# 深入理解Java虚拟机

## 内容分类与环境搭建

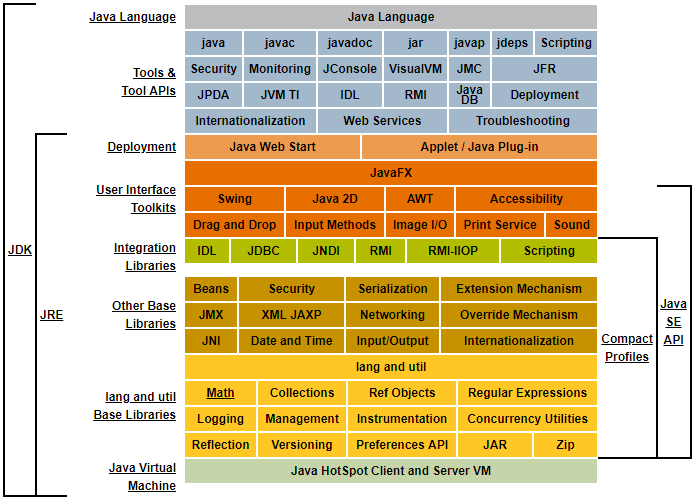
### 1.1概述

1. 了解历史
2. 内存结构
3. 垃圾回收机制
4. 性能监控工具
5. 性能调优方案实战
6. 认识类的文件结构
7. 类加载机制
8. 字节码执行引擎
9. 虚拟机编译及运行时优化
10. Java线程高级

### 1.2环境搭建以及jdk,jre,jvm的关系

1. 下载安装JDK
2. 环境变量配置
3. jdk,jre,jvm

<https://docs.oracle.com/javase/8/docs/>



### 1.3JVM初体验：内存溢出场景模拟

|  |
| --- |
| **public** **class** Demo {  } |

|  |
| --- |
| **public** **class** OutOfMemoryTest {    **public** **static** **void** main(String[] args) {  List<Demo> demoList = **new** ArrayList<>();  **while**(**true**) {  demoList.add(**new** Demo());  }  }  } |

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

at java.util.Arrays.copyOf(Unknown Source)

at java.util.Arrays.copyOf(Unknown Source)

at java.util.ArrayList.grow(Unknown Source)

at java.util.ArrayList.ensureExplicitCapacity(Unknown Source)

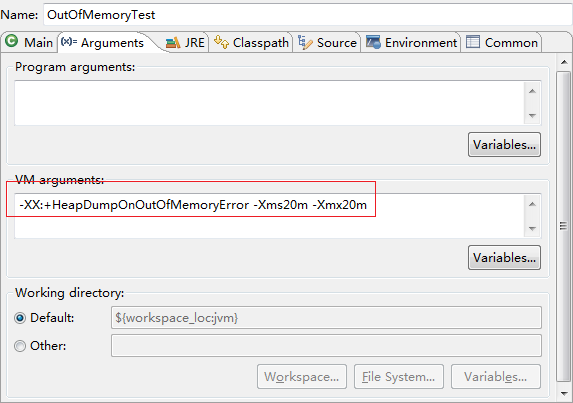
at java.util.ArrayList.ensureCapacityInternal(Unknown Source)

at java.util.ArrayList.add(Unknown Source)

at com.byf.jvm01OutOfMemory.OutOfMemoryTest.main(OutOfMemoryTest.java:11)

