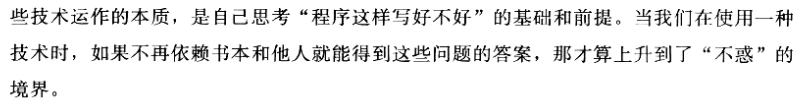
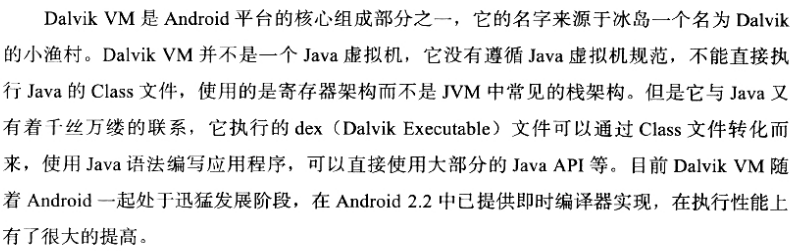
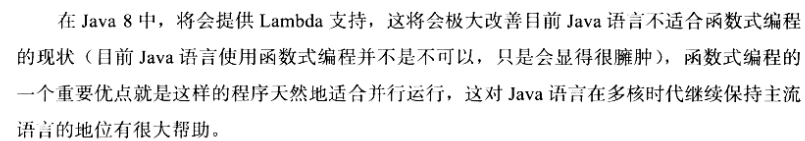
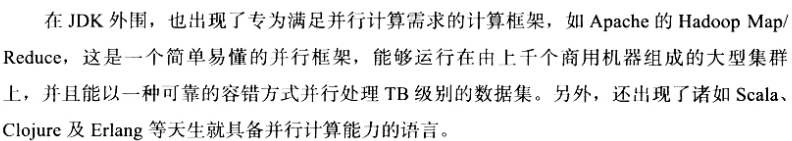
**Jvm**

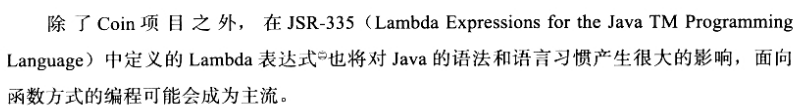
# 走进java



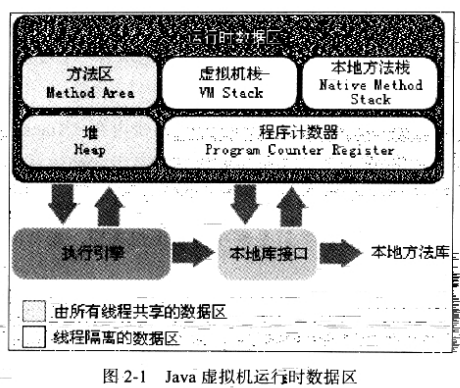




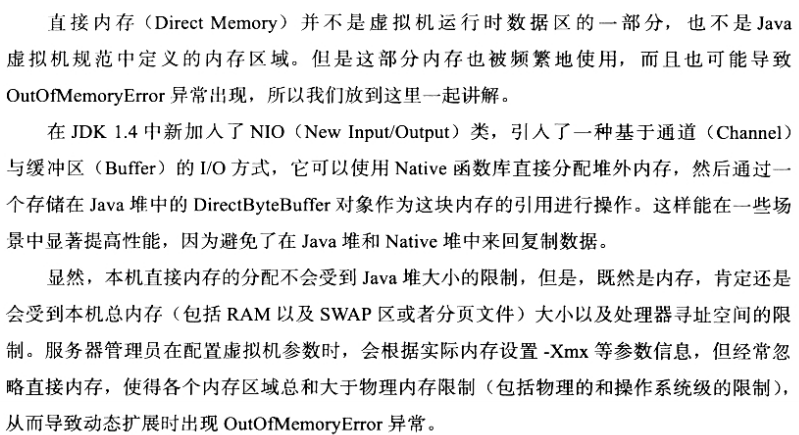




# 虚拟机运行时数据区



**直接内存**



# 内存溢出

## 堆溢出：

|  |
| --- |
| /\*\*  \* VM Args：-Xms20m -Xmx20m -XX:+HeapDumpOnOutOfMemoryError  \* @author zzm  \*/  public class HeapOOM {  static class OOMObject {  }  public static void main(String[] args) {  List<OOMObject> list = new ArrayList<OOMObject>();  while (true) {  list.add(new OOMObject());  }  }  } |

