## 初识 JVM

### JVM的概念

指通过软件模拟的具有完整硬件系统功能的、运行再一个完全隔离的完整的计算机系统

常见的虚拟机有

VMWare

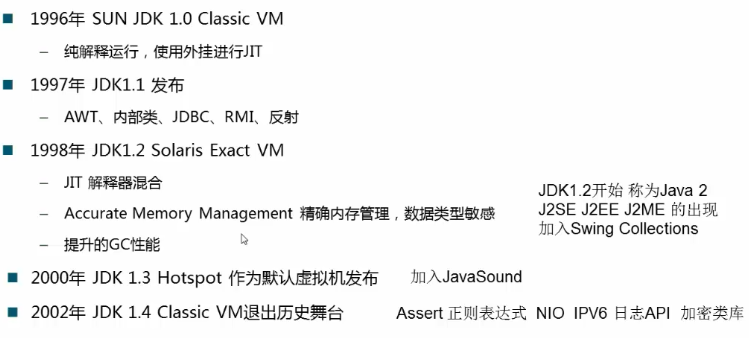
Visual Box

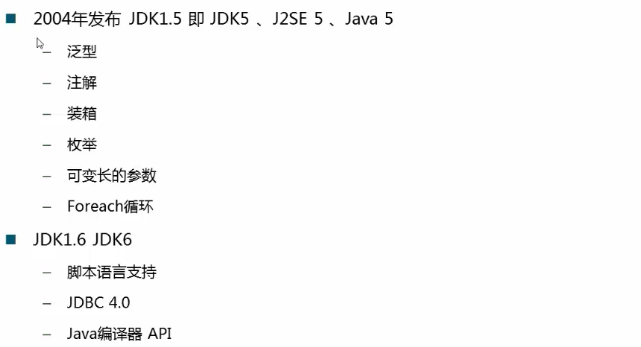
JVM

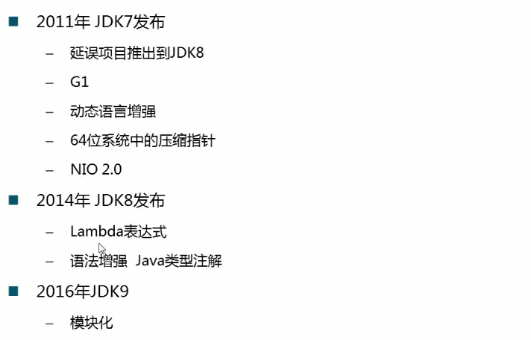
VMWare或者Visual Box都是使用软件模拟物理CPU的指令集

JVM使用软件模拟Java字节码指令集

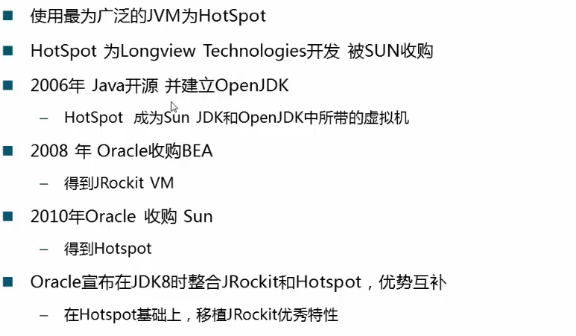
### JVM的发展史







### JVM的种类



### JVM的规范

* **Java语言规范**

语法

|  |
| --- |
| IfThenStatement:  If(Expression)Statement if(true){do sth} |
| ArgumentList add(a,b,c,d);  Argument  ArgumentList, Argument |

语法结构

注释

标识符

关键字

变量

类型

文法

* **JVM规范**

Java语言规范和JVM规范是相对独立的

Groovy

Clojure

Scala

JVM主要定义二进制class文件和JVM指令集

Class文件类型

运行时数据

帧栈

虚拟机的启动

* **虚拟机的指令集**

VM的指令集

|  |
| --- |
| 类型转化  l2i  出入栈操作  aload astore  运算  iadd isub  流程控制  ifeq ifne  函数调用  invokevirtual invokeinterface invokesoecial invokestatic |

* **整数的表达**

原码：第一位为符号位（0为正数，1为负数）

反码：符号不动，原码取反

负数补码：符号位不动，反码加1

整数补码：和原码相同

打印整数的二进制表示

|  |
| --- |
| Int a = -6  For(int i=0;0<32;i++){  Int t=(a&0x80000000>>>i)>>>(31-i);  System.out.println(t);  } |

