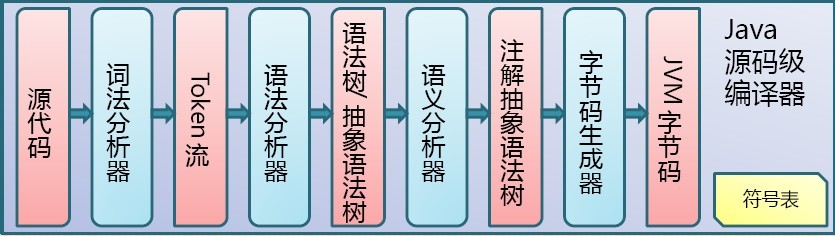
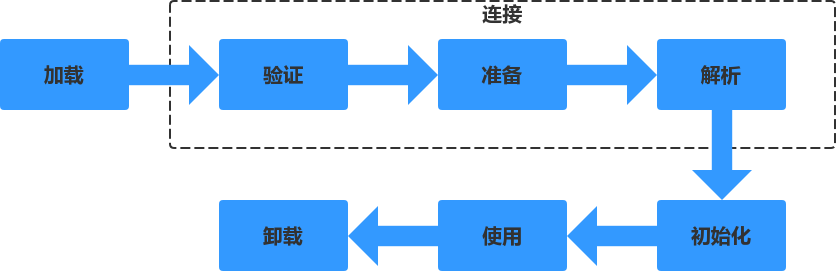
类加载机制与JDK调优监控工具

Java运行时编译源码(.java)成字节码，由jre运行。jre由java虚拟机实现。JVM分析字节码，后解释并执行

# 类的生命周期



## 加载

将.class文件从磁盘读到内存（类的信息存在方法区）

## 连接

### 验证

验证字节码文件(.class)的正确性

### 准备

给类的静态变量分配内存，并赋予默认值，如：int型值默认值为0

### 解析

类装载器装入类所引用的其它所有类

## 初始化

为类的静态变量赋予正确的初始值，上述的准备阶段为静态变量赋予的是虚拟机默认的初始值，此处赋予的才是程序 编写者为变量分配的真正的初始值，执行静态代码块

## 使用

创建对象

1. **卸载**

对象失去引用，会被GC回收

# 类加载器的种类

### 启动类加载器(Bootstrap ClassLoader)

负责加载JRE的核心类库，如JRE目标下的rt.jar，charsets.jar等

### 扩展类加载器(Extension ClassLoader)

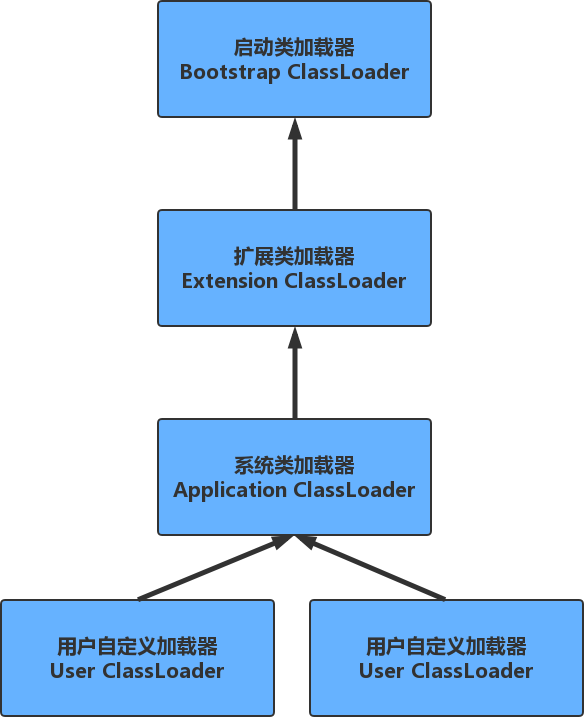
负责加载JRE扩展目录ext中jar类包

### 系统类加载器(Application ClassLoader)

负责加载ClassPath路径下的类包

### 用户自定义加载器(User ClassLoader)

负责加载用户自定义路径下的类包



# 类加载机制

### 全盘负责委托机制

当一个ClassLoader加载一个类的时候，除非显示的使用另一个ClassLoader，该类所依赖和引用的类也由这个

ClassLoader载入

### 双亲委派机制

指先委托父类加载器寻找目标类，在找不到的情况下载自己的路径中查找并载入目标类 **双亲委派模式的优势**

沙箱安全机制：比如自己写的String.class类不会被加载，这样可以防止核心库被随意篡改

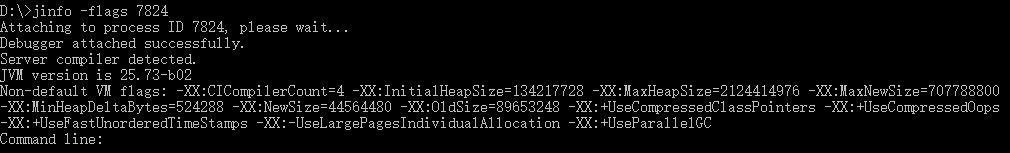
避免类的重复加载：当父ClassLoader已经加载了该类的时候，就不需要子CJlassLoader再加载一次