## 栈溢出

|  |
| --- |
| /\*\*  \* VM Args：-Xss128k  \* @author zzm  \*/  public class JavaVMStackSOF {  private int stackLength = 1;  public void stackLeak() {  stackLength++;  stackLeak();  }  public static void main(String[] args) throws Throwable {  JavaVMStackSOF oom = new JavaVMStackSOF();  try {  oom.stackLeak();  } catch (Throwable e) {  System.out.println("stack length:" + oom.stackLength);  throw e;  }  }  } |

## 方法区溢出

|  |
| --- |
| /\*\*  \* VM Args：-XX:PermSize=10M -XX:MaxPermSize=10M  \* @author zzm  \*/  public class RuntimeConstantPoolOOM {  public static void main(String[] args) {  // 使用List保持着常量池引用，避免Full GC回收常量池行为  List<String> list = new ArrayList<String>();  // 10MB的PermSize在integer范围内足够产生OOM了  int i = 0;  while (true) {  list.add(String.valueOf(i++).intern());  }  }  } |

## 本机直接内存溢出

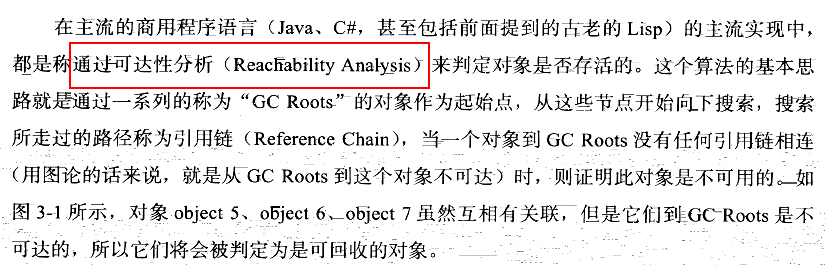
|  |
| --- |
| /\*\*  \* VM Args：-Xmx20M -XX:MaxDirectMemorySize=10M  \* @author zzm  \*/  public class DirectMemoryOOM {  private static final int \_1MB = 1024 \* 1024;  public static void main(String[] args) throws Exception {  Field unsafeField = Unsafe.class.getDeclaredFields()[0];  unsafeField.setAccessible(true);  Unsafe unsafe = (Unsafe) unsafeField.get(null);  while (true) {  unsafe.allocateMemory(\_1MB);  }  }  } |