1）堆内存快照，JVM参数设置

-XX:+HeapDumpOnOutOfMemoryError -Xms20m -Xmx20m

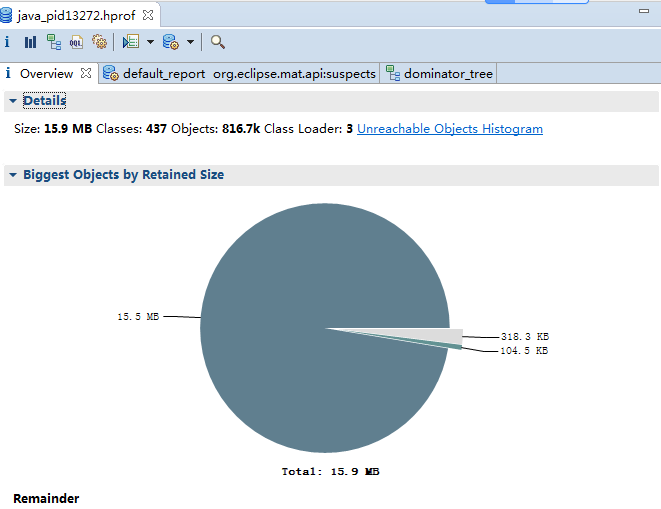


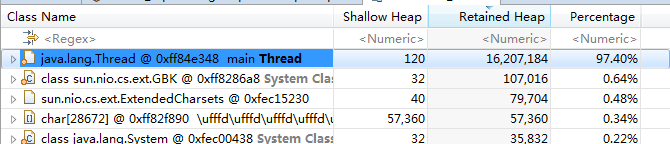
2）生成堆内存快照位置，默认在项目的根路径下：

C:\Users\BYF\workspace\jvm\java\_pid13272.hprof

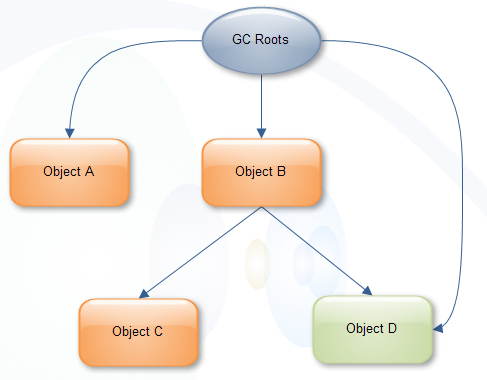
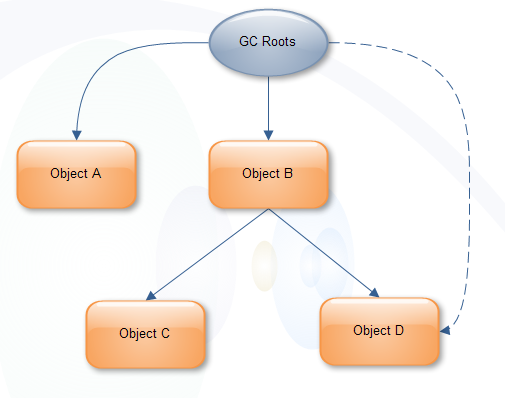
3）下载内存分析工具：

<http://www.eclipse.org/downloads/download.php?file=/mat/1.8.1/rcp/MemoryAnalyzer-1.8.1.20180910-win32.win32.x86_64.zip>

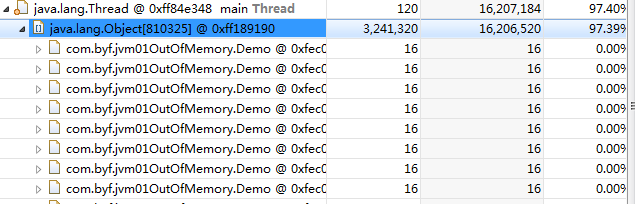




****Shallow Size****   
对象自身占用的内存大小，不包括它引用的对象。   
针对非数组类型的对象，它的大小就是对象与它所有的成员变量大小的总和。当然这里面还会包括一些java语言特性的数据存储单元。   
针对数组类型的对象，它的大小是数组元素对象的大小总和。   
  
****Retained Size****   
Retained Size=当前对象大小+当前对象可直接或间接引用到的对象的大小总和。(间接引用的含义：A->B->C, C就是间接引用)   
换句话说，Retained Size就是当前对象被GC后，从Heap上总共能释放掉的内存。   
不过，释放的时候还要排除被GC Roots直接或间接引用的对象。他们暂时不会被被当做Garbage。

****看图理解Retained Size****   
  
   
  
上图中，GC Roots直接引用了A和B两个对象。   
  
A对象的Retained Size=A对象的Shallow Size   
B对象的Retained Size=B对象的Shallow Size + C对象的Shallow Size   
  
****这里不包括D对象，因为D对象被GC Roots直接引用。****   
如果GC Roots不引用D对象呢？   
  
   
  
