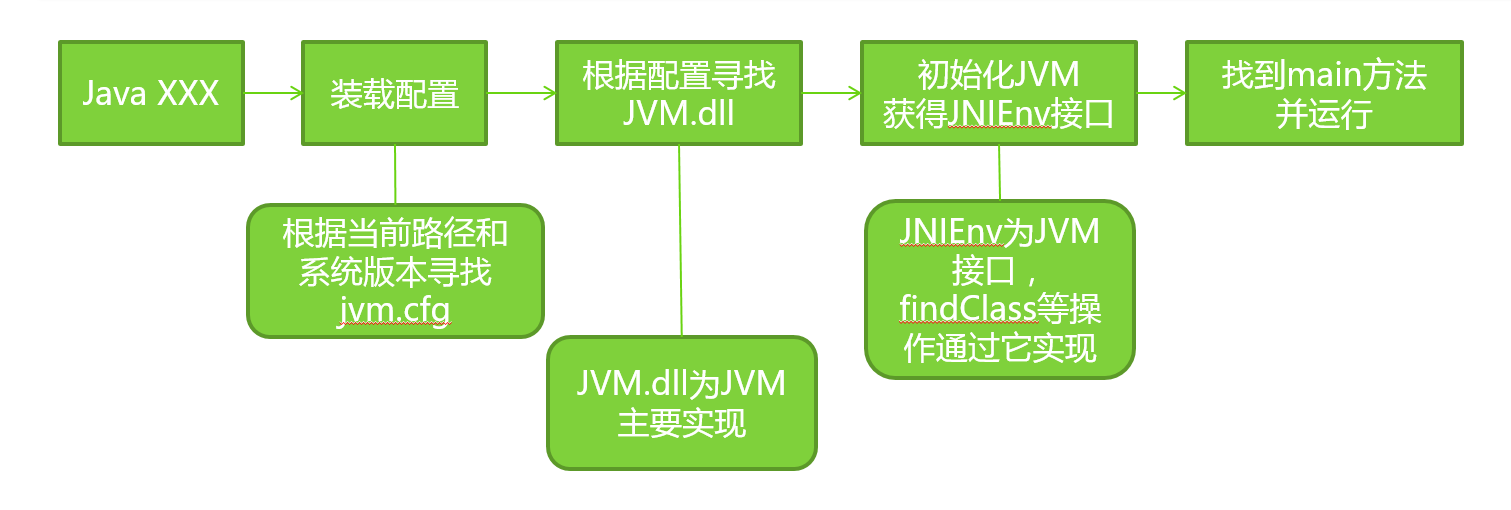
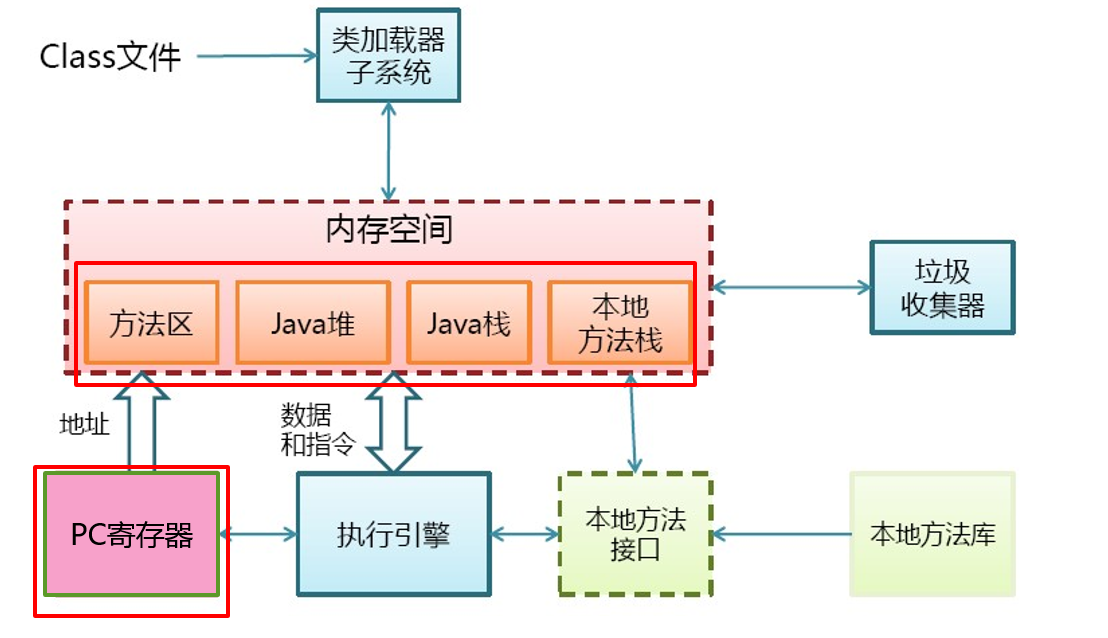
## JVM启动流程