# 垃圾收集和内存动态分配

对象死亡？？

**引用计数法**

**可达性分析法**

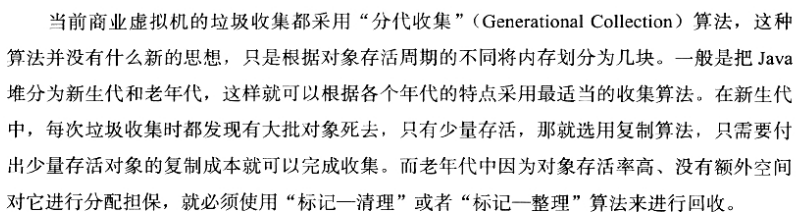
****

不建议使用finalizer方法

方法区回收



垃圾收集算法



**标记-清除算法（Mark-Sweep）**

**两个缺点**：一个是**效率问题 空间问题**

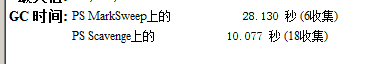
**复制算法（Copying）（针对新生代）**

**缺点**是将内存缩小为原来的一半，代价太高了一点。

**标记-整理算法（Mark-Compact）（针对老年代）**

**分代收集算法（Generational Collection）**

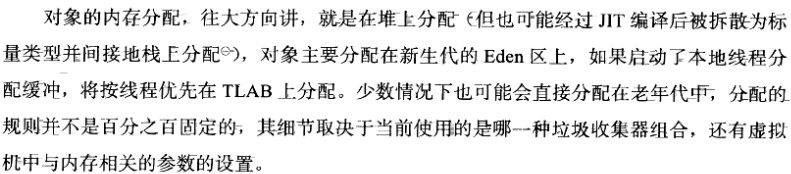
**Jdk1.8默认使用的gc**

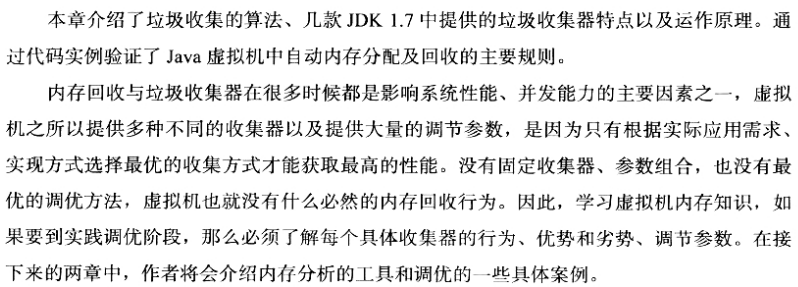


换为G1收集

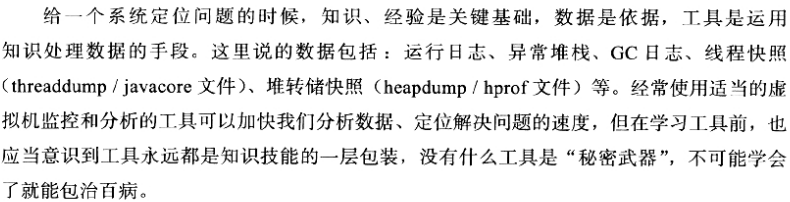
-XX:+UseG1GC

对象的内存分配

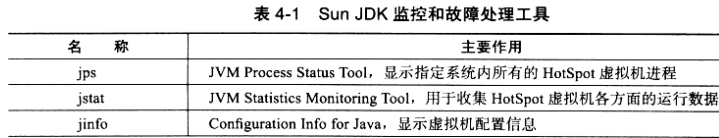


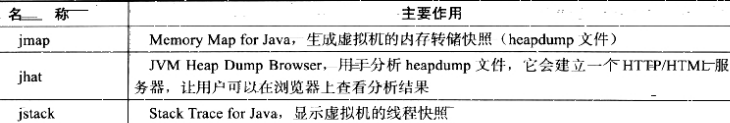