# JVM性能调优监控工具

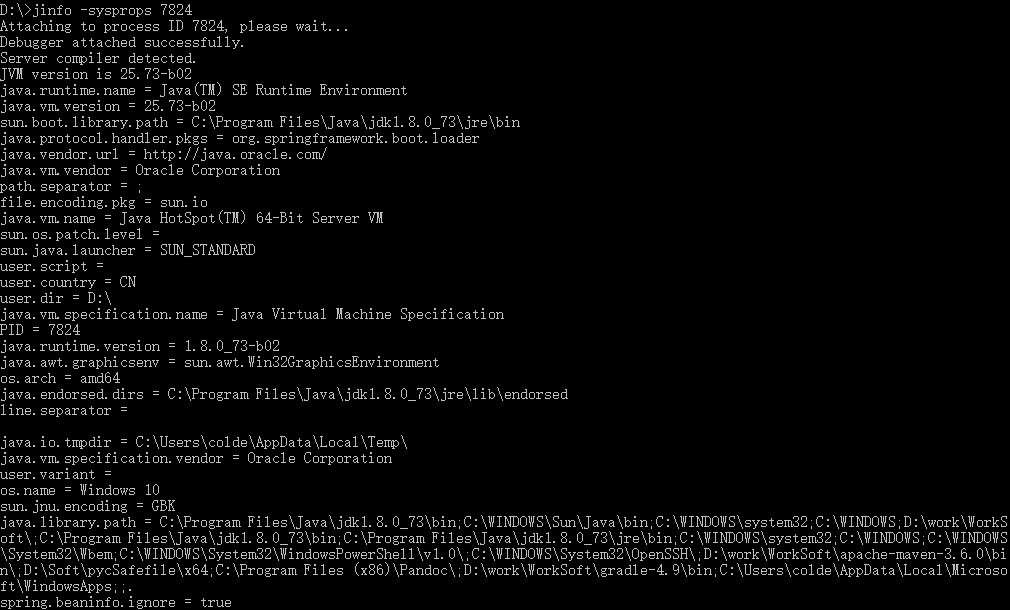
## Jinfo

查看正在运行的Java程序的扩展参数，查看Jvm的扩展参数

### 查看JVM的参数



**查看java系统属性**

等同于System.getProperties()

## Jstat

jstat命令可以查看堆内存各部分的使用量，以及加载类的数量。命令格式： jstat [-命令选项] [vmid] [间隔时间/毫秒] [查询次数]

### 类加载统计



Loaded：加载class的数量Bytes：所占用空间大小Unloaded：未加载数量Bytes：未加载占用空间Time：时间

### 垃圾回收统计



S0C：第一个Survivor区的空间S1C：第二个Survivor区的空间S0U：第一个Survivor区的使用空间S1U：第二个Survivor区的使用空间EC：Eden区的总空间

EU：Eden区的使用空间OC：Old区的总空间

OU：Old区的已使用空间MC：元空间的总空间MU：元空间的使用空间CCSC：压缩类的总空间CCSU：压缩类的使用空间YGC：年轻代垃圾回收次数

YGCT：年轻代垃圾回收消耗时间FGC：老年代垃圾回收次数FGCT：老年代垃圾回收消耗时间GCT：垃圾回收消耗总时间

### 堆内存统计



NGCMN：新生代最小空间NGCMX：新生代最大空间NGC：当前新生代空间S0C：第一个Survivor区空间S1C：第二个Survivor区空间EC：Eden区的总空间OGCMN：老年代最小空间OGCMX：老年代最大空间OGC： 当 前 老 年 代 空 间 OC：当前老年代空间MCMN：最小元空间大小MCMX：最大元空间大小

MC：当前元空间大小CCSMN：最小压缩类空间大小CCSMX：最大压缩类空间大小CCSC：当前压缩类空间大小YGC： 年 轻 代 GC 次 数 FGC：老年代GC次数