* **为什么要用补码**

使我们没有歧义的表示零

使我们能够很好的参与计算机的二进制计算

* **JVM需要对Java Library提供以下支持**

反射java.lang.reflect（反射本身要在JVM底层进行一个实现）

ClassLoader（类加载器）

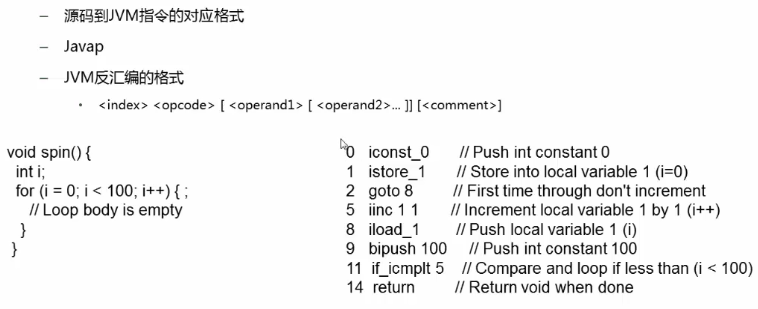
初始化class和interface（类和接口初始化方法）

安全相关java.security

多线程

弱引用

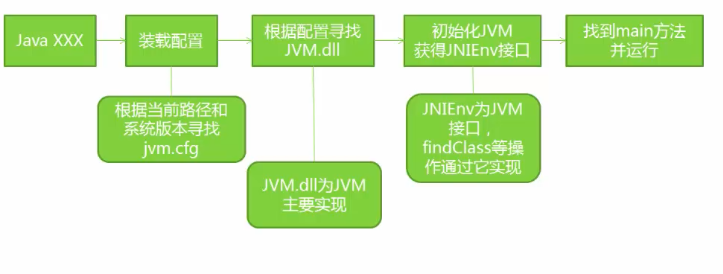
* **JVM的编译**



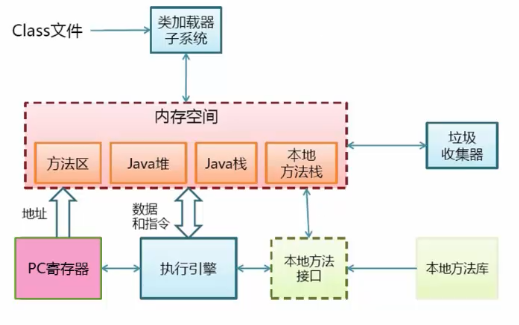
## JVM运行机制

### JVM启动流程

java或javaw启动命令



### 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（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，相对稳定的信息（但不表示它永远不会被回收）

在JDK6时，String等常量信息置于方法区

JDK7时，已经移到了堆中

* **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

堆分代GC（垃圾回收）来说，堆也是分代的

GC的重要工作区间

|  |  |  |  |
| --- | --- | --- | --- |
| eden | s0 | s1 | tenured |

GC是复制算法

* **Java栈**

线程私有

栈是由一系列帧组成（因此Java栈也叫帧栈）

帧保存一个方法的局部变量、操作数栈、常量池指针

每一个方法调用创建一个帧，并压栈

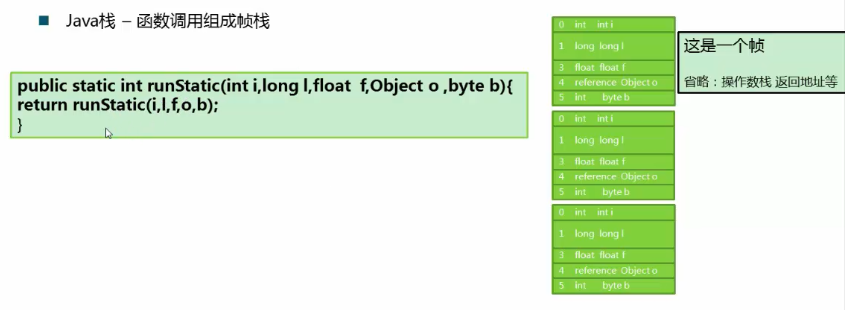
Java栈中的局部变量（局部变量和参数）表，局部变量表里面有很多的槽位，每个槽位最多能够容纳32位的数据类型（long有64位类型就占了两个槽位），对象就是一个引用，也占32位