Java XXX（Java命令）



## JVM基本结构



### PC寄存器

* + 每个线程拥有一个PC寄存器
  + 在线程创建时 创建
  + 指向下一条指令的地址
  + 执行本地方法时，PC的值为undefined

### 方法区

* + 保存装载的类信息（保存类的元信息对类进行描述）
    - 类型的常量池
    - 字段，方法信息
    - 方法字节码
  + 通常和永久区(Perm)关联在一起

保存一些相对静止的数据，虽然会发生变动，但是相对来说是稳定的。

不是一成不变的

JDK6时，String等常量信息置于方法区

JDK7时，已经移动到了堆

### Java堆

* + 和程序开发密切相关
  + 应用系统对象都保存在Java堆中
  + 所有线程共享Java堆（全局共享）
  + 对分代GC来说，堆也是分代的
  + GC的主要工作区间



### Java栈

* + 线程私有
  + 栈由一系列帧组成（因此Java栈也叫做帧栈）
  + 帧保存一个方法的局部变量、操作数栈、常量池指针
  + 每一次方法调用创建一个帧，并压栈

#### Java栈 – 局部变量表 包含参数和局部变量

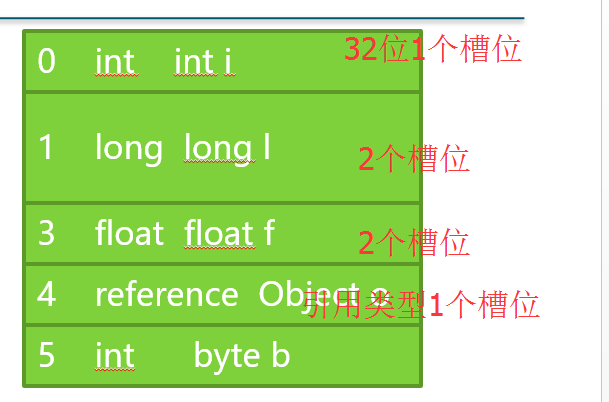
槽位可以容纳32位的数据类型

public class StackDemo {

public static int runStatic(int i,long l,float f,Object o ,byte b){

return 0;

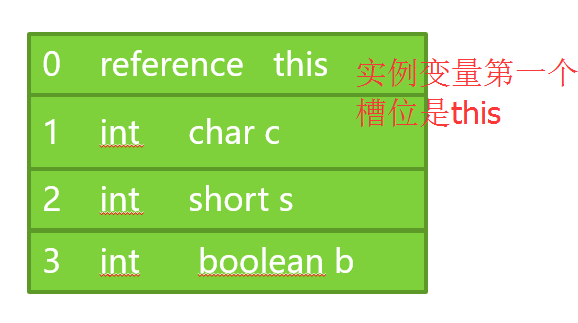
}



public int runInstance(char c,short s,boolean b){

return 0;