  
此时, B对象的Retained Size=B对象的Shallow Size + C对象的Shallow Size + D对象的Shallow Size

5）堆内存溢出的原因，有太多的Demo对象创建。



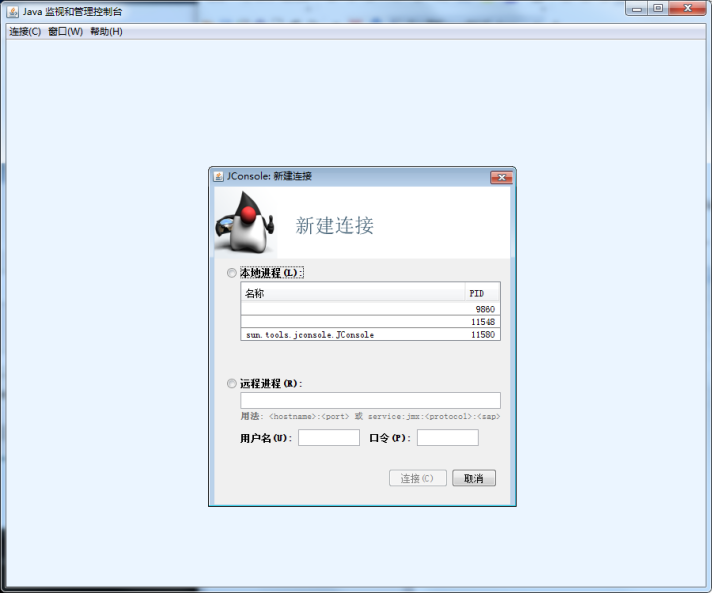
### 1.4JVM再体验：JVM监控工具

性能监控

故障监控

图形化

1. Jconsole



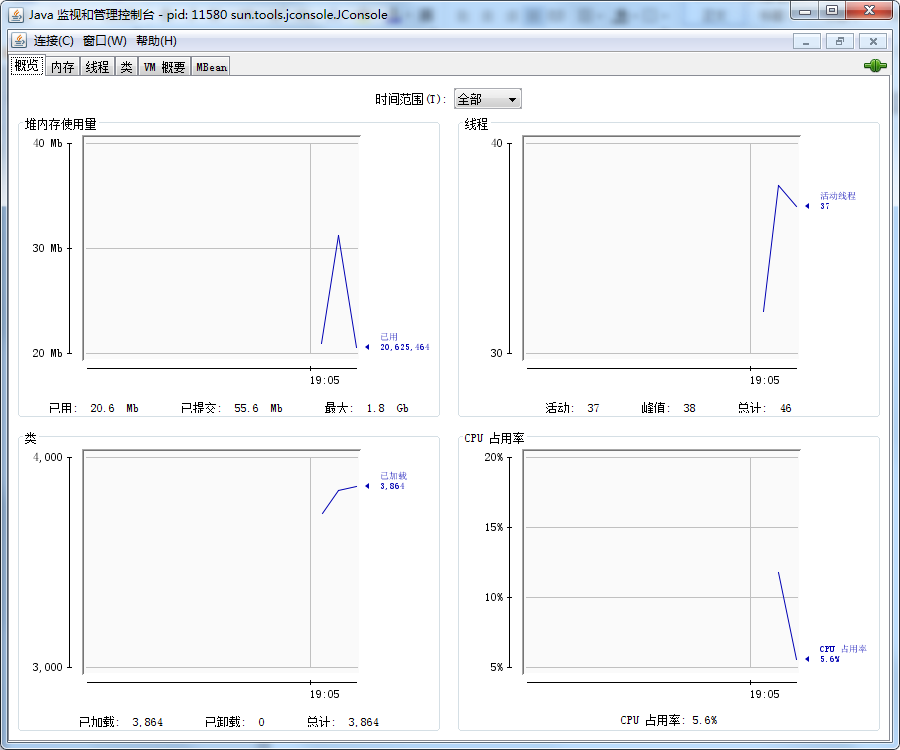
C:\Users\BYF>jps

11428 Jps

9860

11548

11580 JConsole



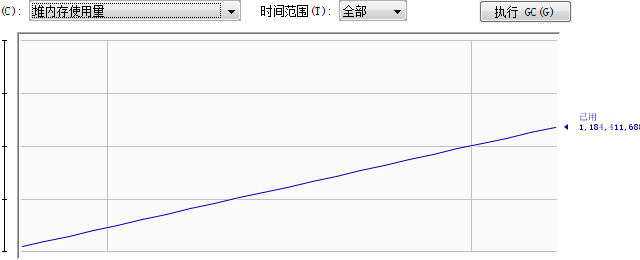
2）测试代码

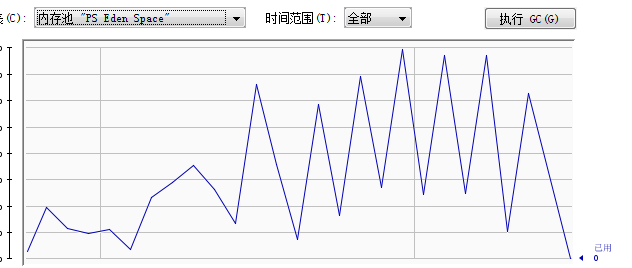
|  |
| --- |
| **public** **class** JConsoleTest {  // public byte[] b1 = new byte[128\*10224];    **public** JConsoleTest(){  **byte**[] b1 = **new** **byte**[128\*10224];  }    **public** **static** **void** main(String[] args) {  **try** {  Thread.*sleep*(5000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  *fill*(1000);  }  **private** **static** **void** fill(**int** i) {  List<JConsoleTest> jList = **new** ArrayList<>();  **for** (**int** j = 0; j < i; j++) {  **try** {  Thread.*sleep*(100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  jList.add(**new** JConsoleTest());  }  }  } |