实例方法局部变量表的第一个槽位传的是当前对象的引用，其他与静态方法一致



当函数发生调用的时候就会有一个帧往栈里去压，当方法调用结束时，就会把这个帧移除掉

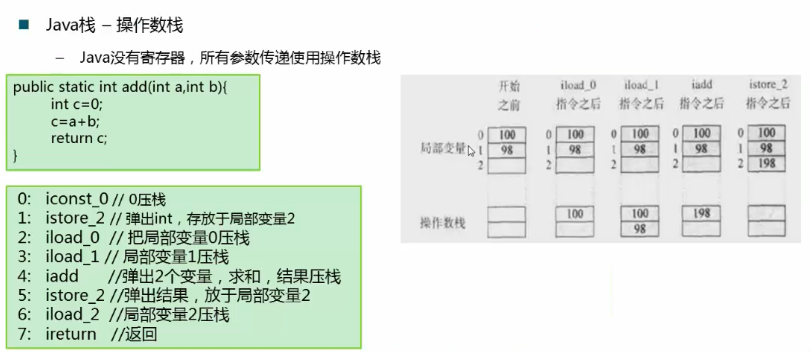
下面的这个递归方法产生的结果就是帧栈溢出



* **操作数栈**

Java中没有寄存器，只有JVM中有寄存器，

Java中所有的参数传递使用操作数栈



* **栈上分配**

c++

堆上分配需要手动删除，不然就会发生内存泄露

栈上分配则不需要

c++的堆上分配（new）和栈上分配

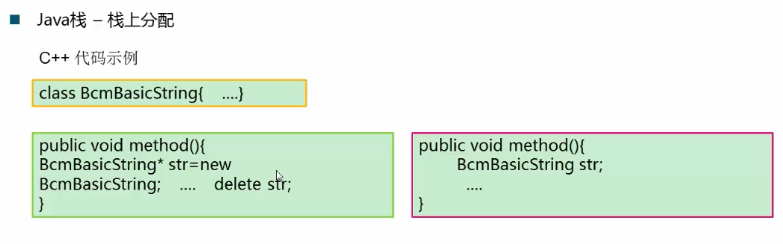
Java

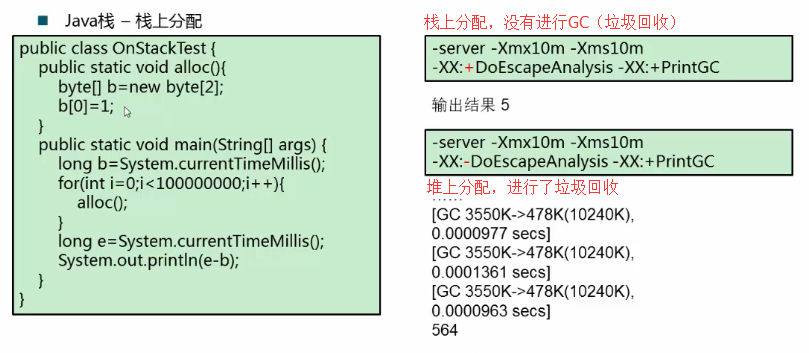
小对象（一般几十个bytes），在没有逃逸的情况下，可以直接分配在栈上

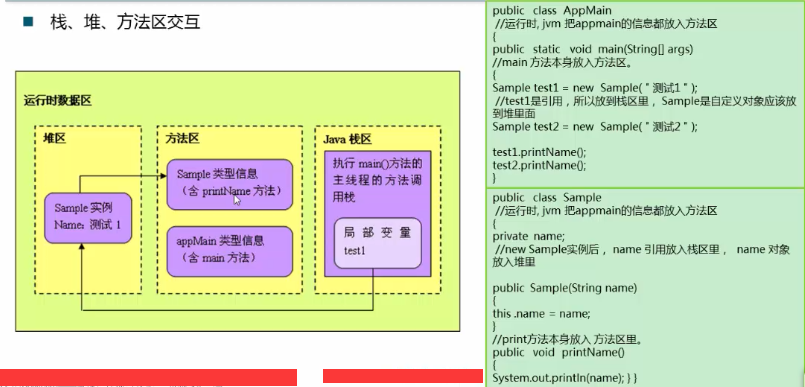
直接分配在栈上可以自动回收，减轻GC压力

大对象或者逃逸对象无法栈上分配

（逃逸对象，如果这个对象除了在我这个线程要用，其他的线程也要用，就不能在栈上 分配，因为栈是线程私有的，这就叫做逃逸对象）



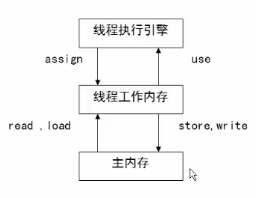
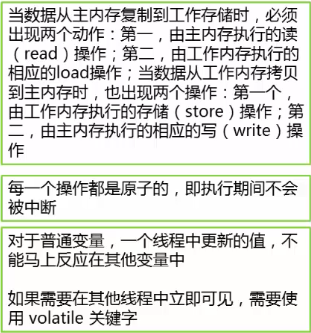


