存在程序编译的时候就已经分配好，这块内存在程序整个运行期间都存在。它主要存放静态数据、全局static数据和包含常量池。

堆：堆是不连续的内存区域（因为系统是用链表来存储空闲内存地址，自然不是连续的），堆大小受限于计算机系统中有效的虚拟内存（32bit系统理论上是4G），所以堆的空间比较灵活，比较大。对于堆，频繁的分配和回收内存会造成大量内存碎片，使程序效率降低。堆内存用于存放引用的对象实体、成员变量全部存储于堆中（包括基本数据类型，引用和引用的对象实体，因为它们属于类，类对象终究是要被new出来使用的）。

Java内存玩的就是虚拟机划分的一亩三分地，而大小是由系统设置的，内存的分配与回收都是虚拟机负责的。申请的内存超过了一亩三分地就会oom，内存不足会触发gc，而gc又分串行gc与并行gc，art虚拟机优化了gc环节，大大缩短了全线程block的时长，但是如果明显的内存抖动还是会造成卡顿问题。

好了，虚拟机内存管理暂时不分析了，之后有机会再单独开系列来分析，内存管理基础暂时就写这么多，歇了。

参考：