### 新生代垃圾回收统计

S0C： 第 一 个 Survivor 区 空 间 S1C：第二个Survivor区空间S0U：第一个Survivor区的使用空间S1U：第二个Survivor区的使用空间TT：对象在新生代存活的次数MTT：对象在新生代存活的最大次数DSS：期望Survivor区大小

EC：Eden区的空间

EU：Eden 区 的 使 用 空 间 YGC：年轻代垃圾回收次数YGCT：年轻代垃圾回收消耗时间

### 新生代内存统计



NGCMN：新生代最小空间NGCMX： 新 生 代 最 大 空 间 NGC： 当 前 新 生 代 空 间 S0CMX：最大第一个Survivor区空间S0C：当前第一个Survivor区空间S1CMX：最大第二个Survivor区空间S1C：当前第二个Survivor区空间ECMX： 最 大 Eden 区 空 间 EC： 当 前 Eden 区 空 间 YGC： 年 轻 代 垃 圾 回 收 次 数 FGC：老年代垃圾回收次数

### 老年代垃圾回收统计



MC：元空间的总空间MU：元空间的使用空间CCSC：压缩类的总空间

CCSU：压缩类的使用空间OC：Old区的总空间OU：Old区的已使用空间YGC：年轻代GC次数FGC：老年代GC次数

FGCT：老年代垃圾回收消耗时间GCT：垃圾回收消耗总时间

### 老年代内存统计



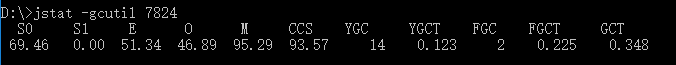
OGCMN：老年代最小空间OGCMX：老年代最大空间OGC： 当 前 老 年 代 空 间 OC： 当 前 老 年 代 空 间 YGC： 年 轻 代 GC 次 数 FGC： 老 年 代 GC 次 数 FGCT：老年代垃圾回收消耗时间GCT：垃圾回收消耗总时间

### 元空间内存统计



MCMN：最小元空间大小MCMX： 最 大 元 空 间 大 小 MC： 当 前 元 空 间 大 小 CCSMN：最小压缩类空间大小CCSMX：最大压缩类空间大小CCSC：当前压缩类空间大小YGC： 年 轻 代 GC 次 数 FGC： 老 年 代 GC 次 数 FGCT：老年代垃圾回收消耗时间GCT：垃圾回收消耗总时间

### 总垃圾回收统计



S0：第一个Survivor区当前使用比例S1：第二个Survivor区当前使用比例E：Eden区使用比例

O：Old区使用比例M：元空间使用比例

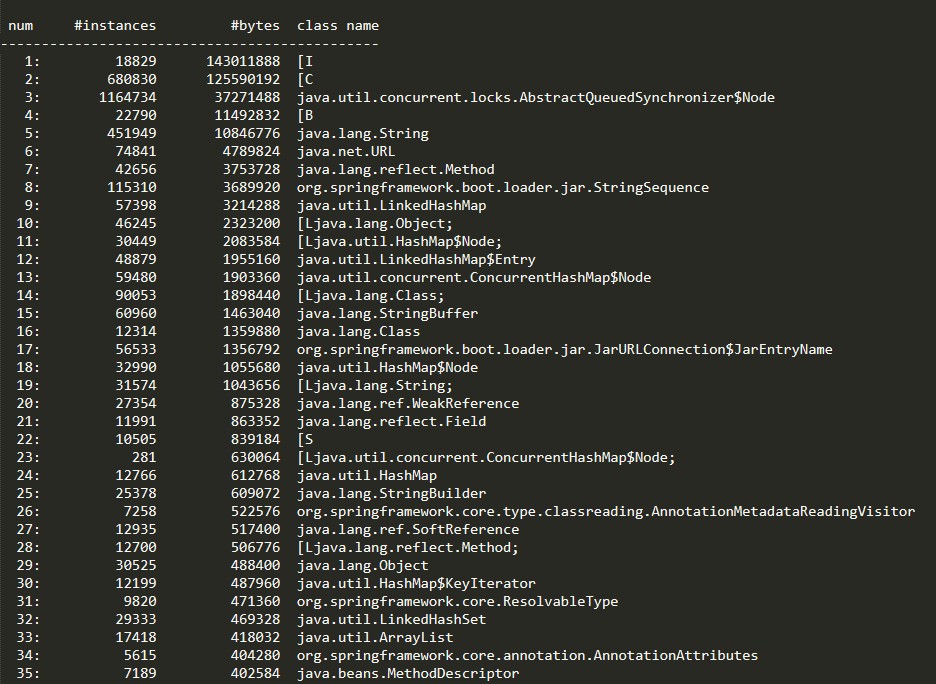
CCS： 压 缩 使 用 比 例 YGC：年轻代垃圾回收次数FGC：老年代垃圾回收次数FGCT：老年代垃圾回收消耗时间GCT：垃圾回收消耗总时间

