**运行时数据区包括**：虚拟机栈区，堆区，方法区，本地方法栈，程序计数器

**虚拟机栈区** ：也就是我们常说的栈区，线程私有，存放基本类型，对象的引用和 returnAddress ，在编译期间完成分配。

错误：StackOverflowError、OutOfMemoryError

**堆区** ： JAVA 堆，也称 GC 堆，所有线程共享，存放对象的实例和数组， JAVA 堆是垃圾收集器管理的主要区域。

新生代与老年代（8：1：1）

Eden、From Survior、To Survivor

**方法区** ：所有线程共享，存储已被虚拟机加载的类信息，常量，静态变量，即时编译器编译后的代码等数据。这个区域的内存回收目标主要是针对常量池的对象的回收和对类型的卸载。

别名永久代，

OutOfMermoryError

包含运行时常量池：动态可以存放常量，运行时将新的常量存入常量池。

**程序计数器** ：线程私有，每个线程都有自己独立的程序计数器，用来指示下一条指令的地址。

**本地方法栈：**本地方法栈则是为虚拟机使用到的 Native方法服务

**GC Roots：**

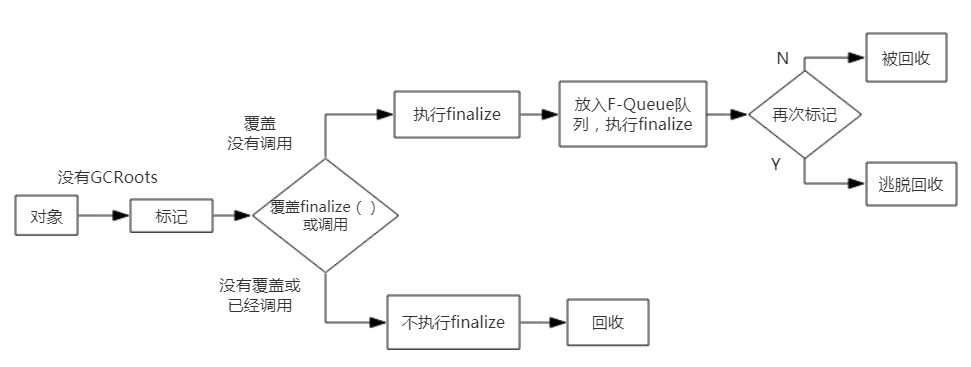
虚拟机栈中引用的对象

方法区中类静态属性引用的对象

方法区中常量引用的对象

本地方法栈中引用的对象

**对象的回收判断：**



minor GC：年轻代

full GC：整个堆空间

major GC：老年代

**永久代的回收**

hotspot的方法区存放在永久代中，因此方法区被人们称为永久代。永久代的垃圾回收主要包括类型的卸载和废弃常量池的回收。当没有对象引用一个常量的时候，该常量即可以被回收。而类型的卸载更加复杂。必须满足一下三点，该类型的所有实例都被回收了，该类型的ClassLoader被回收了，该类型对应的java.lang.Class没有在任何地方被引用，在任何地方都无法通过反射来实例化一个对象。

**垃圾收集算法**

mark-sweep算法

copy算法：内存会被分为两块，每次使用其中的一块

mark-compact（标记整理）算法，标记后让所有存活的对象向一端移动，然后直接处理掉端边界以外的内存。

**类加载的全过程**

加载

通过类名获取定义此类的二进制流

将二进制流代表的静态存储结构转化为方法区运行的数据结构

在内存中生成一个代表这个类的class对象，作为方法区这个类的各种数据的访问入口

验证：确保class文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机自身的安全。

准备：为类变量分配内存并设置类变量的初始值

类变量为static 变量

初始值指的是一般设为0，不为0的在之后的初始化会设为相应的初始值

如果变量为常量，则会赋值为相应的值。

解析：将常量池内的符号引用替换为直接引用的过程

初始化：真正开始执行定义的java程序代码

**类加载器**

通过一个类的限定名来加载类的二进制代码这个动作放在了虚拟机的外部，以便让程序自己决定如何去加载类，实现这个动作的代码快为类加载器。

类的唯一性：类加载器必须一致，如果不相同则即使来源于一个class文件被同一个虚拟机加载，只要加载他们的类加载器不同，那这两个类就必定不相等。

加载器类型：

启动类加载器（Bootstrap ClassLoader），c++编写，属于虚拟机的一部分

其他，独立于虚拟机外部，java语言，继承与java.lang.ClassLoader

常见的三类加载器

启动类加载器（Bootstrap Classloader）

扩展类加载器（Extension ClassLoader）

应用程序类加载器（Application ClassLoader）