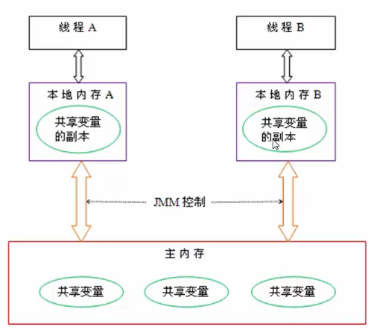


### 内存模型

每一个线程都有一个工作内存和主存独立

工作内存存放主存中变量的拷贝



volatile关键字的个人理解



**可见性**

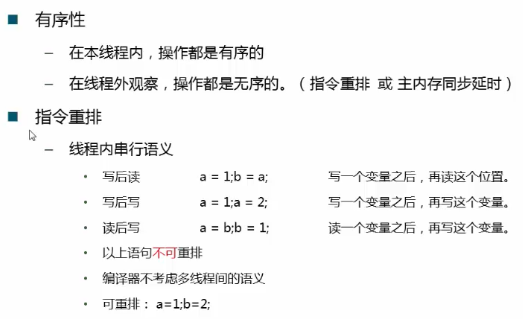
一个线程修改了变量，其他线程可以立即知道

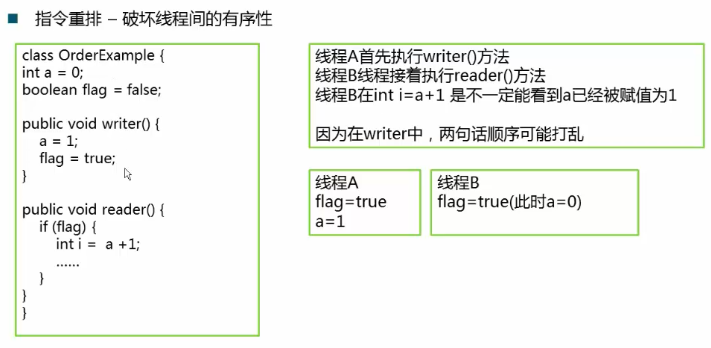
**保证可见性的方法**

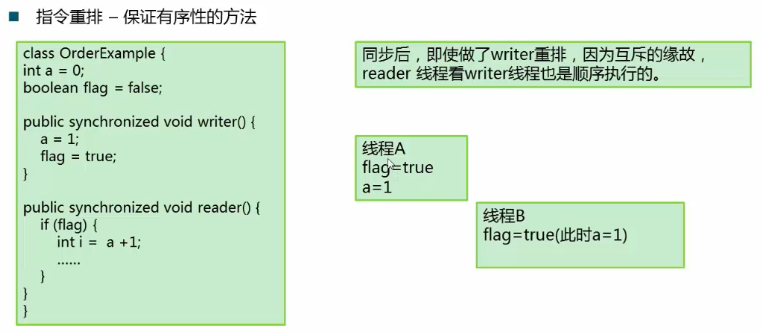
volatile

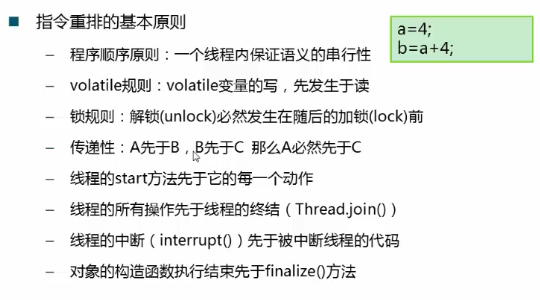
synchronized(unlock之前，写变量值回主存)

final（一旦初始化完成，其他线程就可以见）









### 编译运行和解释运行的概念

* **解释运行**

解释执行以解释方式运行字节码

解释执行的意思是：读一句，执行一句

* **编译运行（JIT）**

将字节码编译成机器码

直接执行机器码

运行时编译

编译后性能有数量级的提升

## JVM常用配置参数

### Trace跟踪参数

verbose：gc

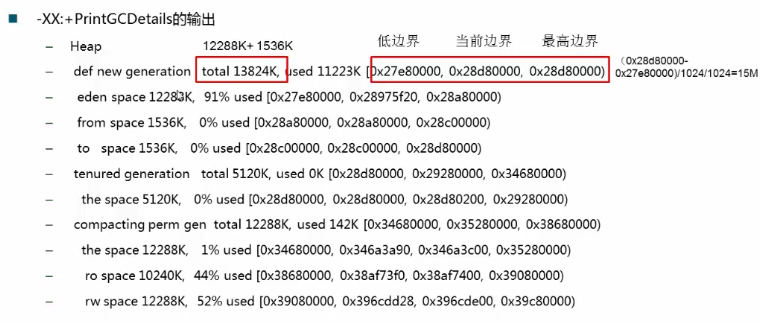
-XX:+printGC

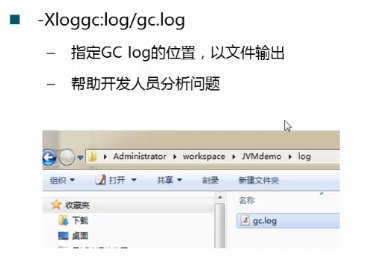
-XX:+PrintGCDetails

打印GC的详细信息

-XX:PrintGCTimeStamps

打印GC发生的时间戳





### 堆的分配参数

### 栈的分配参数