}



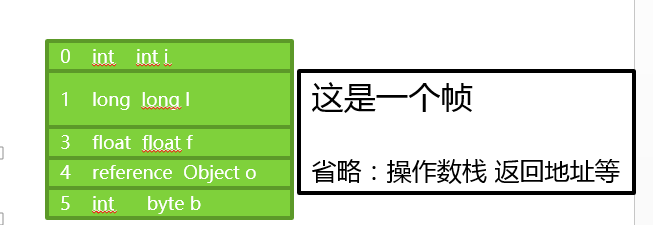
}

#### Java栈 – 函数调用组成帧栈

public static int runStatic(int i,long l,float f,Object o ,byte b){

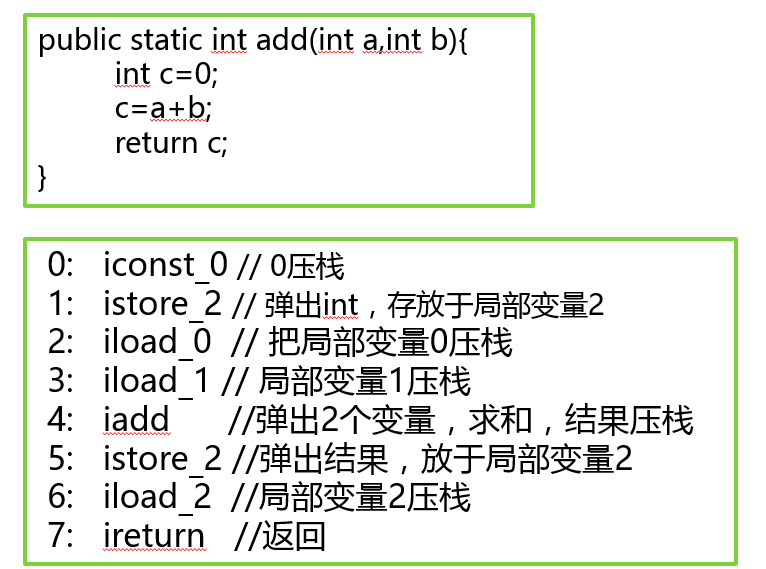
return runStatic(i,l,f,o,b);