区别：

// public byte[] b1 = new byte[128\*10224];

// 放在外边，没有被回收



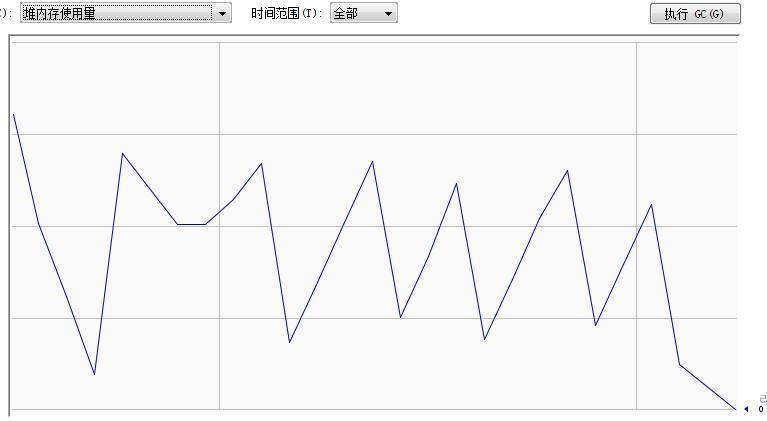


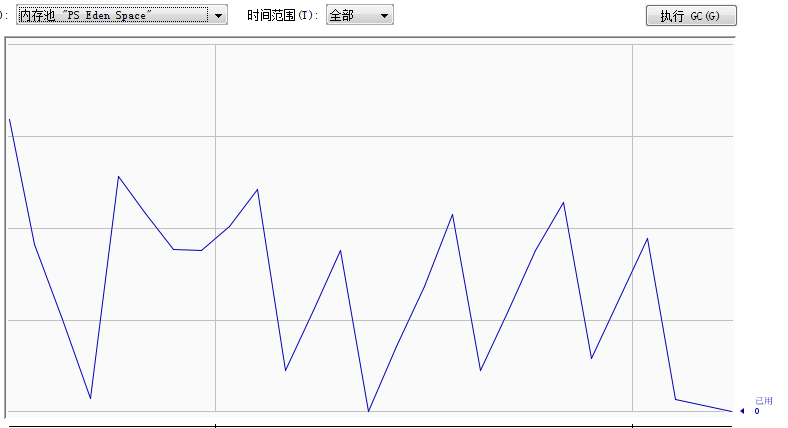
**public** JConsoleTest(){

// 局部变量，没人使用，过段时间就会被JVM回收，放在堆内存，被回收

**byte**[] b1 = **new** **byte**[128\*10224];

}





局部变量被回收，堆内存和Eden变化差不多近似

### 1.5杂谈

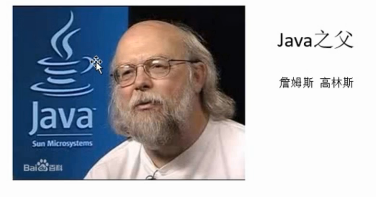
谈Java

java中间件：tomcat/jetty

Hadoop

Lucene搜索

### 1.6Java的发展史



Oak：嵌入式编程语言

Applet

1995.05：Java1.0 Write once run anywhere

1996.01：jvm sun Classic VM

1996.9首届JavaOne大会

1997.02 jdk1.1 内部类，反射，jar文件格式，jdbc，javabeans，rmi

1998 jdk1.2 Java EE Java ME Java SE swing，jit（Just in time），Hotspot VM

2000.5 jdk1.3 Timer，java2d

2002.02 jdk1.4 Structs、Hibernate，Spring1.X，正则表达式，Nio，日志，Xml解析器

2004 jdk1.5 tiger 自动装箱拆箱，泛型，注解，枚举，变长参数，增强for循环，Spring2.X，JUC

2006 jdk1.6 j2ee JDK6 提供脚本语言支持（动态语言），提供了编译api以及http服务器api，开源

2009 jdk1.7 Lambda Jigsaw OSGI

2011.07 jdk1.7

2014.3 jdk1.8

### 1.7Java技术体系

1. Java程序设计语言
2. 各硬件平台上的Java虚拟机
3. Class文件格式
4. Java API
5. 第三方java类库

Java SE Standard edition（桌面）

Java ME Mobile Edition（移动）

Java EE 13种技术综合

### 1.8jdk8的新特性

接口的默认方法和静态方法

Java8使用两个新概念扩展了接口的含义，默认方法和静态方法。

默认方法

可以在不破坏二进制兼容性的前提下，往现存接口中添加新的方法，即不在强调哪些实现类该接口的类也同时实现这个新方法。

Java8引入Lambda的利弊

1. 不用再写大量的匿名内部类，可读性提高
2. 集合操作改善，引入StreamAPI 把Map，Reduce，filter这样的基本函数式编程的概念与Java集合紧密集合，可以告别for、while、if。
3. 学习成本