## Jmap

可以用来查看内存信息

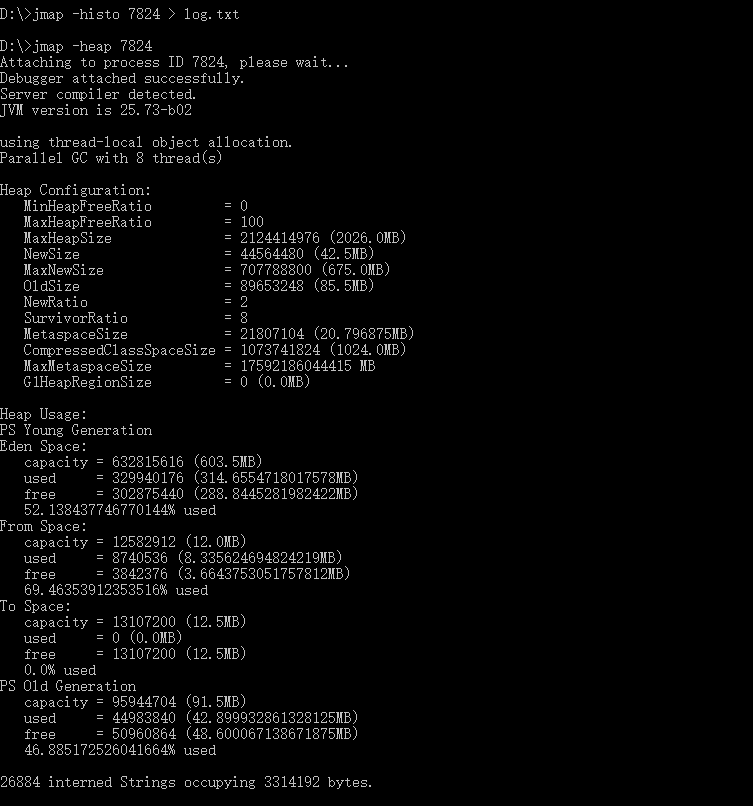
### 堆的对象统计

jmap -histo 7824 > xxx.txt

如图：

Num：序号Instances：实例数量Bytes：占用空间大小Class Name：类名

### 堆信息



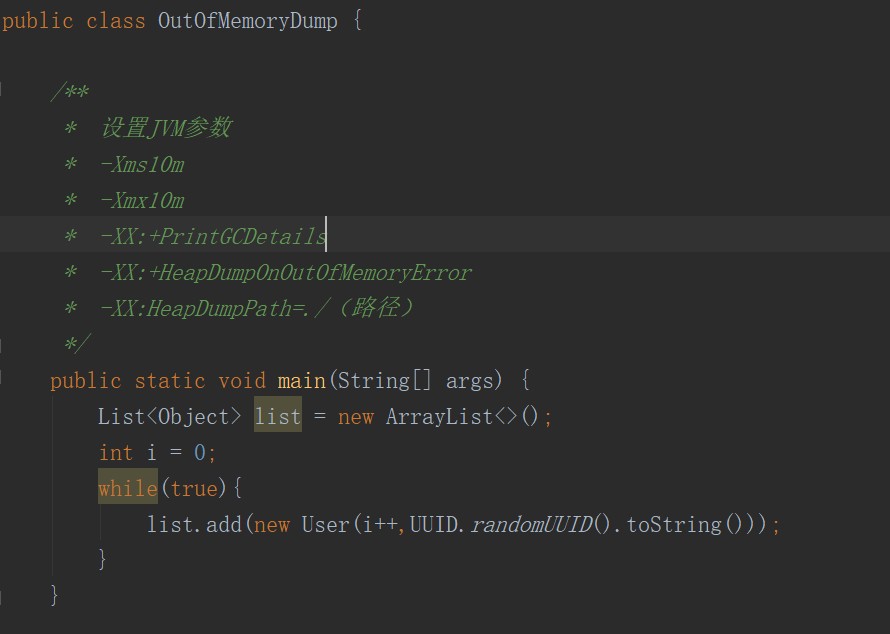
**堆内存dump**



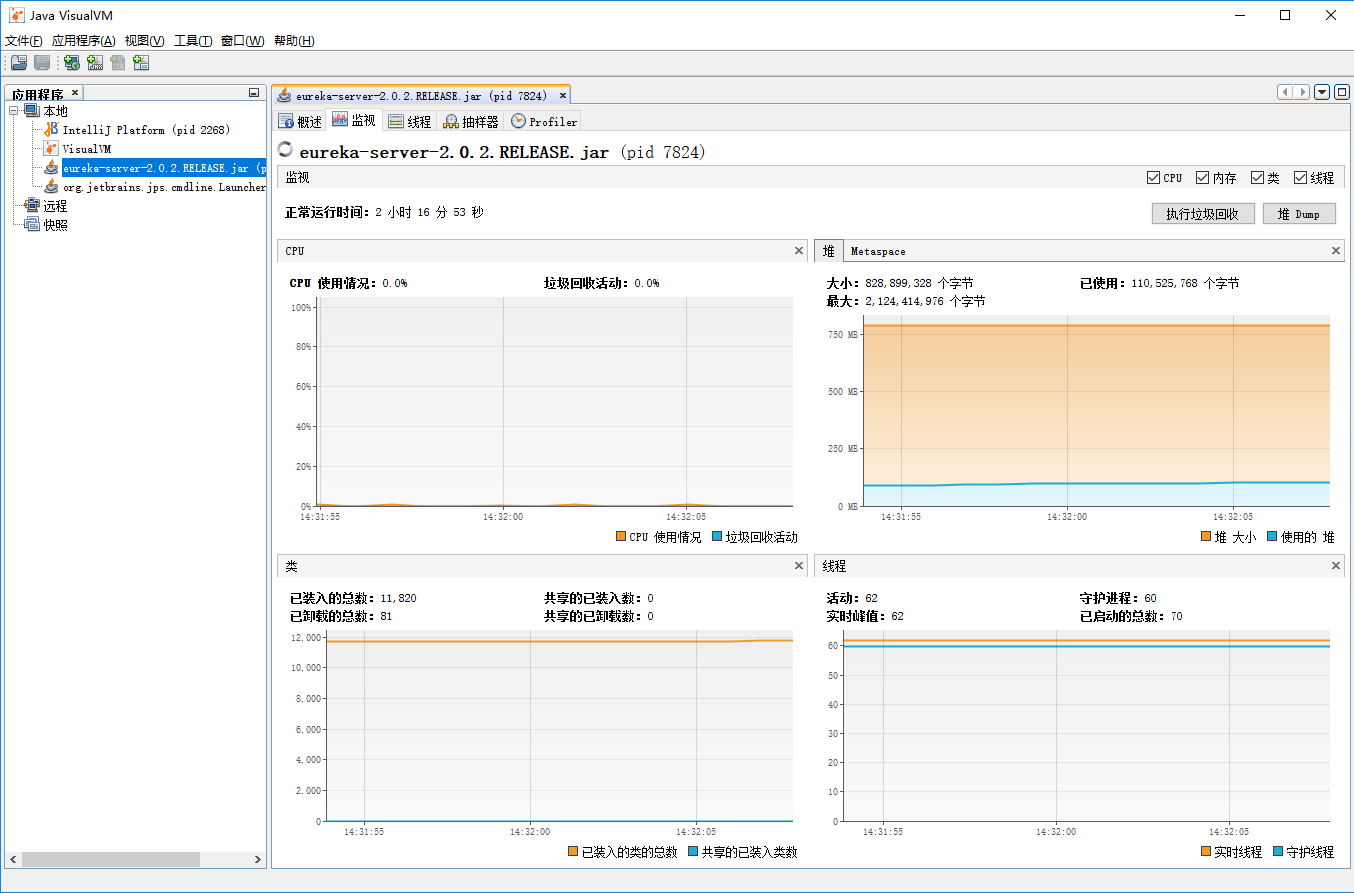
也可以在设置内存溢出的时候自动导出dump文件（内存很大的时候，可能会导不出来）

1.-XX:+HeapDumpOnOutOfMemoryError 2.-XX:HeapDumpPath=输出路径

-Xms10m -Xmx10m -XX:+PrintGCDetails -XX:+HeapDumpOnOutOfMemoryError - XX:HeapDumpPath=d:\oomdump.dump



可以使用jvisualvm命令工具导入文件分析



## Jstack

jstack用于生成java虚拟机当前时刻的线程快照。



## HSDB

运行HSDB工具，该工具可以查看代理类的情况

java -classpath "%JAVA\_HOME%/lib/sa-jdi.jar" sun.jvm.hotspot.HSDB