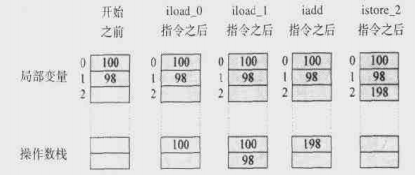
}



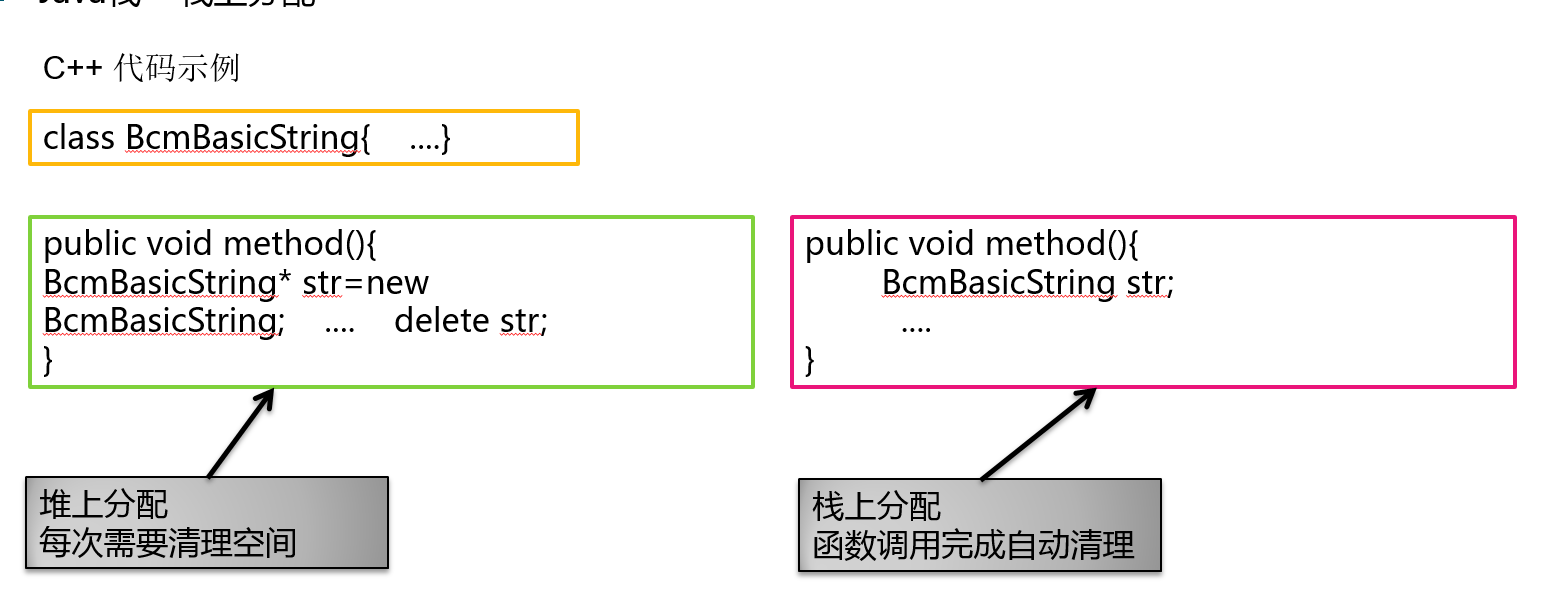
#### Java栈 – 操作数栈

* + Java没有寄存器，所有参数传递使用操作数栈

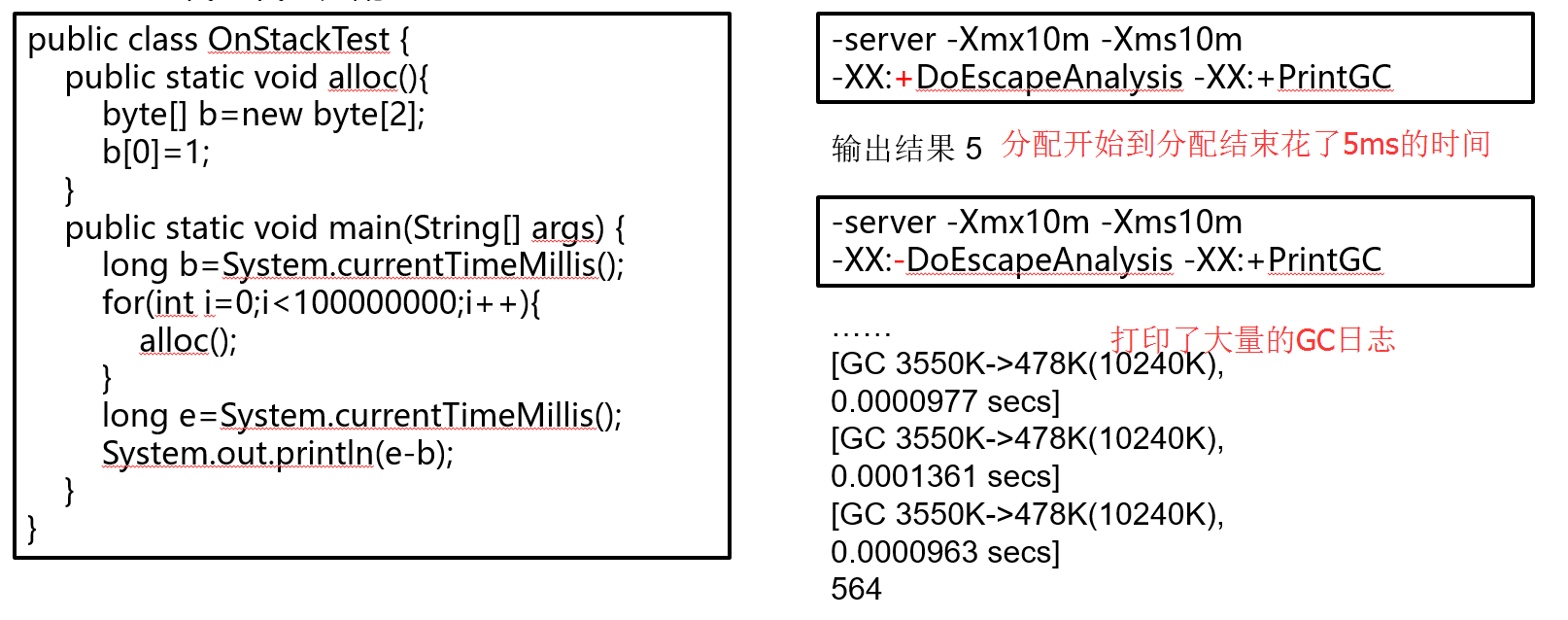




#### Java栈 – 栈上分配



如果内存不够大，分配了大量的内存之后一定会发送GC。

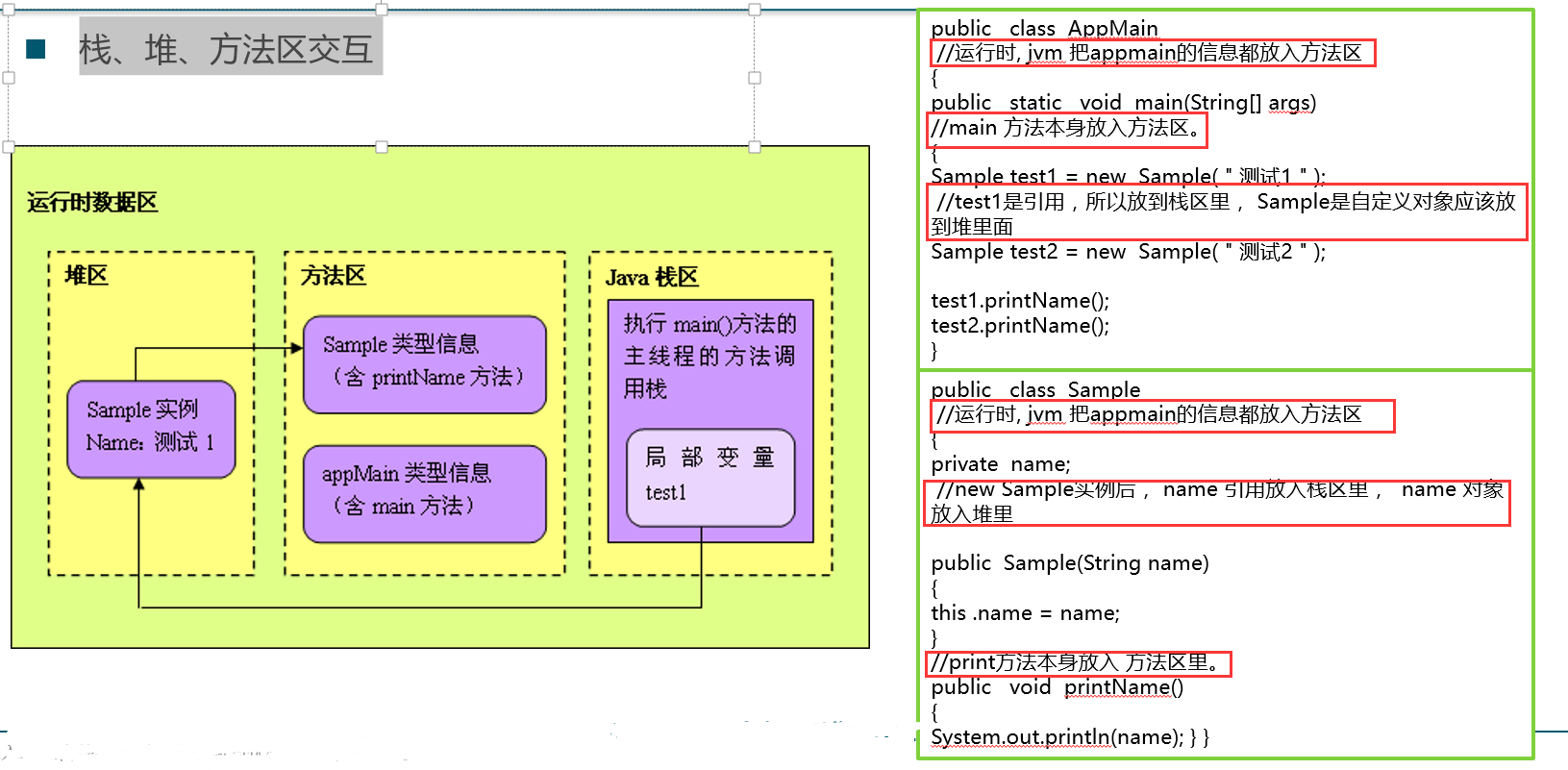


在栈上分配会减轻GC的压力

* + 小对象（一般几十个bytes），在没有逃逸的情况下，可以直接分配在栈上
  + 直接分配在栈上，可以自动回收，减轻GC压力
  + 大对象或者逃逸对象无法栈上分配

**只要这个引用有可能会被多个线程所引用，这就发生了逃逸**。

#### 栈、堆、方法区交互



对象本身是在堆中，栈中存放的是引用；

类的信息的描述，以及包括类当中的方法的字节码是在方法区的；

堆中保存的类的实例；

22：56

内存模型

编译和解释运行的概念