Data API

重复注解

更好的类型推断：Java编译器的优化

Nashorm JavaScript引擎

使用Mataspace代替持久代PermCen space。在JVM参数方面，使用-XX:MetaSpaceSize和-XX:MaxMetaspaceSize代替原来的-XX:PermSize和-XX:MaxPermSize

### 1.9lanmbda表达式简介

New JButton.addActionListener(event -> System.out.println(“hello”));

## 第2节Java虚拟机

### 2.1classicvm

### 2.2ExactVM

### 2.3HotSpotVM

### 2.4kvm

### 2.5JRockit

### 2.6j9

### 2.7dalvik

### 2.8MicrosoftJVM

### 2.9高性能Java虚拟机

### 2.10TaobaoVM

## 第3节Java内存区域

### 3.1简介

### 3.2Java虚拟机栈

### 3.3程序计数器

### 3.4本地方法栈

### 3.5堆内存

### 3.6方法区

### 3.7直接内存和运行时常量池

### 3.8对象在内存中的布局-对象的创建

### 3.9探究对象的结构

### 3.10深入理解对象的访问定位

## 第4节垃圾回收

### 4.1概述

### 4.2判断对象是否存活算法-引用计数法详解

### 4.3判断对象是否存活算法-可达性分析法详解

### 4.4算法-标记清除算法

### 4.5算法-复制算法

### 4.6算法-标记整理算法和分代收集算法

### 4.7收集器-serial收集器详解

### 4.8收集器-parnew收集器详解

### 4.9收集器-parallel收集器详解

### 4.10收集器-cms收集器详解

### 4.11最牛的垃圾收集器-g1收集器详解

## 第5节内存分配

### 5.1概述

### 5.2Eden区域

### 5.3大对象直接进老年代

### 5.4长期存活的对象进入老年代

### 5.5空间分配担保

### 5.6逃逸分析与栈上分配

## 第6节虚拟机工具

### 6.1介绍

### 6.2jps详解

### 6.3jstat详解

### 6.4jinfo详解

### 6.5jmap详解

### 6.6jhat详解

### 6.7jstack详解

### 6.8可视化工具-Jconsole内存监控

### 6.9可视化工具-Jconsole线程监控

### 6.10死锁原理以及可视化虚拟机工具-Jconsole线程死锁监控

### 6.11VisualVM使用详解

## 第7节性能调优

### 7.1概述

### 7.2案例1

### 7.3案例2

### 7.4案例3

## 第8节前半部分内容整体回顾

## 第9节Class文件

### 9.1简介和发展历史

### 9.2结构概述

### 9.3设计理念以及意义

## 第10节文件结构

### 10.1魔数

### 10.2常量池

### 10.3访问标志

### 10.4类索引

### 10.5字段表集合

### 10.6方法表集合

### 10.7属性表集合

## 第11节字节码指令

### 11.1简介

### 11.2字节码与数据类型

### 11.3加载指令

### 11.4运算指令

### 11.5类型转换指令

### 11.6对象创建与访问指令

### 11.7操作树栈指令

### 11.8控制转移指令

### 11.9方法调用和返回指令

### 11.10异常处理指令

### 11.11同步指令

## 第12节类加载机制

### 12.1概述

### 12.2类加载时机

### 12.3类加载过程

### 12.4类加载过程-验证

### 12.5类加载的过程-准备

### 12.6类加载过程-解析

### 12.7类加载过程-初始化

### 12.8类加载器

### 12.9双亲委派模型

## 第13节虚拟机字节码执行引擎

### 13.1运行时栈帧结构

### 13.2局部变量表

### 13.3操作数栈

### 13.4动态连接

### 13.5方法返回地址和附加信息

### 13.6方法调用-解析调用

### 13.7方法调用-静态分派调用

### 13.8方法调用-动态分派调用

### 13.9动态类型语言支持

### 13.10字节码执行引擎小结

## 第14节总结与回顾

### 14.1happens-before简单概述

### 14.2重排序问题

### 14.3锁的内存语义

### 14.4volatile的内存语义

### 14.5final域内存语义