# 性能监控与故障处理工具

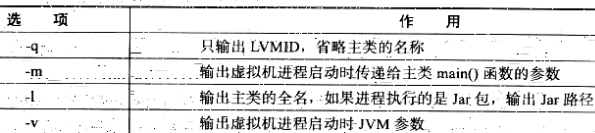


工具包：tools.jar

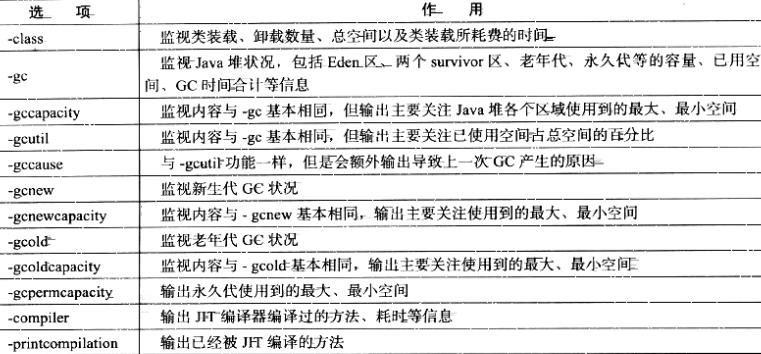




## Jps



## jstat

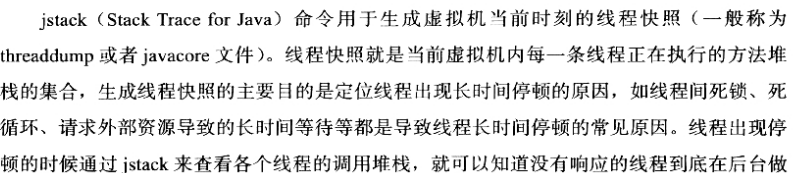


## Jinfo

## Jmap

## Jhat

## Jstack



## Hsdis(jit生成代码反汇编)

## Jvm可视化监控工具

### Jconsole

### Visualvm

# Jvm故障案列

# 类初始化

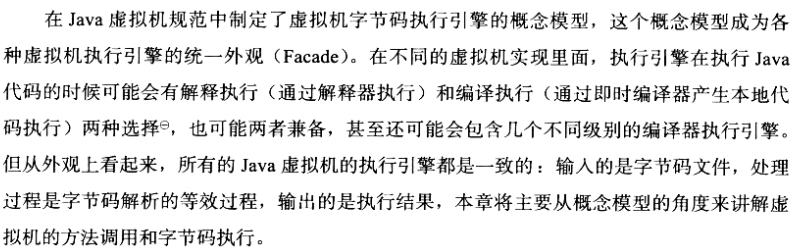
Clinit：线程安全的，只执行一次！！

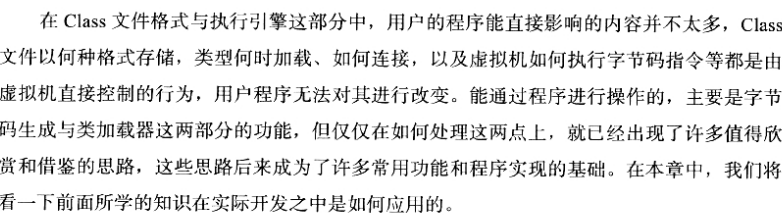
# 类加载器

双亲委派模型：

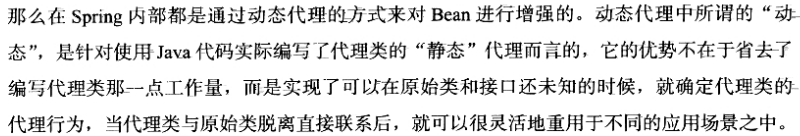
启动类加载器，扩展及应用程序加载器

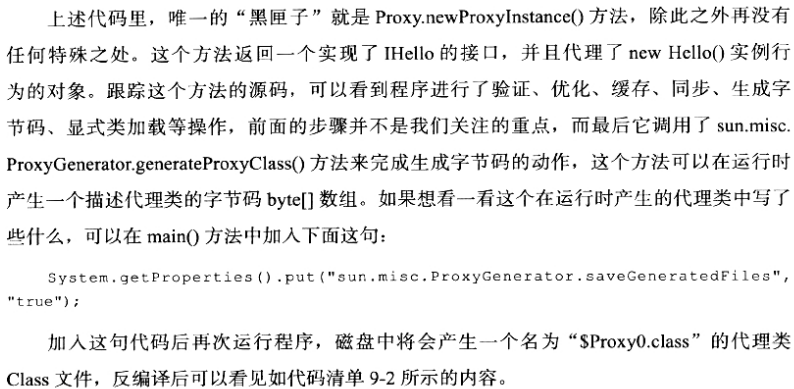
# 字节码执行





# 动态代理





# 程序编译与代码优化

## 早期优化

Javac

## 晚期优化

Jit